DISSERTATION

submitted to

the
Joint Faculties for Natural Sciences and Mathematics

of the
Ruprecht-Karls-Universität
Heidelberg, Germany

for the degree of
Doctor of Natural Sciences

presented by
Dipl. Phys. Andreas Grübl
born in Erlangen, Germany

Date of oral examination: July 4, 2007

VLSI Implementation of a Spiking Neural Network

Referees: Prof. Dr. Karlheinz Meier
Prof. Dr. René Schüffny

VLSI Implementierung eines pulsgekoppelten neuronalen Netzwerks

Im Rahmen der vorliegenden Arbeit wurden Konzepte und dedizierte Hardware entwickelt, die es erlauben, großskalige pulsgekoppelte neuronale Netze in Hardware zu realisieren. Die Arbeit basiert auf dem analogen VLSI-Modell eines pulsgekoppelten neuronalen Netzes, welches synaptische Plastizität (STPD) in jeder einzelnen Synapse beinhaltet. Das Modell arbeitet analog mit einem Geschwindigkeitszuwachs von bis zu 10^5 im Vergleich zur biologischen Echtzeit. Aktionspotentiale werden als digitale Ereignisse übertragen. Inhalt dieser Arbeit sind vornehmlich die digitale Hardware und die Übertragung dieser Ereignisse. Das analoge VLSI-Modell wurde in Verbindung mit Digitallogik, welche zur Verarbeitung neuronaler Ereignisse und zu Konfigurationszwecken dient, in einen gemischt analog-digitalen ASIC integriert, wobei zu diesem Zweck ein automatisierter Arbeitsablauf entwickelt wurde. Außerdem wurde eine entsprechende Kontrolleinheit in programmierbarer Logik implementiert und eine Hardware-Plattform zum parallelen Betrieb mehrerer neuronaler Netzwerkchips vorgestellt. Um das VLSI-Modell auf mehrere neuronale Netzwerkchips ausdehnen zu können, wurde ein Routing-Algorithmus entwickelt, welcher die Übertragung von Ereignissen zwischen Neuronen und Synapsen auf unterschiedlichen Chips ermöglicht. Die zeitlich korrekte Übertragung der Ereignisse, welche eine zwingende Bedingung für das Funktionieren von Plastizitätsmechanismen ist, wird durch diesen Algorithmus sichergestellt. Die Funktionalität des Algorithmus wird mittels Simulationen verifiziert. Weiterhin wird die korrekte Realisierung des gemischt analog-digitalen ASIC in Verbindung mit dem zugehörigen Hardware-System demonstriert und die Durchführbarkeit biologisch realistischer Experimente gezeigt. Das vorgestellte großskalige physikalische Modell eines neuronalen Netzwerks wird aufgrund seiner schnellen und parallelen Arbeitsweise für Experimentierzwecke in den Neurowissenschaften einsetzbar sein. Als Ergänzung zu numerischen Simulationen bietet es vor allem die Möglichkeit der intuitiven und umfangreichen Suche nach geeigneten Modellparametern.

VLSI Implementation of a Spiking Neural Network

Within the scope of this thesis concepts and dedicated hardware have been developed that allow for building large scale hardware spiking neural networks. The work is based upon an analog VLSI model of a spiking neural network featuring an implementation of spike timing dependent plasticity (STDP) locally in each synapse. Analog network operation is carried out up to 10^5 times faster than real time and spikes are communicated as digital events. This work focuses on the digital hardware and the event transport. Along with digital logic for event processing and configuration purposes, the analog VLSI model has been integrated into a mixed-signal ASIC by means of an automated design flow. Furthermore, the accompanying controller has been realized in programmable logic, and a hardware platform capable of hosting multiple chips is presented. To extend the operation of the VLSI model to multiple chips, an event routing algorithm has been developed that enables the communication between neurons and synapses located on different chips, thereby providing correct temporal processing of events which is a basic requirement for investigating temporal plasticity. The functional performance of the event routing algorithm is shown in simulations. Furthermore, the functionality of the mixed-signal ASIC along with the hardware system and the feasibility of biologically realistic experiments is demonstrated . Due to its inherent fast and parallel operation the presented large scale physical model of a spiking neural network will serve as an experimentation tool for neuroscientists to complement numerical simulations of plasticity mechanisms within the visual cortex while facilitating intuitive and extensive parameter searches.

Contents

Introduction

Understanding the human brain and the way it processes information is a question that challenges researchers in many scientific fields. Different modeling approaches have been developed during the past decades in order to gain insight into the activity within biological nervous systems. A first abstract description of neural activity was given by a mathematical model developed by McCulloch and Pitts in 1943 [MP43]. Even though it has been proven that every logical function could be implemented using this binary neuron model, it was not yet possible to reproduce the behavior of biological systems. On the way to getting closer to biology, the introduction of the Perceptron by Rosenblatt in 1960 [Ros60] can be seen as the next major step, since this model provides the evaluation of continuously weighted inputs, thereby yielding continuous output values instead of merely binary information.

The level of biological realism was once again raised by the introduction of neuron models using individual spikes instead of the static communication inherent to the afore existing models. Spiking neuron models eventually allow incorporating spatio-temporal information in communication and computation just like real neurons [FS95].

However, biological realism and the complexity of the model description increased at the same time. One of the most accurate models available has been already proposed by Hodgkin and Huxley in 1952 [HH52] and describes the functionality of the neural network by means of a set of differential equations. Many spiking neuron models are based on this early work and it is the fast development of digital computers that nowadays facilitates the numerical simulation of such complex models at a feasible speed, thus, within acceptable time.

To investigate the key aspect regarding the understanding of the brain, namely the process of development and learning, it is necessary to model the behavior of adequately complex and large neural microcircuits over long periods of time. Moreover, it is of importance to be able to tune different parameters in order to test the level of biological accordance and thereby the relevance for biological research. Both aspects require long execution times, even on the fastest currently available microprocessors.

Developing a physical model, which mimics the inherent parallelism of information processing within biological nervous systems, provides one possibility to overcome this speed limitation. The hitherto only known system used for these purposes is analog very large scale integration (VLSI) of complementary metal oxide semiconductor (CMOS) devices, the latter also being used for the realization of modern microprocessors. Microelectronic circuits can be developed, which mimic the electrical behavior of the biological example. Thereby, important physiological quantities in terms of currents and device parameters like capacitances or conductances can be assigned to corresponding elements within the physical model.

Several VLSI implementations of spiking neuron models have been reported for example by Häflinger et al. [HMW96] and Douence et al. [DLM$^+$99]. In both approaches the motivation is not primarily the operating speed, but rather the accuracy of the continuous time behavior of the model.

A new approach is based on the research carried out within the Electronic Vision(s) group at the Kirchhoff Institute for Physics located in Heidelberg [SMM04]. The analog VLSI architecture implements a neuron model representing most of the neuron types found within the visual cortex. Moreover, the synapse model includes plasticity mechanisms allowing the investigation of long term and short term developmental changes. One important advantage of the physical model compared with the biological example can be clearly pointed out: its operating speed. Due to device physics, the analog neuron and synapse implementations operate at a factor of up to 10^5 times faster than biological real time.

Spikes are communicated between neurons and synapses as digital pulses and no information is encoded in their size and shape. Instead, information is encoded in the temporal intervals in which action potentials are generated by the neurons or transported by the synapses. In particular, the mutual correlation between spikes received and generated by a neuron contributes to plasticity mechanisms that are supposed to be one of the keys towards understanding how learning processes in the brain work. The digital nature of the spikes enables their transport over long distances using digital communication techniques and does not restrict the size of the neural network to one single application specific integrated circuit (ASIC).

The fact that information is supposed to be encoded in the spike timing arises high demands on the digital communication. Neural events are not only to be communicated based on their location (address) within the network, but also the point in time of their occurrence has to be correctly modeled by the digital communication to reflect the spatio-temporal behavior of the biological example. Thereby, the speed-up factor of 10^5 requires a temporal resolution in the order of magnitude of less than 1 ns. Long-range digital communication at this level of accuracy cannot be realized by asynchronous communication techniques as utilized for example in [HMW96]. For the latter reasons, it is required to operate the entire neural network in a synchronous system and on the basis of a global system time. As a consequence, the continuous time operation of the model has to be ensured for a system comprising several digitally interconnected analog VLSI components. Innovative strategies are required to keep track of event timing and to transfer events between the continuous time domain of the analog VLSI implementation and the clocked time domain of the digital communication.

It is especially important that these strategies are not restricted to specific network topologies. Instead, it should be possible to implement different biological examples in order to provide flexible modeling. Furthermore, the realization of recurrent networks is of special interest. Recurrent connections make the network response dependent on its history, allowing for regulatory cycles as well as short-term memories. Specifically, the essential difference is that feed-forward networks are static mappings from an input to an output, while recurrent networks are dynamical systems. While the analog VLSI implementation already features asynchronous local feedback, recurrent connections between different chips require the design of carefully timed bidirectional communication channels. To provide this functionality and facilitate the setup of biologically realistic experiments, the connections provided by these communication channels have to feature a guaranteed and fixed delay at an appropriate speed. By this means, also the successful operation of the STDP implementation is assured and the investigation of temporal plasticity is made possible.

Since the hardware model will only cover cutouts of biological neural systems, biological realism requires one more feature: artificial stimuli, e.g. external input, along with background neural activity from the biological surrounding have to be generated. Furthermore, the emerging neural activity as well as the outcome of ongoing plasticity mechanisms has to be recordable to eventually provide ways of analyzing and understanding the activity of the neural network.

The aim of the work described within this thesis is the development of the components that constitute large scale spiking neural networks based on the analog VLSI implementation of the physical model. In conjunction with the model itself, the presented work will enable the realization of spiking neural networks comprising multiple chips with a total number of neurons in the order of magnitude of 10^4 neurons and more than 10^6 synapses.

The thesis is structured into six chapters: the first chapter gives a short overview of the biological background and introduces the analog VLSI model. Thereby, expected communication demands are introduced defining the requirements on the event communication. Chapter 2 describes the design methodology that has been developed for the integration of the analog VLSI model and accompanying digital logic into the entire mixed-signal chip presented within this thesis.

An algorithm suited for the transport of neural events between different chips is described in chapter 3, together with a hardware platform for the operation of multi-chip networks. Parts of this algorithm are realized on the presented chip whose actual implementation is described in chapter 4 with an emphasis on the event transport and the digital functionality.

The realized operating environment for one chip including hardware platform, programmable logic and software is described in chapter 5. Finally, chapter 6 deals with the measurements that have been carried out to characterize the functionality of the developed digital control logic and the event transport functionality within the fabricated chip. An initial biologically realistic experiment is presented demonstrating the successful implementation of the chip as well as the entire framework.

Chapter 1

Artificial Neural Networks

This chapter describes the neural model and its analog VLSI implementation within the ASIC presented in this thesis. The first section introduces the biological background and gives reasons for the selection of the particular model. In the second section, the neuron model and the synapse model, including plasticity mechanisms, are described. Furthermore, the physical implementation is outlined as it serves as a basis for the development of the mixed-signal ASIC and the neural event transport strategies developed within this thesis. The third section discusses the analog VLSI implementation, which integrates conductance based integrate-and-fire neurons operating with a speed several orders of magnitude larger than real time, whereas the neural events are communicated digitally. The dynamics exhibited by bursting neural networks or networks being in a high conductance state are discussed and important parameters regarding bandwidth and latency of the communication are derived as they will serve as constraints for the digital transport of the events.

1.1 Biological Background

The computational power of biological organisms arises from systems of massively interconnected cells, namely neurons. These basic processing elements build a very dense network within the vertebrates' brain (in Latin: *cerebrum*). Most of the cerebral neurons are contained within the *cerebral cortex* that covers parts of the brain surface and occupies an area of about $1.5\,\mathrm{m}^2$ due to its nested and undulating topology. In the human cortex, the neuron density exceeds 10^4 neurons per cubic millimeter and each neuron receives input from up to 10,000 other neurons, with the connections sometimes spread over very long spatial distances.

An overview of a neuron is shown in figure 1.1 a. The typical size of a mammal's neuronal cell body, or *soma*, ranges from 10 to $50\,\mu\mathrm{m}$ and it is connected to its surroundings by a deliquesce set of wires. In terms of information, the *dendrites* are the inputs to the neuron, which has one output, the *axon*. Axons fan out into axonic trees and distribute information to several target neurons by coupling to their dendrites via *synapses*.

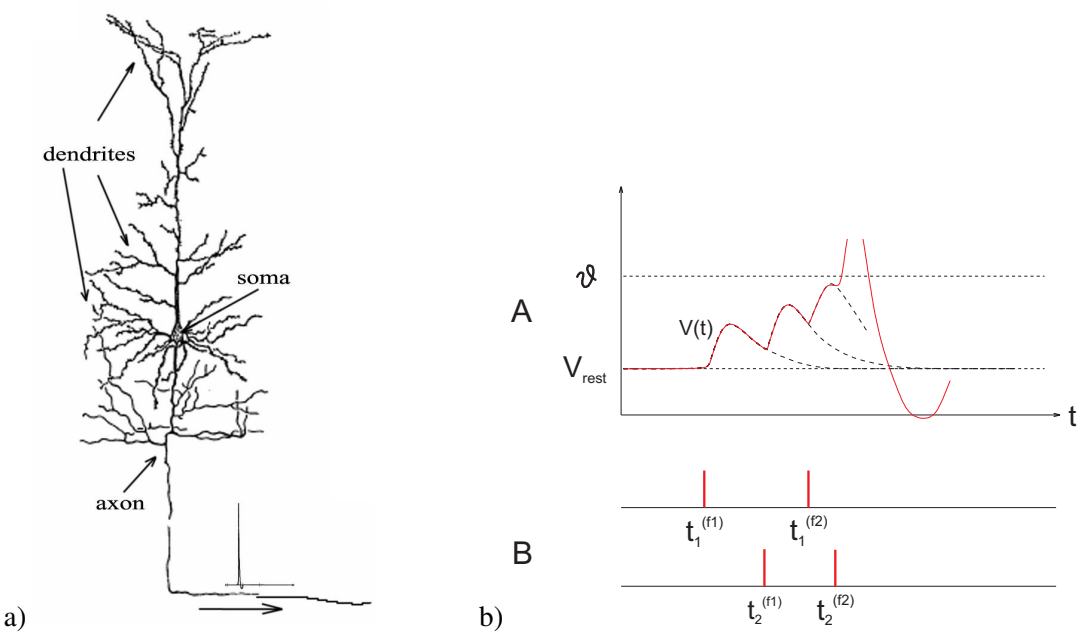


Figure 1.1: a) Schematic drawing of a neuron after Ramón y Cajal; dendrites, soma and axon can clearly be distinguished. Figure taken out of [Ger99]. b) Membrane potential $V(t)$ (A) depending on presynaptic inputs (B). Many postsynaptic potentials (PSPs) are superimposed and eventually make $V(t)$ cross the spiking threshold ϑ. In this case an action potential (which exceeds the scale of the plot) is fired. Afterwards $V(t)$ runs through a phase of hyper-polarization. Figure adapted from [Brü04].

The synapses act as a preprocessor of the information arriving at a neuron's dendrite in terms of modulating the effect on the postsynaptic neuron, which is located after the synapse. Modulating in this sense means weighting the presynaptic input in terms of its strength and also its sign. The term *sign* requires a closer look at the cell membrane of the neuron that divides the intracellular from the extracellular space and has a certain membrane capacitance C_{m}. Without any external input, the potentials on either side of the membrane differ and the *resting potential* V_{rest} is developed over C_{m}. It represents the steady state of concurring ion channels increasing, respectively decreasing, the *membrane potential* which is also called *polarization* in neuroscience. External input that increases the membrane potential is said to be *depolarizing*, the according synapse with a positive sign is *excitatory*. The contrary process is called *hyper-polarization* and the according synapse is *inhibitory*.

The functionality of a neural system strongly depends on the configuration of its synapses. The possibility of dynamically changing the synapse weights is called *plasticity*. This change in the effect of a presynaptic signal on the postsynaptic neuron forms the basis of most models of learning and development of neural networks.

Neurons communicate with action potentials, or spikes, which is illustrated in figure 1.1 b. Such a spike is a short (1 ms) and sudden increase in voltage that is created in the soma and travels down the axon. Reaching a synapse, the spike triggers a change in the synapse's post-synaptic potential (PSP) which adds to the membrane voltage and then slowly decays with time. Several PSPs eventually rise the membrane voltage over a threshold ϑ. In this case, the neuron itself generates an action potential and after some time of *hyper-polarization* the membrane returns to the resting potential, if no further input is provided. As the spikes sent

out by one neuron always look the same, the transported information is solely coded within their firing times.

1.1.1 Modeling Biology

Modeling of neural systems can be classified into three generations as proposed by Maass in his review on networks of spiking neurons [Maa97]. These classes of models are distinguished according to their computational units and the term *generations* also implies the temporal sequence in which they have been developed.

The *first generation* is based on the work of McCulloch and Pitts [MP43] who proposed the first neuron model in 1943. The McCulloch-Pitts neurons are a very simplified model of the biological neuron that accept an arbitrary number of binary inputs and have one binary output. Inputs may be excitatory or inhibitory and the connection to the neuron is also established by weighted synapses. If the sum over all inputs exceeds a certain threshold, the output becomes active. Many network models have their origin in McCulloch-Pitts neurons, such as multilayer perceptrons [MP69] (also called threshold circuits) or Hopfield nets [Hop82]. It was shown by McCulloch and Pitts that already the binary threshold model is *universal* for computations with digital input and output, and that every boolean function can be computed with these nets.

The *second generation* of neuron models expands the output of the neuron in a way, that an *activation function* is applied to the weighted sum of the inputs. The activation function has a continuous set of possible output values, such as *sigmoidal functions* or piecewise linear functions. Past research work done within the Electronic Vision(s) group has focused on multi-layer perceptron experiments that were performed on the basis of the HAGEN chip [SHMS04] developed within the group. See for example [Hoh05, Sch05, Sch06].

Both, first and second generation models work in a time discrete way. Their outputs are evaluated at a certain point in time and the temporal history of the inputs is neglected during this calculation. For an approximate biological interpretation of neural nets from the second generation, the continuous output values may be seen as a representation of the firing rate of a biological neuron. However, real neurons do code information within spatio-temporal patterns of spikes, or action potentials [Maa97]. Knowing this, a spike-based approach, which predicts the time of spike generation without exactly modeling the chemical processes on the cell membrane, is a viable approach to realize simulations of large neuron populations with high connectivity. The integrate-and-fire model [GK02] follows this approach and reflects the temporal behavior of biological neurons. The *third generation* of neuron models covers all kinds of spiking neural networks exhibiting the possibility of temporal coding. The analog VLSI implementation of a modified integrate-and-fire model will serve as a basis for the work described in this thesis. Note that a quantitative neuron model has been developed by Hodgkin and Huxley in 1952 [HH52]. This model uses a set of differential equations to model the current flowing on the membrane capacitance of a neuron. It has been shown by E. Mueller, that it is possible to fit the behavior of the integrate-and-fire model to that of the Hodgkin-Huxley model with regard to the timing of the generated action potentials [Mue03].

1.2 The Utilized Integrate-and-Fire Model

In this section, the selected model for the VLSI implementation is described. It is based on the standard integrate-and-fire model and allows the description of most cortical neuron types, while neglecting their spatial structure. The selection of the model is a conjoint work of E.

Mueller and Dr. J. Schemmel, whereas the analog implementation was carried out by Dr. J. Schemmel. The integration within the presented ASIC is one subject to this thesis.

To be able to predict the requirements for the interfaces to the VLSI circuits as well as the the communicational needs in the entire system, the dynamics of the neural network are discussed after the model description.

1.2.1 Neuron and Synapse Model

The chosen model, in accordance to the standard integrate-and-fire model, describes the neuron's soma as a membrane with capacitance C_m in a way that a linear correspondence exists between the biological and the model membrane potential V. If the membrane voltages reaches a threshold voltage V_th, a spike is generated, just as in the case of the biological neuron. The following effect of hyper-polarization is modeled by setting the membrane potential to the reset potential V_reset for a short time, e.g. the refractory period, where the neuron does not accept further synaptic input. The simple threshold-firing mechanism of the integrate-and-fire model is not adequate to reflect the near-threshold behavior of the membrane. The very near-threshold behavior has been observed in nature [DRP03]. Therefore, the circuit has been designed such that the firing mechanism not only depends on the membrane voltage, but also on its derivative.

In biology, spikes are generated by ion channels coupling to the axon. In contrast to this, the VLSI model contains an electronic circuit monitoring the membrane voltage and triggering the spike generation. To facilitate communication between the neurons, the spike is transported as a digital pulse. As it will be described in the following section, these pulses are either directly connected to other neurons on the same die which preserves the time continuous, asynchronous operation of the network. Furthermore, the time of spike generation may be digitized and the spike may be transported using synchronous digital communication techniques. Regardless of the underlying communication, the digitized neuron outputs are connected to the membrane of other neurons by conductance based synapses. Upon arrival of such a digital spike, the synaptic conductance follows a time course with an exponential onset and decay.

Within the selected model, the membrane voltage V writes

$$C_\mathrm{m}\frac{dV}{dt} = g_\mathrm{leak}\,(V-E_\mathrm{l}) + \sum_j p_j(t)\,g_j(t)\,(V-E_\mathrm{x}) + \sum_k p_k(t)\,g_k(t)\,(V-E_\mathrm{i})\,. \tag{1.1}$$

The constant C_m represents the total membrane capacitance. Thus the current flowing on the membrane is modeled multiplying the derivative of the membrane voltage V with C_m. The conductance g_leak models the ion channels that pull the membrane voltage towards the leakage *reversal potential*[1] E_l. The membrane finally will reach this potential, if no other input is present. Excitatory and inhibitory ion channels are modeled by synapses connected to the excitatory and the inhibitory reversal potentials E_x and E_i respectively. By summing over j, all excitatory synapses are covered by the first sum. The index k runs over all inhibitory synapses in the second sum. The time course of the synaptic conductances is controlled by the parameters $p_{j,k}(t)$. To facilitate the investigation of the temporal development of the neural network model, two plasticity mechanisms are included in the synaptic conductances $g_{j,k}(t)$, which are modeled by

$$g_{j,k}(t) = \omega_{j,k}(t)\cdot g_{j,k}^\mathrm{max}(t)\,, \tag{1.2}$$

[1] The reversal potential of a particular ion is the membrane voltage at which there is no net flow of ions from one side of the membrane to the other. The membrane voltage is pulled towards this potential if the according ion channel becomes active.

with the relative synaptic weight $\omega_{j,k}(t)$ and a maximum conductance of $g_{j,k}^{\max}(t)$. Developmental changes (like learning) within the brain, or generally within a neural network, are described by plasticity mechanisms within the neurons and synapses. The herein described model includes two mechanisms of synaptic plasticity: STDP and short term synaptic depression and facilitation. The actual implementation of these models has already been published in [SGMM06] and [SBMO07]. To motivate the need for precise temporal processing of neural events the STDP mechanism will be described in the following.

Spike Timing Dependent Plasticity (STDP)

Long term synaptic plasticity is modeled by an implementation of STDP within each synapse. It is based on the biological mechanism as described in [BP97, SMA00]. Synaptic plasticity is herein realized in a way that each synapse measures the time difference Δt between pre- and postsynaptic spikes. If $\Delta t < 0$, a causal correlation is measured (i.e. the presynaptic signal contributed to an output spike of the according neuron) and the synaptic weight is increased depending on a modification function. For acausal correlations the synaptic weight is decreased. The change of the synaptic weight for each pre- or postsynaptic signal is expressed by a factor $1 + F(\Delta t)$. F is called the STDP modification function and represents the exponentially weighted time difference Δt. It is defined as follows:

$$F(\Delta t) = \begin{cases} A_+ \exp(\frac{\Delta t}{\tau_+}) & \text{if } \Delta t < 0 \quad \text{(causal)} \\ -A_- \exp(-\frac{\Delta t}{\tau_-}) & \text{if } \Delta t > 0 \quad \text{(acausal)} \end{cases} \tag{1.3}$$

1.2.2 Terminology

Two terms regarding neural activity, which will be used throughout the thesis, shall be defined in this section: *spike train* and *firing rate*

Spike Train

According to the definition in [DA01] action potentials are typically treated as identical stereotyped events or spikes, although they can vary somewhat in duration, amplitude, and shape. Ignoring the brief duration of an action potential (about 1 ms), a sequence of action potentials can be characterized simply by a list of the times when spikes occurred. Therefore, the spike train of a neuron i is fully characterized by the set of firing times

$$F_i = \{t_i^{(1)}, ..., t_i^{(n)}\} \tag{1.4}$$

where $t_i^{(n)}$ is the most recent spike of neuron i. Figure 1.2 illustrates the activity recorded (the spike trains) from 30 arbitrarily selected neurons within the visual cortex of a monkey.

Firing Rate

Considering the large number of neurons in the brain it is nearly impossible to evaluate every single spike with respect to its exact timing. On this account, it has traditionally been thought that most, if not all, of the relevant information was contained in the mean firing rate of the neuron which is usually defined as a temporal average. Within a time window $T = 100\,\text{ms}$ or

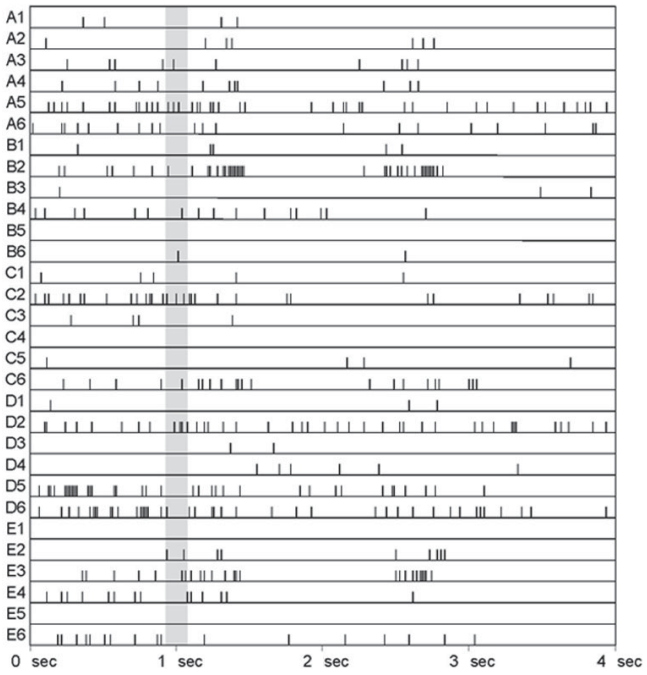


Figure 1.2: Spatio-temporal pulse pattern. The spikes of 30 neurons (A1-E6, plotted along the vertical axes) out of the visual cortex of a monkey are shown as a function of time (horizontal axis, total time is 4 seconds). The firing times are marked by short vertical bars. The grey area marks a time window of 150 ms. Within this time, humans are able to perform complex information processing tasks, like face recognition. Figure taken from Krüger and Aiple [KA88].

$T = 500\,\text{ms}$ the number of occurring spikes $n_{\text{sp}}(T)$ is counted. Division by the length of the time window gives the mean firing rate

$$f = \frac{n_{\text{sp}}(T)}{T} \tag{1.5}$$

in units of s^{-1} or Hz. More definitions of firing rates can be found in [DA01], where the firing rate is approximated by different procedures, e.g. by using a Gaussian sliding window function. To evaluate the performance of the event processing techniques introduced within this thesis, the definition in equation 1.5 will be used.

Note that it cannot be all about firing rates. The boxed grey area in figure 1.2 denotes $T = 150\,\text{ms}$ and obviously no mean firing rate can be given for this time window. Nevertheless is the brain capable of performing tasks like face recognition already within this time span. Consequently, the spatio-temporal correlation of the single spikes has to be considered when modeling a nervous system. While this is fulfilled by fact for the integrate-and-fire model itself, it is important to keep in mind that the underlying event transport mechanisms need to provide transport latencies that preserve the latencies introduced by biological synaptic connections.

1.2.3 Expected Neural Network Dynamics

The aim to model biological neural systems with dedicated hardware on the one hand requires the selection of a specific model which has been described in the preceding section. On the

other hand, the communication of neural events within the system has to be considered. In the selected model spikes are communicated as digital pulses between the neurons. The development of appropriate communication channels requires the knowledge of spike rates that are to be expected and other temporal constraints that arise from the topology of the biological system. Three items are selected to serve as basic constraints for the work presented within this thesis. Regarding spike rates neurons being in a *high-conductance state* and synchronized behavior of neural networks are selected to account for average and peak spike rates. Furthermore, connection delays observed in nature are taken as a minimum delay constraint for the different communication channels.

The High-Conductance State

The term *high-conductance state* describes a particular state of neurons within an active network, as typically seen in vivo, such as for example in awake and attentive animals. A single neuron is said to be in the high conductance state, if the total synaptic conductance received by the neuron is larger than its leakage conductance. First in vivo measurements proving the existence of the high-conductance state have been performed by Woody et al. [WG78]. In a review paper by Destexhe et al. [DRP03] the mean firing rate f_{hc} of a neuron being in the high conductance state is found to be:

$$5\,\mathrm{Hz} \;<\; f_{\mathrm{hc}} \;<\; 40\,\mathrm{Hz}\,. \tag{1.6}$$

Synchronized Behavior

Synchronized behavior is used in this context to describe the coherent spike generation of a set of neurons. Periodic synchronization has been observed within simulations by E. Mueller [MMS04] at a rate of $5-7\,\mathrm{Hz}$ within a network of 729 neurons. Stable propagation of synchronized spiking has been demonstrated by Diesmann et al. also by means of software simulations [DGA99]. To estimate the consequences of this behavior on the transport of the digitized events within a digital network (cf. chapter 3), the synchronized behavior is modeled within this thesis using a rate-based approach[2]. For a set of neurons, an average firing rate f_{av} is assumed for the time t_{off} in between the synchronized spikes and a firing rate of f_{peak} during synchronized spiking, which lasts for t_{on}. The values for f_{peak} and t_{on} will be chosen in a way that the neuron generates at least one spike within t_{on}.

Connection Delays

The connection delay between two biological neurons comprises the signal propagation delay along the axon of the neuron and the delay through the synapse with the subsequently connected dendrite. Depending on the spatial structure of a neural network, the delay between different neurons varies with their relative distance. A physical model that is based on analog VLSI circuits and the transport of digitized events, exhibits the same behavior and its delay should be tunable to reflect the biological values.

Specific delay values strongly depend on the spatial structure of the simulated network, i.e. the benchmark for the hardware model. On the one hand, minimum delay values for synaptic transmission of 1–2.5 ms have for example been observed during in vitro measurements of the rat's cortex by Schubert et al. [SKZ+03]. On the other hand, during software simulations

[2] Spikes are generated with a probability p at a point in time. p is the average firing rate per unit time. Spike generation is uncorrelated to previous spikes (intervals are poisson distributed).

commonly used delay values are in the range of 0.1–1.5 ms for networks with a size of approximately 10^3 neurons [Mul06]. As a consequence of these figures, it can be said that hardware simulated delays should continuously cover possible delay values from approximately 0.1 ms upwards. Considering a speed-up factor of 10^5 for the hardware, this yields a minimum delay requirement of approximately 1 ns.

1.3 VLSI Implementation

We will now have a closer look at the actual implementation of the network model in analog VLSI hardware. It has already been stated in the previous section that parts of the circuits have already been published. However, a concise description of the whole model and especially of the neuron and synapse circuits is in preparation by Dr. J. Schemmel et al. and will therefore be omitted here. The following qualitative description is intended to give an outline of what served as a basis for the concepts developed, and the ASIC design that has been done throughout this thesis work.

1.3.1 Neuron Functionality

The operating principle of the neuron and synapse circuits is shown in figure 1.3. Circuits for synaptic short term plasticity are located within the synapse drivers and are omitted for simplicity. The circuits implementing STDP are located within each synapse and implement equation 1.3 all in parallel. They are also omitted for reasons of simplicity. The circuit topology is an array of synapses where every column of synapses connects to one neuron circuit. Each neuron circuit contains a capacitance C_m that physically represents the cell membrane capacitance. Three conductances model the ion channels represented by the three summands in equation 1.1. The reversal potentials E_l, E_x and E_i are modeled by voltages that are common for groups of several neurons each. This is the same for the threshold and the reset voltages V_th and V_reset. The membrane leakage current is controlled by g_leak which can be set for each neuron.

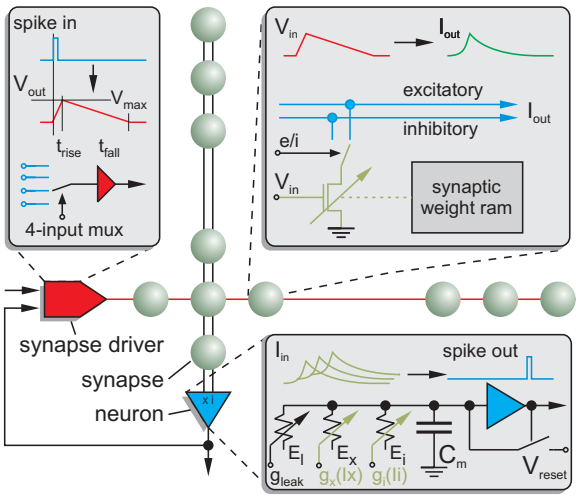


Figure 1.3: Block diagram of the neuron and synapse functionality in the analog VLSI implementation. Figure taken out of [SGMM06].

In biological systems the synapse conductances are controlled by a chemical process within each synapse. In contrast to this, the analog VLSI implementation follows a different approach: the synapses produce a current output and the conductances are separately controlled for all excitatory and all inhibitory synapses at once. Two conductances, $g_x(I_x)$ for the excitatory and $g_i(I_i)$ for the inhibitory synapses, are located within the neuron. Thereby the current generated by the active synapses within the array is summed to an excitatory sum I_x and an inhibitory sum I_i. Two wires exist in each column to collect these current sums. The type of synapse can be set row wise and the synapses of one row all add their output current to the same wire.

The spike generation process is triggered by a comparator that compares the membrane voltage with the threshold voltage. If a spike is generated, the membrane is pulled to the reset potential for a short period of time which hyper-polarizes the membrane in accordance with the integrate-and-fire model. In contrast to biology, the axon of the neuron is electrically isolated from the membrane and carries a digital pulse after the spike has been generated. The comparator is tuned in a way that its dependency on the derivative of the membrane voltage makes its behavior resemble that of the Hodgkin-Huxley model [SMM04].

1.3.2 Synapse Functionality and Connectivity

Due to the array structure, the number of synapses determines the area demand of the VLSI implementation. The size of a single synapse is therefore kept small by relocating some of its functionality into the row drivers that are placed at the left edge of the array, as illustrated in figure 1.3. As a result, the presynaptic signal is common for one synapse row and all associated neurons simultaneously receive the postsynaptic signals generated by the different synapses. The connectivity inside the array is illustrated in figure 1.4: presynaptic signals are shown as dotted lines and are horizontally routed through the array. Synapses are drawn as spheres and connect the presynaptic signal to the postsynaptic signal within each column. The postsynaptic signals use dashed arrows and are vertically routed to the neuron below. Only one of the two wires for I_x and I_i is displayed for simplicity.

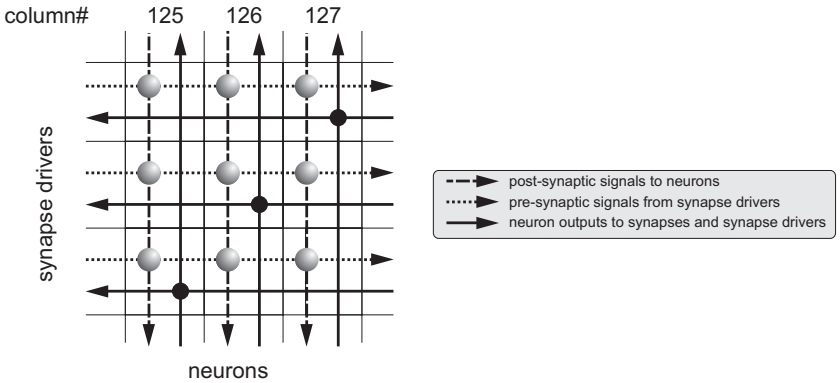


Figure 1.4: Connection scheme used within the synapse array. A cutout of rows 125 to 127 is shown, which have been selected arbitrarily. Figure adapted from [SMM04].

The axon of the neuron is routed back into the array. On the one hand this allows the STDP circuits within the synapses to measure the time distance between the presynaptic and the postsynaptic signal. On the other hand, a horizontal line in each row connects the neuron with the corresponding number (the connection points are set on the diagonal of the array) to the

synapse driver at the edge. The input to the synapse driver can be selected out of this signal or three different external sources which is denoted by the 4-input multiplexer in figure 1.3. Further details regarding this connectivity will be given in section 4.2.2.

The synapse driver converts a digital spike to a triangular voltage slope with adjustable rise and fall time and adjustable peak voltage. This ramp is converted to a current slope with exponential onset and decay by a current sink within each synapse. The current sink consists of 15 n-type metal oxide semiconductor (NMOS) transistors and the overall strength of the synaptic current is controlled by a 4 bit static weight memory within each synapse. Depending on the stored weight, the according number of transistors are activated and add to the synaptic current. As a result, the synaptic weight can be set with a resolution of 4 bit. A control line connected to the synapse driver sets the complete row to add its current to the excitatory or the inhibitory line.

The storage of the synaptic weight within a digital memory requires a larger area than capacitance based solutions which have been used within a previous ASICs developed within the Electronic Vision(s) group [SMS01, SHMS04]. Digital weight storage within each synapse has still been chosen for two reasons:

- The need for a periodic refresh of the capacitively stored weight introduces a significant communication overhead over the external interface of the chip.
- The implementation of STDP within the chip requires the modification of the synaptic weights depending on their current value. Weights that have been modified by the STDP algorithm cannot be reflected by an external memory without transmitting the modified weight off chip.

1.3.3 Operating Speed and Power Consumption

Operating Speed of the Model The analog VLSI model aims to complement software simulations and digital model implementations. To justify this demand, the realization in silicon is designed such that it makes good use of the available resources in terms of area and speed. Many of the implemented silicon devices (transistors, capacitors, etc.) are close to the minimum size geometry. As a result, almost 100,000 synapses and 384 neurons fit on one die (cf. section 4.2). The small size results in low parasitic capacitance and low power consumption. Furthermore, the scaling of capacitances, conductances and currents leads to time constants in the physical model which lead to a speed-up factor of 10^5 for the model time compared to biological real time, i.e. 10 ns of model time would equal 1 ms in real time. The possibility to adjust parameters like rise and fall time of the synaptic current, membrane leakage current and the threshold comparator speed allow for a scaling of this factor within one order of magnitude resulting in a speed-up of 10^4 to 10^5.

Power Consumption According to [SMM04] the static power consumption of one neuron is about $10\,\mu$W. It is mainly caused by the constantly flowing bias currents and the membrane leakage. This low power density demonstrates the advantage of a physical model implementation compared to the usage of standard microprocessor for digital simulations. The power dissipation even of large chips with 10^6 neurons will only reach a few Watts.

No definite value for the overall power consumption of the spiking neural network model can be given, because the synapse circuits do only consume power during spike generation. However, a good approximation of the maximum power consumption is to calculate the dynamic power consumption of the maximum number of synapses that may be active, simultane-

ously at an average firing rate which reflects the maximum expected rate. A rough estimation will be given in section 4.4.3.

1.3.4 Network Model and Potential Topologies

The network model is based on the transport of digital spikes from one neuron to multiple destination neurons reflecting the fan out of the axon of the biological neuron to the dendrite trees of several destination neurons. Thanks to the digital nature of the spike, the transmission may take place over large physical distances, also to neurons located on different chips.

One possibility to transmit digital neural events is to use an address event representation (AER) protocol. An address defining the neuron is asynchronously transmitted together with an enable signal within this protocol which is for example used in VLSI spiking neural networks that operate in real time developed by Douglas et al. [OWLD05, SGOL$^+$06]. Problems arise for the presented model due to the asynchronous nature of this protocol: if an accuracy of less than 0.1 ms biological real time is desired for the spike transmission, a maximum skew of about 1 ns on an AER bus would be required which is not feasible within large VLSI arrays or between several chips. For this reason, events that are to be transported off chip leave the continuous time domain and are digitized in time. The digitization process is described in the following section which also gives an overview of the overall analog VLSI implementation.

The following network topologies can be realized with this model: in principle, a digitally generated spike can be duplicated any number of times and then be distributed to the synapse drivers located on several chips. Each synapse driver provides the fan out of its associated synapse row. Neglecting the signal propagation delay within the row, the axonal connection delay is the same for all target neurons within one row. One of the two goals of the presented thesis is the development of a communication protocol and the according hardware which together enable this event transport between different chips. This concept will be described in chapter 3.

1.3.5 Overview of the Implementation

A block diagram of the VLSI implementation of the physical model is shown in figure 1.5. The central synapse array is organized in two blocks, each containing 192 $\times$ 256 synapses. Each synapse consists of its four bit weight memory, the according digital-to-analog converter (DAC) and the current sink that produces the desired output slope. Furthermore, each synapse contains circuits measuring temporal correlations between pre- and postsynaptic signal, that are used for long term synaptic plasticity, e.g. STDP. Synapse control circuitry and the correlation readout electronics have digital interfaces that are part of an internal data bus that is connected to synchronous digital control logic.

The process of spike generation within the neuron has already been described in the preceding sections. Digital spikes generated by the neurons are not only fed back to the synapse drivers but are also digitized in time to be transported off chip. The digitization happens with respect to a clock signal and involves several steps: the spiking neuron is identified by an asynchronous priority encoder that transmits the number (address) of the neuron to the next stage. If more than one neuron fires within one clock cycle, the one with the highest priority[3] is selected and transmitted within that cycle. In the following cycle, the number of the next neuron is transmitted and so on.

[3]The priority is determined by the address of the neuron. Lower address means higher priority.

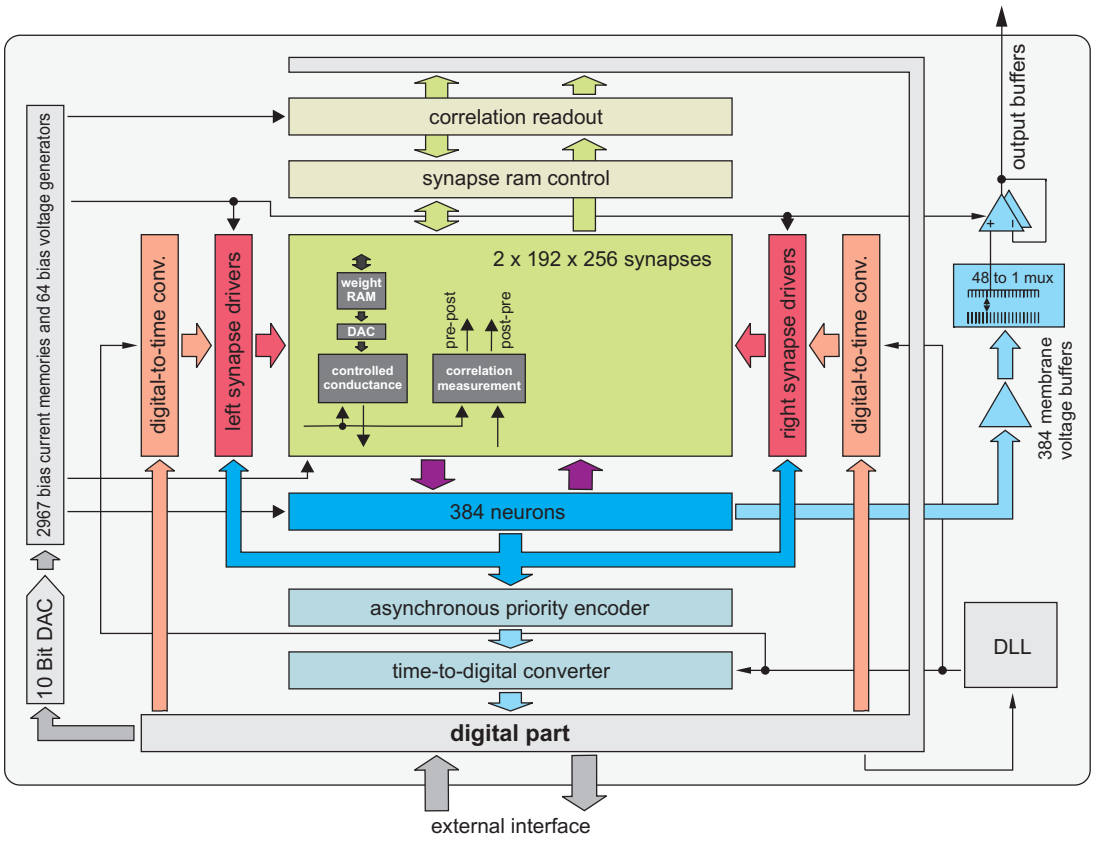


Figure 1.5: Block diagram of the analog VLSI implementation of the physical model. Figure adapted from [SMM04].

Two actions are carried out by the time to digital converter (TDC): on the one hand, it registers the number of the selected neuron to synchronize it to the clock signal. The stored value is the first part of the digitized event's data, the address. On the other hand, the TDC measures the point in time of the spike onset. This is done by means of a delay-locked loop (DLL) dividing one clock cycle into 16 equally spaced time bins. If two neurons fire within one clock cycle, the second spike will thus have a maximum error of one clock period. In addition to the time bin information, a system time counter synchronous to the digitization clock is required. Its value during the digitization process completes the time stamp of the event and thus the event data. Digitized events are forwarded by digital control logic to be sent off the chip.

The process of external event generation is initiated by sending an event to the digital control logic. The event data consequently consists of an input synapse address and a time stamp which itself consists of the system time and the time bin information. The digital control logic delivers the event within the desired system time clock cycle to the *spike reconstruction* circuits. These contain an address decoder and a digital to time converter (DTC), which operates based on the same DLL as the TDC. A digital pulse is generated at the input of the addressed synapse driver within the according time bin.

To account for the high speed-up factor of the model, the event generation and digitization circuits are designed to operate at clock frequencies of up to 400 MHz. Thus, the precision of

the time measurement theoretically will reach at best 156 ps which equals 1/16 of the clock period. In biological time, this results in an accuracy of 15.6 μs.

No two biological neurons or synapses are alike. Different surroundings, different physical size and connectivity lead to varying properties like the membrane capacitance, the leakage currents or differences in the synaptic behavior. The implementation of the model by means of analog VLSI has similar implications: no two transistors or generally no two analog devices are alike. While this deems desirable, it should still be possible to tune model parameters as to mimic a specific behavior and furthermore to eliminate drastic differences which may occur due to mismatching of two circuits. The behavior of analog circuits can be tuned by means of bias voltages or currents or other parameter voltages. This is done by a DAC that connects to different current memory cells and parameter voltage generators storing parameters like the leakage conductance or the reversal potentials. A more precise description of the parameter generation will be given in chapter 4 and a complete list of parameters can be found in appendix C.

Chapter 2

System on Chip Design Methodology

The implementation of full custom analog electronics in contrast to the design of digital ASICs is commonly carried out using different design methodologies. The key aspect covered within this thesis is the implementation of analog VLSI circuits comprising several millions of transistors together with high-speed digital logic into one mixed-signal ASIC by the usage of a customized, automated design flow. This chapter describes this design flow, whereas the actual implementation and the properties of the ASIC are covered in chapter 4. The necessary data preparation needed for the logical synthesis and the physical integration is described followed by a general outline of the logical synthesis process. For the physical implementation, Cadence First Encounter is used and besides the common implementation process, developed strategies for the system integration and the implementation of high speed digital interfaces are described. As a result, the analog electronics introduced in chapter 1 are seamlessly integratable together with the digital logic detailed in the following chapters. The verification process, including timing closure and physical verification, closes the chapter.

Generally, the process of designing an ASIC can be divided into two phases called the *front end* which is the starting point of the design process, and the *back end* where the physical implementation is carried out. Concerning digital logic, the front end design starts with a behavioral description of the ASIC using a high-level hardware description language, such as Verilog or VHDL[4]. The following digital circuit synthesis involves the translation of the behavioral description of a circuit into a gate-level netlist comprising generic logic gates and storage elements. For the physical implementation, these generic elements need to be mapped to a target technology. This technology consists of a library of standard logic cells (AND, OR, etc.) and storage elements (flip-flops, latches) that have been designed for a specific fabrication process. This mapped gate-level netlist serves as a starting point for the digital back end.

[4]Very High Speed Integrated Circuit Hardware Description Language.

Within a purely analog design process, the front end consists of the initial schematic[5] capture of the circuit. It is tuned and optimized towards its specifications by means of analog simulations. The final schematic subsequently serves as a basis for the physical implementation using the back end.

The back end in general involves the physical implementation of the circuit, be it analog or digital. It consists of an initial placement of all components followed by the routing of power and signal wires. The mixed-signal design process described within this chapter allows for the automated integration of complete analog blocks into one ASIC together with digital logic in the back end process.

Different, commonly rather generic design flows for purely digital and even mixed-signal ASICs are available through the online documentation of the design software vendor [Cad06b] or are available in text books (see for example [Bha99]). However, the implementation of large mixed-signal ASIC or system on chip (SoC)[6] requires the development of non-standard solutions throughout a design flow, due to the special demands that arise from the (in many cases) proprietary interfaces and block implementations.

The novel aspect of the presented flow is the automatic integration of complete, fully analog VLSI designs together with high speed digital logic into one ASIC. To integrate both analog and digital circuits, the following approach is used: The analog circuits are divided into functional blocks (later referred to as analog blocks) that together form the analog part of the chip. A top-level schematic for the analog part is drawn which interconnects these blocks and contains ports for signals needed for digital communication or off-chip connections. The developed scripts enable the digital back end software to import this analog top-level schematic together with the digital synthesis results. By partitioning the design into an analog and a digital part, each part is separately physically implemented using different sets of rules. The final chip assembly which involves the interconnection of analog and digital part, connections to bond pads, final checks and the functional verification finishes the design. It is done together for both, analog and digital circuitry, by a common set of tools.

The design flow is illustrated in figure 2.1 as an initial reference. In the following section, some prerequisites are introduced to clarify the terminology used throughout this chapter. The subsequent description of the design flow will intermittently refer to figure 2.1.

2.1 Prerequisites

Two very basic concepts of digital design are explained to clarify the strategies described later on. Following this, the data necessary for the implementation of hierarchical blocks using an automated design flow is explained. While this data is commercially available for the digital design, the corresponding analog design data is generated by a set of scripts which have been developed throughout the presented work.

2.1.1 Digital Design Fundamentals

Static Timing Analysis The static timing analysis (STA) allows for the computation of the timing of a digital circuit without requiring simulation. Digital circuits are commonly characterized by the maximum clock frequency at which they operate which is commonly determined using STA. Moreover, delay calculations using STA are incorporated within numerous steps

[5] Schematic will be used as the short form of *schematic diagram*, the description of an electronic circuit.

[6] System on chip design is an idea of integrating all components of an electronic system into a single chip. It may contain digital, analog and mixed-signal functions.

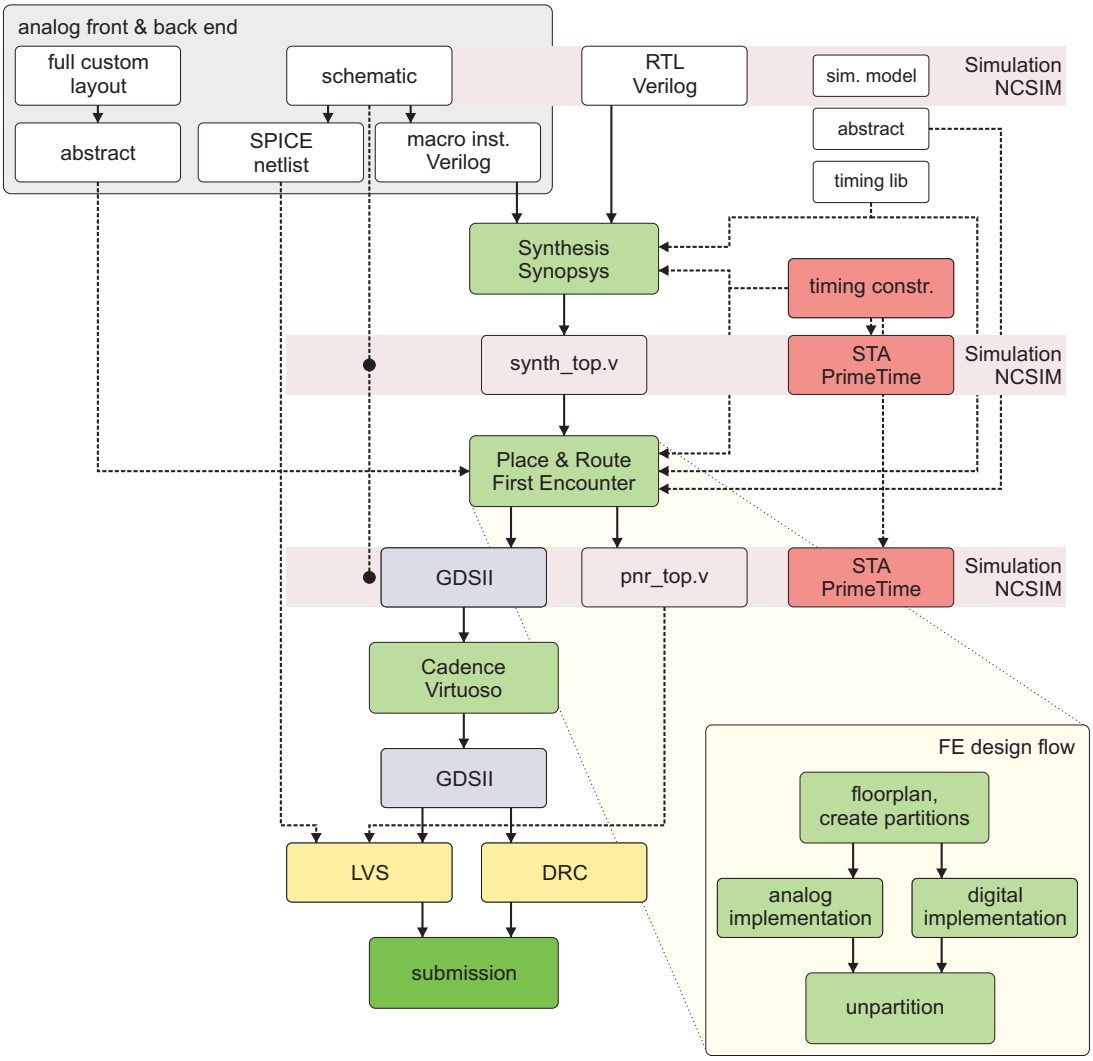


Figure 2.1: Illustration of the developed design flow for the implementation of large mixed-signal ASICs. Solid arrows denote subsequent steps in the flow. Dashed arrows show dependencies on input data and constraints.

throughout the (SoC) design process, such as logic synthesis and the steps performed during physical implementation in the digital back end. The speed-up compared to a circuit simulation is due to the use of simplified delay models and the fact that its ability to consider the effects of logical interactions between signals is limited [Bha99].

In a synchronous digital system data is stored in flip-flops or latches. The outputs of these storage elements are connected to inputs of other storage elements directly or through combinational logic (see figure 2.2). Data advances on each tick[7] of the clock and in this system two types of violations are possible: a *setup violation* due to the signal arriving too late to be captured within the setup time t_{SU} before the clock edge. Or a hold violation due to the signal changing before the hold time t_{HD} after the clock edge has elapsed.

The following terms are used throughout the design flow in conjunction with STA:

[7]The flip-flops may be triggered with the rising, the falling or both edges of the clock signal.

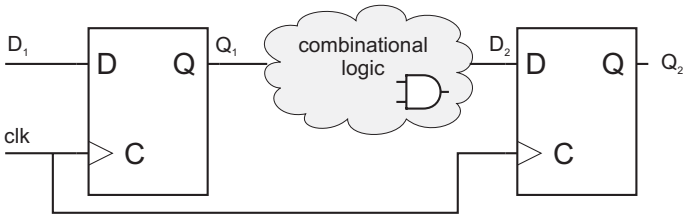


Figure 2.2: Illustration of a synchronous digital system. Ideally, the clock signal `clk` arrives simultaneously, data is delayed by the clock to output time T_{co} and the delay through the combinational logic.

- The *critical path* is defined as the path between an input and an output with the maximum delay. It limits the operating frequency of a digital circuit.
- The *arrival time* is the time it takes a signal to become valid at a certain point. As reference (time 0), the arrival of a clock signal is often used.
- The *required time* is the latest time at which a signal can arrive without making the clock cycle longer than desired.
- The *slack* associated with a path between two storage elements is the difference between the required time and the arrival time. A negative slack implies that the circuit will not work at the given clock frequency.

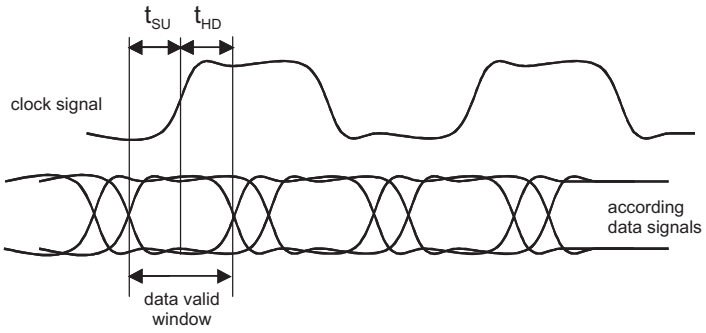


Figure 2.3: Timing specification for the source synchronous HyperTransport interface [Hyp06]. To maximize the data valid window, t_{SU} and t_{HD} are to be maximized.

Source Synchronous Data Transmission In contrast to synchronous digital systems, where one clock is distributed to all components of the system, in a source synchronous system a corresponding clock signal is transmitted along with the data signals. Source synchronous data transmission is commonly used for high-speed physical interfaces such as the HyperTransport interface [Hyp06] or the double data rate SDRAM (DDR-SDRAM) interface [Mic02]. Source synchronous interfaces are point-to-point connections where only the relative timing of the clock signal and the corresponding data signals need to be specified. One advantage is that these signals may be routed throughout the system with equal delay and no care has to be taken about the clock distribution to other parts of the system as in synchronous systems. Moreover, the circuits generating clock and data signals are located on one die. As a consequence, delay

experienced by the data through a device tracks the delay experienced by the clock through that same device over process variations. Figure 2.3 illustrates the timing specified for the HyperTransport physical interface which is implemented in the presented chip.

2.1.2 Required Technology Data

The technology data needed for the integration of analog blocks using the digital back end is described in this section.

The Standard Cell Library The standard cell library consists of a set of standard logic elements and storage elements designed for a specific fabrication process. The library usually contains multiple implementations of the same element, differing in area and speed, which allows the implementation tools to optimize a design in either direction. Basis for each standard cell is an analog transistor circuit and the corresponding layout. To make use of this cell for a digital design flow, it is characterized regarding its temporal behavior on the pin level and its physical dimensions. A behavioral simulation model written in a hardware description language (HDL), the timing information and the physical dimensions are stored within separate files that together define the standard cell library. The integration of analog blocks as macro cells that are to be placed and routed using primarily digital back end tools requires the availability of this very data.

The Technology Library A technology library describing standard cells contains four types of information (the format introduced by Synopsys is used and a complete description of the technology library format is given in [Syn04b]):

- Structural information. Describes the connectivity of each cell to the outside world, including cell, bus, and pin descriptions.
- Timing information. Describes the parameters for pin-to-pin timing relationships and delay calculation for each cell in the library. This information ensures accurate STA and timing optimization of a design.
- Functional information. Describes the logical function of every output pin depending on the cell's inputs, so that the synthesis program can map the logic of a design to the actual ASIC technology.
- Environmental information. Describes the manufacturing process, operating temperature, supply voltage variations, all of which directly affect the efficiency of every design.

For the implementation of analog blocks only the structural information and the timing information are important. The environmental information is predefined by the vendor within the standard cell library and functional information is not required as the block is to be implemented as-is and no optimization is to be performed on it.

Different delay models are available for the description of the timing. Two commonly used ones are the CMOS linear and the CMOS non-linear delay models. For cells with a straight forward functionality like standard cells there are tools available[8] that automatically characterize the cells using methods described for example in [SH02]. If no automatic characterization

[8]No such tools are available at the Kirchhoff Institute. One example is SiliconSmart from Magma. (http://www.magma-da.com)

a)
```
pin(clk400) {
  clock : true ;
  max_transition : 0.15 ;
  direction : input ;
  capacitance : 0.319 ;
}
```

b)
```
PIN clk400
    DIRECTION INPUT ;
    USE CLOCK ;
    PORT
    LAYER ME2 ;
    RECT  10.03 0.01 10.31 0.29 ;
    END
END clk400
```

Figure 2.4: a) Description of a clock pin using the Synopsys Timing Library Format and the linear delay model. b) Geometric description of the same pin using LEF.

tool is available, the linear model should be used. Therein the delay through a cell or macro is calculated based on the transition time of the input signal, the delay through the cell without any specific load and the *RC*-delay at the corresponding output that results from the output resistance of the cell's driver and the capacitive load [Syn04b]. The intrinsic cell delay as well as the output resistance can be obtained by means of analog simulations. Values obtained this way are then embedded as timing information into a technology library. Several technology libraries are usually made available to reflect the best, worst and typical fabrication process variations. Figure 2.4 a shows a very simple example for the definition of a clock pin using the linear delay model.

The Abstract View The layout data of an analog block cannot directly be used by the digital back end. Therefore, for the physical implementation, an abstract view is generated which reduces the analog layout to a block cell like it is the case for the cells within the standard cell library. The abstract view provides information like the cell name or its boundary. Pin names, locations, metal layers, type and direction (in/out/inout) are included and besides the pins the locations of all metal tracks and vias in the layout are included as obstructions. The abstract view is generated in the Cadence analog environment using the Cadence Abstract Generator. It is then converted to a text file in library exchange format (LEF) which is being read by the digital back end. An example for the LEF syntax is shown in figure 2.4 b. Obviously this text file becomes very large if all metal geometry of an analog VLSI block completely would be included. Especially inside analog VLSI blocks this information is surplus. The amount of information is reduced by merging small geometries (in the order of magnitude of the minimum width) to few large obstructing geometries. Merging is done in two ways:

- Use the resize option provided by the software. All geometry data is first being widened, closing small gaps and then shrunk, thereby leaving over only coarse geometries.
- Many overlapping shapes exist after this process and to further reduce the amount of data, these shapes are automatically merged by scripting the Virtuoso layout environment.

Pins are generated automatically within the abstract view at locations labeled accordingly within the layout. To support non-quadratic or rectilinear pin geometries, the developed scripts support so called *shape pins*[9] that override the automatically extracted pin information. This is especially useful for the generation of multiple connections to large power grids or connections to heavily loaded signal pins requiring a wide metal routing. To ease pin access for the router, the pins are preferably to be placed near the cell boundary.

[9]This is a geometry that s natively supported by the layout tool.

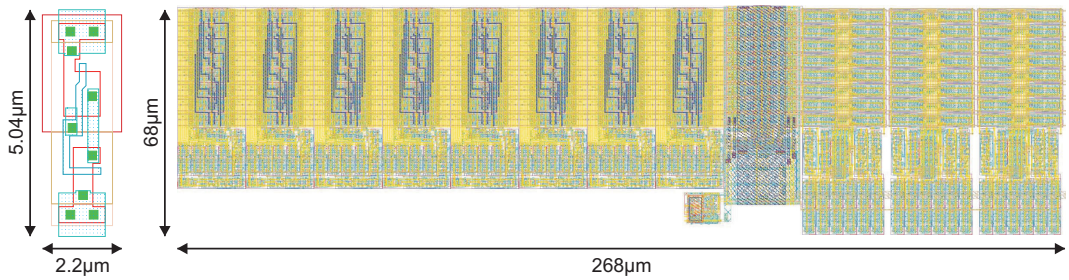


Figure 2.5: Layout view of a standard cell inverter compared to the layout of an analog block. The inverter is comparably easy to access by an automated router, because only one metal layer is occupied by its layout. In contrast, care has to be taken to retain this accessibility at the complex layout of the multi-metal layered analog block to keep the automatic integration feasible.

On its left side, figure 2.5 shows the standard cell implementation of an inverter compared to the layout of an exemplary selected analog block (it is the block containing both the DTCs and the TDCs). The standard cell is designed to occupy only one metal layer and to have end-to-end rails for power connections at the top and the bottom side. Pins can easily be connected by an automatic router by directly connecting to one of the pins from the second metal layer. Five of the six metal layers available in the utilized United Microelectronics Corporation (UMC) fabrication process are available for routing. In contrast to this, the analog block has many pins distributed on different metal layers over the whole area. To still enable an automatic router to connect to these pins, it has to be ensured that they are accessible from at least one direction.

The Technology LEF File The process details relevant to the digital back end are specified in this text file. Preferred routing directions are defined for the different metal layers within this file[10] as well as minimum metal wire width and routing pitch, physical parameters like thickness and sheet resistance, slotting and antenna rules, and rules for the via generation among the layers. According to these rules, the router performs routing at minimum width and pitch. Analog nets that require wider spacing or wider metal, require the manual specification of non-default rules additional to the standard rules. These non-default rules at least include the redefinition of the routing width, the spacing and the via generation - for reasons of simplicity, the reader may refer to the LEF language reference for details [Cad05a].

2.2 Design Data Preparation

The following paragraphs will briefly describe the steps that are taken from data preparation to the final submission of the layout data to fabrication according to figure 2.1.

Analog Design Data The analog design includes the schematic capture and the physical layout of the macro blocks using the Cadence Virtuoso and AMS Designer platform. Each block is designed by hand and a physical check including design rule check (DRC)[11] and layout versus schematic (LVS)[12] is performed. The blocks may comprise small building blocks

[10] Routing direction is either horizontal or vertical. The router tries to route wires in the preferred direction first and only routes short segments in the other direction.

[11] Checks for geometric errors.

[12] Verifies the matching between devices in the layout and the schematic.

as for example single neuron or synapse circuits. In the case of the presented chip, VLSI blocks are implemented that already contain the connections among these small building blocks.

To integrate a whole set of analog blocks including their interconnections automatically, the library data and abstract view described in the preceding section and a Verilog netlist are required. This netlist is generated by the following steps:

- Create a schematic of the whole analog part including block interconnections and required ports.
- Create empty behavioral descriptions of the analog blocks including only port descriptions. These are needed for correct instantiation and are generated automatically.
- Make use of the netlisting functionality provided by the Cadence AMS Designer suite and create a Verilog netlist only consisting of the block instances. This is done script based within the presented flow to avoid errors after design changes.

Digital Design Data The digital design includes the RTL[13] or behavioral description of the digital part of the chip. Above all it includes the very top-level of the hierarchical design that is to be implemented using the proposed SoC design methodology. This top-level module is a Verilog netlist with ports representing physical pins of the chip instantiating digital and analog top-level modules (of course, several are possible). No glue logic must be used at this level as it only defines the connectivity and no optimization by the digital front end or back end is performed.

For top-level verification purposes, the top-level module may be integrated into different simulation testbenches, be it behavioral or RTL descriptions of the controller, and simulated using either Cadence NCSIM for mixed-signal simulations or Mentor Graphics ModelSim for solely digital simulations[14]. If digital macros like memory arrays are present in the design, the according library data has to be available before proceeding with the flow.

2.3 Logic Synthesis and Digital Front End

The task of the digital front end is to transform the RTL or behavioral description of an ASIC into a gate-level netlist. This process is called synthesis. After the initial translation into a netlist consisting of generic logic functions and storage elements (compile phase), this netlist is then mapped to the target technology using the elements available in the standard cell library (mapping phase). The design is then optimized for area, speed and design rule violations like maximum transition time or maximum capacitance violations (optimization phase). Especially the timing optimization requires the definition of consistent constraints to the software tools. Synopsys Design Compiler is used for the digital front end within this flow [Syn04a]. The timing constraints are defined using the tool command language (TCL) syntax which is supported by all other software throughout the flow that requires the specification of timing constraints for the design. This ensures consistent STA results at every stage of the flow.

The synthesis is divided into two major steps within the presented design flow. First, the digital part is synthesized and optimized to meet its timing requirements. The correct specification of the timing is crucial for the correct functionality of the synthesized netlist. The periods of all clock signals need to be specified, thereby defining the maximum delay between

[13] Register Transfer Level: coding style that describes registers using a hardware description language.

[14] Analog blocks are treated as black boxes using only the empty modules in this case.

two clocked elements. Apart from these synchronous constraints, asynchronous paths have to be identified and constrained with minimum or maximum delay, whichever is appropriate. To avoid unnecessary optimizations, paths that do not underly any timing constraints are marked manually as well. In a second step, the digital netlist obtained in the first step is combined with the analog netlist to form the top-level netlist of the chip. Only interface signals between analog and digital modules are optimized during this step.

The top-level netlist is evaluated for timing problems using Synopsys PrimeTime and a functional verification is performed using the simulation tools ModelsSim or NCSIM, depending on the setup of the testbench. The purpose of the verification at this stage of the flow is to identify faulty timing constraints that lead to incorrect implementations. The final implementation is subject to the digital back end.

2.4 Digital Back End and System Integration

For the digital back end and the system integration, Cadence SoC Encounter is used. This software provides a complete hierarchical design solution, including features like floorplanning and virtual prototyping, hierarchical partitioning and block placement, logic optimization, timing optimization, signal wire and power routing, geometry verification, connectivity and process antenna verification and the generation of stream data (GDSII[15]). During the phase called virtual prototyping, it is possible to estimate the design's performance very quickly by means of a fast trial-route algorithm, a built in *RC*-extraction and delay calculation algorithm, and a built-in STA algorithm. As SoC Encounter can be run in batch mode, the whole back end flow has been set up using scripts that are executed from within a make file. Reports are written at each step of the flow and the design process is fully reproducible.

2.4.1 Design Import and Partitioning

Apart from the data that is supplied by the standard cell vendor like LEF data and technology libraries for I/O cells and standard cells, the following data has to be generated respectively specified and imported into First Encounter:

- The synthesized gate-level netlist of the chip (output of digital front end).
- Consistent LEF data for the standard cells, digital macros and the analog blocks as well as the according technology library information. Besides the LEF and technology data for the standard cells, the same data has to be available for I/O cells that are to be used for the pad connections of the chip. These cells are also supplied by the standard cell vendor.
- A timing constraint file. The constraint file used by the digital front end is included by this file to ensure the usage of identical constraints. Moreover, constraints that only apply to the back end can be given here.
- A capacitance table file which serves as a basis for the *RC* calculations. This file is comprised of metal-metal capacitances and resistances for various configurations which are stored within look-up tables (LUTs). This file has to be generated separately and can have different levels of accuracy (the lower the accuracy, the faster the calculation).

[15] GDSII is a hierarchical file format to exchange 2D design data. It contains only geometry information for the physical layers of an ASIC and is used to transfer the design data to the foundry.

- An I/O assignment file which defines the positions of the I/O cells. If the pinout is not fixed prior to the design start, this file may be changed iteratively during the initial floorplanning stages.
- A very initial floorplan specifying the coordinates of the die boundary, the I/O ring and the core area.
- A list of power and ground net names that are globally valid for the design. Power routing can only be done for these nets. This list is very important as the imported Verilog netlist does not contain any power connections and the power connections for the analog blocks are later on performed using this list.

The sum of this data is called the *configuration* for SoC Encounter.

Floorplan Floorplanning involves the definition of placement regions for modules, defining boundaries for top-level modules that will serve as partition boundaries, and macro cell placement. In the presented flow, the macro cell placement is done semi-automatic by placing macro cells with relative coordinates. One example is given in the following code snippet where a set of macros `no_<0:3>_` is placed relative to another, absolutely placed macro `nb_0_` with a pitch of 100 μm in x-direction:

```
placeInstance analog_chip/nb_0_ 381.4 1800 MY

for {set i 0} {$i<4} {incr i} {
  relativeFPlan --relativePlace analog_chip/no_{$i}_ TR \
                analog_chip/nb_0_ BR [expr 100+($i*100)] 0.0 R0
}
```

The use of the for-loop allows for the automatic placements of large arrays of custom macro cells. Provided that a suitable algorithm exists, even the single compartments of a complex neural network array could be placed using this technique. Particularly regarding the digital part of the chip, the quality of the floorplan and the macro cell placement is crucial to the result of the following timing driven placement of the standard cells and the subsequent timing optimizations. An optimal macro cell and block floorplan not only speeds up this placement, but also guarantees superior results in terms of timing and reduced congestion [Bha99].

Power Plan After the floorplan is complete, the power distribution network (PDN) for the complete chip is generated. At the beginning, the global power nets defined in the configuration are assigned to power pins of standard cells and macros in order to prepare the semi-automatic and automatic power routing. To ensure a correct power netlist, this is to be done as in the following example where the pins `vdda` of all present macros are connected to the global power net `avdda`:

```
globalNetConnect avdda -module analog_chip -inst * \
    -type pgpin -pin vdda -override -verbose
```

It is especially important to clearly separate analog and digital power domains in this step. In the following, filler cells are placed within the IO-Ring to form a closed power ring[16]. Wide metal structures are generated around the core (core rings) and around macro cells to predefine a coarse structure for the PDN. These structures should be planned such that the final automatic

[16] If the chip requires different power supplies, it is possible to leave open gaps in the I/O-Ring to separate these power domains from each other (see section 4.4.3 for the implementation on the presented chip).

power routing only has to perform rather obvious connections (from a human point of view) because these are supposed to be routed in a straight manner without excessive via insertion by the power routing software[17]. Figure 2.6 illustrates the predefined power structures in the area of an exemplary analog block before and with the power routing done.

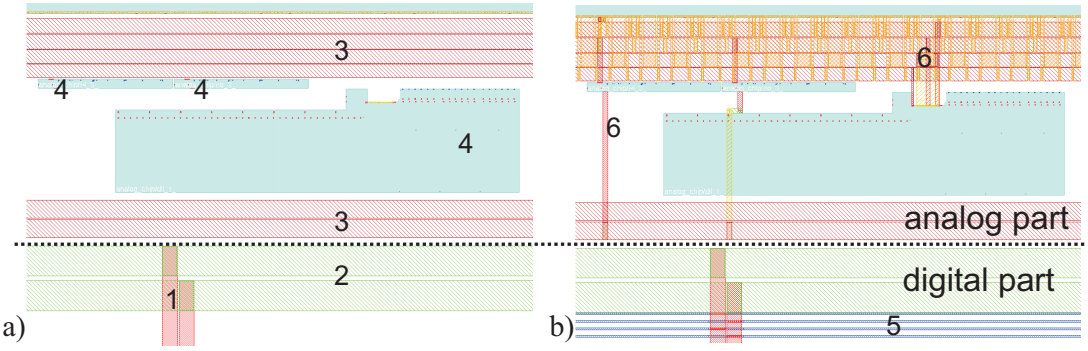


Figure 2.6: a) Power structures before automatic power routing and b) after automatic power routing. Note the automatically generated connections to the macros in the analog part and the power rails for the standard cells in the digital part. 1, 2: script generated structures in the digital part. 3: script generated structures in the analog part. 4: pre-placed analog blocks. 5: automatically generated power grid within the digital part. 6: automatically generated power connections within the analog part.

Partition the Design The design is subdivided into partitions that are then implemented separately. Partition boundaries are drawn as module boundaries in the top-level floorplan and the connectivity, more precisely the location of the different partition pins is determined by two steps. First, a timing driven placement of the whole design is performed. It has to be ensured that the software is allowed to place standard cells only within the foreseen digital areas by inserting placement blockages in the analog areas. Second, a trial-route run is performed and the partition pins are generated where the signal wires hit the partition boundary. The result of the trial routing may not lead to optimal pin placement. Therefore, critical signals should be investigated manually and the according pins should be placed like in the following example for a clock pin (`clk200[0]`):

```
preassignPin analog_chip {clk200[0]} -loc $c20_xpos 0.0 -layer 6
```

During the partitioning process itself, the timing of all signals crossing partition boundaries is analyzed and the timing budget within the partitions is derived by estimating parasitic capacitances and routing delays based upon the results of the initial placement and trial routing. The design data of each partition, including the derived timing constraints, the netlist of the partition, macro cell placement and power routing is then stored within separate directories for implementation. The top-down design flow recommended by Cadence [Cad06b] recommends the partitioning of all modules present at the top-level of the design such that no optimization is necessary there and the modules just have to be routed together. For two reasons this strategy cannot be followed for the presented chip within which the digital part is not partitioned separately:

[17] The power structures have to be capable of delivering the required current to the circuits. Furthermore, voltage drop on metal lines has to be considered. A very good chapter on the estimation of power consumption and the design of the power distribution system can be found in [CBF00].

- To correctly derive the timing budget for synchronous signals that cross partition boundaries, it is necessary to know the latency of the according clock signal generating these signals. As the clock tree[18] of these clock signals has not been synthesized yet within the according partition, no wire delays or buffer delays can be assumed. The actual delay after clock tree synthesis therefore invalidates the timing budget, and for example the implementation of the synchronous signals to and from the DTCs/TDCs in the analog part would become erroneous.

- The I/O signals of the chip are distributed over the whole left and right edges of the chip (cf. section 4.1). As the digital part is located within the bottom third of the floorplan, the routing delays among these signals need to be exactly known to achive correct interface timing which is not the case after the trial routing but only after the final routing step. Furthermore, the clock trees of the interface clocks have to be synthesized which imposes the same problem as in the first point.

Consequently, after partitioning there are two subdesigns: the analog part and the top-level of the design including the digital part. Important details for the implementation of the two will be described in the following two subsections.

2.4.2 Analog Routing

The analog partition comprises the placement of the analog blocks, the predefined power routing, the partition boundary and the pin locations on the boundary. The only step to perform is the routing of the signal wires among the macro cells, which is crucial due to the analog nature of the signals. To gain good routing results, the following steps are performed prior to automatic routing:

- Check for the connection directions of signal busses with minimum pitch (horizontal/vertical). If the metal layer the bus is located on has a different routing direction defined in the technology LEF file, the design will probably be not routable because the routing algorithms are biased in the wrong direction (cf. section 2.1.2). In this case, the preferred routing directions in the technology LEF file could be modified appropriately which has been done for the analog part of the presented ASIC.

- Assign appropriate non-default rules to analog nets requiring wide metal routing or wide spacing.

- To improve the quality of the analog routing, reasonable routing blockages are defined in areas where routing is undesirable, as for example above sensitive analog circuits.

- Although the routing software supports signal integrity (SI) driven routing, the routing of very sensitive signals is performed in a directed, script based manner. In the case of the presented chip, several analog signals are to be routed in parallel with the maximum available spacing over a distance of approximately 3 mm. To ensure straight parallel routing these signals can be manually routed in the following way:

[18] The clock tree, or clock distribution network, distributes the clock signal(s) from a common point (the clock's root) to all the elements that need it (leafs).

```
for {set i 0} {$i<4} {incr i} {
  setEdit -nets vmemout\[$i\] -shape None -force_regular 1 \
      -layer_horizontal M6 -layer_vertical M6 -snap_to_track_regular 1 \
      -width_horizontal 0.280 -width_vertical 0.440 \
      -spacing_horizontal 0.280 -spacing_vertical 0.440
  editAddRoute    [expr $startx + $i*$pitch] $boty
  editCommitRoute [expr $startx + $i*$pitch] $topy
}
```

Four vertical wires are drawn on metal layer 6 with a pitch of `$pitch` from `$boty` to `$topy`. In the following routing stage, these wires are left unchanged and the router connects to either end of the wire.

Signals requiring the routing with non-default rules are routed first using the Cadence Ultra Router. In contrast to the recommended digital back end software (NanoRoute), this router correctly connects to wide metal pins - NanoRoute tapers to minimum width just before the pin [Cad06b]. The remaining wires are routed using NanoRoute.

The completely routed design is verified for geometry, antenna and connectivity errors and two output files are generated: a GDSII stream for inclusion in the top-level GDSII, and a data exchange format (DEF) file of the design which is later on used to un-partition the design and perform full-chip timing analysis.

2.4.3 Top-Level Placement and Routing

The following steps are preformed to establish the final placement and routing of the standard cell netlist and the connectivity to the IO-pads:

1. Timing driven placement: Due to the digital part being spacial restricted to a certain area of the chip (i.e. the lower third on the presented chip), placement blockages are created within the remaining areas prior to the actual placement. After placement, a trial-route run is performed to determine if the placement leads to highly congested areas[19]. Bad trial-route results indicate a poor placement and require an iterative optimization of the floorplan.

2. Pre-clock tree synthesis in-place optimization (Pre-CTS IPO): Nets that violate maximum transition time and/or maximum capacitance design rules are fixed in this step. Furthermore, the timing is optimized for positive slack by means of gate resizing (changing the drive strength of standard cells) or buffer insertion (buffers are inserted if the max. drive strength is not sufficient).

3. Clock tree synthesis: Clock networks are heavily loaded by the clock pins of the flip-flops they are driving. For synchronous designs, the propagation delay through the clock network to the clock pins needs to be the same for all destination pins (i.e. the clock skew has to be minimized[20]) which requires the buildup of a balanced buffer tree. This is done by a dedicated algorithm in this step[21]. The capabilities of this algorithm

[19] The congestion output is the percentage of wires that could not be routed due to high routing density. Rule of thumb: as of the experience of the author, values of more than approx. 1.5 % will later on lead to unroutable areas.

[20] In the literature, skew is mainly defined as the difference between actual and nominal interarrival times of a pair of clock edges in integrated circuits or printed circuit boards (PCBs), or as the difference between a pair of data and clock signals in parallel data transmissions. It depends on process variations, environmental variations (voltage and temperature), wire RC delay, and clock loading.

[21] The algorithm also supports a *useful skew* mode to improve very tight timing. The clock signal at a receiver flip-flop is delayed with respect to the clock at the sender to increase the required time. This feature should be used with care when optimizing clocks that cross partition boundaries [Cad06b].

include the definition of clock groups where a set of clocks is balanced for equal delay. On the one hand this is used to synthesize different clock signals on the presented ASIC with equal delay and on the other hand this is exploited to balance the routing of the source synchronous link signals (cf. section 2.4.4).

4. Post-CTS IPO: The design containing the synthesized clock tree is again in-place optimized, as in the first IPO step. To gain good routing results the optimization is also performed for hold violations in this step, after setup violations have been fixed (positive slack). The outcome of this step is the pre-final placed but not routed design.

5. Routing: The design is now being routed in detail while taking into account the timing constraints. The complete set of options specified to the router can be found in appendix D.1. The completion of this step without any design rule or process antenna violations finalizes the design of the digital part.

6. Post-Route IPO: This step only slightly improves the timing, as the routing is not substantially changed, anymore. What makes this step necessary is that hold violations possibly introduced by the router could be fixed.

7. Stream Out: The top-level GDSII file is generated by including GDSII files for all instantiated standard cells, macro cells, and the one for the analog part into the top-level design. The generated GDSII file is then read into the Cadence analog environment again (stream in) to have the chance of visually inspecting the design. An automated stream out of this data yields the final GDSII file for the design.

8. Sign-off *RC*-extraction: For each process corner available in the technology libraries, a detailed extraction is performed. Delay calculation and STA are performed and final timing reports are written. The final verification of the timing is performed using Synopsys PrimeTime (see section 2.5.1).

2.4.4 Intermezzo: Source Synchronous Interface Implementation

The strict interface timing of source synchronous interfaces as shown in figure 2.3 arises challenges on the implementation. On the one hand, STA based timing optimization software optimizes a design for positive slack, regardless of the exact slack values. On the other hand, the implementation of a source synchronous interface requires the slack for all signals to be the same in order to maximize the data valid window.

Another requirement of the HyperTransport specification is the 90° phase relation between clock and data which ensures the clock to be always in the center of the data valid window. A representative configuration of a transmitter and receiver configuration is shown in figure 2.7. The transmitter uses a phase-locked loop (PLL) to generate the clock for the output data and a phase shifted version for the output clock. The receiver uses a phase recovery first-in first-out (FIFO) for the data capture. To achieve reliable phase recovery for variable frequencies, DLL circuits are commonly used. These systems are also called *clock recovery circuits* or *bit synchronization circuits* [Raz96] because they perform the function of generating clock signals in synchronization with the data incoming to a receiver circuit allowing the recovery of the data. The fact that neither a PLL with the required outputs, nor a DLL have been available for the development of the presented chip, together with the demand for equal slack on the signal lines, led to the implementation technique that is described in the following. On the one hand, the capabilities of the CTS software are exploited, and on the other hand, delay elements are manually placed.

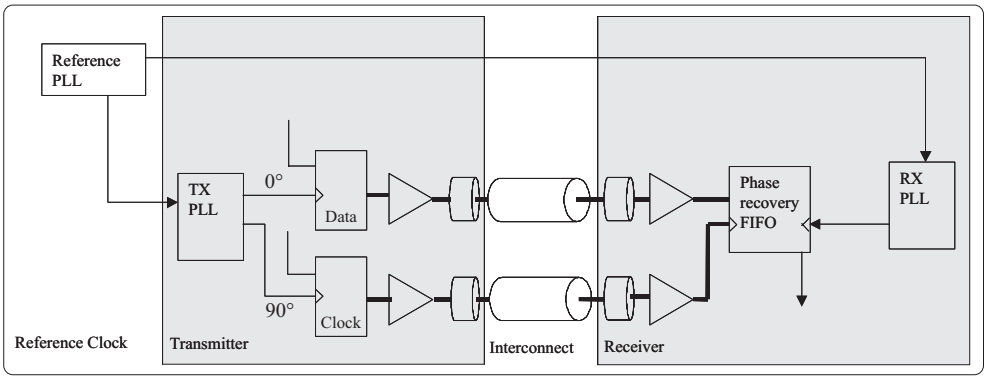


Figure 2.7: Representative transmitter and receiver configuration according to the HyperTransport specification. The "TX PLL" provides two clocks with a 90° phase shift to separately generate clock and data signals. Figure adapted from [Hyp06].

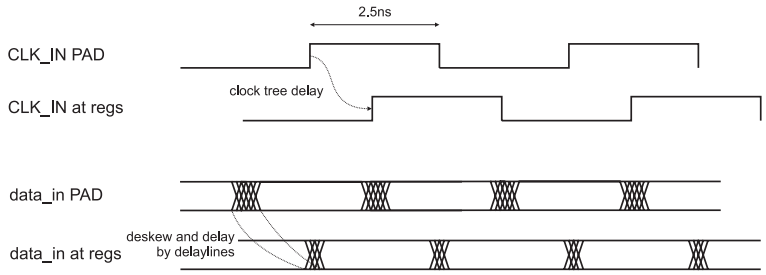


Figure 2.8: Illustration of the desired functionality for the skew reduction at the input interface. Adjustable delay elements are used to minimize the skew and to introduce appropriate delay on the data signals.

Input At the input of the chip it is first of all desirable to reduce the skew among the input signals to increase the size of the data valid window. Furthermore, to forward the clock-data phase relation present at the chip boundary to the input registers, the data signals require a delay which is equal to that of the clock network. The solution is to use adjustable delay elements that introduce the necessary delay and can be controlled to minimize the skew. The desired timing at the pads of the chip and the input registers is illustrated in figure 2.8. To achieve identical routing delays on all input signals, the CTS software of First Encounter is used in the proposed implementation technique. The basic idea is illustrated in figure 2.9 a. However, the latency t_{ct} of the input clock tree strongly depends on the number of clocked registers and can initially not be predicted [Cad06b]. Therefore, the following steps are necessary:

- From an initial First Encounter run, the value t_{ct} that the software has achieved for the clock tree of the input clocks is determined.

- Using a delay element for each data signal, the additional delay required for each signal is calculated as

$$t_{add} = t_{ct} - t_{del}, \tag{2.1}$$

with t_{del} being the default delay through an (adjustable) delay element.

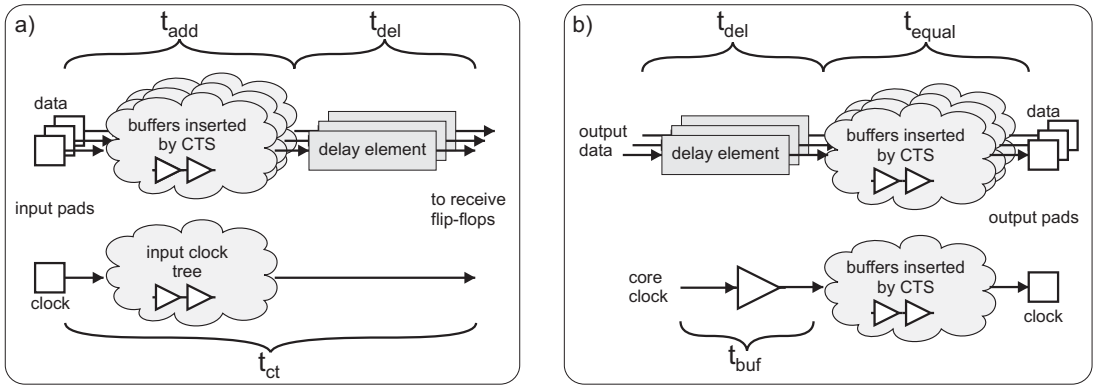


Figure 2.9: Concept for the implementation of source synchronous interfaces. a) receive side b) transmit side. The output data is meant to originate at flip-flops clocked with core clock.

- All input signals are defined as clocks to the clock tree synthesizer where the physical pad is the root pin and the input to the delay element is the only leaf to the respective clock. Combining the signals of one link into one clock group ensures that the clock tree synthesizer routes these signals with equal delay and minimum skew. The borders for the allowable delay should be set within $\pm 100\,\mathrm{ps}$ of t_{add} to not overconstrain the CTS software.

As a result, the clock tree synthesizer inserts buffers as required into the routing of the input data signals as illustrated in figure 2.9 a:. The straight forward approach using this strategy would be to let the clock tree synthesizer analyze the whole path from the input pad to the D-input pins of the input registers. However, the clock tree synthesizer will in most cases not be able to analyze the path through the delay element. For this reason, the input pins to the delay elements are set as leaf pins to the respective clock trees as described above and CTS stops tracing at the input.

Output The routing of output signals with identical delay t_{equal} again is achieved by means of the CTS software. The generic setup is illustrated in figure 2.9 b. Delay elements are on the one hand used to compensate for external or internal skew. On the other hand, they introduce a fixed time shift t_{del} on the data signals with respect to the clock. The buffer depicted within the clock path serves as an endpoint (leaf pin) to the core clock and as a root to the artificially introduced output clock connected to the pad. The time shift t_{shift} and thus the output phase can be calculated as

$$t_{\mathrm{shift}} = t_{\mathrm{del}} + t_{\mathrm{cto}} - t_{\mathrm{buf}} \,, \tag{2.2}$$

with t_{cto} being the clock to output delay of the flip-flops generating the output data (not shown in the figure). This technique results in implications for the interface timing:

- No real phase relation exists between clock and output data but rather a fixed delay. This limits the maximum data rate in a way that no valid data transmission would be possible if one bit time[22] would be smaller than t_{shift}—provided that the receiver meets the specifications and cannot tune it's link in the right direction.

[22] A bit time is the time needed to transmit one bit of data. In the case of double data rate, one bit time equals one half of the clock period.

- The shift between clock and data for the typical process corner is approximately -90° and not 90° as specified. To still capture the data with the correct edge of the clock, the clock needs to be shifted by 180° which is done within the receiver of the presented ASIC and the controller respectively.

The correct functionality of the interface as well as the successful implementation of this methodology is demonstrated in section 6.2.3 by means of the measured performance.

2.5 Verification

2.5.1 Timing Closure

Achieving timing closure on a design is the process of creating a design implementation that is free from logical, physical, and design rule violations and meets or exceeds its timing specifications. For a production chip, all physical effects, such as metal fill and coupling, have to be taken into account before it can be confirmed that timing closure has been achieved [Cad06a].

The timing of the design is verified throughout all stages of the presented design flow by means of *RC*-extraction, delay calculation, STA and timing report generation. By monitoring the results of the different optimization steps it can be determined already in the early design phase whether the final implementation will meet the timing specifications or not. Large negative slack values require a redesign of the causing logic in the Verilog description. Over-constraining the timing yields worse results than relaxing the constraints if an improvement by redesign is not possible [Bha99].

The final timing closure is based on the detailed *RC*-extraction data of the design flow and requires two steps:

- Import the final gate-level netlist and the extracted parasitic data into PrimeTime. On the one hand, STA is performed on the data using the globally defined timing constraints. In contrast to the STA engine built into First Encounter, PrimeTime performs a detailed analysis on unconstrained paths and thus may reveal erroneous constraints. On the other hand, PrimeTime is used to generate delay data for the back annotated simulation of the gate-level netlist in standard delay format (SDF[23]).

- Perform back annotated simulation. The gate-level netlist is simulated using the SDF file. The timing specifications are met if the behavior of the gate-level netlist matches the behavior of the RTL description of the design for the targeted clock frequency using the worst process corner. This is verified by means of automated test procedures included within the software that produces the input test data (cf. section 5.3). Potential hold violations are detected by performing the simulation with the best process corner SDF data. The correct timing on the interface to the analog part has to be verified manually by means of a mixed-signal simulation, as the design flow in the current state does not support automatic verification of this interface.

[23]The Verilog simulation models for the standard cells contain parameterized delay values for the signal paths through the cell as well as for setup and other timing checks. Based on the process corner, the technology library and the extracted parasitics, these values are calculated and written to the SDF file. Interconnect delays are also written to the SDF file. The simulator loads the SDF file together with the gate-level netlist and replaces the parameters' default values. Erroneous library data required modifications to the simulation models and the SDF file after generation. These modifications are executed in a script based manner as to have reproducible data.

2.5.2 Physical Verification

The physical verification involves three steps, namely the DRC, the process antenna check and the verification of LVS. All checks are performed using Mentor Graphics Calibre.

- DRC: The complete design is verified for accordance to the topological design rules of the fabrication process. Digital macro cells for which no GDSII data is available are excluded from the checks and their GDSII data is inserted by the manufacturer. To avoid wide metal spacing violations in the periphery of these macros, in the technology LEF file, the options `OBSMINSPACING OBS/PIN` have to be set to `OFF`[24].
- Antenna check: The design is verified for large metal structures connected to isolated gates (antennas). In the digital part these are repaired by the place and route software. However, antennas generated by automatic routing of analog blocks are not detected by First Encounter due to the abstraction of the VLSI structures and require manual rework. The method used for the presented chip is the insertion of diodes to provide alternative discharge paths.
- LVS: This check is performed by first extracting the devices present in the layout and generating an according SPICE netlist which represents the layout view of the ASIC. The source netlist, i.e. the schematic view, is generated with the following steps special to this flow:
  - The top-level solely exists as Verilog netlist. It is converted to SPICE syntax first.
  - The netlist of the analog part including all hierarchical instances down to the transistor level is automatically exported from the Cadence environment using CDL export.
  - Include statements are inserted into the top-level netlist for the analog netlist as well as for cells that were not present in the Verilog netlist, like power pads.
  - Global power net names are inserted into the top-level netlist.

  Both the layout netlist and the complete source netlist are compared. If the layout netlist and the source netlist match, the design is ready for fabrication and the GDSII data can be sent to the manufacturer.

2.6 Concluding Remarks

A design flow has been introduced allowing for the integration of complex analog circuits together with high speed digital logic into one ASIC by means of a modified traditional digital back and design flow. Thereby, a technique for the implementation of source synchronous interfaces has been presented. What has not been explained in detail is the simulation environment that is being used for the analog, digital and mixed-signal verifications. The verification of the analog circuits is performed using Cadence AMS Designer [Cad05b]. This software is also capable of performing full-chip mixed-signal simulations which are carried out using the digital testbench. The digital simulation environment will be described in chapter 5. The extensive simulation of the design together with the timing closure and the physical verification during and after implementation provide an elaborate design flow that reliably yields fully operational ASICs.

[24] These violations occurred in the first version of the neural network chip after macro insertion. The technology LEF file was corrected manually.

Chapter 3

Large Scale Artificial Neural Networks

This chapter describes the developed concept for the transport of digitized neural events. The concept relies on a hardware platform that is introduced first. It allows the connection of multiple ASICs using high speed serial links. The strategies applied in the following are closely related to the physical implementation of the neural network ASIC and therefore some of the hardware specific details will need to be anticipated throughout this chapter. The protocol for the communication with the chip is introduced. A generic setup for single chip experiments is shortly outlined, followed by a detailed description of the communication between different neural network ASICs using the configurable logic resources of the hardware platform. The key aspect here is the sorting of different event streams targeted for one chip. As the implementation of this concept has not been finished throughout this thesis, implementation considerations are given and simulation results are presented that demonstrate its functionality. Resource estimations that support the feasibility of this integration are given.

3.1 Existing Hardware Platform

The hardware platform serving as a basis for the construction of large scale artificial neural networks (ANNs) has been developed by the Electronic Vision(s) group prior to the development of the neural network ASIC presented in this thesis [Grü03, FGP+04, Sch06, Sch05]. It has generally been designed for the parallel operation of mixed-signal ANN ASICs. In particular, research has been done based on the Perceptron based ANN ASIC HAGEN[25] that has also been developed within the group [SHMS04, Sch05]. The hardware platform consists

[25] The acronym stands for Heidelberg Analog Evolvable Neural network.

of 16 network modules[26] interconnected by a high-speed backplane. Each individual Nathan module hosts one ANN ASIC with the according infrastructure, a programmable logic device field programmable gate array (FPGA), and local memory resources. The backplane hosts the modules and allows high-speed digital communication between them. The platform can thus be used to digitally transport neural events between different neural network ASICs which supports the implementation of the neural network model described in section 1.3.4.

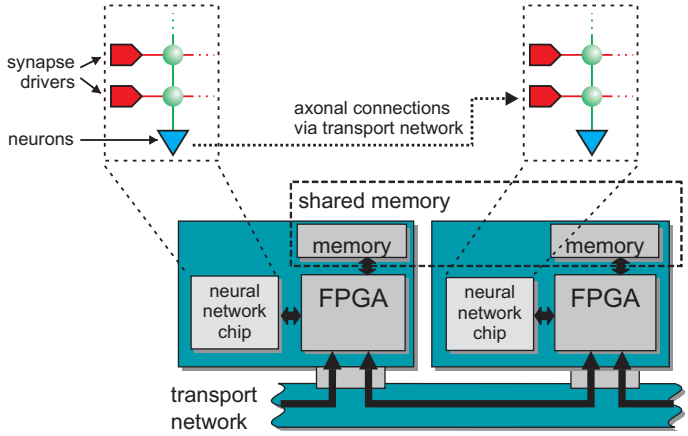


Figure 3.1: The transport network spanning several Nathan modules. Note the distinction between the neural network data transmitted between the network chips and the shared memory data.

In order to operate the Nathan modules in parallel, strategies are necessary to coordinate their distributed resources: the neural network model constantly requires input spikes and generates output spikes in a time continuous way. Since the spikes are transferred digitally, they can easily be transported by digital communication technologies, and the spikes generated by one ANN ASIC can be fed to the synapses of another one to scale up the size of the neural network. Maintaining the continuous communication between the ASICs requires a carefully designed connectivity.

The necessary high connectivity between the Nathan modules is realized by a high-speed transport network which links the modules via the backplane. This network is capable of transporting data, e.g. neural events, between the Nathan modules with a fixed and guaranteed latency which eventually enables the realization of axonal connections between neurons and synapses on different neural network ASICs as these require this very fixed connection delay (cf. section 1.2.3). To give a first glance this is illustrated in figure 3.1.

Furthermore, the transport network provides the exchange of large amounts of data between the Nathan modules to a high-level controlling software and the programmable logic. For this purpose the same transport network is used to create a large shared memory [HP95] which allows the remote access to memory resources on any Nathan module by means of a global shared memory address space.

3.1.1 The Nathan PCB

Figure 3.2 illustrates the main components of the Nathan module as well as the control PC needed for user interaction. The Nathan module contains a socket for the afore mentioned

[26]The network modules will be called *Nathan modules* in the following, not to confuse with the terms network module and neural network ASIC.

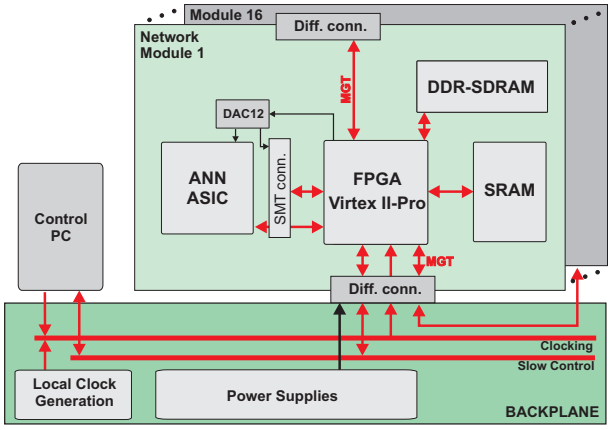


Figure 3.2: Overall schematic of the hardware platform showing the Nathan modules connected to the backplane. Figure adapted from [Grü03].

Perceptron based ANN ASIC as well as daughter card surface mount technology (SMT) connectors where the carrier printed circuit board (PCB) for the chip presented in this thesis will be mounted on. The FPGA can be configured to control all connected components and in addition to its configurable logic features eight embedded multi-gigabit transceivers with a data rate of up to 3.125 Gbit/s that are used for the high-speed transport network [Xil02a]. Four of them are routed to the backplane over a differential connector and the remaining four are available at the top of the module over an additional connector that is also suitable for wire based interconnects. The multi-gigabit transceivers (MGTs) require an accurate reference clock which is globally generated on the backplane.

For the operation of mixed-signal ANN ASICs, analog support circuitry and dedicated power supplies are required and provided on the Nathan modules. As the communication interface with the presented ANN ASIC is kept digital, it will directly be connected to the FPGA using a daughter card plugged into the SMT connectors.

To store configuration data for the ANN ASIC and experiment data, the FPGA has been connected to two 64 bit wide memory interfaces: a designated DDR-SDRAM socket which carries one memory module with an addressable capacity of up to 2 Gbyte, and two static random access memory (SRAM) chips with an overall capacity of 512 kbyte.

3.1.2 The Backplane

The backplane provides the infrastructure necessary for the Nathan modules, a serial interface to a control PC (the *Slow Control*) and the physical connections between the different Nathan modules, thereby enabling data transport using the MGTs located within the FPGAs on the Nathan modules. Four of the MGTs are used for the backplane connectivity and on one backplane 16 Nathan modules are connected in a 2D-torus manner. The resulting topology of the transport network is illustrated in figure 3.3. Due to the realization as a 4×4 matrix, the communication of two non-adjacent modules requires routing functionality within the intermediate modules, and in the worst case routing has to take place over three nodes. To satisfy higher connectivity demands, the Nathan modules may be connected using the additional MGTs available at the top of the Nathan modules by means of appropriate cables.

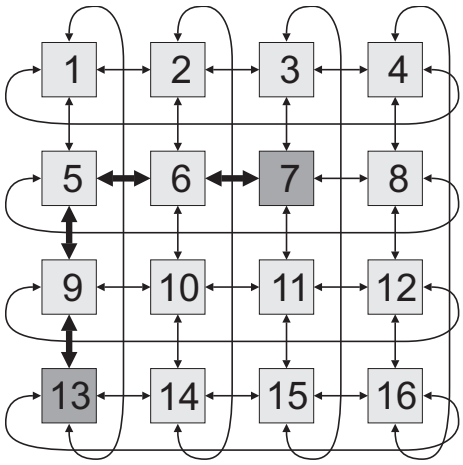


Figure 3.3: Illustration of the 2D torus topology present on the backplane. The worst case routing over passing three intermediate nodes is highlighted.

3.1.3 Transport Network

The transport network described within this section has been developed within the Electronic Vision(s) group by S. Philipp [Phi07, PGMS07] with the aim to realize a general purpose transport network among the Nathan modules located on either one or multiple backplanes and to provide the required quality of service (QoS)[27] to the transport of different types of data in this distributed system. To keep the transport network as flexible as possible and yet suited for the underlying hardware, the data to be transported is classified into two different levels as illustrated in Figure 3.1. The different needs regarding QoS depend on the particular level; therefore, a short description of these levels of data transport is given in the following.

Neural network data is referred to as level 1 data or *high priority traffic*. This data consists of neural events that are to be transmitted between the neural network chips. The biological counterpart of connections transporting these data are the axonal connections to target synapses which results in the following QoS requirements:

- Constant and small delay for all connections. The constant delay is necessary to realize the desired neural network model which assumes axonal connections with fixed delay. Another requirement is to make the connection delay as small as possible to achieve biologically realistic connection delays while maintaining the desired speed-up factor of 10^5.
- Spiking neural networks are expected to exhibit synchronized or bursting behavior (cf. section 1.2.3). The transport network needs to provide the appropriate high bandwidth to handle these peak event rates. To achieve this, a fixed fraction of the overall bandwidth needs to be reserved for each interconnection between neural network chips.
- Error correction cannot be performed for this type of data as the retransmission of an event would presumably exceed the allowable latency on the particular connection. Of course, error detection is needed and false data should be discarded.

[27] Quality of service generally refers to the quality of data transmission in terms of bandwidth, latency and reliability for a specific type of network traffic.

- The overall setup and the routing within the transport network depend on the type of neural network to be implemented and are therefore known prior to experiments. During the experiment the topology does not change.

Shared memory data is referred to as level 2 data or *best-effort traffic*. The content in the case of a multi-chip spiking neural network could be configuration data for the analog parameters or the synapse configuration data for the network chip. Level 2 traffic exhibits the following demands on the QoS:

- Connections are constantly set up and closed with different endpoints during operation and the bandwidth needs can hardly be predicted. This requires an intelligent routing algorithm to transport this type of data in the system.
- To keep the performance of the system on a convenient level the latency for the level 2 data should be as small as possible and the data throughput should be maximized.
- Error correction is needed, since the loss of e.g. neural network configuration data cannot be accepted.

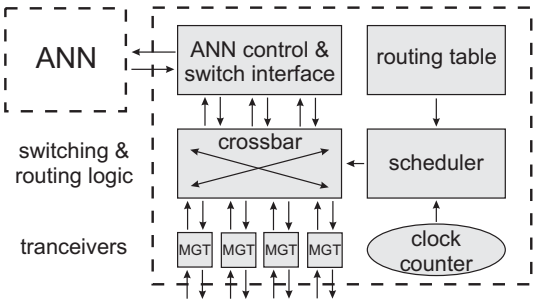


Figure 3.4: Illustration of the transport network related FPGA logic at one Nathan module. The MGT ports are external inputs and outputs to the network. The ANN chips and the shared memory (not shown) are sources and sinks of data. Figure adapted from [PGMS07].

Figure 3.4 shows an overview of the logic implemented to build up the transport network. Communication among the FPGAs, thus, the Nathan modules takes place over the Gigabit connections supplied by the built-in MGT transceivers with up to 3.125 Gbit/s. The interface to the FPGA fabric is a parallel bus with an internal width of 16 bits and a clock rate of up to 156.25 MHz[28] for the parallel interface.

Data coming from the transceivers and going to the transceivers is treated as external inputs and outputs to the transport network, whereas the data produced by the logic assigned to the network chip (*ANN control*) logic is treated as internal sources and sinks. The same applies for the level 2 data, which is not shown in figure 3.4. Data is transferred to and from the switch via *switch ports* that are symbolized by arrows pointing to and from the *crossbar switch fabric*. All of these ports operate at the rate of the MGT links with 16 bit at up to 156.25 MHz.

Depending on the switching it has to be decided during each clock cycle, which output gets its data from which input. These connections are physically made by the *crossbar* switch fabric whereas the *scheduler* configures the crossbar dynamically in each clock cycle. The task of scheduling requires an intelligent algorithm, especially for the level 2 data connections,

[28] Depending on the global clock source it is possible to adjust this frequency to up to 156.25 MHz

which have to be established dynamically on the arrival of the respective data packets. An algorithm suited for this task called *iSLIP* [McK99] has been customized for the transport network [Phi07]. In this case, data is being stored within queue memories at the input of the switch and the scheduler takes its decisions based on the content of these input queues. As a consequence of this, an optimal configuration for the crossbar can be determined with a latency of down to two clock cycles.

Isochronous Connections

The isochronous[29] transport of level 1 data is guaranteed by the transport network by sorting the different data streams to be routed prior to the operation of the network. Based on the assessable maximum bandwidth requirements of each connection and the overall number of connections that have to be routed through the switch, this sorting is accomplished by dividing the time axis into periods of the same size. Each period consists of a globally equal number of time slots. Each connection gets a certain number of such time slots assigned depending on the percentage of the overall bandwidth this connection will presumably need[30]. Each time slot is assigned to only at most one source at a time. Now the scheduling is reduced to a mere *timetable problem* and an optimal input-output assignment has to be found for each time slot. This assignment is stored in a routing table configuring the scheduler within each time slot (cf. figure 3.4). The length of one such period is set in a way that it sums up to the transmission delay introduced by the MGT connections between the Nathan modules. As a result, isochronous network operation is achieved, and a packet assigned to a time slot s_2 and a starting time t_{source} at the source switch will arrive at

$$t_{\text{dest}} = t_{\text{source}} + \Delta t_{\text{period}} \quad \text{with} \quad \Delta t_{\text{period}} = \sum_i \Delta t(s_i) + \Delta t(s_{\text{sync}})$$

at the destination switch and will still be assigned to time slot s_2 (cf. figure 3.5). Thereby, $\Delta t(s_i)$ is the time required for one time slot, $\Delta t(s_{\text{sync}})$ the duration of one synchronization slot and i loops over all time slots within one period. As a consequence, a constant delay of Δt_{period} is achieved on all connections, thereby fulfilling the definition of an isochronous network.

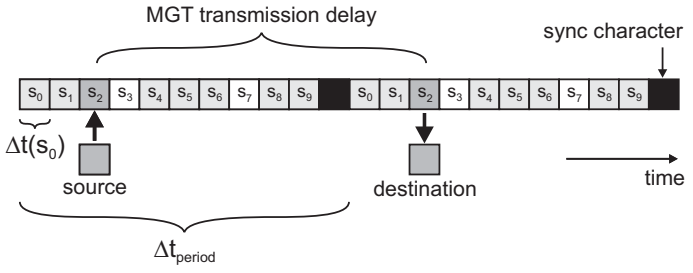


Figure 3.5: Illustration of two periods of time of the switch operation. In this case, the overall transmission latency of the MGT connection requires the time of 11 time slots. A single time slot is used to constantly check the state of synchronization. All slots but s_3 and s_7 are assigned to level 1 connections in this example and only s_2 is actually used for transmission.

[29] Network connections with nearly constant bandwidth and delay between two network nodes are called *isochronous* connections.

[30] A fraction of the time slots will be reserved for level 2 data, thus reducing the overall bandwidth available for the level 1 traffic.

To ensure that the time slots at the source and at the destination are the same, the system needs to be globally synchronized. This synchronization strategy does only work for systems having a global clock source as even small deviations in frequency lead to a divergency of the counters and therefore a loss of synchronization within split seconds [Phi07]. A global clock source is available for an entire set of 16 Nathan modules hosted on one backplane (cf. section 3.1.1). Different backplanes cannot be supplied with one global clock source in the current version. A successor PCB is currently being developed by D. Husmann [HdO06] which supports the distribution of one global clock among different backplanes. Once synchronization has been achieved, it is continuously checked by inserting a special synchronization slot after each switch period.

A Word on Event Rates and Bandwidths

On the one hand, the MGTs and thus the whole transport network operate at a maximum internal clock frequency of $f_{\max} = 156.25\,\text{MHz}$ and with a granularity $g = 16\,\text{bit}$. Based upon this, the following formula can be used for calculating the net data rate that is achievable with one port of the switch:

$$R_{\text{d,net}} = f \cdot g \cdot \frac{\text{n}_{\text{data}}}{\text{n}_{\text{data}} + \text{n}_{\text{sync}}} \quad [\frac{\text{bits}}{\text{s}}], \tag{3.1}$$

where n_{data} is the number of time slots reserved for data transmission and n_{sync} is the number of synchronization characters required.

On the other hand, an event consists of at least 9 address bits and 12 bits for the time stamp in the current implementation (cf. section 4.3.2). Therefore, the transmission of an event will require two 16 bit slots. The achievable event rate is thus given as

$$R_{\text{e,net}} = f \cdot (\text{n}_{\text{event}})^{-1} \cdot \frac{\text{n}_{\text{data}}}{\text{n}_{\text{data}} + \text{n}_{\text{sync}}} \quad [\frac{\text{events}}{\text{s}}], \tag{3.2}$$

where n_{event} is the number of time slots needed for one event.

3.2 Principles of Neural Event Processing

The construction of large scale artificial neural networks as it is presented in this thesis is based on the analog VLSI hardware implementation described in chapter 1 and the hardware platform described in section 3.1. Basically, events that are generated by the neuron circuits on one neural network ASIC are either directly fed back to local synapse drivers or shall be transported off chip and communicated to other neural network ASICs to form larger networks. Regarding the demands on the communication channels between the different chips, the digitized events are treated as a data stream that has to be processed with a fixed delay. This demand is derived from the biological specimen, where inter neuron connections are established by the axons having a certain fixed delay (cf. section 1.2.3).

The communication of neural events is subdivided into two domains within the system: on the one hand, events need to be physically sent to and received by the chip. The bundling of event streams within this domain requires the definition of a communication protocol with the chip itself and an according controller which is shortly described in section 3.2.1. On the other hand, the distribution of the events between the chips requires another protocol that accounts for the processing of events received from the controller and the transmission of these events to a target controller using the isochronous transport network. This inter-chip event processing is described in section 3.2.2.

Furthermore, the correct timing of the events within the neural network ASIC and throughout the system has to be assured as all information carried by a digitized spike is contained within the point in time of its digitization or creation. Event processing in the analog VLSI implementation is carried out synchronous to a clock signal that is available in the digital part of the chip (see section 4.3.3). Counters are provided within the chip as well as within the controller that serve as a time base for the event processing and are globally synchronized. The according event processing algorithm is described in section 3.2.3.

3.2.1 Communication with the Neural Network Chip

Communication that is directly related to the chip is subdivided into the chip internal communication and the external communication. The data flow is illustrated in figure 3.6.

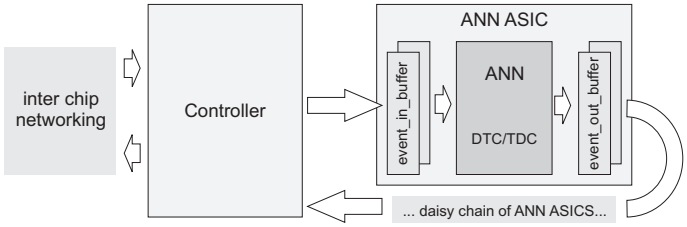


Figure 3.6: Illustration of the daisy chain of ANN ASICs. The ANN block in the ASIC comprises the full analog VLSI circuitry including TDC, DTC, synapses and neurons.

Internal Event Processing Events need to be transported to the DTCs and from the TDCs. Figure 3.6 illustrates the very basic concept. Two modules are realized accomplishing this task:

- *event_buffer_in*: Events are sent to the chip by a controller and are stored temporarily within this module. One at a time is associated to a DTC. It comprises a FIFO memory, for the following reason: the bandwidth requirement for peak event rates may exceed the bandwidth of the interface to the chip. In this case, events are sent earlier, are distributed to the different *event_buffer_in* modules and are stored there until generation. By the time the time stamp of an event matches the current system time, it is sent over to the DTC with the address of the according synapse driver and the time bin information for its generation. The controller has to ensure that events arrive in time by means of comparing their time stamp with the current system time, calculating the transport delay, and scheduling the transmission of the events.

- *event_buffer_out*: One of these modules at a time is associated to a TDC. Events that are digitized by the TDC are captured and are stored temporarily within the module until their transmission off chip. This module also comprises a FIFO memory, for similar reasons as in the case of *event_buffer_in*. During peak event rates, events are stored until sufficient bandwidth is available on the interface of the chip. The capture of a digitized event involves the generation of the event's time stamp (the current system time) and the generation of its address (with respect to the overall number of TDCs).

External Communication Event data as well as configuration data for the chip is transported via the physical interface of the chip, from the controller and to the controller. To allow

the implementation of larger neural networks it is desirable to operate more than one ASIC with a common controller. Generally spoken, different topologies are possible for the connection to a common controller. Several ASICs may be connected in an astral manner, connected to a common bus or in a daisy-chained fashion. The star topology requires a complete interface for each chip at the controller which limits the number of chips by the maximum available I/O pins at the controller. The implementation of high-speed bus topologies arises challenges to the layout of the carrier PCB as well as it requires a rather complex communication protocol and arbitration scheme (see e.g. [SA99, JG93]).

For these reasons, the daisy chain topology was chosen. The connections between the chips and the controller are point-to-point connections and the communication is packet based. Packets are generated by the controller; they contain a chip address and are handed around by the chips. Each chip gets an address in the chain and only acts upon a matching address in the packet. In this case, the chip processes the data contained in the packet and is allowed to fill the packet with its own data (i.e. events). The continuous data flow through the daisy chain reflects the streaming nature of the event data, and the electrical interface is due to its point-to-point connections suited for high-speed digital data transmission. Furthermore, the arbitration among the different chips is easy to implement by means of a suited address order in the packet stream.

The full implementation of the communication protocol will be described in chapter 4. For the following considerations these facts are of importance: to achieve a higher event rate on the interface events are stored within the data packets in a compressed fashion. This is possible, since at high event rates events will have similar time stamps. For this reason, the upper nibble[31] of the time stamp is stored only once for all events within a packet. Some figures that reflect the specific implementation of the protocol for the presented chip are summarized in table 3.1. They serve as a basis for the following considerations which are nevertheless of general nature.

Parameter	Value
events per packet	3
width of event time stamp	8 bit
compressed bits	4 bit
time bin resolution	4 bit
event address width	9 bit

Table 3.1: Figures related to the event packet format. Anticipated from chapter 4.

Furthermore, the latencies introduced by the daisy chain communication are of relevance for the temporal processing of neural events. Particularly, the following latencies are of interest for the following considerations:

$$\begin{aligned} n_{\text{ctrl}} &= \text{from controller to first chip in chain} \\ n_{\text{ch}} &= \text{through 1 chip} \\ n_{\text{eg}} &= \text{from chip input to earliest event generation.} \end{aligned}$$

Unless otherwise noted, all latencies are given in clock cycles of the clock signal used for digital event generation and digitization.

[31] A nibble is a four-bit aggregation or half a byte.

3.2.2 Inter-Chip Event Transport

The protocol described in the preceding section is only capable of transporting events between the controller and one specific chip in the daisy chain at a time. No further processing respectively routing can be performed there, not even among the chips within one daisy chain. Therefore, an event processor has been developed, which implements an event processing algorithm and accounts for the transmission of events using a protocol, which is based on the features provided by the isochronous transport network. The basic setup of this system is illustrated in figure 3.7 for one Nathan module.

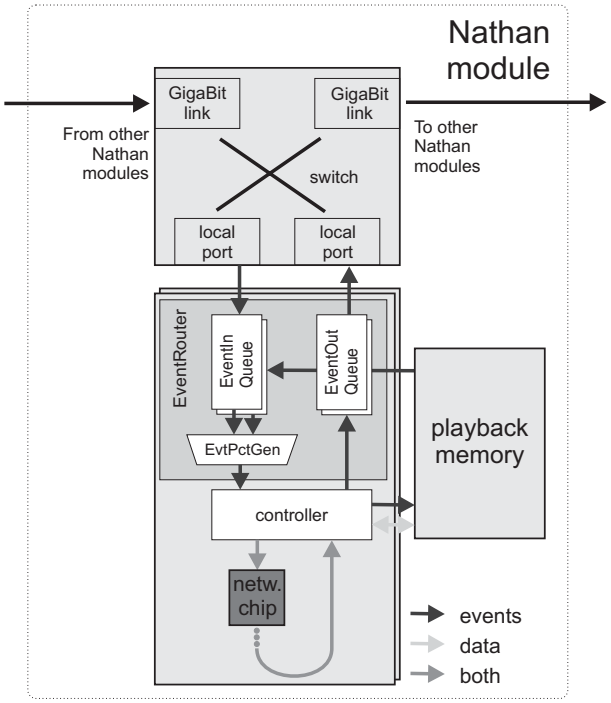


Figure 3.7: Overview of the system used for inter-chip event transport. The arrows denote the direction of data flow. The daisy chain of network chips, the controller and the *EventRouter* module form one unit connected to a switch that transports data among several Nathan modules. Several of these units may be connected to one switch if larger neural networks are desired. The playback memory is used to locally generate and record events as well as to send configuration data to the chip, and store data read back from the chip.

Keeping to figure 3.7, the following elements provide the inter-chip event transport:

- The **switch** realizes the isochronous transport network. Communication with the event processing logic takes place solely via local ports of the switch. Thereby, the protocol used for this communication has to ensure the receipt and delivery of event data within the correct time slot of the isochronous network, as inter-Nathan module connections are defined by these pre-sorted time slots (cf. section 3.1.3).
- Events generated by the chips in the chain are processed within the ***EventRouter*** and the corresponding destination events have to be produced. This is done within the *EventOutQueue* module. Events arriving from external sources are merged into the data stream sent to the daisy chain which is handled by the modules *EventInQueue* and *EventPctGen*. More concise descriptions of these modules will be given in section 3.3.

- The **controller** accounts for the communication with the connected daisy chain of neural network chips. Furthermore, it merges the data streams coming from the playback memory and from the event processing modules. It will be described in detail in section 5.2.3

- The **playback memory** serves as a local source and sink of event data as well as auxiliary configuration data for the neural network chip. It uses the local memory resources and can also be used for single chip experiments which will shortly be outlined below.

Generic Single Chip Experiments

As indicated above, the playback memory has been developed to serve as a local source and sink of event data as well as configuration data to the neural network chip. For this purpose, the local DDR-SDRAM is used as a mass storage device for both directions of transmission. A program sequence can be stored in the DDR-SDRAM which contains the data to be sent to the chips and the playback memory logic executes this program while keeping to a deterministic timing with respect to the system time counter. In parallel, the data received from the chips is written to the DDR-SDRAM and can be retrieved to the control PC after execution of the playback memory program.

The capability of real time event generation and recording is used to locally provide background activity to the neural network chips and to record their event output for later evaluation. Obviously this can also be used without the isochronous transport network present and with only one single chip in the daisy chain. This very constellation is the generic single-chip experiment setup which will also be the basis for the results presented in chapter 6.

The playback memory not only can be used to record events but it is also possible to intermittently read back the results of correlation measurements to evaluate ongoing plasticity developments within the active neural network.

3.2.3 Event Processing Algorithm

Basic Concept

The isochronous connections of the transport network are ideally suited as a basis for the fixed-delay axonal connections of the neural network to be implemented. The data flow of such a fixed-delay connection is depicted in figure 3.8. After an event has been digitized on the neural network chip, it is sent to the off-chip controller via the digital interface. Based on this source event, one or more target events are generated. The target event is addressed to a synapse row in the destination neural network chip. Depending on the synapse configuration it can therefore be routed to all neurons associated with one synapse row. In biological terms, the transmission of the event represents the axon of the source neuron, whereas the dendrite tree is realized by the synapse column associated to one neuron on the destination chip.

The time stamp of the destination event t_d is obtained by adding the according axonal delay $t_\mathrm{del,ax}$ of the connection to the time stamp of the source event, t_s:

$$t_\mathrm{d} = t_\mathrm{s} + t_\mathrm{del,ax} \ . \tag{3.3}$$

Depending on the current state of the network, this new event is either stored in an output FIFO of the *EventOutQueue* module, which is described in section 3.3.1, or it may be dropped. Consequences of event dropping are discussed in section 3.4.2.

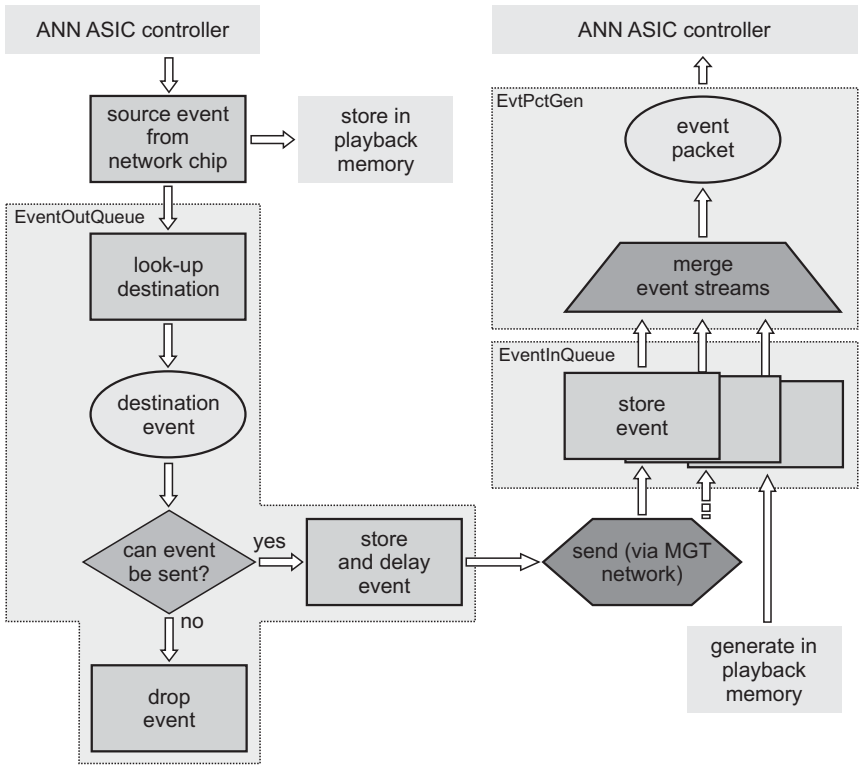


Figure 3.8: Data flow in the *EventRouter*. The modules *EventOutQueue*, *EventInQueue* and *EventPctGen* will be described in the following sections. The symbol for the MGT network implies source and destination switch and the MGT connections.

Depending on the configuration of the switch the events stored in the output FIFO memory are fetched during the according time slot and are then sent via the isochronous network. At the output of the destination switch the event is stored in the input FIFO memory of the *EventInQueue* module which will be described in section 3.3.1. Events available at the front of the FIFO memories within the *EventInQueues* are then merged into one event packet by the *EventPctGen* module which is described in section 3.3.2. Finally, the generated event packet is sent to the connected controller and subsequently to the neural network chips.

Implementation

Event streams originating at different neural network chips are processed as described above and result in event streams that have to be merged at the corresponding destination. In contrast to commonly applied methods for data stream processing (see [MAG+02] for an overview) where the number of data packets in a stream and their order is normally known or defined by special packets, this system has to handle event streams with a rather random rate and random relative timing. As the randomness of the event data is an inevitable fact, an algorithm has been developed that minimizes the sorting effort at the destination and guarantees an optimum network bandwidth utilization. To minimize the sorting effort, the following steps have to be taken:

- Introduce *virtual connections* that combine several neuron-neuron connections with equal axonal delay. Since one neural network chip produces events with ascending time

stamps, the time stamps of target events will still be in ascending order, provided that the connection delay is the same for all combined connections. This first step greatly reduces the number of event streams that have to be merged at the target. One virtual connection is made up of a pair of one *EventOutQueue* and one *EventInQueue*.

- Events have to be *delayed at the source* before sending them to the switch until the latest possible time slot of the switch's transmission cycle. As illustrated in figure 3.9 all events arriving at a particular destination are only valid within a certain time window and thus have only to be sorted within this time window. In section 3.3.3 it is shown that this is a viable way of sorting as the sorting task can be fulfilled with a manageable number of comparators as long as the time window is small enough, thereby requiring small comparators for the time stamps.

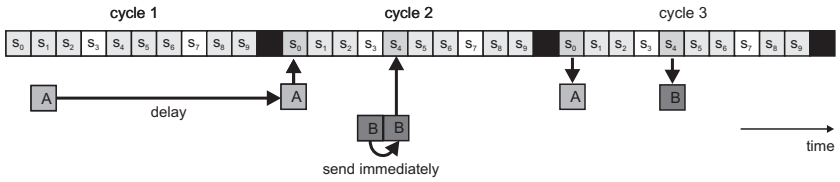


Figure 3.9: Illustration of the strategy to delay events at the source. Events A and B are both due for generation in cycle 3. Event A arrives too early and is thus delayed. Event B is actually ready for sending on arrival but it is sent in time slot s_4, because s_3 is not assigned to level 1 connections in this case. Events A and B need to be temporally sorted in cycle 3 only.

An important aspect of the presented event processing algorithm is the handling of high event rates, especially in case the event rate exceeds the available bandwidth of the isochronous network. Different solutions to this problem can be found in [Tan03]. One generic approach to handle peak rates would be to provide FIFO memories that are large enough to temporarily store the data at the source until sufficient bandwidth is available. The problem with neural events is that their time stamp may become invalid during the storage in the FIFO which would require the events to be dropped. Even worse, if this situation occurs and the event rate stays at a high level, all events after the first one with an invalid time stamp may have to be dropped because they have been stored at the source for too long. A solution to this problem is the implementation of an extended *Leaky Bucket* algorithm [Tan03].

Before the afore mentioned submodules and the event processing algorithm itself are described in more detail in section 3.3, the following section deals with different possible topologies for the event processing.

3.2.4 Layers of Event Processing

One virtual connection basically consists of one *EventOutQueue* and one *EventInQueue*. In principle, the connection between them can be made arbitrarily and is not bound to the usage of the MGT connections. This suggests the categorization of the event transport into four layers[32]:

[32]The term *layer* is used here to distinguish the description from the categorization of the transport network traffic into *levels*.

- Layer 0: Local feedback connections on the neural network chip. These connections feature a fixed delay which may vary from approximately 0.5 ns to 2 ns depending on the position within the *network_block* of the involved neuron and synapse circuits.
- Layer 1: Connections within one daisy chain hosted by one controller. The switch is not needed for these connections and the output and input of the corresponding *EventOutQueue* and *EventInQueue* can directly be connected. The latency of this layer depends on the known latencies of the neural network chip, the controller, and the latencies of the event processor logic. The latter will be estimated in section 3.3.3.
- Layer 2: Connections between network chips hosted by different controllers that are connected to one single switch. The latency of this layer is the sum of the layer 1 latency and the latency of the isochronous network. If this type of connection is used, the latency of the isochronous network will be reduced to the time it takes the event data to propagate through the local switch on one Nathan module.
- Layer 3: Connections between chips that are located on different Nathan modules and are connected via the MGT network. This is the connection with the longest physical range and with the highest delay. In this case, the MGT network latency is included into the overall latency. As the current hardware only supports single-chip operation on one Nathan module (cf. section 5.1.1), this is the layer on which the focus of the simulations performed in section 3.5 is set.

The minimum axonal connection delay for a layer 3 connection can be calculated as

$$t_{\mathrm{d,min}} = (n_{\mathrm{co}} + n_{\mathrm{fi}} + n_{\mathrm{po}} + n_{\mathrm{in}} + n_{\mathrm{pi}} + n_{\mathrm{fo}} + n_{\mathrm{ci}})/f \ , \tag{3.4}$$

where
$n_{\mathrm{co,ci}}$ = cycles needed for event digitization and generation in the neural network chip including daisy chain delay
$n_{\mathrm{fi,fo}}$ = input and output latency of the controller
$n_{\mathrm{pi,po}}$ = input and output latency of the *EventRouter*
n_{in} = latency of the isochronous network
f = clock frequency.

The clock frequency f is always given in terms of the clock used for event generation and digitization as this defines the lowest granularity in clock cycles. The actual values for $n_{\mathrm{co,ci}}$ can be found in section 4.3.1, the values for $n_{\mathrm{fi,fo}}$ are given in section 5.2.3. The delays of the local feedback connections on the network chip are excluded from these considerations. Delay values for layers other than layer 3 can be obtained by setting the appropriate number of clock cycles to zero.

Table 3.2 summarizes the latencies of the different layers in terms of biological real time. The latency values of the event processing logic have been anticipated from section 3.3.3. Layer 0 has a fixed axonal delay of approx. 0.1 ms in terms of biological real time whereas the latency of the other layers starts with a minimum of 4.5 ms. Whereas the gaps between layers one and above may be bridged by setting the actual delay of the respective virtual connection to an appropriate value, the gap between layer 0 and layer 1 is inherent to the system and cannot be bridged. The resulting gap of min. 4.5 ms leads to inconsistencies of the neural network model and a variety of problems: as described in section 1.2.3 and [MMS04], one of

layer	$n_{\min}$ [Cycles]	$t_{\min,312}$ [ns]	$t_{\min,400}$ [ns]	$t_{bio,312}$ [ms]	$t_{bio,400}$ [ms]
0	-	1	1	0.1	0.1
1	9	57.6	45	5.76	4.5
2	11	70.4	55	7.04	5.5
3	37	236.8	185	23.68	18.5

Table 3.2: Minimum connection delays of the different layers of event processing. Number of cycles is given in terms of the event clock of the neural network chip and times are calculated for 312 MHz and 400 MHz respectively. The according biological axonal delays are calculated for both clock frequencies and a speed-up factor of 10^5 is assumed compared to biological real time.

the key aspects in the dynamics of spiking neural networks is the development of synchronized activity over time. This activity is supposed to be mainly caused by very short range excitatory connections with axonal delays t_d in the range $0\,\text{ms} < t_d \leq 5\,\text{ms}$. Even with very high developmental effort, the latency of the layer 1 event processing cannot be made much smaller as the latency is mainly caused by the already implemented neural network chip and the optimization of the controller logic would gain a maximum improvement of a total of possibly 4 clock cycles. Another possibility to decrease this gap is to decrease the operating speed of the analog neural network circuits in a way that the time constants of the model described in equation 1.1 are increased. The following parameters would have to be modified (cf. section 1.3.2 and 1.3.1): the rise and fall times of the spike reconstruction circuitry have to be increased and the leakage current of the membrane as well as the comparator speed for the spike generation have to be decreased. The lowest possible speed-up factor is approx. 10^4 compared to biological real time [Scha] which would result in a minimum latency of 0.45 ms for a layer 1 connection and 0.01 ms for a layer 0 connection. In this setup, the gap is decreased by one order of magnitude. The investigation of the consequences of the very presence of this gap are subject to future work and will be done by D. Brüderle [Brü07].

3.3 Neural Event Processor for Inter-Chip Communication

In this section, the implementation of the previously described event processing algorithm within the different modules *EventInQueue*, *EventOutqueue*, and *EvtPctGen* is described. Packet size and data widths are chosen according to the values given in table 3.1 and could be adapted to different scenarios without loss of generality .

3.3.1 Event Queues

EventOutQueue

The functionality of this module can be divided into two parts: first, the destination events are generated and a customized version of the leaky bucket algorithm is implemented. Second, the destination event is delayed and not sent before the latest possible time slot.

Figure 3.10 shows the data flow within the module in detail. Events received from the neural network chip are stored by the controller, keeping the original packet structure with up to three events per packet. The output of this register is applied to all *EventOutQueues* in parallel. As the module has to be able to process up to one event packet per clock cycle, it is necessary to generate the destination events for all three event slots in parallel. In a first step, the destination event is generated based on the mapping described in section 3.4.2. The mapping of source

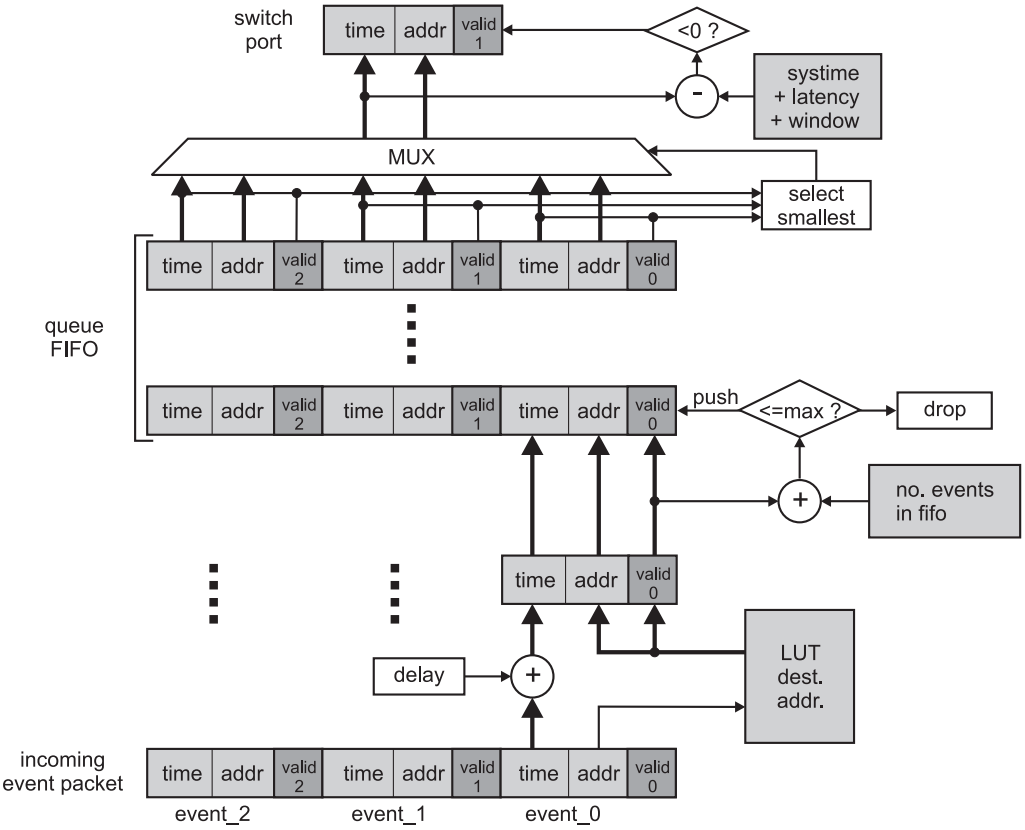


Figure 3.10: Data flow in the *EventOutQueue* module. Systime has to be connected to the controller's system time counter. The logic connected to event_0 is the same for event_1 and event_2.

neuron addresses to target synapse addresses is stored within a LUT and if an entry exists, the source address of the event will be replaced by the target address. Source chip address and destination chip address are equal for all of these connections as well as the axonal delay. Consequently, this information only needs to be stored once per *EventOutQueue*.

In a second step, the events will be stored within the FIFO memory, if the size of the leaky bucket is not exceeded. This size is calculated based on the maximum allowable delay $t_{\mathrm{stor,max}}$ imposed by the storage of the event in the FIFO memory,

$$t_{\mathrm{stor,max}} = t_{\mathrm{del,ax}} - t_{\mathrm{lat}} \ , \tag{3.5}$$

where $t_{\mathrm{del,ax}}$ is the axonal delay and t_{lat} is the transmission latency of the virtual connection. Values for t_{lat} are given in section 3.2.4.

Using the event rate $R_{\mathrm{e,net}}$ from equation 3.2 and the number of events currently stored in the FIFO $\mathrm{n_{ev}}$, the storage time within the FIFO is

$$t_{\mathrm{stor}} = \frac{\mathrm{n_{ev}}}{R_{\mathrm{e,net}}} \ . \tag{3.6}$$

Using equations 3.5 and 3.6, the maximum number of events and thus the size of the leaky bucket can be calculated as

$$\mathrm{n_{ev,max}} = (\mathrm{t_{del,ax}} - \mathrm{t_{lat}}) \cdot \mathrm{R_{e,net}} \ . \tag{3.7}$$

If the additional storage of an event exceeds this number, the event will be dropped. $n_{ev,max}$ has a constant value and only requires the number of events within the FIFO to be calculated during operation.

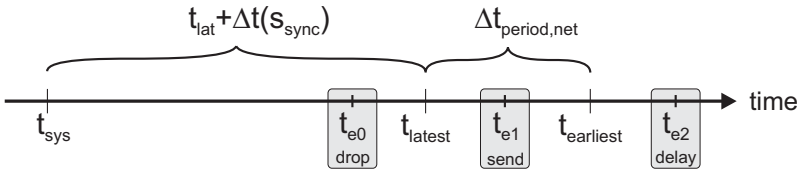


Figure 3.11: Time window used to delay events at the output of *EventOutQueue*. An event with time stamp t_{e0} would arrive too late and needs to be dropped. An event with time stamp t_{e2} would arrive too early regarding the time window allowed for sorting at the destination and is delayed. The event with time stamp t_{e1} is sent.

If an event packet appears at the output of the buffer, the connected logic first has to determine the contained event with the smallest time stamp. The strategy to delay the events at the source is the following: only if its time stamp occurs within a sliding time window, the event will be sent. Figure 3.11 illustrates this window which is calculated as

$$t_{\text{latest}} \leq t_{\text{e}} < t_{\text{earliest}} \,, \tag{3.8}$$

$$\begin{aligned} \text{with} \quad t_{\text{latest}} &= t_{\text{sys}} + t_{\text{lat}} + \Delta t(s_{\text{sync}}) \\ t_{\text{earliest}} &= t_{\text{latest}} + \Delta t_{\text{period,net}} \end{aligned}$$

and t_{sys} being the current system time. Furthermore, t_{lat} is the overall latency from transmission of the event until event generation within the neural network chip (cf. equation 3.4). $\Delta t_{\text{period,net}}$ is the net time of one switch period excluding the time for synchronization frames $\Delta t(s_{\text{sync}})$ and t_{latest} is the latest possible point in time a particular event can be generated within the neural network chip. Hence, events with a time stamp smaller than t_{latest} need to be dropped. According to the window size, events with time stamps greater than t_{earliest} are temporarily delayed. Dropping does actually not happen at this point as the *leaky bucket* algorithm prevents the storage of events that would arrive too late.

Depending on the mapping of the neural network to the hardware there may be more than one virtual connection assigned to one switch port and the bandwidth is shared by these connections. In this case, several *EventOutQueues* are assigned to dedicated time slots and only transmit data within appropriate time slots. The according fraction of the maximum event rate $R_{\text{e,net}}$ has then to be used for the calculation of equation 3.7.

EventInQueue

The *EventInQueue* module is the counterpart of the *EventOutQueue* module within one virtual connection. As in the case of *EventOutQueue*, several *EventInQueues* can be connected to one switch port and can be assigned to different time slots. Events are only processed during time slots that a particular *EventInQueue* is assigned to. The need for data storage at the destination arises from the fact that several event streams need to be merged (see the following section 3.3.2) and not all incoming events may be immediately packed into an event packet. Storing events at the destination leads to a delayed delivery and might lead to invalid time stamps just like in the case of the *EventOutQueue*. This becomes particularly problematic when many

streams with high rates have to be merged and the overall rate exceeds the data rate of the link to the neural network chips. To avoid the invalidation of all following events, the size of the buffer memory is also limited and as a result, another leaky bucket is implemented. The actual optimum size depends on the specific setup. Example results will be given in section 3.5.

For downstream event processing, the time stamp of the foremost event in the FIFO needs to be normed on the system time in terms of its daisy chain delay:

$$t_{\mathrm{ev,norm}} = t_{\mathrm{ev}} - (a \cdot t_{\mathrm{lc}} + t_{\mathrm{le}} + t_{\mathrm{pg}}) \ , \tag{3.9}$$

where t_{ev} is the time stamp of the event, a is the destination chip address in the daisy chain, t_{lc} is the latency through one chip in the chain, t_{le} the latency from the input of the neural network chip until event generation and t_{pg} is the latency of the *EventPctGen* module.

3.3.2 Event Packet Generator

The *EventPctGen* module finally merges incoming event streams at the destination, chronologically sorts the events and generates event packets that are then sent to the connected daisy chain of neural network chips by the controller. Besides these tasks this module also has to avoid deadlocks of the *event_buffer_in* modules on the neural network chip as described for the software generating the playback memory program in section 5.3.2. Since no software is involved in the event processing, the evaluation of the event time stamps and the state of the *event_buffer_ins* has to be done in hardware.

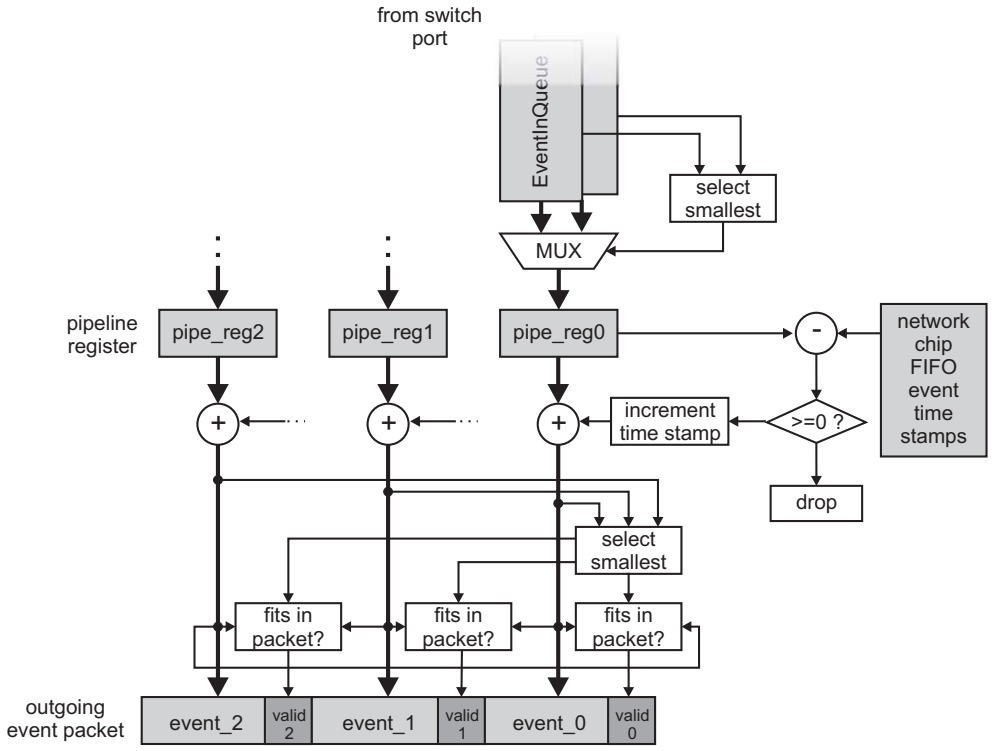


Figure 3.12: Data flow in the *EventPctGen* module. Logic connected to the pipeline registers pipe_reg1 and pipe_reg2 has been omitted for clarity.

For clarity, the following description refers to a setup with only one neural network chip in the daisy chain. Figure 3.12 shows the data flow within the *EventPctGen* module. To minimize

the number of multiplexers and the routing complexity, fixed assignments of *EventInQueues* to slots within the event packet are made. All *EventInQueues* present in the system should be distributed over the packet slots with respect to the expected event rate while providing equal overall rates for all slots.

The first step is to determine the event with the smallest time stamp within all *EventInQueues* connected to one packet slot. This event is then stored in a pipeline register.

In the second step, the time stamps of the stored events are compared with the time stamps of the most recently stored events within the according *event_buffer_in* module on the neural network chip[33]. A possible deadlock of this module will be avoided by incrementing the time stamp of the stored event, if it is equal to the most recently stored one. If the time stamp of the stored event is smaller than the most recent one, it would have arrived too late and would need to be dropped[34]. According to this correction, the valid event with the smallest time stamp is determined which defines the common upper nibble of all time stamps within the packet (cf. section 3.2.1). If the remaining pipeline registers also contain valid events, those will also be transmitted—provided that their time stamp matches the upper nibble of the packet and that the events need not to be stored within identical *event_buffer_ins* on the neural network chip. Such collisions are avoided by a simple priority scheme:

- The event with the smallest time stamp is always sent.
- If the above condition is not true and the same buffer is addressed, then slot i will be prioritized over slot $(i+1)$ mod 3 with $i \in \{0,1,2\}$.

This checking is done in the box denoted by "fits in packet?" in figure 3.12.

Subsequently, the event packet is assembled and a decision has to be made on whether to send it immediately or not. A straight forward implementation would be to store the events in the FIFO memories within the *EventInQueues* until the very latest possible time and to transmit them only then. However, this implies the problem that even very short peak rates cannot be handled by the system as there might be many events with equal time stamps arriving within the same clock cycle and only three of them can be transmitted in one packet. The chosen solution to avoid this problem is to set the latest possible time for event transmission for a few clock cycles earlier as to gain some headroom to handle peak rates. The packet is then sent as soon as the following condition is met:

$$t_{\text{sys}} \leq t_{\text{ev,norm}} < t_{\text{sys}} + \Delta t_{\text{window}} \ , \tag{3.10}$$

where $t_{\text{ev,norm}}$ is the value obtained from equation 3.9 and Δt_{window} is the allowed timing margin into the future. Here, the problem arises that, if transmission starts too early (Δt_{window} is too large), events arriving after the early transmission will run late and will have to be dropped at the pipeline register stage. For the simulations presented in section 3.5, a value of

$$\Delta t_{\text{window}} = \frac{10}{f} \ \text{ with } f \text{ being the clock frequency,}$$

has been proven to yield the best results.

If $n > 1$ neural network chips are attached to the daisy chain, the logic described in this subsection should be implemented in a parallel manner for each of them. n event packets are

[33] These time stamps are monitored by registers in the box "network chip FIFO event time stamps".

[34] To catch the case the most recently stored time stamp has also been corrected, the developed algorithm offers the option to increase expired time stamps within a certain margin. This can be done with the logic used for the regular correction with only marginal modifications.

then available, each with its own value for the smallest $t_{\text{ev,norm}}$. In this case, the packet with the overall smallest $t_{\text{ev,norm}}$ is to be sent first. The time stamps of the events sent are finally marked as most recently sent to the according *event_buffer_in* module on the neural network chip.

3.3.3 Implementation Considerations

The implementation of the *EventRouter* is subject to future work and therefore—within the presented thesis—only the concept and recommendations for the final implementations are shown. In the first version, the *EventRouter* will presumably be implemented in the FPGA of the Nathan module whose resources have to be shared among the logic needed for the implementation of the logic modules comprising the whole system: the basic environment of the FPGA consists of a memory controller and an external control interface (see section 5.2.1). For the operation of large scale spiking neural networks, four additional components are required: the controller logic, the playback memory, the switch and the *EventRouter*. This section gives first recommendations on the HDL[35] implementation of the *EventRouter* and based upon these, the resource consumption of the final design is calculated to have an estimate whether the design will fit into the FPGA or not.

Concerning the whole system, the following points can already be predefined:

- The maximum clock frequency for the MGTs is 156.25 MHz. This should also be chosen as the operating frequency for the controller and the neural network chip itself because asynchronous operation would introduce additional delay.
- As described in section 3.1.3, a single neural event consists of at least 12 bit of timing information and 9 bit for the neuron/synapse address. As a result, two 16 bit switch time slots at a time are required to transmit one event and the resulting rate for the resulting *event time slots* within one switch period is 78.125 MHz.

Recommendations for the Different Modules

EventOutQueue The *EventOutQueue* module (cf. figure 3.10) has to provide three interfaces. For the incoming events this should be a direct connection to the registers located within the controller, where incoming events are stored for the first time. On the output side, an adaption to the data structure expected by the VHDL implementation of the switch will be necessary for two reasons: first, data is being exchanged between the two synchronous clock domains which differ by a factor of two in frequency. Second, the current time slot of the switch has to be evaluated and a decision has to be made whether the module is addressed within this time slot or not. Besides the interfaces needed for event communication, an interface for configuration purposes and for monitoring features is needed which connects to the global FPGA infrastructure (cf. section 5.2.1).

As the virtual connection to which an *EventOutQueue* is assigned automatically defines the destination chip address and the axonal delay, these two values need only to be stored once inside the module. The mapping of a source neuron address to a destination synapse row address has to be stored within a LUT. The *Block SelectRAM* resources of the Virtex-II Pro FPGA [Xil02a] are ideally suited for this task when storing the value of the destination address at the Block SelectRAM memory address representing the source neuron. This look-up has

[35] The *EventRouter* could be implemented using either Verilog or very high speed integrated circuit (VHSIC) hardware description language (VHDL) as a HDL.

Component	# required	BRAMs	Flip-Flops	LUTs	Slices
Comparator $a < b$, 8 bit	3	–	–	32	16
Subtractor, 2 bit	1	–	–	40	25
Adder/Subtractor, 8 bit	3	–	–	40	25
Adder, 12 bit	3	–	60	60	30
Smallest out of 3, 8 bit	1	–	–	30	15
Multiplexer 3-in-1, 21 bit	1	–	–	21	21
FIFO 66×64, BRAMs	1	2	153	78	96
Register for data path	100	–	100	–	50
Memory for constants	20	2	20	–	10
Total		**4**	**309**	**221**	**241**

Table 3.3: Logic components needed for the implementation of *EventOutQueue*. The resource consumption has been either obtained by synthesizing exemplary VHDL code or is based on the information supplied by the Xilinx CORE Generator [Xil05c] for generic components like comparators.

to be done for up to three incoming events in parallel. One Block SelectRAM features two independent address ports, so three LUTs will consume two such modules. The calculation of the destination time is done based on the delay value stored globally for the *EventOutQueue* module and requires a 12 bit adder. Destination time and address are stored after calculation which requires another 21 registers.

The calculation of the number of events currently stored within the FIFO memory requires one 2 bit subtracter, one adder with a width of max. 8 bit and an according comparator. The subsequent storage of the events could then be done in a FIFO memory based on two Block SelectRAMs that account for the data width of 66 bit which is needed to store up to three events in parallel.

On the pop-interface of the FIFO memory suitable asynchronous logic is needed that determines the event with the smallest time stamp. A suggestion for the implementation of this circuitry is the *Parallel Rank Computer* described by Hirschl et al. in [HY04]. The event with the smallest time stamp is then multiplexed to the output by a 21 bit 3-to-1 multiplexer. Finally, equation 3.8 needs to be calculated and the event can then be sent. Along with the multiplexer logic, this requires one comparator, one adder and one subtracter. Based on these considerations, the required components and their estimated resource consumption are summarized in table 3.3.

EventInQueue This module basically acts as a FIFO memory. The depth of this FIFO may be relatively small as described in section 3.3.1. In particular it has been proven by simulations that depths over 16 entries are not required (data not shown). Knowing this, another feature of the Virtex-II Pro FPGA can be exploited: each LUT of the FPGA fabric may be configured as 16 bits of *distributed SelectRAM+* which can be seen as memory resources equally distributed over the FPGA fabric. The implementation of FIFO memories using distributed SelectRAM+ resources is described by Xilinx, Inc. in [Gop05] and a customized version of this FIFO has exemplarily been synthesized for the purpose of resource estimation. The only arithmetics required within *EventOutQueue* is the calculation of the front event's time stamp normed on its transmission latency in the daisy chain based on equation 3.9 which requires one 8 bit subtracter. As in the case of *EventOutQueue*, some glue logic to interface to the switch ports

is additionally needed. The required components and their estimated resource consumption are summarized in table 3.4.

Component	# required	BRAMs	Flip-Flops	LUTs	Slices
Adder/Subtractor, 8 bit	1	–	–	8	5
FIFO 21 × 16, BSRAMs	1	–	34	38	20
Register for data path	45	–	45	–	23
Memory for constants	20	–	20	–	10
Total		**–**	**99**	**46**	**58**

Table 3.4: Logic components needed for the implementation of *EventInQueue*. The resource consumption has been either obtained by synthesizing exemplary VHDL code or is based on the information supplied by the Xilinx CORE Generator [Xil05c] for generic components like comparators.

EventPctGen In section 3.3.2 the structure of the event packet generation logic (*EventPctGen*) has been described (see also figure 3.12). As indicated there, the actual implementation strongly depends on the number of virtual connections terminating at this module. If this number exceeds the number of events that can be packed into one data packet for the network chip, the *EventInQueue* modules are clustered at the single packet slots. Pipelining is necessary and in the first cycle, the *EventInQueue* containing the event with the smallest time stamp for delivery has to be determined. Resource consumption is estimated using the *Parallel Rank Computer* that has been used to select an event out of *EventOutQueue* in section 3.3.3. After this comparison, the three selected events are stored within three 21 bit pipeline registers.

The next step involves the comparison of the time stamps of the selected events with the time stamps of the events that have most recently been stored in the *event_buffer_in* modules on the neural network chip. In figure 3.12 it is indicated, that this requires one 8 bit comparator and one 8 bit adder for each event slot in the data packet. The elements used for the storage of the time stamps within the *event_buffer_ins* depend on the timing margin left during implementation. Regarding timing (e.g. positive slack), the most promising approach would be an implementation using flip-flops but this requires $\#\,\textit{event\_buffer\_ins} \cdot \text{width} = 16 \cdot 8 = 128$ registers per neural network chip. If timing is not an issue, an implementation using Block SelectRAM or Distributed SelectRAM would be most resource efficient.

VHDL code implementing the event packing algorithm described in section 3.3.2 has been written and synthesized to estimate the resource consumption of the afore described logic as well as the logic needed for the selection of the event with the smallest time stamp. Depending on the timing margin that results from the implementation, it might become necessary to introduce another pipeline stage between the existing pipeline registers and the registers for the final event packet. To have a conservative estimate, this is considered in the resource consumption of the module. Simulation results are not supposed to change due to this pipeline stage, since it only increases the minimum transmission latency that can be achieved on one virtual connection and has no influence on the other stages of the event processing. Table 3.5 summarizes all these components including the additional pipeline stage and the 128 additional registers. The numbers are calculated for three *EventInQueues* connected to the module.

This module needs to be adapted to the existing FPGA modules at two points. Data sent by the playback memory and the event packets generated herein need to be merged. This integration can be done seamlessly as the output data path of the controller is already implemented in a way that prioritizes event packets (see section 5.2.3). Therefore, event processing should be

Component	# required	BRAMs	Flip-Flops	LUTs	Slices
Comparator $a < b$, 8 bit	3	–	–	24	12
Adder/Subtractor, 8 bit	3	–	–	24	15
Smallest out of 3, 8 bit	1	–	–	30	15
Multiplexer 3-in-1, 21 bit	1	–	–	21	21
Fits-in-Packet logic	3	–	–	57	33
Register for time stamp storage	128	(3)	128	–	64
Register 2 $\times$ pipeline	132	–	132	–	66
Register for event packet	70	–	70	–	35
Memory for constants	64	–	64	–	32
Total		**–**	**394**	**156**	**293**

Table 3.5: Logic components needed for the implementation of *EventPctGen*. The resource consumption has been either obtained by synthesizing exemplary VHDL code or is based on the information supplied by the Xilinx CORE Generator [Xil05c] for generic components like comparators.

disabled during the initialization and configuration phase of the chip to make sure all data are completely transmitted.

Background Activity for the Neural Network As described in chapter 1 the simulation of neural networks either in software or in hardware will always be restricted to a very small fraction of the biological system. Thus, to develop a most realistic scenario, it is important to stimulate the set of simulated neurons with some sort of background activity that mimics the context of the biological network for the modeled neural network. In particular, this is required to induce the high-conductance state of the simulated neurons [DRP03, MMS04]. This background activity is provided locally on each Nathan module by events that are generated by the playback memory. This concept is also being used to provide the entire input for single-chip experiments with the presented chip by D. Brüderle [Brü07] and can easily be implemented together with the *EventRouter* logic.

The respective outputs of the playback memory are connected to a corresponding *EventInQueue* within the *EventRouter* as shown in figure 3.7. Events providing the background activity are then sent by the playback memory in a way that their timing is equal to the timing of the incoming event streams. To accomplish this, the algorithm to generate the playback memory program (see section 5.3) needs to be modified in a way that it does not use the earliest possible point in time to transmit an event, but rather the latest possible point in time. Doing so, the events supplied by the playback memory will have similar time stamps as the ones coming from the switch.

3.3.4 Estimated Resource Consumption

An exact estimation of the resource consumption of the final design is not possible. Several factors need to be considered for a rough estimation: to begin with, the estimations given above only consider the components that are supposed to consume the most resources and neglect the functionality required for the integration into the existing logic framework. It is also not predictable to what extent the routing will consume resources. For reason of these considerations it has been decided to include a certain overhead into the calculations. Only a small error is expected for the number of needed flip-flops, the overhead is set to 20 %. The

overhead for LUT and slice utilization is set to a conservative value of 40 % as these values are harder to predict.

Table 3.6 lists the resource consumptions including the overhead for the *EventRouter*. The values for the controller are taken from the final mapping report and correspond to the complete design including the external interface and the memory controller. The values for the switch represent a complete design including the synchronization logic with two switch ports to the MGT network and two to the *EventRouter*[36]. No overhead has been calculated for the usage of Block SelectRAMs.

Component	BRAMs	Flip-Flops	LUTs	Slices
EventRouter	11	1571	1064	1345
Controller	22	3451	3911	3203
Switch	0	319	768	500
Total Available	44	9856	9856	4928
Util. EventRouter	25.0%	15.9%	10.8%	27.3%
Util. Total	75.0%	54.2%	58.3%	102.4%

Table 3.6: Compilation of the resource consumption of the switch, the controller and the estimated overall resource consumption of the module *EventRouter* including an overhead. The numbers are calculated relative to the resources of the Virtex-II Pro FPGA XC2VP7 that is being used on the Nathan module [Xil02a].

The most reliable numbers in this calculation are the numbers of registers that will be used by the final design because it became obvious in the preceding sections how many data will have to be stored within one clock cycle. Logic functions that need to be implemented have been pre-synthesized and the calculated numbers include a conservative overhead. The final value for the occupied slice resources does not only depend on the logic functions to be implemented but will also be strongly affected by the routing of the design as the router often has to use entire slices as mere route-through logic. In contrast to this, the placer is able to combine storage elements and logic functions independent of each other into one slice which is referred to as resource sharing [Xil05b]. Due to resource sharing, the slice utilization is supposed to drop below 100% for the final design. This argument is supported by the exemplifications in [Xil05b].

To conclude, it can be stated that a design which is able to transport the data of a neural network consisting of four neural network chips and one random event source (e.g. the playback memory) per Nathan module should fit into the logic resources provided by the local FPGA.

3.4 Simulation Environment

The presented event routing algorithm has been developed to transfer event data between several neural network chips that constitute one large scale neural network. In the previous sections the concept and the ideas behind this algorithm were introduced and a complete description of the algorithm was given. To be able to assess the quality of the presented algorithm, the functionality of all modules is implemented in an object-oriented manner using C++ [Str97]. Based upon this, a complete simulation environment has been developed that includes behav-

[36]The values have been obtained by S. Philipp [Phi07], whose PhD thesis work is the implementation of the isochronous transport network.

ioral models of the presented network chip as well as models of the switch and for the physical layer of the transport network.

3.4.1 Operation Principle of the Simulation Environment

In figure 3.7 an overview of the whole system for the event routing is given. For the simulation, each depicted module is realized as a separate C++ class. Each class provides appropriate access functions and methods for the communication with other modules and for internal data processing. Communication with other modules is solely done via container classes that do only contain interface-specific data payload and auxiliary functions for verification purposes. Thereby, it is ensured that the functionality of each module has no dependency on data, other than the data that is planned to be exchanged via the physical interface between the modules. The communication data structures are:

- *Event* contains the full address and time information of one event. The address includes the address of the switch, the connected controller and the neural network chip within the corresponding daisy chain. For events generated by the network chip the neuron address field means the neuron number that generated an event. In all other cases the destination synapse row is addressed.

- *SpikeyPacket* contains up to three *Event* structures. According to the actual format of the packets processed by the network chip (cf. section 4.3.1), the identical upper nibble for all events is stored additionally.

- *RouterPacket* is the data processed by the switch and the MGT network. Besides some status bits that resemble the physical interface of the switch, this structure stores the time slot within which it has been sent. The data payload of *RouterPacket* may be defined arbitrarily but in the current setup it only carries *Event* structures.

The neural network chip is internally modeled by six neuron blocks according to the real structure described in subsection 4.3.5. Each of these blocks can randomly generate events for up to 64 neurons. Output event packets are assembled in the same way as in the real chip. Additionally, the ability to read text files is implemented for the neuron blocks to inject the activity yielded from a software simulation into the network. No such data is available so far, so the simulations are run with randomly generated events.

To model the physical layer of the transport network, the transmission latencies of all involved modules starting from the input of an MGT transmitter on the FPGA and ending at the output of an MGT receiver are modeled using a double ended queue from the C++ standard template library (STL)[37]. The length of this double ended queue is fixed and matches the overall transmission latency of the physical layer in clock cycles. The data stored within this container is of type *RouterPacket*. All other data structures, either FIFO-like queues of *EventInQueue*, *EventOutQueue* or other array structures, are also implemented using the STL containers where possible.

[37]The STL provides a ready-made set of common classes, such as containers and associative arrays that can be used with any built-in type and with any user-defined type that supports some elementary operations (such as copying and assignment). The container introduced here, the *double ended queue*, can be treated as a shift register if it is used with fixed length. The length can be kept constant by simultaneously executing a push and a pop operation within the same cycle after an initialization with a certain length. FIFO operations like pop and push are supported by almost all STL containers.

The simulation of the system is done cycle-accurate in order to have an accurate simulation of the timing of the network. Cycle-accurate simulation means the negligence of analog timing properties. The system is thereby only evaluated at discrete points in time (clock edges). Due to the isochronous nature of the transport network and the clock-cycle wise digitization/generation of events in the neural network chip, this strategy gains reliable results regarding the departure and arrival times of single events within the system. Registers are in this case simply modeled as variables that are updated at most once per simulation cycle. Delay stages or points where latency is introduced due to pipelining or data path delays are modeled with an appropriate number of these register pendants. A more detailed description of cycle-accurate simulation strategies can be found in [RD06].

Pseudo code of the developed behavioral simulation is shown in the following algorithm 1. The presence of a global clock signal is simulated by an outer loop triggering the execution of the single modules for a certain number of times until the desired simulation duration has been reached. To achieve results that are comparable to the HDL simulation of the final implementation, the cycle-accurate simulation mimics the behavior of an RTL-design for each cycle: the output of the previous cycle is first read at each level of the hierarchy before the respective data is being updated.

Algorithm 1 Pseudo code for the event routing simulation. The outer loop simulates the presence of a global clock signal by triggering all contained actions. All data is evaluated backwards with respect to the direction of data flow. This simulates the behavior of registers updating their outputs with the result of the previous clock cycle.

```
for i ← 0 to number of cycles − 1 do
    for n ← 0 to number of switches − 1 do
        mgtdelays[n] ← result of switch->switchOut()[n] from prev. cycle
        function NATHAN->EXECUTENATHAN()                                   ▷ execute one Nathan module
            SWITCH->SWITCHIN()          ▷ process EventOutQueue output from prev. cycle as input to the switch
            for s ← 0 to number of daisy chains − 1 do
                function CONTROLLER->EXECUTECONTROLLER()   ▷ executes EventRouter and neural network chips
                    popOutQueues()                                          ▷ process EventOutQueues
                    pushOutQueues()
                    for s ← 0 to number of chips in daisy chain − 1 do
                        function SPIKEY->EXECUTESPIKEY()                    ▷ execute neural network chip
                            NetworkBlock->fire()
                            genEvPacket()
                        end function
                    end for
                    genEvOutPacket()   ▷ evaluate EventInQueues and generate SpikeyPacket for the next cycle
                    pushInQueues()                                  ▷ read switch output into EventInQueues
                end function
            end for
            SWITCH->SWITCHOUT()              ▷ execute the crossbar switch and generate switch output data
            setTime()                                                       ▷ increment system time
        end function
    end for
end for
```

3.4.2 Neural Network Setup

Plenty of research is being done in the field of neuroscience using software simulators for the neural network like NEST[38] or NEURON[39]. One of the first goals for the experiments done

[38] NEST [The07a] is a software provided by the Neural Simulation Technology Initiative and is intended as a *'simulation system for large networks of biologically realistic (spiking) neurons'*. It allows the custom definition of point-neuron models and is used as a verification tool for the model implemented within the Spikey chip.

[39] NEURON is a simulation environment for modeling individual neurons and networks of neurons [HC97].

with the chip presented in this thesis would presumably be the comparison of the obtained results with the results of software simulations. First results for single-neuron experiments will be given in section 6.8. The comparison of simulations of complex neural networks would demonstrate the correct functionality of the implemented model. Even more important, it would demonstrate the speed advantage of the hardware model, which operates 10^5 times faster than biological real time, compared to software simulations. These barely simulate networks with a complexity in the order of 10^3 neurons in real time with a simple integrate-and-fire model and conductance based synapses [Mue03].

Commonly used neural network topologies for software simulations can be found for example in the work of Mueller et al. [MMS04] or Maass et al. [MNM04]. The neural networks modeled there feature approximately 1000 neurons and are only small portions of cortical areas. Neurons are arranged on a lattice and synaptic connectivity between these neurons is supposed to be random and uniformly distributed without any spatial preference. The number of output synapses per neuron is parameterized by the connection factor r_{con} with values between 5 % and 20 % of the neuron number. Figure 3.13 shows the connectivity of one example neuron within a $9 \times 9 \times 9$ lattice.

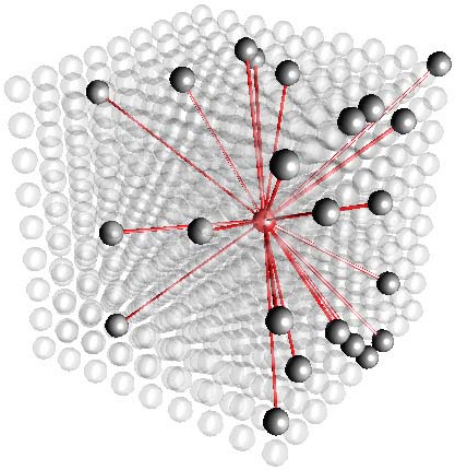


Figure 3.13: Schematic drawing of the synaptic connections made by a representative neuron (center, red) in the lattice. Figure taken out of [MMS04] with kind permission.

Mapping of the Neural Network to the Hardware

Prior to simulation, this network needs to be mapped to the resources available in hardware, e.g. the neural network chip and the accompanying event transport network. This mapping process is a major challenge due to the biological diversity of parameters and the comparably limited or quantized features of the hardware. The development of a mapping software is subject to the collaborative work within the FACETS project and the mapping process will shortly be outlined in the following. For more specific details, please refer to [EME+06].

Two input sets of data to the mapper can be identified. On the one hand, there is a netlist of neurons and synaptic connections. This netlist can be generated using a scripting language like Python that is commonly used for programming neural network simulators like NEST. On the other hand, there is the description of the hardware system including the topology and

the bandwidth information of the transport network and constraints regarding connectivity on the neural network chips. Furthermore, the hardware description covers the parameters for the neural circuits on the network chip with all constraints in terms of grouping of parameters and quantization (cf. section 4.2.1). Based on this input data the mapping process takes place in two steps:

1. Map the netlist of the neural network to a number of chips. To facilitate this process, the number of chips and the number of neurons to be used on each chip has to be defined before the mapping starts. The input netlist is then reordered as to maximize local connections on the neural network chips. As an outcome, the inter-chip connections and thus the required bandwidth on the transport network are minimized. At the end of this step all connections between two neural network chips that have an identical or almost identical axonal delay are combined to virtual connections. The number of axonal connections contained within one virtual connection then allows the estimation of the bandwidth required for this connection by summing up the estimated maximum firing rates of all involved neurons. Another outcome of this step is the content for the LUTs of each *EventOutQueue* and the configuration of the on-chip feedback connections of the neural network chips.

2. Map the virtual connections onto the resources provided by the transport network. To facilitate this process the user has to take two decisions before the mapping starts: first, the actual Nathan modules that will be part of the transport network have to be chosen. Second, the size of the switch has to be chosen based on the available logic resources in terms of the number of ports to the controller and to the MGT links. This configuration predefines the available bandwidth between the Nathan modules and the bandwidth to and from the controllers. The mapper then allocates one or more time slots of the switch period to each virtual connection and distributes the virtual connections over the available switch ports. Route-through connections over several hops will also be considered if necessary (cf. section 3.1.2). The result of this step is the configuration table for the scheduler with an input-output assignment for each time slot.

It turned out that a randomly connected network, even with a low connection density of $r_{\text{con}} = 5\,\%$, after the first mapping step still requires one axonal connection between a neuron on one network chip and at least one neuron on all other network chips in the system. Consequently, each virtual connection needs to transport all events generated by neurons on the attached source neural network chip and in this case, identical bandwidth is required for the communication between all neural network chips. Since the mapping tool is in a very preliminary stage of development and due to the fact that a detailed mapping result is not supposed to have any impact on the performance of the event routing algorithm, it has been decided to manually create a rather simple netlist for a neural network for the benchmark simulations.

Assuming the viability of an implementation using two virtual connections per Nathan module, the second step of the mapping process has been performed with this netlist consisting of 1000 neurons distributed over four neural network chips. An illustration of the (transport) network topology is given in figure 3.14. The Nathan modules are arranged in a square topology that matches one square out of the transport network on the backplane of the hardware platform. Each Nathan module has two active MGT connections that serve one virtual connection each, thus each switch has two ports to the MGT links. To have the full MGT bandwidth available at the controller, two user ports are allocated to this module on each switch.

The reduction of the communication between two network chips to exactly one virtual connection is due to the limited resources that are available on the currently available FPGA. It

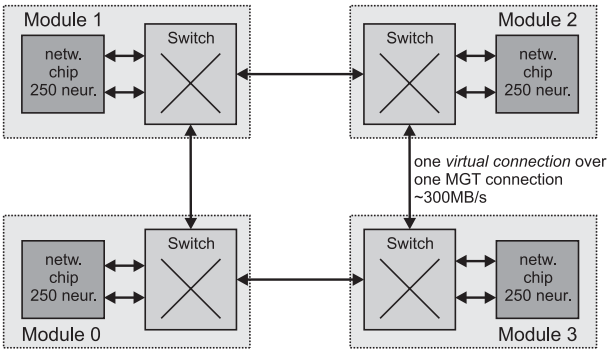


Figure 3.14: Topology of the neural network used for the benchmark tests of the event routing algorithm. In addition to the virtual connections shown, each network chip gets one *EventInQueue* connected to the local Playback Memory to inject background activity into the neural network.

has one severe implication for the realizable neural network models: all axonal delays between two chips are required to have the same delay. This limitation may only be overcome by the usage of an FPGA with large logic resources that might enable the implementation of more than one virtual connection per network chip connection. However, this number will still be small compared to the diversity of axonal delays within biological systems. To what extent this restricts the quality of the simulations of large scale neural networks cannot be quantified at the moment and will be subject to future research.

3.5 Simulation Results

Extensive simulations were performed in order to verify the functionality of the event routing algorithm with different configurations and under various load conditions. In the following, characteristic simulation results are shown that prove the functionality of the algorithm and also show some drawbacks resulting from bandwidth limitations within the system. First, the performance under static load conditions is shown. Second, simulations of networks with synchronized activity follow and finally the expected drop rates against the axonal delay of a virtual connection are shown.

It has to be noted that the results were obtained with the very simple neural network described in the previous section. As only the firing rates of the neurons are of interest, this should not restrict the validity of the results. Axonal delays for the virtual connections have been chosen randomly between 40 ms and 60 ms biological time for the following simulations.

3.5.1 Static Load

The goal of an analysis with a static load mainly is to verify that the event drop rates yield no unacceptable values for all activity that might occur within the neural network. Firing rates are kept constant for a biological time of about 6.5 seconds. At the chosen speed-up factor of $n_{speedup} = 10^5$ and an assumed clock frequency of $f = 156\,\text{MHz}$, this corresponds to $2 \cdot 10^4$ clock cycles for the simulation.

If all 250 neurons of one chip are firing at the same rate f_n, the overall firing rate f_t can be superimposed to $f_t = 250 \cdot f_n$. Using equation 3.2 the maximum achievable firing rate in terms of biology can be calculated as

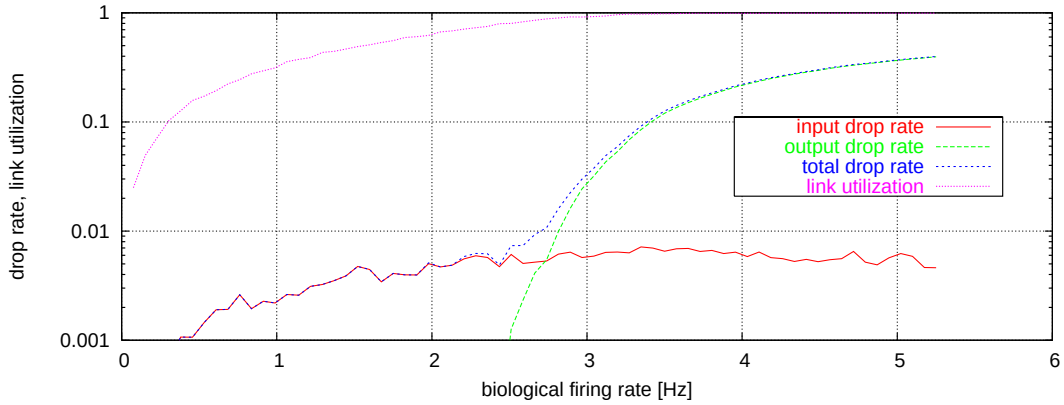


Figure 3.15: Drop rates at *EventInQueue* (input drop rate), *EventOutQueue* (output drop rate) and the resulting total drop rate (total drop rate) against the average firing rate of all neurons. Firing rates are kept constant for approx. 6.5 seconds of biological real time.

$$f_{\mathrm{n,bio}} = \frac{R_{\mathrm{e,net}}}{\mathrm{n}_{\mathrm{speedup}} \cdot 250} \simeq 3\,\mathrm{Hz}\ . \tag{3.11}$$

Figure 3.15 shows the response of the system for static firing rates from $f_{\mathrm{n,bio}} = 0.07\,\mathrm{Hz}$ to 5.2 Hz. The simulation is carried out with the setup described in the previous section. Four chips with 250 neurons each were simulated, where the average firing rate is the same for all neurons. Each *EventPctGen* module has one additional source for random events connected that is constantly sending events to 64 synapse rows at an average rate of 3 Hz per synapse row. The plotted results are the mean values of all modules in the system.

Within the *EventOutQueues* the first drops occur at 2.5 Hz and the drop rate then increases linearly with the firing rate. At the expected maximum of 3 Hz it has reached 2.5 %. This drop rate is due to the fill level of the queue memory operating near the maximum. Short peaks in the firing rate immediately lead to an overflow and thus to dropped events. Nevertheless, the implemented leaky bucket algorithm performs well and the link utilization almost reaches 100 % at the expected maximum firing rate. No decrease of the link utilization is observed even if the link is heavily overloaded.

At the destination, events yet need to be discarded by the *EventPctGen* module at comparably low firing rates. These drops occur due to events arriving with a time stamp smaller than the one of an event that has already been sent to the neural network chip. The algorithm that checks for collisions within the *event_buffer_ins* on the network chip does not correct the time stamp of these events and discards them. The fact that this drop rate does not exceed an acceptable value of 0.7 % shows two things: first, the strategy to start the transmission of events to the network chip within a certain time window (cf. section 3.3.3) leads to low drop rates as expected. Second, the leaky bucket algorithm at the source performs well because it does not send events that would arrive too late at the destination. These events would be dropped immediately by the *EventInQueues* and would show up as an increase of the input drop rates under heavy load conditions.

The above setup is simulated with an average firing rate for all neurons that is slowly increased over time in order to evaluate the performance under moderate load transients. Start and stop values for the firing rate are chosen equal to the static analysis. The result is shown in figure 3.16. In this case, only the drop rate averaged over the whole system and the link

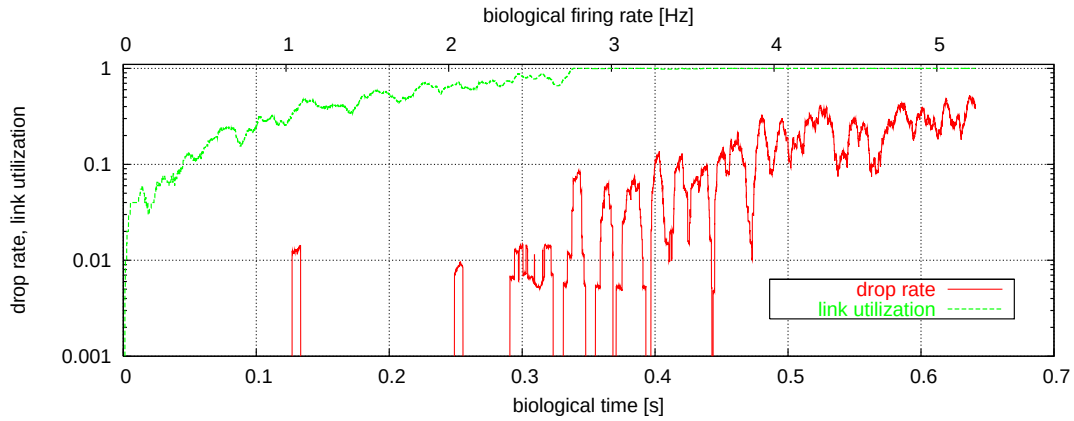


Figure 3.16: Drop rates described in figure 3.15 for a transient, constant increase of the firing rate of all neurons.

utilization are plotted. The overall behavior confirms the above results. For firing rates well below the expected maximum of 3 Hz, the overall drop rate does not exceed 2 %. Furthermore, it can be seen that the link utilization has reached 100 % at about 2.75 Hz while the drop rate has a peak value of about 8 %. This shows that the transport network may become overloaded for short times due to fluctuations in the event rate if it is operating close to the maximum rate. This fact recurs in the results presented in the following section and will have to be taken into account during the planning of neural network experiments using this system.

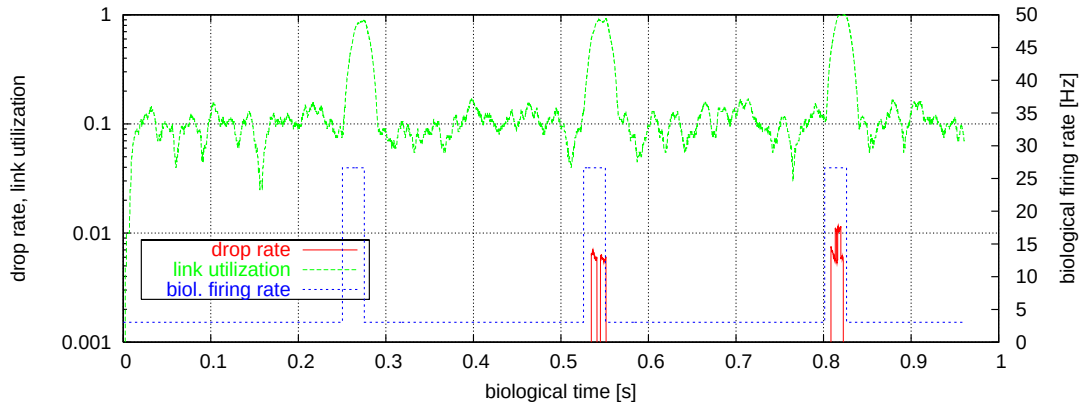


Figure 3.17: Simulation of a network with synchronized activity and peak rates that can be handled by the transport network.

3.5.2 Synchronized Activity

This section describes generic simulations of a neural network exhibiting synchronized activity aimed to evaluate the functionality of the event routing algorithm under heavy load transients as they are expected for this kind of neural network activity. The basic setup is identical to the setup used in the previous sections. Four network chips, all with 250 active neurons, are interconnected via next-neighbor connections and each chip has a local source of randomly generated events that serve as background activity and are injected into 64 dedicated synapse

rows on the network chip. The average biological firing rate for this background activity is 3 Hz.

As a basis for the network simulations, simulation results obtained with NEURON for a set of 729 neurons are taken [MMS04]. Based on this data, the neural network is expected to develop a synchronized behavior. These synchronized *bursts* are in the following modeled as an increase of the firing rate of all involved neurons, over periods of 25 to 50 ms on biological time scale, at a rate of 2 to 5 Hz. At a speed-up factor of 10^5 the system is intrinsically not able to handle event rates over 3 Hz. As a consequence of this, this factor is reduced to 10^4 for the following simulations which matches the minimum operating speed of the analog circuitry of the neural network chip. The duration of the synchronized burst was 25 ms at a rate of 3.5 Hz for all simulations.

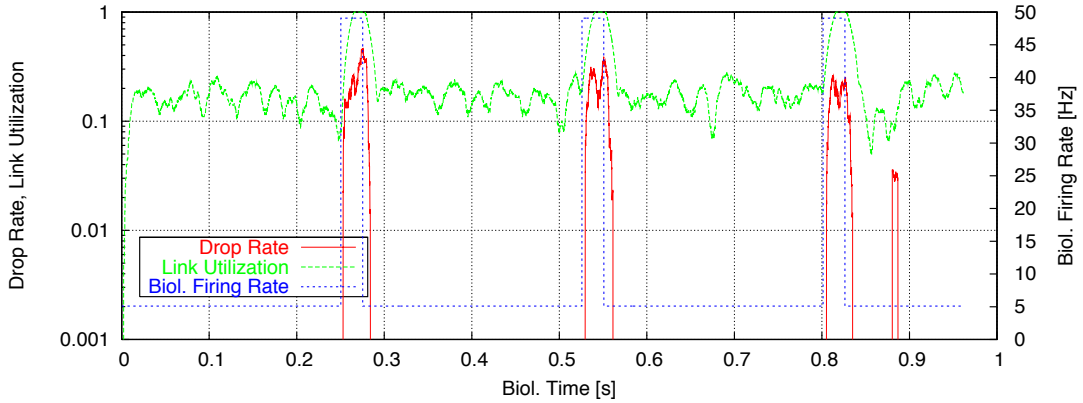


Figure 3.18: Simulation of a network with synchronized activity and peak rates that exceed the capacity of the transport network.

The result for an idle firing rate of 3 Hz and a burst firing rate of 28 Hz is shown in figure 3.17. At a speed-up factor of 10^4, the system is expected to handle peak rates close to 30 Hz on the basis of the previous simulation results which is approved by this simulation. The overall drop rate does not exceed approximately 1.2 % during a burst.

For an idle firing rate of 5 Hz, a burst firing rate of 48 Hz and a speed-up factor of 10^4, the simulation results are shown in figure 3.18. As expected in this case, up to 40 % of the events need to be dropped during a burst. The advantage here is, that the system immediately recovers from the overload and continues to reliably transport events. As a consequence of this, if the enabling of bursting activity at high rates is desired, either the number of axonal connections per virtual connection will have to be decreased or the physical bandwidth will have to be increased.

3.5.3 Drop Rates and Connection Delay

In the preceding subsections, the simulated axonal delay is randomly chosen between 40 ms and 60 ms biological time which is well above the minimum value of 24 ms for a layer 3 connection at a speed-up factor of 10^5 (cf. section 3.2.4). The dependency of the output drop rates against the axonal delay shall now be investigated. The simulated network is again the four-chip system with 250 active neurons per chip. Firing rates are kept at different constant values for each chip: 1.125 Hz for all neurons on chip 0, 1.5 Hz for chip 1, 2.25 Hz for chip 2

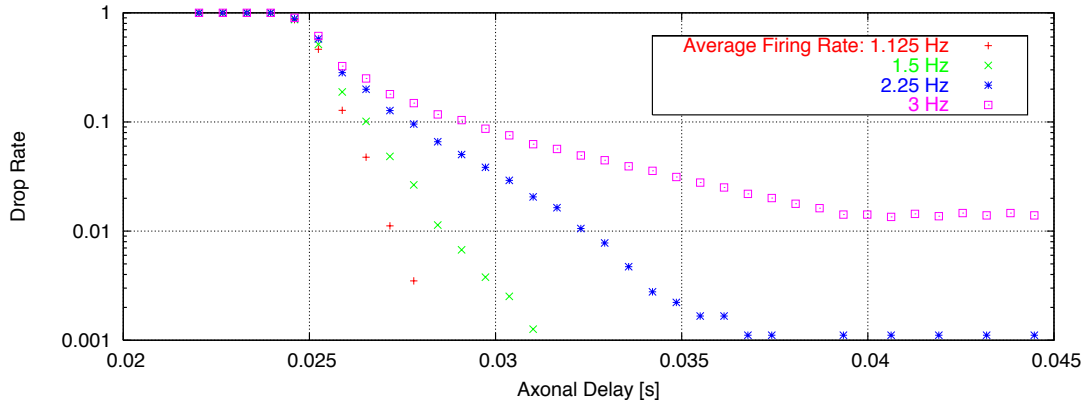


Figure 3.19: Dependency of drop rates at *EventOutQueue* from the axonal delay for different average firing rates. The speed-up factor is 10^5 compared to biology for this simulation.

and 3 Hz for chip 3. Figure 3.19 shows the result of a simulation where the axonal delay was globally swept from 22 ms to 45 ms for all virtual connections.

At delay values below the minimum of 24 ms the drop rate is 100 %, as expected. For values slightly above this minimum delay, the drop rate does not decrease immediately to a certain minimum but rather approaches this minimum with a slope proportional to the average event rate that is transmitted by the corresponding *EventOutQueue*. This minimum drop rate equals zero for firing rates below 2 Hz and otherwise approaches constant values of up to 1-2 %, as it is expected. As a consequence, if a virtual connection is supposed to operate at the maximum possible event rate, the minimum feasible axonal delay for this connection will sum up to about 40 ms for drop rates below 2 %.

3.6 Concluding Remarks

An algorithm suited for the transport and routing of neural event data between several neural network chips, based on the transport network of the FACETS[40] *Stage I* hardware system, is presented. To evaluate the performance of the proposed system, a C++ simulation environment has been developed and the algorithm has been tested under various conditions. The results show that the bandwidth resources of the transport network are optimally utilized under static and transient load conditions.

By means of the simulation of a network of 1000 neurons comprised of four network chips it was shown that the algorithm can handle the traffic of a neural network exhibiting synchronized behavior. This proves the presented concept for building large scale spiking neural networks based upon the hardware presented in this thesis. Besides the simulation results, an estimation on the resource consumption of the final implementation of the algorithm has been given. It turns out that with the current hardware, at least the implementation of the simulated four-chip network should be feasible. Maybe even three virtual connections per Nathan module are possible, which would allow the realization of a fully connected neural network.

Some drawbacks arise due to the constraints imposed by the presented hardware system. Due to the ever limited logic resources, only a comparably small number of virtual connec-

[40]FACETS: Fast Analog Computing with Emergent Transient States. An Integrated Project funded by the EU [Mea05].

tions per neural network chip is feasible. This basically restricts the number of inter-network chip connections and thus the overall number of neural network chips in the system. Additionally, the diversity of axonal delays is restricted to one discrete value per virtual connection. Moreover, within the current system, the size of this delay will presumably be greater or equal to 40 ms in biological terms at a speed-up factor of 10^5. The bandwidth requirements of a bursting network operating with this speed-up factor cannot be satisfied using only one virtual connection for the transmission of the whole event data produced by one chip.

There are two possible solutions to these problems: the first possibility is to build a new dedicated hardware substrate facilitating the operation of several neural network chips connected to one large FPGA. Depending on its logic resources, this would enable the implementation of layer 1 and layer 2 connections. Their availability would introduce the ability to implement shorter and more diverse axonal delays among the neurons distributed over the neural network chips. The second possibility to achieve a more biologically realistic behavior of the network is the reduction of the speed-up factor to 10^4. This would shrink the minimum axonal delay by a factor of 10 to about 4 ms on the currently available hardware system and would furthermore provide enough free bandwidth to possibly enable synchronized behavior with only few virtual connections.

Future work will integrate the developed simulation environment into a convenient framework for the system simulation and the development of the FACETS hardware. Mainly the mapping tool, which is currently being developed [EME+06], will provide the actual neural networks to be realized in hardware. Prior to the final implementation, the performance of the system may then be estimated using the presented simulation environment. A planned integration into a SystemC-based simulation environment will in the future also enable system simulations that integrate the actual hardware models.

Chapter 4

Implementation of the Chip

This chapter describes the implementation of the neural network ASIC by means of the design flow described in chapter 2 while incorporating the analog VLSI components described in chapter 1. An overview of the chip architecture is given and the analog and the digital part are separately discussed. The analog part implements the functionality of the neural models and is described on a functional level. The features of the digital part are introduced in the following sections with, on the one hand, a special emphasis on the interface of the chip. On the other hand, the implemented logic for the event transport is explained in detail. The physical implementation using Cadence First Encounter is described and the performance of the final chip is estimated based on results obtained from the timing closure using Synopsys PrimeTime. Two versions of the chip have been fabricated which will be referred to as Spikey 1 and Spikey 2. Unless otherwise noted, this chapter refers to Spikey 2 and it closes with an overview of improvements of the second version.

The presented chip has been fabricated in a standard UMC 180 nm CMOS process [Cor] with one layer of poly silicon and six metal layers. A complete set of standard cell and I/O pad libraries as well as generators for customized static memory arrays and a PLL are available for this process. Moreover, multi-project wafer (MPW) runs for the UMC 180 nm process are made available by the Dutch research institute IMEC which renders possible the low priced production of ASIC prototypes in quantities of tens to hundreds.

4.1 Chip Architecture

In figure 4.1 a micro photograph of the Spikey chip is shown and the essential functional blocks are marked with arrows. The chip comprises two identical network blocks, each consisting of the synapse array, 256 according synapse drivers and 192 leaky integrate-and-fire neurons, resulting in 49,152 synapses per network block, hence, 100,000 synapses per Spikey chip. The synapses have a size of about $10 \times 10\,\mu\text{m}^2$ which made it possible to integrate the circuits on a die size of $5 \times 5\,\text{mm}^2$. The models described in chapter 1 are implemented in the neuron

and synapse circuits and the intrinsic time constants result in a speed-up factor between 10^4 and 10^5 compared with biological real time. DTCs and TDCs generate and digitize events at a digital clock rate of nominal 400 MHz and a time bin resolution of 1/16 clock period (156 ps). About 3000 analog parameters can be set to allow for a rich model diversity as well as for calibrating the analog circuits.

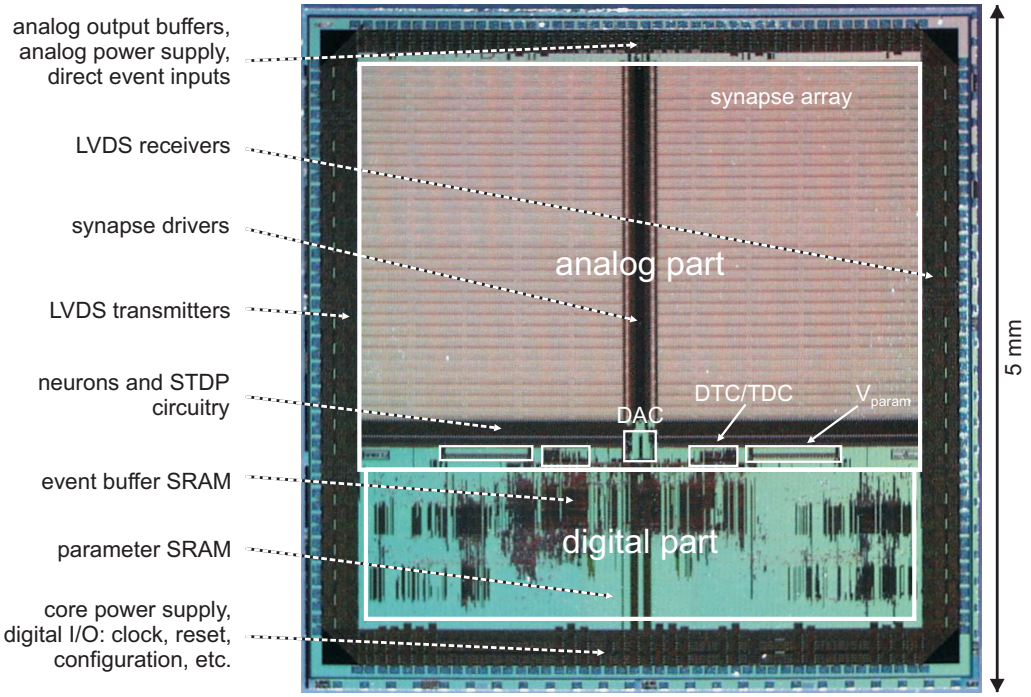


Figure 4.1: Micro photograph of the Spikey chip.

As already indicated, the chip is divided into an analog and a digital part and the analog part occupies the upper two thirds of the core area. The synapse array, the synapse drivers and the neuron circuits are combined in one block which will be referred to as *network_block*. Two *network_blocks* are present on the chip and thus all blocks that are directly related to their functionality are implemented twice. Both blocks that are placed in the center below each *network_block* provide 20 parameter voltages V_{param} and the TDC and DTC circuits are integrated into one block at a time. The DAC that is globally used for the model parameter generation is placed between the *network_blocks*. Analog supply voltages, external biasing, the membrane voltage outputs and other auxiliary signals for the analog part are confined to the top edge of the chip.

The digital part occupies the lower third of the core area. Requirements for its functionality are derived in the following sections. In the micro photograph, the memory arrays for the event processing logic and the model parameter storage are marked. The I/O pad cells of the digital interface are placed along the right and the left edges of the chip in a way to facilitate the physical interconnections of the daisy chain topology with the data flowing through the chip from right to left (cf. section 3.2.1). Auxiliary pads that are required for the digital part

are confined to the bottom edge and include power pads, daisy chain address and reset pins, the external core clock input and pads of the JTAG[41] interface.

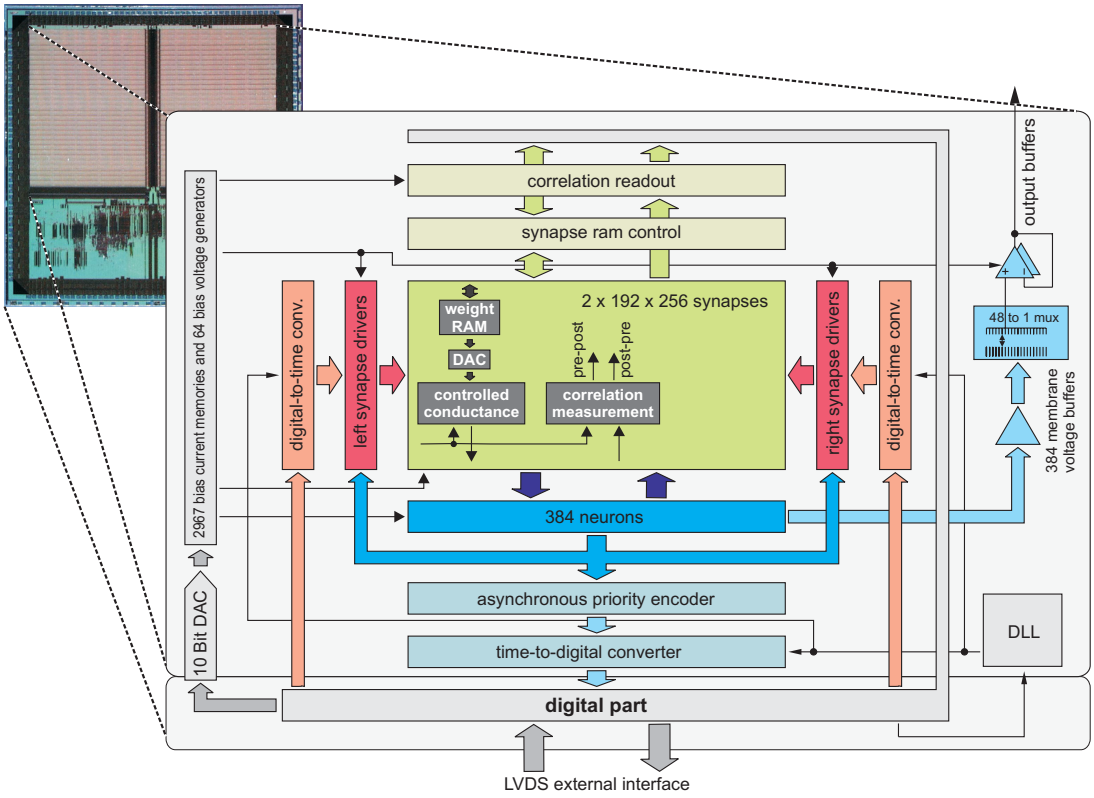


Figure 4.2: Illustration of the analog part of the Spikey chip that contains the neuromorphic circuits. The realization of the digital part, which occupies the lower third of the core area, together with the integration of the analog blocks into the final chip, is a main aspect of this thesis work.

4.2 Analog Part

The functionality of the neuromorphic circuits is outlined in chapter 1. Within this section, the model parameter generation is briefly described and the relevant data for the automatic integration of the different blocks is collected.

4.2.1 Model Parameter Generation

The ability to control the diversity of the neuromorphic circuits is an important aspect of the presented chip, as it is desired to model the variability observed in nature [AGM00]. Control of diversity is implemented by means of manipulating certain parameters of the analog circuits either group wise or for single circuits. As it is not possible to manipulate geometrical properties of silicon devices, the parameters have to be adjusted in the form of currents or voltages. One example is the leakage current I_{leak}, which can be set for each neuron individually. A complete list of available analog parameters can be found in appendix A.

[41] The joint action test group (JTAG) interface is a four-pin serial interface. It is implemented for testing purposes and provides backdoor access to the I/O pads of the chip.

Current Memory Cell The total amount of parameters sums up to 2967 and therefore a custom current memory cell has been implemented [SMM04]. Two types of this cell are available: a current source and a current sink. It is used within the circuits and replaces bias current sources or sinks where they are relevant for the model parameter generation. The current memory is implemented as a voltage controlled current source/sink, and the control voltage is stored on a capacitance within the cell. Due to leakage currents, this capacitance requires periodic refresh. For this purpose, all current memories are connected to the output of the 10 bit DAC and a common address bus along with the digital control logic constantly loops over the current memories refreshing their values. The cell is designed such that it keeps its old output current while it is being refreshed, thereby avoiding output glitches during programming.

10 Bit DAC The 10 bit DAC has a current steering output and requires the supply of an external reference current I_{refdac} and an external bias voltage V_{casdac}. The output current can be set from 0 to $I_{\text{refdac}}/10$ with a nominal resolution of 10 bits. The interface to the DAC is fully asynchronous: the five least significant bits of the value have to be supplied as binary value whereas the five upper bits are thermometer coded. In contrast to the remainder analog and digital circuits, which operate from 1.8 V supply voltages, the DAC operates from a 3.3 V supply to facilitate the realization of a rail-to-rail output within the 1.8 V domain [Scha].

The output of the DAC drives a large capacitive load as it is connected to all current memory cells in parallel. Therefore, the *RC* time constant at the output and thus the output settling time becomes relatively large, especially for small output currents. To overcome this issue, the DAC features a *boost* input which in parallel connects the output to 9 dummy current memory cells located within the DAC. To achieve the desired current at the output of the DAC, the input value has to be multiplied by 10 when using this pin. This increased output current shortens the settling time. Therefore, the boost pin can be used to shorten the refresh time for small current values.

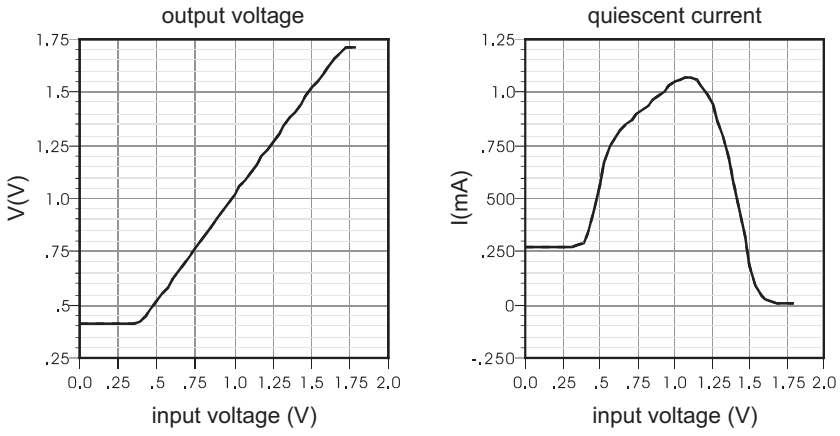


Figure 4.3: DC response of the buffer used for the parameter voltage generation to an input voltage sweep from 0 V to 1.8 V at a bias current of 0.2 μA. Left: output voltage. The rails are not reached on purpose. Right: quiescent current. The large quiescent current of up to 1.1 mA has to be taken into account during power planning.

Current to Voltage Converter Several model parameters, e.g. the reversal potentials E_l, E_x and E_i, require the supply of voltages rather than currents. These voltages are generated by cells called *vout_block* located below the *network_blocks*. These cells are addressed by the same bus as the current memories but they convert the output current of the DAC to a voltage by means of a poly silicon resistor of $R_{cv} = 10\,\mathrm{k}\Omega$. The voltage developed at the resistor is first stored as a charge on a capacitance C_1. After programming, the DAC is disconnected and C_1 is connected to a second capacitance C_2. C_1 and C_2 share their charge and after a sufficient number of programming cycles the desired output voltage is developed over C_2. As a consequence, the output of the cell operates glitch free. C_2 is connected to a low-impedance output buffer that is capable of driving up to 5 mA by means of a push-pull output stage with equally sized transistors ($L = 240\,\mathrm{nm}$ and $W = 250\,\mu\mathrm{m}$). The output impedance and the quiescent current of this buffer are also controlled by a current memory cell. Figure 4.3 shows the DC response of the output buffer to its input swept from 0 to 1.8 V with a bias current of $0.2\,\mu$A (the regime in which the buffer is supposed to operate). The rails are not reached on purpose which limits the voltage parameter range to

$$0.45\,\mathrm{V} \leq V_{\mathrm{param}} \leq 1.7\,\mathrm{V}\ . \tag{4.1}$$

As a consequence, parameter voltages requiring lower voltages are supplied externally. These voltages are listed in appendix A together with an explanation of their purpose.

The quiescent current reaches up to I_{qv} =1.1 mA. This value may be reduced by a lower bias current. However, this is not desirable due to the increased output impedance. For this reason, the power supply for *vout_block* needs to be designed appropriately (cf. section 4.4.3). The large quiescent power dissipation increases the self-heating of the chip near the *network_block*. It was chosen to still use this buffer and to tune the bias currents in order to reduce power consumption.

The space requirement of the poly silicon resistor and the output buffer prohibits the integration of the current to voltage converters into every neuron or synapse circuit. For this reason, groups of neurons and synapses share parameter voltages. All available parameters as well as a list of grouped parameters is given in appendix A. A more concise description and the characterization of the current to voltage converters can be found in the diploma thesis of B. Ostendorf [Ost07].

4.2.2 The Network Block

Figure 4.4 shows a block drawing of the *network_block*. Its core is formed by the synapse array of 256×192 synapses. This size has been iteratively determined during the layout phase of the synapse array. A maximum of 384 columns fit into the available core area of the chip after placement of the I/O cells and bond pads. To reduce the wire capacitance, thus, the signal propagation delay and the required drive strength for the row drivers, the array of 384 columns is split into two *network_blocks*. Another advantage of the splitting is the availability of twice as much synapse drivers per chip (only half the synapses share one common driver) resulting in an increased flexibility for the mapping of neural networks to the hardware.

Row Circuitry The row circuits beneath the synapse array contain the spike reconstruction circuitry to drive an input spike into the associated synapse row. Moreover, circuits implementing short term synaptic plasticity are integrated common for one row. Available analog parameters for one row are listed in table 4.1. The input of the synapse driver can be selected

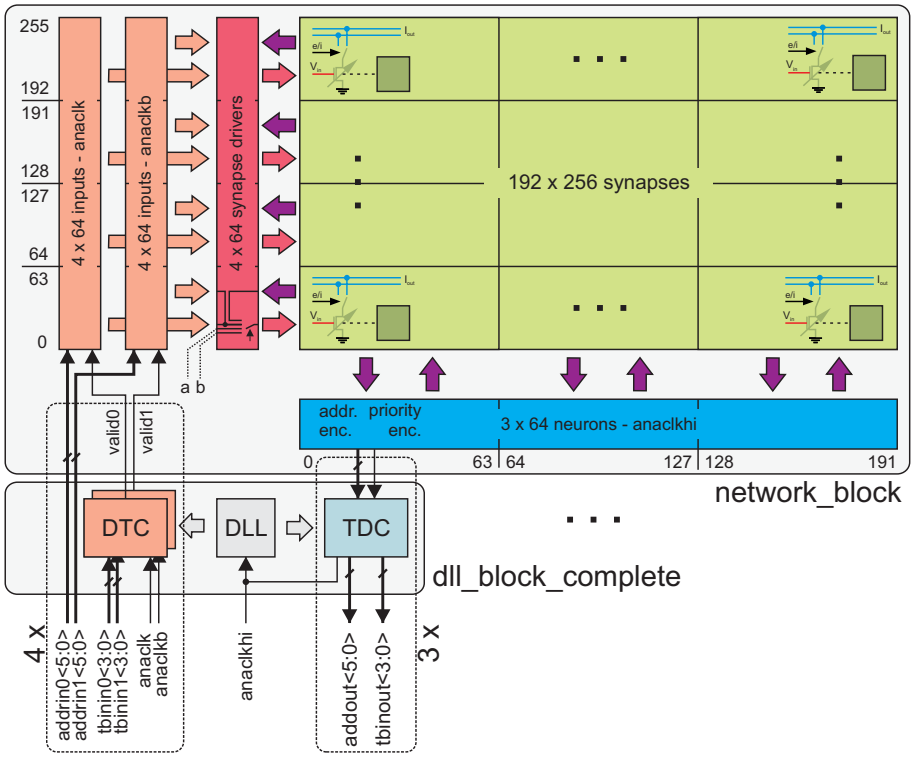


Figure 4.4: Block diagram of the analog blocks *network_block* and *dll_block_complete*. The illustration is reduced to the event generation and digitization functionality. The DTCs are implemented fourfold for each of the two clock domains `chip_clk` and `chip_clkb`, resulting in eight address interfaces. The TDCs are implemented threefold for the clock domain `clkhi`, resulting in three address interfaces. The according valid signals are not shown for reasons of simplicity.

either from an external source, local feedback on the *network_block*, feedback from the adjacent *network_block*, or from an adjacent row to group two synapses into one with double precision [SGMM06]. Feedback from an adjacent *network_block* and grouped synapses are symbolized with *a* and *b* in figure 4.4. The selected input is stored by means of two configuration bits for each row. Six necessary row configuration bits and the column configuration bits are summarized in section 4.3.6. The row configuration bits as well as the associated row within the synapse array are accessed by a common address bus. The data bus carrying the row configuration bits is implemented separately from the column data bus and is bidirectional [Scha] which requires the implementation of tri-state signals within the digital part.

Regarding event generation, the rows are divided into four blocks of 64 rows with the following consequences:

- Each block has two 6 bit address busses `addrin*<5:0>`[42] plus enable bits `valid*` that are used for external event input. The outputs of the two address decoders and the enable bits are connected such that they may produce events simultaneously. Input events are generated from within two clock domains `chip_clk` and `chip_clkb`. Event generation from within the digital part will be described in section 4.3.5. It is the parasitic capacitance of the address lines, which extend over the whole height of

[42] The asterisk $\star$ replaces possible numbers, in this case `0` and `1`.

Address	Parameter
0x0	synapse output driver
0x1	delay element before synapse driver
0x2	pulse shape rise time
0x3	pulse shape fall time
0x0	neuron membrane leakage (g_{leak})
0x1	neuron V_{th} comparator speed

Table 4.1: List of analog parameters available in each column and row circuit. All parameters are bias currents supplied by a current memory. The actual physical addressing is given in appendix A.

the *network_block*, that limits the operating speed of the event input and made this separation necessary.

- Externally supplied model parameters relevant for short term synaptic plasticity are shared among alternating blocks of 64 rows (cf. appendix A).

Column Circuitry Besides the neuron circuit containing the membrane capacitance C_{m} and the conductances modeling the ion channel currents, the columns below the array also contain the following functionality (not shown in figure 4.4):

- Two column configuration bits which enable or disable the analog output of the membrane voltage V_{mem} and the spike digitization. Spikes of disabled neurons are neither digitized nor are they sent to the local feedback connections. Four $50\,\Omega$ output buffers per *network_block* are available at the top edge of the chip. Every fourth V_{mem} is connected to one of them, thereby enabling the monitoring of a single neuron's membrane potential, or an average of a set of membrane potentials.

- Column address decoders that either decode the address for the configuration bits within the column circuits or the column address within the synapse array.

- Associated with the column address decoders is the bit line control of the according synapse memory column including the readout sense amplifiers.

- The STDP readout circuitry. The results of ongoing correlation measurements are read out for one row at once and are processed in each column.

- The neuron parameter storage (see table 4.1 for a list), the neuron comparator and event readout circuits (cf. figure 1.3) and the asynchronous priority encoders together with the neuron address encoders (cf. figure 4.4).

The *network_block* is organized in three blocks of 64 columns with the following consequences:

- A memory access to the configuration memory or a column within the synapse array affects the three blocks simultaneously. Each block has a data bus with a width of 4 bit[43] and consequently a column access has an address space of 64 entries (6 bit) and an overall data width of 24 bit for two *network_blocks*.

[43] Only two bits are actually used within the column configuration memory.

- Every 64 neurons are associated to one priority encoder, which is implemented in several stages[44]. The number of a spiking neuron is determined and its address is sent to the TDC module to code the spike into an event. In the case where more than one neuron fires at the same time (within one clock cycle), the one with the highest priority (lowest address) is transmitted first and the remaining spikes are delayed and then transmitted in subsequent clock cycles. This provides a total of six event digitization channels per chip reducing the probability of a quantization error due to collisions by a factor of six.

Synapse Array Together with the linear voltage ramp generated by the row driver each synapse produces an exponentially increasing and decreasing input current to the connected neuron (cf. section 1.3). As described in section 1.3, each synapse connects the presynaptic row signal to its associated postsynaptic column. Action potentials that are generated by the neuron are routed back into the column and are used within each synapse for correlation measurement. Furthermore, each synapse having the same physical address as its associated neuron connects to a horizontal line available at the row driver (see the afore described row circuitry), which is indicated by the dark blue arrows in figure 4.4. The synaptic weight is stored in a 4 bit static memory located within the synapse. It is accessed by the row and column address lines, which are also used to address the respective configuration bits. The same data bus as for the column configuration bits is used resulting in an address space of 64×256 entries and a data width of 24 bit for two *network_blocks*.

Power Connections and Distribution The synapse array as well as the directly connected neuron circuits and the synapse drivers are connected to a power grid inside the synapse array, which is to be connected to the external power supply at the top edge of the chip. For this purpose $20\,\mu$m wide power and ground pins with a pitch of $70\,\mu$m have been defined on the top edge to ensure equally distributed power grid connections. This analog power is also available at power outlets at the bottom edge of the *network_block*. Power structures supplying the remaining analog circuitry will be connected there. A separate power supply is required for the digital event input and output to and from the *network_block*.

Technology Library Information

The TDCs and DTCs, which are described in the following section, solely account for the synchronization of the event generation and digitization process to the digital clock signal. Furthermore, all data, address and control signals to the internal memory resources are operated directly by a synchronous controller, which is located within the digital part of the chip. Therefore, no clock-synchronous interface timing is to be defined for the technology library of the *network_block*. Nevertheless, the grouping of signals to busses is defined to obtain equal automated routing lengths. Moreover, constraints like maximum capacitive load and minimum input transition times are obtained as described in section 2.1.2 and are defined for input and output pins to ensure an adequate signal quality.

4.2.3 Event Generation and Digitization

The analog block *dll_block_complete*, which is located below the *network_block*, performs the neural event generation and digitization. The central component is a DLL comprising a

[44] Parts of the priority encoder logic are located outside the *network_block* within the *neuronout_pe_top* module. As this module solely contributes to the priority encoder functionality, it will not be described separately.

delay line of 16 delay elements that are realized with current starved inverters. It is designed to operate at a frequency of up to 400 MHz and the delay of the single elements is controlled by a voltage V_{ctrl}, which can be initially set to a fixed value by means of a reset pin[45], and an according voltage parameter V_{resetdll} ($V_{\text{resetdll}} = V_{\text{ctrl}}$, if reset = 1). If the DLL is locked to the applied clock period, the outputs of the 16 delay elements will serve as a time base for event generation and digitization within *dll_block_complete*. To achieve the locked state, V_{ctrl} is automatically fine tuned. For monitoring purposes, V_{ctrl} is connected to a buffer whose output can be driven off chip by the circuits described in the following section.

Time-To-Digital Converter The TDCs are implemented threefold, reflecting the group structure of the *network_block*. Time to digital conversion takes place within the 400 MHz domain. Digital spikes that arrive from the last priority encoder stage are represented by a 6 bit neuron address and a valid signal. The address, together with the number of the active delay element upon the arrival of the spike, are synchronized to the 400 MHz clock. In the subsequent clock cycle they are provided to the digital part at registered outputs and represent the raw event data. The specification of the interface timing will be given in section 4.3.5.

Digital-To-Time Converter The DTCs are implemented twofold for each group of 64 synapse drivers which results in 8 DTC units per *dll_block_complete*. The twofold implementation is reflected by two independent clock interfaces that are to be operated with two clocks running at half event clock speed (200 MHz) and with a phase shift of 180°. As event data input, the DTCs require a valid bit together with the 4 bit time bin information. Within the 400 MHz clock cycle following the cycle of an event receipt, the spike is generated within the associated synapse block by triggering the enable signal within the desired time bin. The actual synapse address has to be supplied directly to the according address lines within this clock cycle by the digital control logic. It has to be noted that no address processing is performed by the DTCs.

Technology Library Information

To ensure correct implementation of the synchronous interfaces of *dll_block_complete*, its interface timing has been determined by means of analog simulations [Scha] and the timing on input pins and output pins has been constrained accordingly. The clock pins related to the input signals are the pins `clk200` and `clk200b` that are internally connected to the DTCs and externally to the signals `anaclk` and `anaclkb` (cf. figure 4.4). The internal DLL as well as the output signals operate synchronous to the 400 MHz clock `anaclkhi`, which is connected to the pin `clk400`.

4.2.4 Monitoring Features

Monitoring of Membrane Voltages Four membrane voltage outputs are present in each *network_block* and are driven off chip for monitoring purposes. The fastest transient expected on these signals is the membrane voltage being pulled to V_{reset} after a spike has been generated. The fall time of this signal is approximately $t_{\text{f}} = 1$ ns [Scha] which results in a bandwidth requirement of $f_{\text{cutoff}} \simeq 300$ MHz for the output driver. To accommodate these requirements, a 50 Ω output buffer has been designed which fits into the I/O pad ring at the top edge of the

[45] The reset pin of the DLL is connected to a control register output within the digital part and can be activated by setting the control register appropriately. See section 4.3.6.

chip. It is capable of driving a 50 Ω transmission line due to its matched output impedance, which is realized by means of a 50 Ω series poly silicon resistor at the output.

The bias currents for the off-chip drivers is stored within eight current memory cells that are located above the *network_blocks*. One additional current memory cell is present there whose output is connected to the pin `IBTEST` of the chip for characterization purposes.

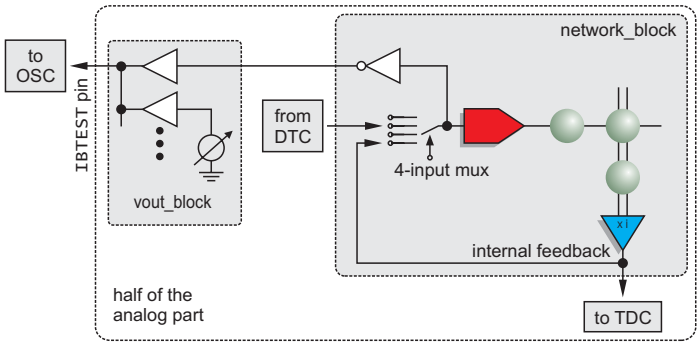


Figure 4.5: Illustration of the `IBTEST` pin connectivity for one half of the analog part. On out of all buffers within *vout_block* can be selected to drive the `IBTEST` pin. Besides the parameter voltages, also the digital output of one synapse driver's input multiplexer is available. On this account, parameter voltages as well as the input signal to the synapse driver can be recorded by an oscilloscope.

Monitoring of Parameter Voltages Each output buffer within *vout_block* has an attached transmission gate, which is controlled by a local flip-flop's Q and $\overline{\text{Q}}$ outputs. The outputs of all transmission gates are shorted to one wire, common for all outputs within both *vout_blocks*. The flip-flops are connected in series as a 1 bit shift-register and by shifting a bit string containing exactly one active bit into the chain, the output of a specific buffer is multiplexed to the common output wire. This wire is connected to the `IBTEST` pin parallel to the current memory cell, thus enabling the monitoring of either a parameter voltage or the output of the current memory. Three buffers are available in *vout_block* in addition to the ones for the voltage parameter generation:

- Digital monitor outputs, available at the *network_block*: the output of the multiplexer that selects one out of the four possible input signals to the synapse driver with physical address `0x1` (the second synapse row) and the output of the adjustable delay element within this synapse driver that is connected to the output of the multiplexer. Both signals are connected to additional output buffers within *vout_block* and may be multiplexed to `IBTEST` in the same way as the parameter voltages.

- The third output buffer is connected to the control voltage node of the DLL within *dll_block_complete*. Hereby, it is possible to verify if the DLL has achieved its locked state (the voltage is stable, then). This is especially useful as there is no other way provided to verify this state.

The connectivity is illustrated in figure 4.5 neglecting the transmission gate circuitry. For reasons of readability, only one parameter voltage buffer and the output of the synapse driver's multiplexer is shown.

4.2.5 Specifications for the Digital Part

Based on the functionality of the analog blocks, specifications for the digital control logic are derived.

- Clock Frequency: Two 200 MHz clocks (`anaclk` and `anaclkb`) are required for the DTC operation. Therefore, the clock frequency for the core logic is set to $f_c = 200\,\text{MHz}$ which allows for synchronous operation with the nominal operating frequency of the DTCs. To provide synchronous interfaces to the TDCs as well, an additional 400 MHz clock (`anaclkhi`) is required for the synchronous operation of these interfaces.
- Event Delivery and Capture: An overall of eight DTCs is available per *dll_block_complete* with four of them assigned to one of the 200 MHz clocks at a time. On the output side of *dll_block_complete* digitized events supplied by three TDCs need to be stored and processed. Having two *dll_block_complete* blocks on the chip, 16 *event_buffer_in* modules for the event delivery to the DTCs and 6 *event_buffer_out* modules for the event processing after the DTCs are needed (cf. section 3.2.1).
- System Time Counter: The time base for the events' time stamps in the sense of 400 MHz clock cycles needs to be provided by a system time counter. The previously addressed modules for event delivery and processing process events based on this counter. It has to be loadable to synchronize the chip with its environment.
- Parameter Storage and Refresh: Local memory resources are required that hold the digital value for all of the current memories. Additional logic has to periodically apply the stored values to the DAC and address the according current memory.
- Synapse Ram Controller: The static memory distributed over the synapse array and holding the synapse weights has an asynchronous interface which requires the control signals, address and data lines to be operated in the correct order. Moreover, the configuration memory located within the synapse drivers and the neuron circuitry needs to be accessed over shared data and address lines.
- Communication: Apart from the physical interface, a high level communication protocol is required that allows for packet-based communication with the packet content being decoded by the digital part. Furthermore, the proposed daisy chain topology has to be supported by the protocol (cf. section 3.2.1).

4.3 Digital Part

According to the initial specifications that have been collected in the previous section, a short overview of the single modules in the digital part and the data flow through the chip will now be given. The following subsections explain these modules in more detail.

Figure 4.6 shows an overview of the presented chip, which is focused on the digital part. Data flows through the chip from bottom to top and the clock domains operating at 200 MHz and 400 MHz frequency are separated by light grey boxes and marked with the respective frequency value.

A communication protocol has been developed that specifies the communication with the chip. The communication is based on data packets with a size of 64 bit and the protocol is classified into three layers:

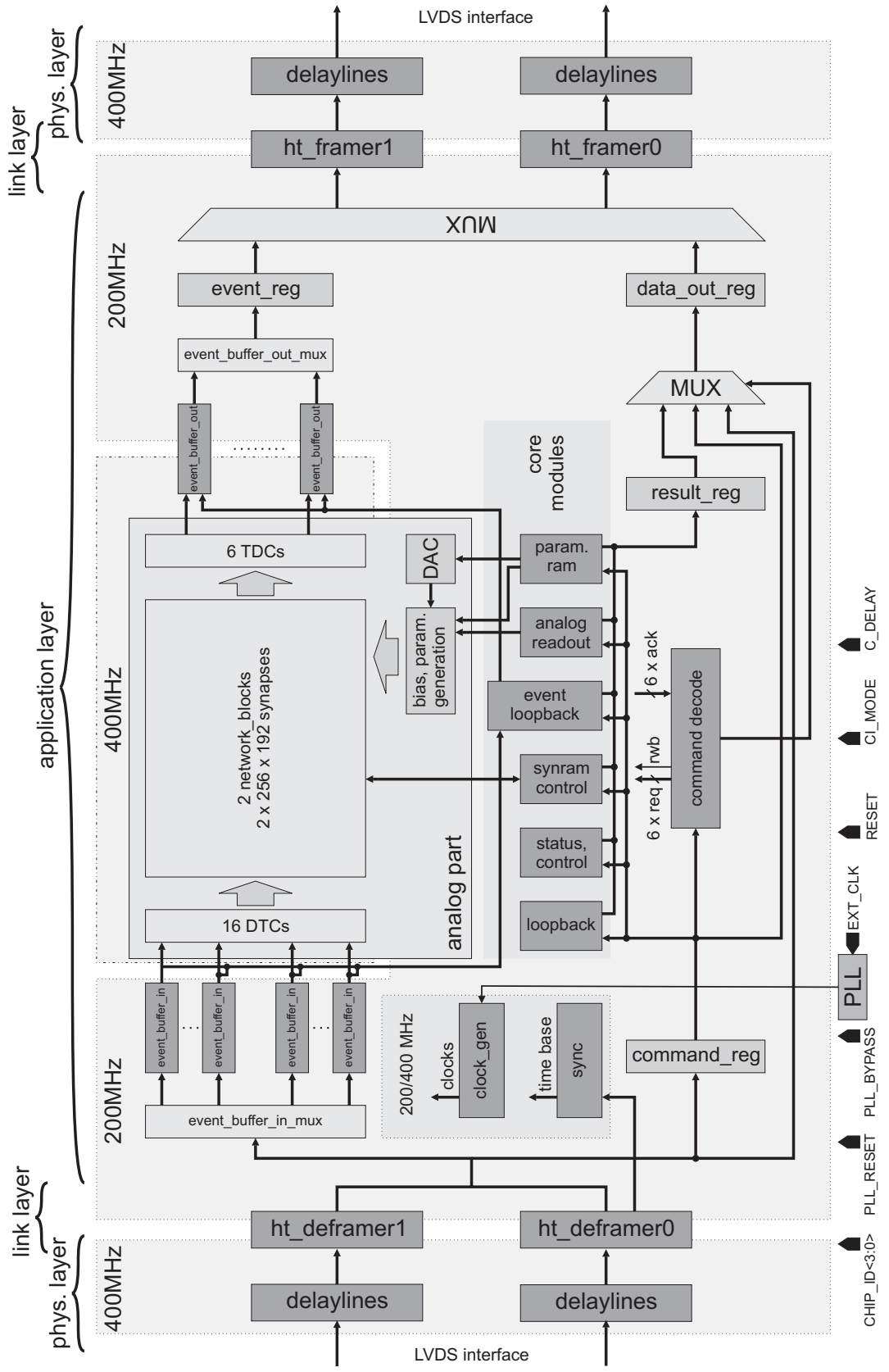


Figure 4.6: Simplified architecture of the presented chip focused on the digital part. The illustration of the clock signals and auxiliary signals is omitted for clarity. A detailed illustration of the analog components is given in figure 4.2.

- The *physical layer*, which is solely used to transport data to and off the chip through low voltage differential signaling (LVDS)[46] I/O pads. It comprises the modules *delaylines* and parts of the modules *ht_deframer** and *ht_framer**. The physical layer will be described in section 4.3.1.
- The *link layer*. Assembly and disassembly of data packets is performed by this layer. The chip also features a bypass mode (not displayed in figure 4.6) that is controlled by the pin `CI_MODE` and will be explained together with the link layer in section 4.3.1.
- The *application layer*, which resembles the digital core logic and the analog part of the chip. It accounts for data packet decoding and encoding and the digital control functionality. The different types of packets will be described in section 4.3.2 and the realization of the core modules connected to the command decoder is described in section 4.3.6.

The way data is processed by the chip results in 64 bit packets continuously being clocked through the chip, hence, through a daisy chain of chips. Data packets are addressed to a specific chip and each chip in the daisy chain is only allowed to transmit its own data (be it events or data packets) within a packet slot that contains a packet addressed for this chip.

External inputs to the chip that are relevant for the digital part are shown in figure 4.6 and their functionality is described in table C.1. Auxiliary modules depicted in figure 4.6 include a PLL which is used for the generation of internal clocks and the *clock_gen* module where derived clocks are generated (see section 4.3.3). The *sync* module includes the system time counter with synchronization functionality which is described in section 4.3.4.

4.3.1 Interface: Physical and Link Layer

Physical Layer

The physical layer of the interface consists of two 8 bit, source synchronous and unidirectional links that physically comply to the HyperTransport I/O Link Specification defined in [Hyp06]. Each of the links transports 8 bit of data and one frame bit; the complete pinout of the chip can be found in appendix C.1. To transport the 64 bit content of one data packet within one 200 MHz clock cycle, the link clock frequency is set to 400 MHz and data is transmitted with double data rate (DDR) at both clock edges resulting in a data rate of 800 Mbit/s on each data signal and overall 1.6 Gbyte/s for both the input and the output of the chip. Signals are transmitted differentially using the LVDS signaling standard [lvd00] to account for the high-speed requirements arising from this high data rate. The used LVDS pads have been designed by Wegener et al. [sup03] and convert the differential signal levels to the CMOS signal levels used inside the chip and vice versa.

As described in section 2.4.4, custom strategies have been developed to realize the interface timing required by the HyperTransport specification. For clarity, figure 2.9 is redisplayed here illustrating the required components (cf. figure 4.7). So far, a generic delay element has been assumed. In the following, the delay element that has been realized to deskew the signals at the input and the output of the chip will be described.

The Delay Line Figure 4.8 illustrates the design of one delayline which consists of specific standard cells. Seven delay stages are realized resulting in eight possible signal delay values.

[46] LVDS is a signal standard that exploits coupling and cancelation of electromagnetical fields by transmitting data over two inversely driven signal lines for one bit. A concise description of the application of LVDS on the PCB level has been given in [Grü03] and can also be found in [ANS96].

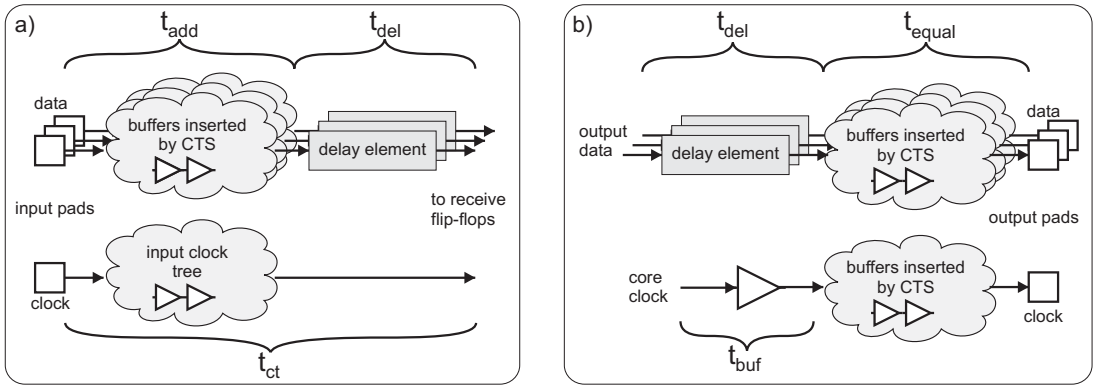


Figure 4.7: Concept for the implementation of source synchronous interfaces. a) receive side b) transmit side. The output data is meant to originate at flip-flops clocked with the core clock.

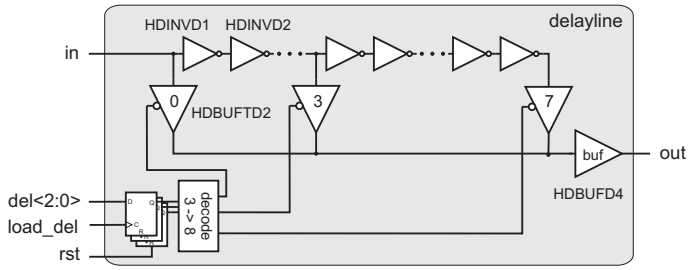


Figure 4.8: Functionality of the delayline. The instantiated cells are named as in the library. Upon reset the tri-state buffer with address 3 is activated.

Two inverters HDINVBD1 and HDINVBD2[47] have been used per stage to compensate for remaining differences in the signal propagation delay for rising and falling signal edges. In between each of these delay stages the signal is tabbed out by a tri-state buffer of type HDBUFTD2 and the outputs of these tri-state buffers are shorted to one single output signal. To ensure equal output load for all tri-state buffers (and thus an equal output *RC*-delay), this signal is buffered by a buffer of type HDBUFD4. The selection of a certain delay value is done by enabling the appropriate tri-state buffer. Three flip-flops store the binary value for the stage to be enabled and an address decoder constantly applies the appropriate enable signals to the tri-state buffers. The procedure to store the values within the flip-flops is explained in section 4.3.1.

The reset signal `rst` is connected to the external pin `PLL_RESET` in a way that optimized delay values are not reset by the global reset pin (`RESET`). Upon reset, delay stage 3 (counted from zero) is selected to have maximum headroom for tuning to both directions. In this setup, the delay sums up to about $t_{\mathrm{dl},3} = 550\,\mathrm{ps}$ for the typical process corner. The two inverters of each stage are selected as to have a total propagation delay of approximately 80 ps in the typical process corner which results in possible delay values theoretically ranging from 0 ps to 640 ps. At the desired link clock rate of 400 MHz, this is enough to deskew signals with a maximum skew of half a bit time. A more exact value for the delay is not given here as the actual delay

[47] The names correspond to the standard cell terminology [Vir04d]. These cells are **B**alanced **INV**erters with a **D**rive strength of 1 and 2 respectively. Drive strength 1 corresponds to an inverter with $W/L_{\mathrm{NMOS}} = 500\,\mathrm{nm}/180\,\mathrm{nm}$ and $W/L_{\mathrm{PMOS}} = 1200\,\mathrm{nm}/180\,\mathrm{nm}$.

varies due to slight differences in the placement of the cells and the routing, and also varies massively with the process corner. The implementations of the delay lines slightly differ at the receive and transmit size, particularly regarding the applied timing constraints. Therefore, the implementation will be described together with the implementation of the physical layer of each side in the following sections.

Receive Side Figure 4.9 illustrates the circuitry implemented for the data capture at the receive side. For the physical layer only the first register stage is relevant which captures the link data. Data signals and the frame bit are treated the same by the physical layer and are registered at both edges of the link clock. The resulting output data of the physical layer is the 18 bit bus `rx_word*<17:0>`, which is forwarded to the link layer at 400 MHz, synchronous to `rx_clk`.

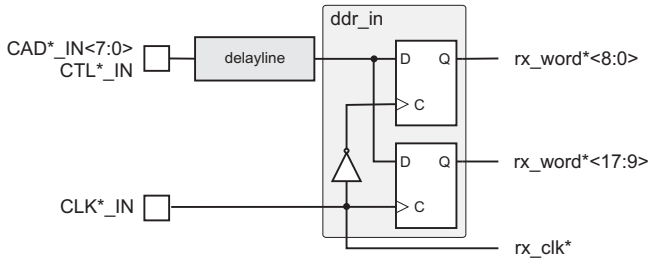


Figure 4.9: DDR input registers with delayline. The squares denote the input pads of the chip, the signals are labeled according to the pinout of the chip given in appendix C.1. The clock phase is inverted for the two halves of *rx_word* to account for the transmit side output timing.

Equal signal propagation delay is required for both clock and data signals to comply to the interface timing defined in figure 2.3. To accomplish a delay on the data signals equal to the clock tree latency of `rx_clk`, the methodology described in section 2.4.4 is used for the physical implementation; implementation results are given in section 4.4.4. One implication of the applied methodology is the modified phase shift of $-90°$ of the link clock with respect to the data signals (cf. section 2.4.4). To capture the data with the correct phase of $90°$, the clock is inverted and thereby shifted by the required $180°$.

Transmit Side The implementation of the transmit side of the physical layer is shown in figure 4.10. Again, data bits and control bit are not distinguished as this is only relevant for the link layer. Data to be transmitted (`tx_word<17:0>`) is provided by the link layer with a width of 18 bit. It is transmitted synchronous to the 400 MHz core clock `clkhi`. Double data rate transmission is achieved by triggering the select pin of the multiplexer with `clkhi`; after a rising edge of `clkhi`, `tx_word<8:0>` is available at the output of the multiplexer and after a falling edge `tx_word<17:9>` is available respectively. The clock tree is balanced only up to the clock pins of the two flip-flops and the input pin of the first buffer before the multiplexer. To avoid race conditions between the clock to output time of the flip-flops and the select to output time of the multiplexer, all components are selected such that the multiplexer switches shortly after the output of the flip-flops is stable.

To compensate skew that is introduced either internally or by external signal routing, a *delayline* is used for each output data signal and the frame bit. The required interface timing is likewise obtained by applying the strategy described in section 2.4.4: when using the default

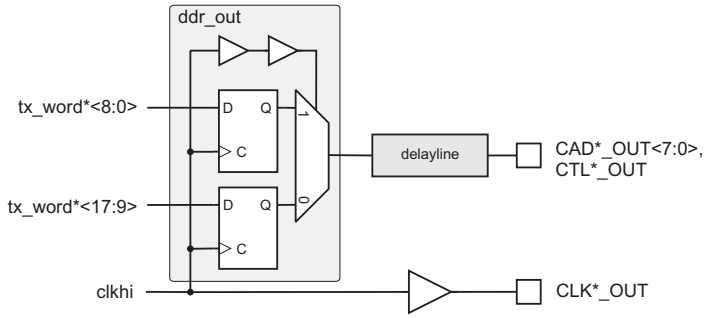


Figure 4.10: DDR output register with output multiplexer and delayline. The squares denote the output pads of the chip, the signals are labeled according to the pinout of the chip given in appendix C.1. The buffer selected for output is set by dedicated signals that are connected to the input data lines of the chip.

delay $t_{\mathrm{dl},3}$ of the delay lines, data signals are valid at the output pads with a shift of $-640\,\mathrm{ps}$ relative to the link clock signal which closely matches a phase shift of -90° at a clock frequency of 400 MHz (clock period of 2.5 ns). The results of the physical implementation of the transmit side will also be given in section 4.4.4.

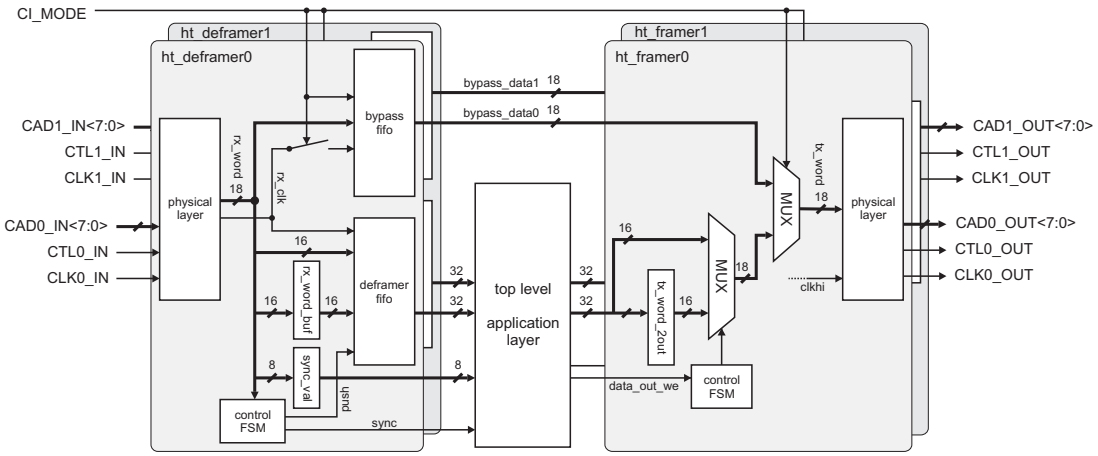


Figure 4.11: Block diagram of the modules *ht_deframer* and *ht_framer* that together comprise the link layer. The squares denote the I/O pads of the chip, the signals are labeled according to the pinout of the chip given in appendix C.1. Only the signals relevant for the data flow are shown.

Link Layer

The link layer of the developed protocol translates requests issued by the application layer into requests for the physical layer and vice versa. More precisely, data packets received by the physical layer are decoded and handed on to the application layer and data packets generated by the application layer are sent to the physical layer. Figure 4.11 shows the data flow through the chip reduced to the functionality of the link layer, which is basically implemented within the modules *ht_deframer* and *ht_framer*.

Besides the interfacing between the physical and the application layer, another fundamentally different functionality is implemented within the link layer. The switching between these

different behaviors is not part of the communication protocol and is done by an external pin to the chip, CI_MODE (cf. figure 4.11). They are described in the following.

Regular Operation CI_MODE = *low*. The *ht_deframer* module performs the decoding of data packets, whereas three types of packets are distinguished on the link layer: idle packets, synchronization packets and regular packets. In table 4.2 the content of these types of packets is summarized. From right to left the bit position within the respective 8 bit link is shown and the leftmost bit denotes the frame (CTL) bit of this link. From top to bottom the bit time for each link is shown in ascending order. Bit time 0 always starts at a rising clock edge of the link clock and as a consequence, within each link clock cycle, 18 bit of data are delivered by the physical layer. Packets other than the idle packet are marked by a change of the frame bit from 1 to 0 in the first two bit times and require four bit times on both links to be transmitted. The frame bit stays 0 for the last two bit times of these packets. Each link receives 32 bits of data during one packet cycle which sums up to the whole 64 bits of data transmitted within one packet.

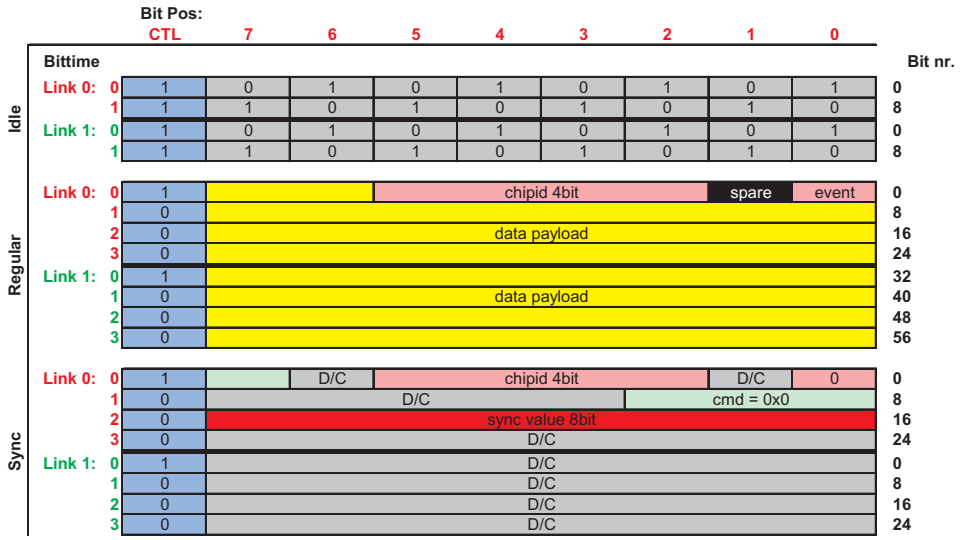


Table 4.2: The three types of packets that are distinguished on the link layer.

Incoming data is processed by a finite state machine (FSM) inside *ht_deframer*, which is implemented straight forward and therefore not described in detail. Idle packets carry no information and no action is triggered within *ht_deframer* when receiving this packet. To avoid the development of static DC voltages on the data lines of the link, the idle packet contains a data pattern that ensures continuous toggling of each data line during idle periods[48].

Upon reception of a regular packet, the 16 data bits received in the first clock cycle are temporarily stored within the register *rx_word_buf*. In the second clock cycle of the packet cycle the content of this register is stored within a FIFO memory together with the current content of *rx_word*. The frame bit is solely needed for the detection of a packet start and is therefore not stored in the FIFO along with the data. The FIFO is taken out of the Synopsys DesignWare [Syn04d] and has two independently clocked interfaces with a width of 32 bits

[48] A discharge of a physical line to a DC voltage due to a constant binary value of e.g. 0 causes jitter in the signal edges which in turn reduces the width of the data valid window. This effect reduces the maximum achievable data rate on the link.

each, whereas the push interface is operated synchronous to the 400 MHz clock belonging to the respective input link and the pop interface is clocked by the 200 MHz `chip_clk`. The pop interface represents the interface to the application layer. Communication is managed by handshake signals that are omitted in figure 4.11 for simplicity.

Synchronization packets are introduced for the purpose of setting the system time counter of the chip (cf. section 4.2.3). This action actually is part of the application layer functionality; it is still implemented within the *ht_deframer* module because the application layer decodes commands at the speed of the 200 MHz core clock and the system time counter runs within the 400 MHz domain. The very clock cycle within which the counter is to be synchronized can therefore not be determined and the synchronization is initiated from the link layer which operates at 400 MHz. A synchronization command is detected during the first clock cycle of a packet cycle on link 0 if the input data conforms to table 4.2. In the second clock cycle of the packet, the 8 bit value for the system time counter is transmitted, which is then stored in the register *sync_val*. The signal *sync* is issued in parallel to the core logic where the actual synchronization takes place (cf. section 4.3.4).

Generally, received data is pushed into the FIFO regardless of the type of packet and the synchronization packets are not processed by the application layer. It has to be ensured that link 1 also detects a packet (content is "don't care") during synchronization.

The *ht_framer* module performs the encoding of data packets that are to be transmitted. Data generated by the core logic is already formatted as 64 bit packet and 32 bit at a time are assigned to each *ht_framer* from within the 200 MHz domain. Transmission to the physical layer takes place within the 400 MHz domain of `clkhi` (see figure 4.11) and the lower 16 bit of `data_out` are directly multiplexed to the physical layer during the first clock cycle. The upper 16 bit are stored for one clock cycle within *data_out_reg*. In the second clock cycle, *data_out_reg* is multiplexed to the physical layer and the packet cycle is complete. In parallel to the multiplexing of the data, the frame bits are generated according to table 4.2 making the output data `tx_word` totally 18 bits wide.

Bypass Operation `CI_MODE` = *high* (cf. figure 4.11). The purpose of this mode is to verify the functionality of the physical layer and to provide the possibility of adjusting the delay value of the single *delaylines* present in the physical layer. In this mode, the link layer is disconnected from the application layer and the chip acts like a shift register. Data received by the physical layer is constantly pushed into the FIFO *bypass_fifo*. The same basic FIFO is used as in the case of *deframer_fifo* but with a width of 18 bit and the complete data of the physical layer including the frame bits is pushed within each cycle of `rx_clk`. The pop interface is also continuously read using the 400 MHz `clkhi` and the output data is multiplexed to the transmit side of the physical layer.

The functionality of this mode does not depend on the content of the transferred data and it is thus ideally suited for thorough testing of the physical layer. The test procedure will be outlined in section 5.3.

Bit Pos:	CTL	7	6	5	4	3	2	1	0
Link 0	delay value 3bit			address of *delayline* 6bit					
Link 1	D/C					chipid 4bit			

Table 4.3: Bit meanings on both links during configuration of the *delaylines*. Bits marked with D/C are "don't care" and are not used.

During bypass operation, the delay of the *delaylines* can be set using the external pin `C_DELAY`. In order to load a certain delay value into one of the *delaylines*, static data has to be applied to the LVDS input pads of the chip according to table 4.3. A specific *delayline* is addressed by six address bits and four bits for the daisy chain address of the chip. The decoded enable signal is AND'ed together with `C_DELAY` and is then connected to the `load_del` input of the *delaylines* (cf. figure 4.8). New values are then loaded on the rising edge of `C_DELAY`. All signals used for the configuration of the *delaylines* are buffered with the smallest available buffer HDBUFDL in order to minimize the additional capacitive load on the input data lines (operating at a bit rate of 800 Mbit/sec).

4.3.2 The Application Layer

The following description of the application layer refers to figure 4.6. Besides alien packets addressed for different chips in the daisy chain, the application layer distinguishes between *event* packets and *control interface* packets. The generic content of these packets is shown in table 4.4. Data flow through the application layer takes place with a delay of one 200 MHz clock cycle per packet. Depending on the type of an incoming packet, either *event_reg* or *data_out_reg* (cf. figure 4.6) is updated in the subsequent clock cycle and the according data is multiplexed to the link layer.

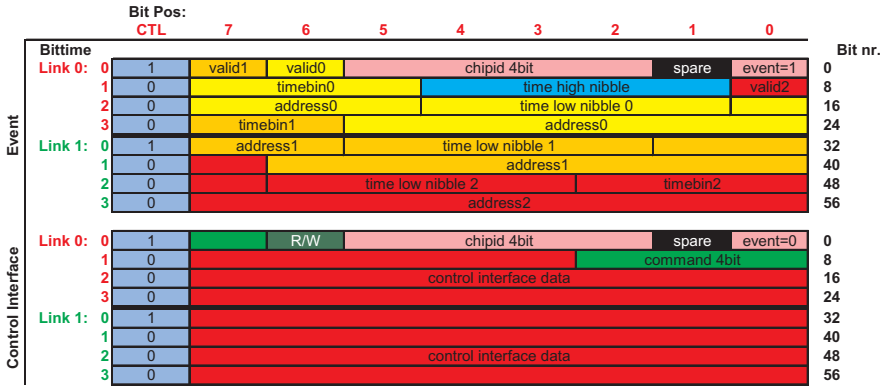


Table 4.4: Generic definition of event packet and control interface packet. The type of packet is determined by bit 0. The `chipid` which addresses a particular chip and a spare bit that may be used for protocol extensions within future revisions are common to both types.

Event Packet Data Flow Event packets may contain the data of up to three events. The *event_buffer_in_mux* module decodes this information and stores the events in the according modules *event_buffer_in*. Sixteen *event_buffer_in* modules are present on the chip, two at a time connected to two DTCs serving 64 synapse drivers.

Spikes generated by the neurons are digitized in time by the TDCs which pass the time bin information and the spiking neuron's address to the *event_buffer_out* module, where the event is temporarily stored. Event packets are assembled within the *event_buffer_out_mux* module and are temporarily stored within the register *event_reg* until transmission. Valid events are marked by means of bits `<12:10>` of the event packet. 9 bits for the neuron address[49], the lower nibble of the events' time stamp and the 4 bit for the time bin are separately stored

[49] For incoming packets, this is the address of a synapse row to drive and for outgoing packets it is the address of the neuron that has fired.

for each event. The upper nibble of the time stamps is the same for all three events. This compression technique makes it possible to store three events within one 64 bit packet.

Using this format, three different types of event packets are defined which trigger the following actions:

- *Regular* event packets are defined as described above and the contained events are delivered to the analog part as described in section 4.3.5. As an answer to this packet, the chip may either transmit own event data that have been assembled within *event_reg* or, if none are available, send an *empty* event packet.
- *Empty* event packets are indicated by the reserved `chipid 0xf` and the valid bits `<12:10>` are set to 0. If an empty event packet is received, the chip may fill this packet with pending event data or forward the packet to the daisy chain. By this means, unused packet slots may be occupied by subsequent chips that do have event data available. This packet can also be used to empty the FIFOs in *event_buffer_out* of all chips in a chain.
- *Error* event packets are defined by a valid chipid and the valid bits <12:10> set to 0. They are generated by the chip if an overflow of either the FIFOs within *event_buffer_in* or *event_buffer_out* occurred and are evaluated by the controller for monitoring purposes.

Command	Meaning
`0x0`	sync (reserved, used by link layer)
`0x2`	control interface loopback
`0x4`	parameter storage and control
`0x6`	chip control and status register
`0x8`	synapse memory control
`0xa`	analog readout module
`0xc`	event loopback module
`0xe`	dummy (reserved, reset state of *result_reg*)
`0x1`	control interface error if OR'ed with one of the above

Table 4.5: Available commands within the control interface packet. If the LSB of the command is set ("0x1"), this indicates an error for the actual command.

Control Interface Packet Data Flow The term "control interface" covers all functionality of the application layer but the event processing. The according data packets contain the address of one of the six core modules depicted in figure 4.6 as a command. This command is stored in bits `<10:7>` of the packet and bit `<6>` indicates a read (`1`) or write (`0`) access. The available commands are listed in table 4.5 and a complete description of these commands is given in appendix B. The content of a control interface packet is stored in the register *command_reg* for further processing by the top level command decoder and also serves as input data to the core modules. The top level command decoder is implemented as a FSM that issues a request (signal `req`) to the addressed module along with the signal `rw` which indicates read or write access. It basically has three states: `SN_idle`, `SN_read` and `SN_write` and after leaving the idle state it only resumes from the active state if the addressed module issues its acknowledge signal `ack`.

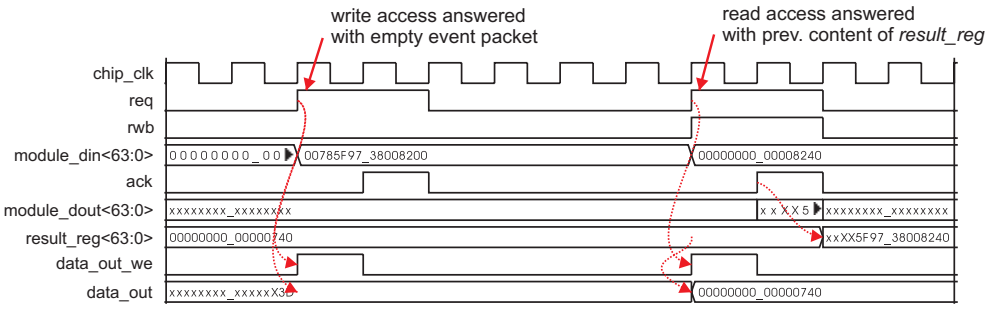


Figure 4.12: Timing diagram for a write and a read access on the interface of the modules present within the application layer at the top level of the chip.

The timing diagram for a write access followed by a read access is shown in figure 4.12, thereby illustrating the answer of the chip:

- Write: Write accesses are executed immediately and are not being acknowledged by the chip. Hence, the thereby available packet slot will be used to transmit event data if available. An error will occur if a previous request is not finished and a write command is issued while the FSM is not idle. In this case, the least significant bit (LSB) of the command is set to indicate the error and the packet is transmitted along with the original content.

- Read: Read requests are also directly issued to the addressed module but the result is soonest available within *result_reg* after the next clock cycle. To still comply to the specified protocol, the content of *result_reg* is nevertheless written to *data_out_reg* and transmitted immediately which effectively causes the result of a previous read access to be transmitted. For this reason, a single read access requires two read commands to be sent and consequently n consecutive read accesses require $n+1$ read commands. The error condition is the same as for the write access and the error bit is set when a read command arrives during the FSM not being idle.

Depending on the functionality of the single modules an access may last over several clock cycles. To avoid errors, the according controller has to take this into account and issue commands with an appropriate distance.

4.3.3 Clock Generation and System Time

The Clock Signals Present on the Chip

The *clock_gen* module (cf. figure 4.6) generates five clock signals that are listed in table 4.6. An external clock source is connected to the differential input pads EXT_CLK_(P/N) and subsequently to the PLL which has a $4\times$multiplier[50]. As a consequence, the chip needs to be supplied with an external clock at 100 MHz which imposes weaker demands on the signal routing on the board level than a 400 MHz clock distribution. In case of malfunction, the PLL may be deactivated with the external pin `PLL_BYPASS`. The external clock is then directly forwarded by the PLL.

[50]The PLL was generated with a PLL compiler supplied by Virtual Silicon Technology, Inc. [Vir04a] and is parameterized to an input frequency of 100 MHz and an output frequency of 400 MHz.

Clock	Frequency	Purpose
chip_clk	200 MHz	core clock for dig part
clkhi	400 MHz	dig event time base, transmit clock
anaclk	200 MHz	DTC: event generation
anablkb	200 MHz	DTC: event generation, 180° phase shifted
anaclkhi	400 MHz	TDC: event digitization

Table 4.6: Clock networks within the digital part of the Spikey chip.

The 200 MHz chip_clk is generated from the 400 MHz PLL output by means of a standard cell flip-flop (cell HDDFFPB2) that has its inverting output fed back to its input, thereby dividing the applied clock frequency by two at its Q output. To still ensure that both chip_clk and clkhi are generated with negligible phase shift, a replica cell has been realized with same internal structure as the HDDFFPB2 flip-flop, but toggling its output synchronous to both edges of the input clock. Thanks to the equal propagation delay within both cells, both clocks are generated in phase and the clock tree can reliably be built for both clocks. The clocks used for event generation are derived from chip_clk and clkhi which are AND'ed together with an according control signal to offer the possibility to switch off all clock signals relating to the analog part.

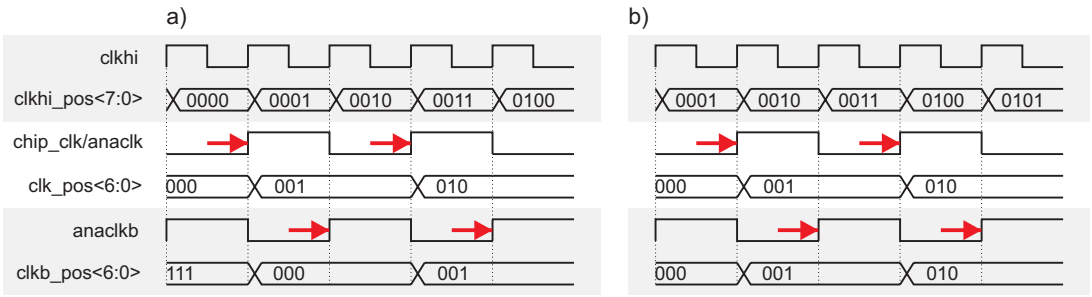


Figure 4.13: The system time counters relevant for event generation. The counters clk_pos and clkb_pos together account for the LSB information of the actual system time, clkhi_pos. Four consecutive clock edges for event generation at system times 0000 to 0100 are marked with red arrows at the according clock edge in both figures. a) events with even time stamp are assigned to "anaclk". b) events with odd time stamp are assigned to "anaclk". The upper nibble of all binary values is omitted for reasons of readability.

System Time Counters

The *sync* module contains the system time counters that serve as a time base for the event generation and digitization on a clock cycle basis within the chip. Three event-related clock domains are present on the chip (cf. table 4.6). To prevent timing arcs crossing these clock domains, three system time counters are implemented and the related logic will operate based on the respective counter:

- clkhi_pos<7:0>: This 8 bit counter is used as a time base for digitized events within the clkhi domain. Its value during event capture from the TDCs completes the time stamp of the digitized event. As clkhi is the fastest clock in the chip, this counter is the reference time for the other counters running at lower frequencies.

- clk_pos<6:0>, clkb_pos<6:0>: These two 7 bit counters are associated with the chip_clk domain and together account for the LSB value of the system reference time. Their purpose is defined such that events that are to be generated within the first half of chip_clk are related to clk_pos and events due in the second half are related to clkb_pos. The relevant clock edges for the generation of four consecutive events are illustrated in figure 4.13.

Assignment of the *event_buffer_in* modules to the DTCs

While the event digitization relating to clkhi is performed deterministically, the LSB value of clkhi_pos during the first and second half of chip_clk is not fixed. Therefore, a decision has to be made on which events (time stamp LSB = 0/1) to assign to which of the two counters. For this reason, the processing of events is now qualitatively described first, to derive the requirements on the synchronization which will then be detailed.

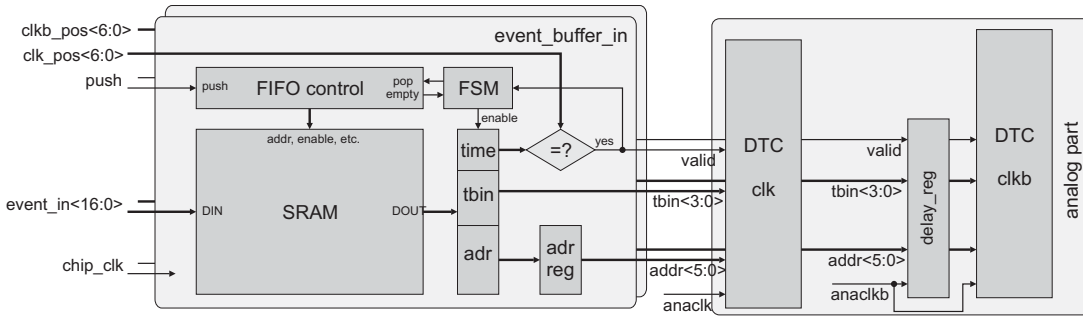


Figure 4.14: Data flow in the *event_buffer_in* module. The module is either directly connected to a DTC which is clocked by anaclk, which is the gated version of chip_clk, or its output is registered once with the 180° phase shifted anaclkb and then shipped to a DTC clocked by anaclkb.

Figure 4.14 shows the implementation of the *event_buffer_in* module which delivers incoming events to the analog part, precisely the DTCs. The input to all modules is synchronous to chip_clk and eight of these modules are assigned to the anaclk domain and to the anaclkb domain respectively with the according counter values (denoted by the two boxes in figure 4.14). The output of the *event_buffer_ins* associated with anaclkb is transferred to this clock domain by means of one register stage and as a consequence, events stored within these modules are delivered half a clock cycle after the events stored in the other domain with respect to one cycle of chip_clk.

It is decided during the synchronization of the counters, whether to store events with an even time stamp (LSB = 0) in the modules associated to anaclk or to anaclkb. Synchronization is described in the following.

4.3.4 The Synchronization Process

Synchronization is required to provide a consistent global time base for systems consisting of several chips. Moreover, the system time counters need to be synchronized to the time of the controller to provide correct temporal processing of events (cf. section 5.3.2). The *sync* module contains a FSM that loads the counters when a synchronization command is captured by the link layer. The sync signal issued by the link layer is generated in the link clock domain and is synchronized by two consecutive registers to the clkhi domain to avoid

metastable register states. Subsequently, the system time counters are loaded with their new values and the assignment to the clock domains is determined. This decision is stored in a register *evt_clk_select* and is valid until the next synchronization or reset.

chip_clk Phase	Sync time LSB	*evt_clk_select*	even time stamp →
low	0	1	`anaclk`
low	1	0	`anaclkb`
high	0	0	`anaclkb`
high	1	1	`anaclk`

Table 4.7: Possible scenarios during the synchronization process. The value of the *evt_clk_select* register and the according assignment of events with even time stamps to the respective clock domain is shown.

Table 4.7 lists four possible scenarios for *evt_clk_select* during the synchronization process which depend on the following conditions:

- Does the synchronization occur in the low phase or in the high phase of `chip_clk`
- The LSB value of the time to synchronize to.

Depending on the value of *evt_clk_select* events with an even time stamp (LSB = 0) are stored within the modules *event_buffer_in* associated to the clock as listed in table 4.7. The same holds true for events with an odd time stamp in an inverted manner.

Another issue is the initialization of the 200 MHz system time counters during synchronization. Two of the four possible scenarios are illustrated in the timing diagrams in figure 4.13. They will be valid if `chip_clk` is low during the arrival of the synchronization command.

In figure 4.13 a, `clk_pos` needs to precede `clkb_pos` by one clock cycle to deliver the events within the correct cycle of `clkhi` because the LSB of `clkhi_pos` toggles on the rising edge of `chip_clk`. In this case, the counters are initialized as follows (in Verilog syntax):

```
clk_pos  <= sync_val<7:1> + 1;
clkb_pos <= sync_val<7:1>;
```

the register `sync_val` stores the synchronization time transmitted along with the synchronization command (cf. figure 4.11). In figure 4.13 b the LSB of `clkhi_pos` toggles on the falling edge and the two 200 MHz counters need to count synchronously. Therefore, they are initialized with identical values:

```
clk_pos  <= sync_val<7:1>;
clkb_pos <= sync_val<7:1>;
```

The two remaining scenarios occur if `chip_clk` is high during arrival of the synchronization command. In this case, the afore described initialization is carried out inversely relative to the LSB toggling of `clkhi_pos`. The reader may refer to appendix C.3 for the commented Verilog source code of the *sync* module.

4.3.5 Event Processing in the Chip

This section deals with the event processing modules within the chip. They complement the external modules that are required for the setup of large scale spiking neural networks as described in section 3.3.1.

Digital Event Input

The structure of the *event_buffer_in* module which performs the event generation has been shown in figure 4.14. The input signals `eventin<16:0>` (the complete event data) and `push` are generated by the *event_buffer_in_mux* module which demultiplexes the incoming events according to the upper three bit of the target synapse and the LSB of the time stamp, as described in the preceding subsection. Therefore, the `eventin` signal consists of a 6 bit address selecting one out of 64 associated synapse row drivers, a 7 bit time stamp, since the LSB information of the system time is contained in the clock domain where the event is being stored, and 4 bit for the time bin information.

Events are stored in a dual ported static memory acting as a FIFO. The FIFO control logic has been taken from the Synopsys DesignWare library [Syn04e] and the dual-port SRAM has been generated using a compiler by Virtual Silicon Technology, Inc. [Vir04c] with a width of 18 bit (only even widths are allowed by the compiler) and a depth of 64 words, which also is the depth of the FIFO.

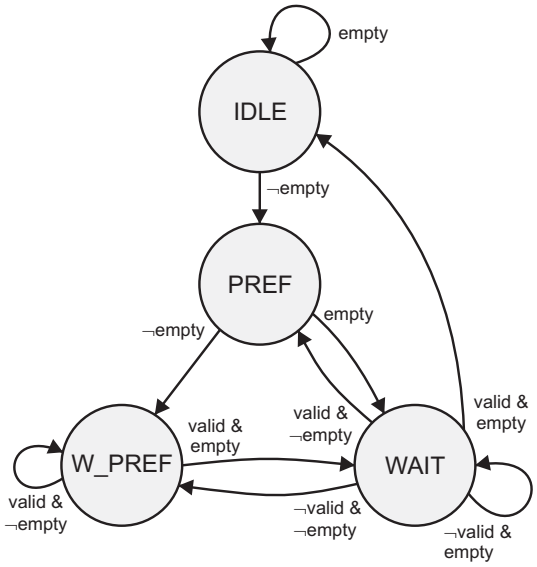


Figure 4.15: State diagram of the FSM in the *event_buffer_in* module. Transitions depend on the `empty` status bit of the FIFO and the signal `valid`, which is active when `clk_pos` has reached the value of the current event's time stamp.

The operation at the output of the FIFO, thus, the event generation is controlled by a FSM (see figure 4.15). If the FIFO contains data, the FSM will leave the `IDLE` state and stay in the prefetch state `PREF` for one cycle in which the FIFO is immediately popped. The event popped out of the FIFO is stored in the register stage illustrated in figure 4.14 within the subsequent clock cycle and depending on the fill level of the FIFO, the FSM simultaneously goes into either of these two states:

- `WAIT`: No more events are present in the FIFO. If `clk_pos` matches the time stamp of the event, the event is generated. A state transition takes place according to the state diagram in figure 4.15. It may be triggered either by the event generation or by the FIFO changing to not empty.

- `W_PREF`: This is the "prefetched wait" state where the content of the register stage is valid and the FIFO contains valid data which is already popped. The FSM will reach this state if more than one event is due and the only possible transition to leave this state is through the WAIT state in which only one event is due and the FIFO is empty.

The actual event generation requires the address signals to be delayed by one clock cycle. The DTC synchronously registers the `valid` and the `tbin` signals and generates the event within the appropriate time bin of the following clock cycle. The delayed address signals are therefore valid within the correct clock cycle.

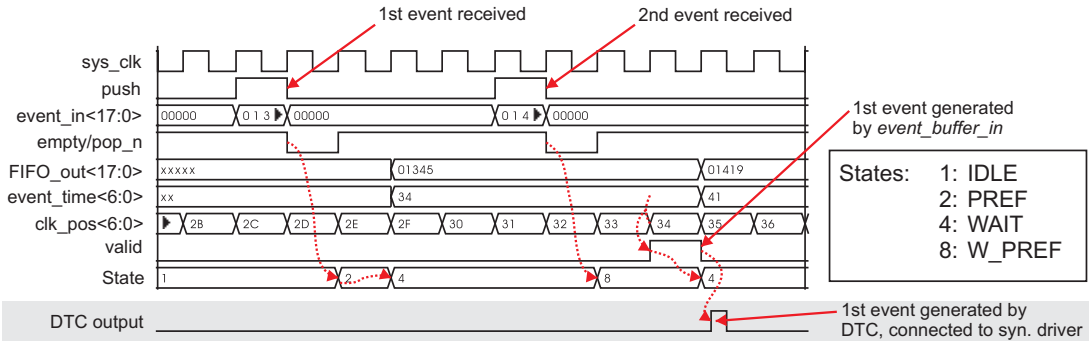


Figure 4.16: Timing diagram for the interface of *event_buffer_in* and the event generation. Two events are received by this module and the timing until the generation of the first one is shown. Note: the output of the DTC is generated within the `clkhi` cycle following the clock cycle of event generation by *event_buffer_in*. The labels for the states correspond to figure 4.15.

Figure 4.16 illustrates the arrival of two events and finally the generation of the first event. It can be seen that the minimum latency through the *event_buffer_in* module sums up to four `chip_clk` cycles. Three cycles are introduced by the FIFO, which is the minimum possible latency using a standard synchronous FIFO together with the SRAM and another cycle is introduced by the register stage. This stage is necessary because of the SRAM having a clock to output delay of up to 2.2 ns in the worst process corner. Alongside with the comparator for the time stamp this exceeds the required time of 2.5 ns for the potential path from `chip_clk` to `chip_clkb`.

Digital Event Output

DTC Output Timing A spike that is generated by the neuron circuits is digitally forwarded to the TDC, along with the address of the spiking neuron. The timing diagram for the following event digitization process is shown in figure 4.17. The digitization is carried out synchronous to the 400 MHz `anaclkhi` and involves the storage of the DLL's current time bin during the spike occurrence. As illustrated in the timing diagram, the wrap around of these time bins is shifted by Δt_{DLL} with respect to `anaclkhi`, which is due to intrinsic component and routing delays caused by the physical layout of the DLL circuitry. For this reason the actual digitization is not directly carried out within the clock cycle preceding the output clock cycle n, but rather between the two falling edges of `anaclkhi` preceding the output clock cycle. To determine whether the spike occurred within clock cycle $n-1$ or $n-2$, the `early` bit is generated together with the value of the time bin. This bit is generated synchronous to `anaclkhi` and is `0` if the spike occurred within $n-2$ and `1` if it occurred in $n-1$. As the time window for the generation of the `early` bit is shifted by 180° with respect to the time

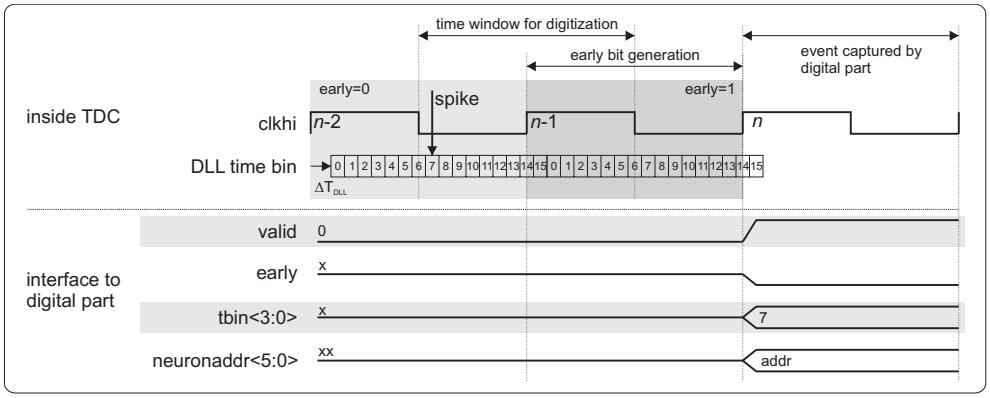


Figure 4.17: Timing diagram of the event digitization process. The event denoted with "spike" will be stored with `tbin`= 7 and the system time value of clock cycle $n-2$. Note: the time window for the `early` bit generation is shifted by 180° relative to the digitization time window.

`tbin` MSB	`early`	Clock Cycle	Clock Cycle (impl.)
0	0	$n-2$	$n-2$
0	1	$n-1$	$n-1$
1	0	$n-2$	$n-2$
1	1	$n-2$	$\mathbf{n}-1$

Table 4.8: Determining a digitized event's time stamp by its time bin's MSB and the `early` bit. The actual implementation for both versions of the Spikey chip is shown in the rightmost column. In the last line, $n-1$ is implemented instead of $n-2$ (see main text).

window for the digitization, the clock cycle the event occurred in can reliably be determined using the truth table 4.8.

The *event_buffer_out* module This module mainly serves as a FIFO memory that stores events generated by the TDCs as described above until they are sent off chip. As the input to the FIFO is clocked with `anaclkhi` and the output is operated with the `chip_clk`, a FIFO controller from the Synopsys DesignWare library is used that supports two independently clocked interfaces [Syn04c], along with a dual ported static memory that is again generated using the compiler from Virtual Silicon Technology, Inc. [Vir04c]. The FIFO has a width of 18 bit and a depth of 128 entries. This large depth will be required to handle peak event rates even if the chip cannot immediately send the stored events off chip[51].

The signals `valid, early, tbin<3:0>` and `neuronaddr<5:0>`, which are generated by the TDC within the analog part (see figure 4.18), are the input to the module. If `valid` is active, the appropriate system time[52] will be selected according to table 4.8 and the FIFO is pushed storing the newly generated event. It consists of the address of the firing neuron, the time bin, and the system time when it occurred.

[51] This situation will possibly occur if the daisy chain is occupied with packets for another chip in the chain. This prevents the remaining chips from using packet slots and sending their event data (see section 4.3.2).

[52] `clkhi_pos` (used as time stamp for $n-1$) is the 400 MHz system time counter discussed in section 4.3.4. `prev_clkhi_pos` (used as time stamp for $n-2$) is generated by registering `clkhi_pos` once with `clkhi`. As a result, `prev_clkhi_pos` = `clkhi_pos` −1 and a subtraction is avoided.

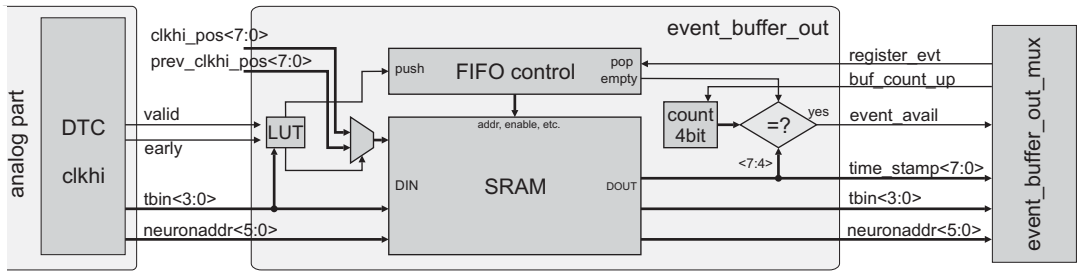


Figure 4.18: Data flow in the *event_buffer_out* module. Events digitized by the TDC within the analog part are stored within the FIFO. The right side is connected to the *event_buffer_out_mux* module, which finally assembles the event packets. Events are requested via the signal register_event.

64 Neuron-Blocks	Packet Slot	Resulting Addresses
0, 3	0	<63:0>, <319:256>
1, 4	1	<127:64>, <383:320>
2, 5	2	<191:128>, <447:384>

Table 4.9: Assignment of 64 neuron-blocks to event slots in the event packet and the resulting neuron addresses. Neuron blocks 0 to 2 are located on the left *network_block*, starting from the middle of the chip. Addressing of blocks 3 to 5, which are located on the right *network_block* starts with 256 in analogy to the synapse addresses. The addresses are also incremented from the middle of the chip outwards.

At the output to the digital part, the pop interface of the FIFO is combined with a 4 bit counter. This counter operates synchronous to `chip_clk` and stores the current high nibble of the event packet to be generated. It is incremented by the signal `count_up` generated within the *event_buffer_out_mux* module, which is described in the following paragraph. Since all *event_buffer_out* modules only would indicate valid event data if the value of this counter matches the high nibble of their currently available event's time stamp, the high nibble for the subsequently assembled event packet is already fixed and assembly is facilitated. The FIFO is popped upon request via the signal `register_event`.

The *event_buffer_out_mux* module This module is not separately illustrated; its position in the data flow is illustrated in figure 4.6. It outputs the data for an event packet. Six *event_buffer_out* modules are connected to it where two at a time are assigned to one of the three event slots of the packet. The global 9 bit neuron address of the events is herein created, based on the 6 bit address stored with the event and the address of the according 64 neuron block. The assignment of the 64 neuron blocks to the packet slots and the resulting addresses are given in table 4.9.

During operation, the `count_up` signal will always be issued until the first *event_buffer_out* flags a valid event, if this is not already the case. The presence of the 4 bit counters ensures correct time stamps for all events flagged valid. These are multiplexed into the packet. If two concurring *event_buffer_outs* should indicate valid events simultaneously, they are read out alternately, and no special priority scheme is used here.

The `count_up` signal will only be issued if the value of the 4 bit counter does not exceed the upper nibble of the current system time. This precaution has not been taken within Spikey 1. Therefore, more recent events with a larger time stamp might be transmitted before

older events in case of a wrap around of this 4 bit counter. This beaks down the correct functionality of the event processing algorithm outside the chip (cf. section 3.2.3), which assumes that events arrive with ascending time stamps.

Shortcomings in the Implementation Instead of the third column in table 4.8, the rightmost column has wrongly been implemented in both versions of the Spikey chip. This results in events that potentially have time stamps with an error of $+1$ LSB. The probability for this error within one clock cycle is

$$p_{\text{error}} = \frac{\Delta T_{\text{DLL}}}{\Delta T_{\text{tbin}} \cdot 16} , \tag{4.2}$$

with ΔT_{DLL} being the phase shift of the DLL time bins' wrap around point with respect to `chip_clk`, as illustrated in figure 4.17, and ΔT_{tbin} being the length of one time bin. The total number of time bins within one clock cycle equals 16. The occurrence of this error is demonstrated by measurements presented in section 6.5.5.

4.3.6 Digital Core Modules

In this subsection, the Verilog modules within the application layer are described (they are illustrated in figure 4.6). The interface to the top level command decoder is standardized for all modules and follows the timing diagram given in figure 4.12.

The Status and Control Register

This module basically consists of a set of registers the either control parts of the functionality of the chip or hold the status of certain other modules inside the chip. For example, the status of the FIFO controllers within *event_buffer_in* and *event_buffer_out* can be read out using this module. One read command is defined for all FIFOs relating to one of the three clock domains at a time. The status bits transmitted for each FIFO are listed in appendix B. Thereby it is possible to monitor the status of the event generation logic during continuous operation. For each of the clock domains, the current values of the associated system time counters are also transmitted and the read answer for the *clkhi* domain additionally contains the common high nibble of the current event packet and the state of the `PLL_LOCKED` signal.

The content of global control registers is modified by writing to this module and the functionality associated with these registers is also listed in appendix B.

The Loopback Module

This module is implemented to verify the basic functionality of the application layer. Upon a read request, it registers the inverted version of the data at its `data_in` port for one clock cycle and acknowledges the read access in the following clock cycle, thereby sending the stored data to the `result_reg`. By using the *loopback* command it can be verified whether the command decoder at the top level is working correctly and whether read and write accesses are answered correctly by the chip.

The Event Loopback Module

The purpose of this module is to verify the functionality of the event processing modules *event_buffer_in* and *event_buffer_out* along with the synchronization of the chip. As illustrated in figure 4.6, event data generated by the *event_buffer_in* modules is additionally connected to

the *event_loopback* module. The module basically acts as a pipeline register and its output is connected to the input of the *event_buffer_out* modules which process the pipelined events as if they were generated by the analog part.

First, the pipeline registers are clocked with the source clocks `anaclk` and `anaclkb` in order to minimize the number of flip-flops in the pipeline clocked with `clkhi`. Second, the content of the registers with the source clock currently being in its low phase is multiplexed to the output, depending on the phase of `clkhi` relative to the source clocks. To test all of the *event_buffer_ins*, the output of two of these modules is multiplexed to the same upper pipeline and the select input of these multiplexers is constantly driven by a control register. Likewise, a constant value for the early bits, sent along with each event output can be set (no early information is generated at the input). Finally, to minimize power consumption if the module is not active, all pipeline registers are connected to one global clock enable signal. This signal also serves as the select input to the multiplexers at the input of the *event_buffer_out* modules.

No timing constraints have been set on this enable signal in order to relax the timing. Therefore, the output of the pipeline presumably contains invalid data for several clock cycles after activation of the event loopback. For this reason, the reset of all *event_(in/out)_buffers* has to be pulsed after each change of the enable signal by means of sending the appropriate commands to the control register.

Parameter Ram Control

The purpose of this module is to digitally store the values of the various model parameters that are applied to the neural network circuits described in section 4.2. These parameters are stored within current memories and need to be periodically refreshed. Therefore, a controller addresses each current memory in a customizable sequence and applies the according parameter value to the DAC.

The storage is accomplished by the usage of a single ported static memory which is generated using the SRAM compiler from Virtual Silicon Technology, Inc. [Vir04b]. The following data is stored for each parameter: the physical address of the parameter within the analog part (12 bit), the value to apply to the DAC (10 bit), and the information about how long to apply the value to the respective current memory (4 bit). The mapping of the physical addresses to the model parameter addresses is listed in appendix A. The DAC can be set from 0 to $I_{\mathrm{refdac}}/10$ with the according 10 bit resolution (cf. section 4.2.1).

In particular, the 4 bits defining the update duration of the parameter contain the 4 bit address of a LUT which actually defines the procedure used to apply the DAC value. One LUT entry contains four values: the number of clock cycles to activate the `boostb` pin of the DAC (4 bit) and the number of clock cycles during which it is inactive (4 bit). The two remaining values are optional and allow for the definition of the number of automatic increments of the physical address (8 bit) with a certain step size (4 bit) while keeping the parameter value and the time unchanged. The default is to set the number of increments to 1.

Functionality The complete configuration of the module requires to correctly set at least one of the LUTs, which can be used as a the update duration for all parameters. To ensure correct operation of the chip, a value for every physical parameter has to be given while the amount of entries in the memory can be reduced by using the described auto-increment functionality. The total number of valid entries within the parameter memory is stored in the register *numparameters* (12 bit). Refresh cycles are performed by a FSM which periodically reads the

memory content from address 0 to *numparameters*−1 and generates the appropriate address signals and DAC values.

The update functionality is controlled by the signal `update_en`, set within the control register. Unless this signal becomes active, all parameters are periodically updated to zero current to prevent the current memories from drifting to their maximum output current [Scha].

The Analog Readout Module

The read out chain for the analog parameter voltages generated within the analog blocks *vout_block* is controlled by this module (cf. section 4.2.1). With a write command, the module (left or right *vout_block*) is addressed and the bit pattern to clock into the flip-flops of the read out chain is transmitted. The update of one chain is initialized by resetting all chains and then serially shifting the bit pattern into the flip-flops of the addressed chain. This way it is ensured that only the addressed chain does contain an active output and shorts between the outputs of both chains to the common `IBTEST` pin are omitted.

While the new bit pattern is shifted into the chain, the output of the chain is stored within a register stage of the module. Its content can be read back to verify the correct functionality of the flip-flop chain within each *vout_block*.

Synapse Ram Control

Several static memory resources are distributed over the *network_block*. The *synapse_control* module mainly acts as a memory controller to these resources by means of a FSM which triggers the data, address and control signals associated with the respective memories in the correct order. The memory resources are divided into four groups:

- Row and Column Configuration Bits: The row circuits are configured with eight bits, two bits are needed for the column circuit configuration. The interface to the column circuits has a width of 4 bit because of the width of the synapse memory. Only bits 1 and 2 are used for the column configuration bits. All configuration bits and their meanings are listed in table 4.11.

- Synapse Memory: This is the 4 bit weight memory located within each synapse. Solely the weight is stored for each synapse, the distinction between excitatory and inhibitory behavior is made within the row drivers.

- Correlation Readout: Reads out the current result of the correlation measurement performed within the synapses (1 bit per synapse). The data width again is 4 bit and four contiguous columns are read out at once. As a consequence, the lower two column address bits are “don’t care” for this access. Please refer to [SGMM06] for further details regarding the correlation measurement.

Note that always several memories are addressed at once and thus accessed with one command. For a row configuration access, the data of both *network_blocks* is transmitted, resulting in 16 bits of data for one access. All other accesses involving column addresses are aligned to the grouping of 64 neurons into one sub block. The two *network_blocks* consist of six of these blocks with each having a 4 bit data bus resulting in 24 bits of data for one access. To summarize, the address space for all accesses covers 256 row addresses and 64 column addresses.

	Bit	Meaning
row config. bits	0 1	`0x0`: ext. event source (*event_buffer_in*) `0x1`: local feedback adjacent *network_block* `0x2`: combine 2 synapses to gain 8 bit resolution `0x3`: local feedback this *network_block*
	2	1: row is excitatory
	3	1: row is inhibitory
	4	1: enable short term depression/facilitation
	5	1: short term depression, 0: facilitation
	6 7	set time constant for short term circuits
column config. bits	1	1: output V_{membrane} via 50 Ω driver
	2	1: enable digital event output

Table 4.11: Description of the configuration bits for row and column circuits. The row configuration bits 0 and 1 together define the functionality described in the last column.

4.3.7 Relevant Figures for Event Transport

Transmission Latencies

The different modules described in the preceding sections as well as the data transport within the physical and the link layer do introduce latencies to the (event) data, which shall be summarized in this section. The data is compiled in table 4.12 and discussed in the following. It has been verified by means of back annotated simulations to ensure correct values for different phase relations between the interface clocks and the core clocks of the chip.

		# clkhi cycles	
		receive	transmit
interface layers	physical layer	2	2
	link layer	4(5)	1
application layer	control interface	2	
	event generation	6(7)	
	event digitization	6(7)	
total latencies	sync. of internal counters	10(11)	
	earliest event generation	12(14)	
	earliest event digitization	9(10)	
	daisy chain delay 1 chip	11(12)	

Table 4.12: Latencies in terms of `clkhi` cycles, which are relevant for the event transport and the synchronization of the chip.

Interface layers The actual latency of the link layer depends on the phase difference between the link clocks `rx_clk*` and `chip_clk`. The minimum of 4 cycles is determined by the two-stage synchronization registers clocked by `chip_clk` of the FIFO within *ht_deframer*.

Application layer Control interface data takes one `chip_clk` cycle to pass through the application layer. The latency for event generation is introduced by the synchronous FIFO within *event_buffer_in* (two clock cycles) and the downstream register stage (one cycle, cf. section 4.3.5). Digitized events are stored within an asynchronous FIFO within *event_buffer_out*, which accounts for three `chip_clk` cycles. The additional `clkhi` cycle present in both cases is introduced depending on whether `chip_clk` is in its low or high state during event generation/digitization within the `clkhi` domain.

Total latencies The uncertain value for the internal counter synchronization requires a careful synchronization method, which is performed by the controller and is described in section 5.2.4. For the setup of large scale neural networks as described in section 3.4.2, the maximum values are considered.

Achievable Event Rates

The event rates achievable by the digital part are an important performance measure, as they indicate the maximum neural activity that can be achieved on both the digital event input and the digital event output of the chip. Since the digital part operates fully synchronous, these rates are best declared in units of events per clock cycle. The according firing rate in biological terms can then be calculated using the speed-up factor s and the digital clock frequency f_c. Based on the architecture of the digital part, these rates are summarized in table 4.13 for $s = 10^5$, a `chip_clk` frequency of $f_{cc} = 156\,\text{MHz}$, and a `clkhi` frequency of $f_{ch} = 312\,\text{MHz}$, which represents the actual operating frequency of the Spikey chip within its operating environment (cf. section 5.2.2).

		events per `clkhi` cycle	events s^{-1} chip [Hz]	events s^{-1} element [Hz]	biol. rate element [Hz]
$r_{\text{ev,max,peak}}$	inp. synapse	8	$1.25 \cdot 10^9$	$2.44 \cdot 10^6$	24.4
	outp. neuron	6	$9.36 \cdot 10^8$	$2.44 \cdot 10^6$	24.4
$r_{\text{ev,max,theo}}$	inp. synapse	1.5	$4.68 \cdot 10^8$	$9.14 \cdot 10^5$	9.14
	outp. neuron	1.5	$4.68 \cdot 10^8$	$1.22 \cdot 10^6$	12.2

Table 4.13: Maximum achievable average event rates. The first column gives the rate in events per `clkhi` cycle for the whole chip. The second column contains the afore rate in Hz for $f_{ch} = 312\,\text{MHz}$. The values are calculated as an average for one single synapse/neuron out of all available within the third column. The last column contains the biological firing rate for one single element at a speed-up factor of $s = 10^5$.

The theoretical maximum rate is determined by the number of DTCs and TDCs. Within one `anaclkhi` cycle, eight events can be generated simultaneously by the available DTCs while the TDCs are capable of simultaneously digitizing 6 events at once. On the input, these rates can be achieved for short periods of time by early enough filling the FIFO memories within *event_buffer_in* (maximum 64 entries) before generating the first event, whereas on the output the duration of the peak rate is limited by the size of the FIFOs within *event_buffer_out* (maximum 128 entries).

On average, the event rate is limited by the physical interface of the chip which transports at maximum one event packet containing up to three events per `chip_clk` cycle. As a result, $r_{\text{ev,max,theo}} = 1.5$ events can be transported per `clkhi` cycle. The values given in table 4.13 are

based on this value and represent the average event rate for one input synapse and one output neuron, out of 512 input synapses and 384 output neurons respectively.

Note that the maximum average biological rate for one single synapse or one neuron can be calculated as $f_{ch}/s = 3120\,\text{Hz}$ using the above values. This hypothetical value will not be used for experiments. Therefore, all values are averaged over the totally available synapses and neurons. During operation of the chip, the rates will presumably range between the average and the theoretical maximum. This is feasible, since on the one hand, not every input synapse or output neuron will require external event transmission. On the other hand, peak rates are expected due to potential synchronized behavior of groups of neurons.

4.4 Mixed-Signal System Implementation

This section deals with the physical implementation of the Spikey chip, which is based on the design methodology described in chapter 2. The RTL-based Verilog description of the complete digital part as well as the set of scripts used for the implementation is available as a SVN[53] repository [EVg].

4.4.1 Timing Constraints Specification

Initially, the clocks `chip_clk` and `clkhi` are specified for a clock period of 5 ns and 2.5 ns respectively. Unfortunately, this timing requirement could not be met for the worst process corner specified in the technology libraries. For optimal implementation results, it is recommended to slightly overconstrain designs by a maximum of 10 % [Bha99]. For this reason, the period constraints were relaxed for the actual implementation. The clocks present in the design and the according period constraints are listed in table 4.14. An uncertainty of 0.1 ns has been set by means of the `set_clock_uncertainty` command for all clocks .

clock name	purpose	period [ns]
`rx_clk0`	link clock, input link 0	3.2
`rx_clk1`	link clock, input link 1	3.2
`ext_clk`	ext clock, input to PLL	12.8
`pll_clk400`	PLL output clock	3.2
`chip_clk`	core clock	6.4
`clkhi`	time base, event dig., output links	3.2
`del_conf`	load value into *delaylines*	20
`jtag_clk`	JTAG tap controller clock	200

Table 4.14: Clock signals present in the chip with the according period constraints. The clocks to the analog part are covered by these definitions as they are traced through the inserted AND gates (cf. section 4.3.3).

The achievable speed is limited by the path with the worst negative slack. According to the STA performed on the synthesis results, this path is located in the event packet generation logic. The pop interface of the FIFO memories within the *event_buffer_out* modules is controlled by asynchronous logic within *event_buffer_out_mux*, which itself evaluates the output of *event_buffer_out* to assemble the event packets. Depending on the optimization during synthesis, this path has a length of up to 21 logic levels. It has been decided not to split up

[53] SVN, also called Subversion, is the versioning system used at the Kirchhoff Institute for Physics.

this path by means of a register stage to keep the latency of one clock cycle required for the event packet generation. A period of 6.4 ns on `chip_clk` results in a clock frequency of 156.25 MHz which matches the desired speed of the FPGA controller (cf. section 3.3.3) and is therefore determined to be sufficient.

The complete set of timing constraints is given in appendix D.2. The following constraints are noteworthy:

- No phase relation exists from the link clocks to `chip_clk` and `clkhi` due to the usage of asynchronous FIFOs. Therefore STA is disabled for all paths crossing these clock domains. The same holds true for all paths from the JTAG clock and the *delayline* configuration clock to the remaining clocks in the design.
- The majority of the FSMs present in the design accepts an asynchronous reset. Therefore, timing analysis is also disabled for all static and slow inputs, such as `RESET`, `CI_MODE` or `CHIP_ID`. The reset signal is synchronized locally by means of two flip-flops, where required.
- The *delaylines* are constrained as described in section 2.4.4.

4.4.2 Top Level Floorplan

A brief overview of the chip's floorplan has already been given in figure 4.1. Now that the components of the chip have been described, the top level floorplan shall be described in more detail by means of figure 4.19, where also the power structures are displayed.

I/O Ring The integration of the large analog VLSI arrays and the manageable number of required I/O and power bond pads make the design of the presented chip rather area limited than pad limited. Therefore, the bond pads are implemented in one single row and not staggered. The I/O ring is split into two parts: first, the top edge which solely comprises power and signal pads related to the analog part. The regular I/O cells supplied by Virtual Silicon Technology (VST) are used in this row as to save space in the die core area. Second, the remaining I/O ring is u-shaped and composed of LVDS and according power pads on the right and left edges. On the bottom edge, miscellaneous single ended signal pads, power pads, one LVDS input pad for the EXT_CLK signal and the PLL are placed in the I/O ring. The LVDS cells as well as the PLL were designed for the staggered I/O cells supplied by VST and therefore the u-shaped part of the I/O ring uses the more area consuming staggered pads. The top row is separated from surrounding corner cells by a 5 μm wide gap which thereby also acts as a power cut.

Analog Part The two *network_blocks* are placed such that the synapse drivers in the middle are opposite to the adjacent *network_block*. By this means, automatic routing of corresponding spike output signals to the inputs of the synapse drivers of adjacent blocks is facilitated. The space that has been saved at the top is mainly used for power distribution and also for signal routing. Six *neuronout_pe_top* modules are placed below the *network_blocks*. Two at a time make up the last stage of the priority encoders for one 64 neuron block. They are placed near the corresponding 64 neuron block for optimal routing. The two *dll_block_complete* blocks containing the DTCs and TDCs are placed in a manner that, on the one hand, the output of the DTCs is close to the address inputs of the synapse driver columns and on the other hand, the wires from the priority encoders' outputs are routable with acceptable lengths. The DAC is

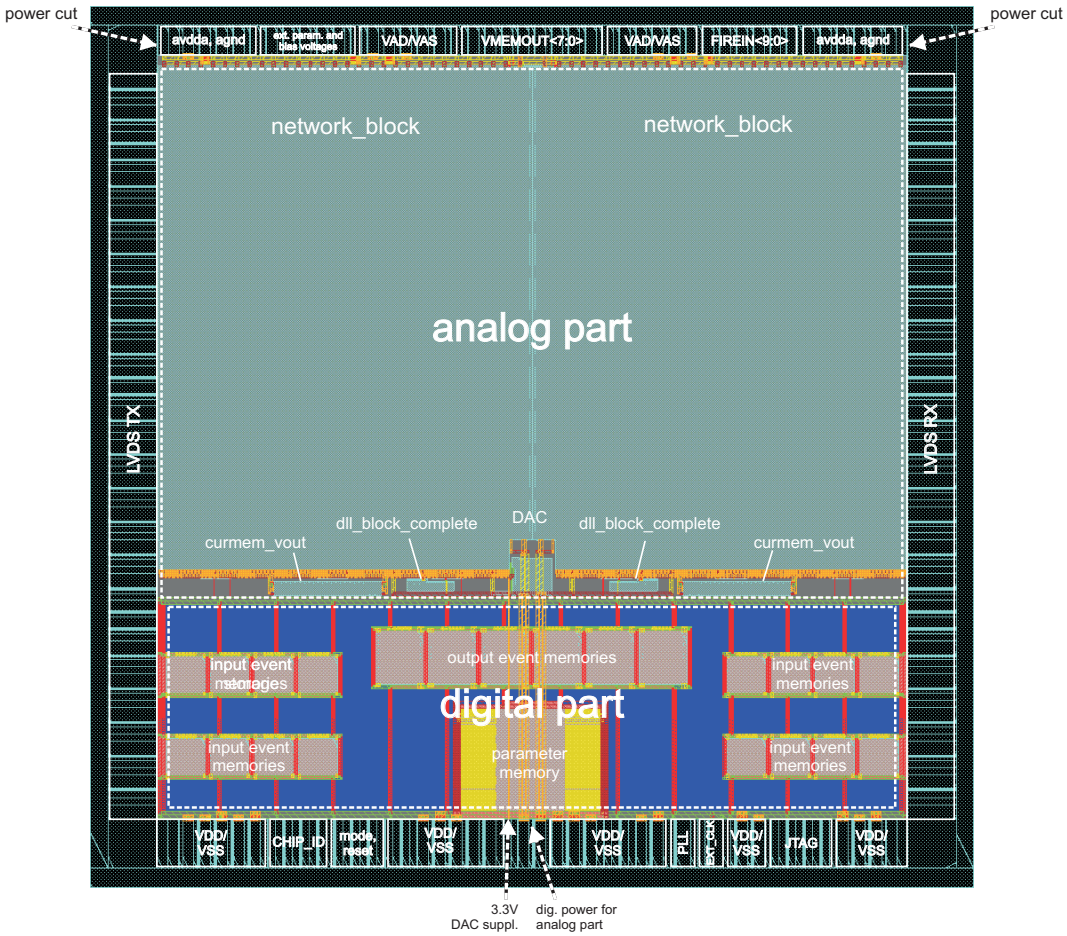


Figure 4.19: Floorplan view of the Spikey chip with power routing done.

placed at a central position, to minimize the wire capacitance of its output signals that need to be routed to all current memory inputs in parallel.

Digital Part The placement of the digital macros has been optimized iteratively and follows two objectives: first, the blocks should be placed in a regular manner to allow for straight forward power connections. Second, the timing of critical signals has been considered while leaving enough room for standard cell placement. For this reason, the memory blocks used for the storage of digitized events are placed within the upper area of the digital part. They receive their input directly from the TDCs and the system time counters within the digital part (cf. section 4.3.5). The area above these macros is needed for the buffers that drive high capacitance nets within the analog part and the registers directly related to the event input. The very presence of these registers allows for the placement of the memory blocks used for the storage of incoming events with larger distance to the DTCs, thereby leaving more space for standard cell placement throughout the core area. The static memory used for the parameter storage is not critical in terms of timing and it is placed in the bottom middle of the digital part to facilitate the access of the application layer logic to its ports.

Net name	Purpose	Supplied from
avdda	1.8 V analog power	top edge
avddhi	3.3 V analog DAC supply	bottom edge
VAD	1.8 V I/O power for 50 Ω outputs	top edge
VDD	1.8 V digital power	bottom edge
VDDL	1.8 V LVDS transmit power	left edge
VDDR	1.8 V LVDS receive power	right edge
V3IO	3.3 V IO power	left, right, bottom edge

Table 4.15: Power nets that are present on the chip.

4.4.3 Estimated Power Consumption and Power Plan

Seven different power domains are present on the chip. They are listed in table 4.15 and the names correspond to the pin names in the top level layout. VAD and avdda are confined to the top edge and are isolated by the above mentioned power cuts. VDDL and VDDR are confined to the LVDS pads within the according edges. They are separated from the core power ring by means of cuts near the bottom corner pads. V3IO is the 3.3 V supply for the I/O cells and electrostatic discharge (ESD) protection structures. It is common for the whole u-shaped I/O ring.

The skeleton of the power structure (stripes and rings) is generated by a script to set up initial anchor points for the following automatic power routing.

Analog Part Given a worst case dynamic power consumption for one *network_block* of $P_{\text{nb,worst}} = 300\,\text{mW}$ according to [Scha] and a worst case quiescent current within the parameter voltage output buffers of $I_{\text{q,worst}} = 1.2\,\text{mA}$, the analog power consumption can be estimated as[54]

$$P_{\text{ana,worst}} = 2 \cdot P_{\text{nb,worst}} + 2 \cdot V_{\text{DD}} \cdot I_{\text{q,worst}} \simeq 700\,\text{mW and} \tag{4.3}$$

$$I_{\text{ana,worst}} \simeq 390\,\text{mA} . \tag{4.4}$$

Note that the power drawn by *dll_block_complete*, the DAC and the auxiliary priority encoder modules is thereby neglected.

Above the *network_blocks*, two 20 μm wide stripes are inserted in order to connect to the power and ground pads of avdda and analog ground respectively, and the power pins of the *network_blocks*. Six pairs of power pads supply these stripes and they are placed in a way that the current flowing in either direction of one stripe equals approximately $I_{\text{stripe}} = 1/8 \cdot I_{\text{ana,worst}}$. With a maximum allowable DC current of $I_{\text{max,dc}} = 3.25\,\text{mA}/\mu\text{m}$[55] for the metal stripes, the worst case power can be sustained by this network.

At the bottom edge of the *network_blocks*, the analog power and ground outlets are collected by means of two additional stripes. The analog power supply for the *vout_blocks* and the other blocks located there is drawn from these stripes. Two power nets for the analog part are supplied from the bottom edge of the chip as it is not possible to route them through the *network_blocks*:

[54] $P_{\text{nb,worst}}$ is assumed for a reasonable setup of the synapse array. In the worst case, given pathologic values for the leakage conductances, the synapse weights and the differences between the different reversal potentials, the current could drastically increase to values even damaging the bond wires.

[55] This value is given for a die temperature of 80°C. Assumed an adequate cooling, the die will in no case heat up above this value.

Component	#	power [mW]	total power [mW]
gate equivalent	83191	0.00628	417
event_in_ram	16	4.3	69
event_out_ram	8	9.9	79
parameter_ram	1	3.3	3.3
Total power P_{core} [mW]			568.3

Table 4.16: Estimated core power consumption.

- `VDD`/`VSS`: One digital power/ground pad pair is reserved for the supply of the digital circuits within the analog part. `VDD` and `VSS` are distributed below the *network_blocks* in the same way as the analog stripes and are connected to the power pins of the neuron circuits. The connection between the pads and the stripes is realized on metal 6, only. Thereby, they are not connected to power structures within the digital part to avoid coupling of switching noise into this supply.

- `avddhi`: The 3.3 V supply of the DAC is implemented with a separate power pad in the bottom row.

Digital Part The digital power consumption is estimated by determining the gate count of the design by means of an initial First Encounter run and calculating the power dissipation P_{diss} using the following formula [Vir04d]:

$$P_{\text{diss}} = ((E_{\text{rise}} + fall_{\text{E}} + C_{\text{fanout}} \cdot V^2) \cdot f_{\text{sw}} \cdot p_{\text{sw}} + P_{\text{stat}}) \cdot n_{\text{gate}} \ , \tag{4.5}$$

where E is the energy for a rise or fall transition, C_{fanout} is the capacitive load at the output, V is the supply voltage, f_{sw} is the switching frequency, p_{sw} is the toggle probability, P_{stat} is the static power dissipation and n_{gate} is the number of gates. The gate count of the design is reported by First Encounter as a multiple of the smallest standard cell (gate equivalent count). The values used refer to this smallest cell (HDBUFBDL). Conservatively, an average load capacity of $C_{\text{fanout}} = 0.02\,\text{pF}$ and an average toggle probability of $p_{\text{sw}} = 0.2$ are assumed[56].

The power consumption of the static memory cells has been estimated using worst case values from the according data sheets and a toggle rate of 16 % for the event rams and 5 % for the parameter ram. The resulting values for a core frequency of $f_{\text{sw}} = 200\,\text{MHz}$ are summarized in table 4.16.

Experience at the Kirchhoff Institute proves that one bond wire tolerates a maximum current of 100 mA [Ach]. Nine power and ground pads supply the core with `VDD` and `VSS` from the bottom edge, which is sufficient for the expected power consumption. Power is distributed over the core area by means of a surrounding metal ring (width: 20 μm) and 12 vertical stripes (width: 10 μm) on metal 2. In a bad scenario (highest standard cell density at the top of the core), it is assumed that half of the estimated current will flow through the whole stripe. The voltage drop for one stripe is calculated using the current through one stripe

$$I_{\text{stripe}} = (0.5 \cdot P_{\text{core}}/1.8\,\text{V})/14 = 11.5\,\text{mA}$$

[56]The low value for p_{sw} is based on the nature of the communication protocol. Only one of the core modules will be active at a time, or in the case of events, three of the 16 input buffers.

	one cell [mW]	20 cells [mW]
dynamic power	1.6	32
static power	6.3	126
total power		158

Table 4.17: Estimated LVDS transmit pad power consumption.

and the stripe resistance (sheet resistance of metal 2: $R_\square = 0.062\,\mathrm{m\Omega/\square}$)[57]

$$R_{\mathrm{stripe}} = \frac{1350}{10} \cdot R_\square = 8.37\,\Omega \ :$$

$$V_{\mathrm{drop}} = I_{\mathrm{stripe}} \cdot R_{\mathrm{stripe}} = 96\,\mathrm{mV} \ .$$

At an early design stage, this has been chosen to be sufficient to supply the digital core of the Spikey chip. Transient loads are buffered by the capacitances within filler cells that are automatically inserted after routing is complete.

I/O power The power consumed by the I/O circuitry is dominated by the LVDS transmit pads. The input pads do only marginally contribute and they are neglected as well as the static single ended pins. The static power consumption of the transmit pads is calculated with the LVDS quiescent current $I_{\mathrm{q,tx}} = 3.5\,\mathrm{mA}$ (see [ANS96]) and results in $P_{\mathrm{q,tx}} = 6.3\,\mathrm{mW}$ per pad neglecting other currents in the cell. Dynamic power consumption with full toggling rate is calculated using the following equation [TMG02]:

$$P_{\mathrm{dyn,tx}} = C_{\mathrm{tot}} \cdot V^2 \cdot f_{\mathrm{sw}} \ , \tag{4.6}$$

with C_{tot} being the driven capacitance, V the switched voltage and f_{sw} the switching frequency. C_{tot} is estimated to be 25 pF including line capacitance and the receiver pin capacitance, $f_{\mathrm{sw}} = 400\,\mathrm{MHz}$ (the link clock frequency) and $V = 400\,\mathrm{mV}$ differential signal swing. The resulting power dissipation is given in table 4.17. The transmit side is supplied by three pairs of power and ground pads equally distributed among the total number of pads.

4.4.4 Timing Closure

In this section, relevant results of the timing closure of the final design are given. These results affect the maximum core operating frequency and thereby the maximum event rate achievable on the digital interface of the chip.

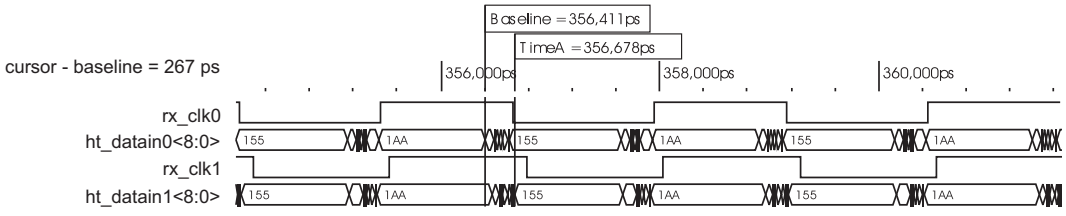


Figure 4.20: Back annotated simulation of the timing at the input DDR registers.

[57] The sheet resistance $R_\square$ is given in Ohms per square, which is indicated by the square $\square$ symbol.

Interface Implementation: Input The fulfillment of the timing requirements on the physical inputs and outputs of the chip cannot be verified by means of STA (cf. section 2.4.4). Therefore, the interface timing is verified with a back annotated simulation of the chip. The timing at the clock and data pins of the DDR input registers is shown in figure 4.20. The clock frequency is 400 MHz and data is sent to the chip with the ideal phase relation of 90°. This simulation has been performed for the typical mean process corner and all *delaylines* are set to default delay. The skew on the data signals equals 267 ps and can easily be compensated by tuning the according *delaylines*. The arrival times of the clock signals differ by about 90 ps and the clocks arrive almost simultaneously with the data. As a result, the data would not be registered correctly at the input. This is a serious implication which either requires the connected sender to change its interface timing or an additional delay of the clock signals on the carrier PCB.

The cause of this erroneous implementation are the minimum and maximum delay values that have to be defined for the interface clock tree constraints. These have to be set with a large gap from $t_{\text{min}} = 1500$ ps to $t_{\text{max}} = 2300$ ps as the CTS software otherwise fails to build the clock tree[58]. As the link clocks and the data signals are defined within different clock groups (cf. section 2.4.4), the implementation of equation 2.1 likely fails.

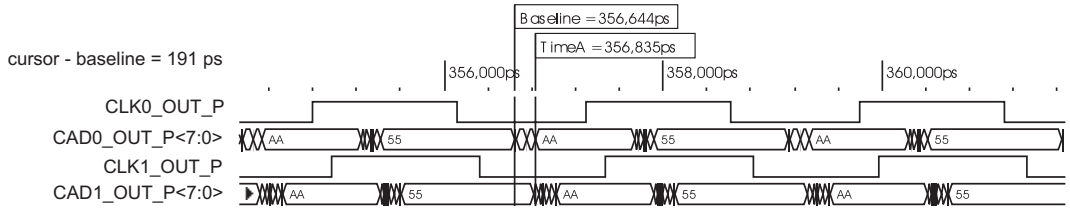


Figure 4.21: Back annotated simulation of the timing at the output pins of the chip.

Interface Implementation: Output The timing of the output signals is shown at the physical pads of the chip in figure 4.21 with the same setup as for the input. In this case, clock and data signals of one link belong to one clock group and it can be seen that the expected $-90°$ phase relation is well fulfilled for both interfaces. The skew among the data signals of one link equals about 191 ps which can again be compensated by tuning the *delaylines*. The time shift between the two links equals about 200 ps. This is of no concern due to the source synchronous nature of the interface. The fact that the compensation of internal skew only requires the usage of one delay element on both input and output interface shows that the interface is tunable to compensate externally introduced skew in the order of magnitude of 200 ps per signal and still provides a maximum data valid window opening.

Static Timing Analysis The critical path identified during synthesis is still the limiting factor for the digital back end implementation. Timing closure has been achieved in the typical mean process corner and the best case process corner for the specified clock frequencies of 156 MHz and 312 MHz respectively. The slack values for the different clock domains are listed in table 4.18. Despite an optimized placement of all FIFO memory cells, the worst case slack is negative. A closer look at the violating paths shows that besides the known critical path, synchronized reset signals within the *clkhi* domain have negative slack. To improve

[58] This is at least true for the version 5.2 of First Encounter which was used for the design.

	$s_{\text{chip\_clk}}$ [ns]	s_{clkhi} [ns]	$s_{\text{rx\_clk0/1}}$ [ns]	max core frequency [MHz]
worst case	−0.71	−0.31	−0.25	145
typical case	0.91	0.63	0.39	180
best case	1.48	1.01	0.62	211

Table 4.18: Worst slack values for the different clock domains on the chip. The values are obtained from the post-layout STA and correspond to a target period of $T_{\text{clkhi}} = 3.1$ ns and $T_{\text{chip\_clk}} = 6.2$ ns. The expected maximum core operating frequency of the chip is given in the last column.

their slack, they should be treated as clock trees constrained to a delay that is smaller than the `clkhi` period T_{clkhi}. However, as this increases the required buffer area and does not reduce the worst negative slack, this solution has not been realized.

A consequence of the reduced maximum core frequency is a reduced accuracy of the event digitization and generation. In the worst case process corner the resolution is 216 ps (equal to 21.6 μs biological time) instead of the originally specified 156 ps. This results in a quantization error of 4 % for the shortest expected axonal delays of 0.5 ms (biological time). This error is chosen to be acceptable as it lies well below the maximum variability of 0.25 ms of spike times required for stable neural network activity. The latter values have been obtained in in vivo measurements [MS95].

Furthermore, the event transmission delay is increased due to the increased clock period of the physical communication with the chip. The scenario with a clock frequency of 156 MHz has already been discussed in section 3.2.4. It is not expected to get back a true worst case sample since the fabrication process is already used for quite a long time (about 5 years) and thus is supposed to be well elaborated.

4.5 Improvements of the Second Version

The first version of the Spikey chip has been taped out in August 2005. Its functionality has been proven by a large series of tests that have been performed on the interface, the digital part and the analog circuits. Some of the results presented in chapter 6 have been obtained with this first version. Nevertheless, there were some shortcomings regarding wiring and the digital logic that justified a second tape out including the corrections and improvements described in this section.

Parameter Voltage Generation In the first version, the parameter voltages are generated by means of a switched capacitor circuit. The voltage is generated by charging a capacitor for a certain period of time with the constant current of one current memory cell. The required control lines and the voltage on the switched capacitor itself introduce crosstalk to the current memory cell that leads to a random error of approximately ± 150 mV on the parameter voltages. The current memory is replaced by a 10 kΩ poly silicon resistor in the second version and the control lines are removed. Both circuits are described in more detail and characterized in the diploma thesis of B. Ostendorf [Ost07]. Thereby, the successful redesign is proven and the random error does not occur anymore.

Parameter Voltage Readout The capability to read out every voltage parameter did not exist in the first version. Instead, probe pads connected to one dedicated voltage are placed on the die that are used for characterization.

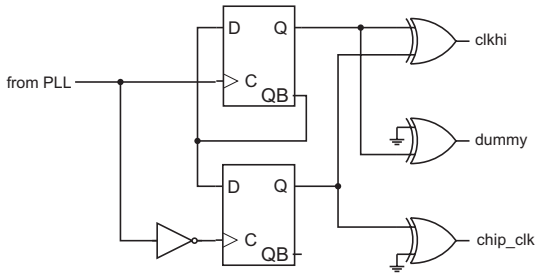


Figure 4.22: Clock generation circuitry used within Spikey 1. The XOR connected to `dummy` ensures symmetrically loaded flip-flop outputs.

Internal Clock Generation Within Spikey 1, the clocks `chip_clk` and `clkhi` are generated using the circuit shown in figure 4.22. The flip-flops are symmetrically loaded to achieve an equal propagation delay through the XOR gates. Differences in the gate propagation delays for rising and falling input transients result in two period values differing by 250 ps for the `clkhi` output depending on the input of the according XOR. This behavior could be reproduced in the back annotated simulation of the design (data not shown). As a result it can be said that the circuit is not suited to generate two clocks with a speed difference of a factor of 2, if a clock period smaller than 10 ns is desired. Instead, the circuit described in section 4.3.3 is used within the second version.

The Event Loopback Module This module was added to the design in the second version. Using Spikey 1, it is only possible to verify the digital event generation and digitization by means of comparing the expected results to a selected membrane voltage on an oscilloscope.

Besides these modifications and additional features, the experiences collected during the design of Spikey 1 were of great benefit for the design of Spikey 2. Especially the automated abstract generation and routing of the analog part have been improved and now gain more predictable results in terms of general routability and crosstalk.

Chapter 5

Operating Environment

This chapter describes the hardware, programmable logic, and software setup that is designed for the test and the operation of the neural network ASIC and has been developed throughout this thesis. An overview of the hardware platform has already been given in chapter 3. Additional hardware hosting the Spikey chip is shown. The programmable logic design of the FPGA acting as a controller to the chip is described together with the implementation of the communication protocol described in chapter 4 followed by a description of the control software which provides the low level access to the chip as well as an abstraction layer for the high level software access. The system is suited for the test and the operation of a single chip and it is shown how the modules needed for the communication of events among several of these can be integrated into this framework.

5.1 Hardware Platform

5.1.1 System Overview

The hardware platform was introduced in chapter 3 with an emphasis on the transport network which facilitates communication between different Nathan modules hosted by a backplane. Figure 5.1 shows a photograph of this platform with two Nathan modules plugged into the backplane. Each Nathan module carries a daughter card that hosts one Spikey chip and provides the necessary infrastructure like a DAC, an analog-to-digital converter (ADC) and voltage regulators for the operation of the chip.

The Nathan Module (Revised)

The Nathan module has a size of $13 \times 8\,\mathrm{cm}^2$ and is fabricated in an eight layer build-up MicroVia process with a minimum feature size of $100\,\mu\mathrm{m}$. It is designed as a general-purpose platform for the distributed operation of mixed-signal ANN ASICs. All required components for the local execution of experiments as well as the interconnection of several modules are

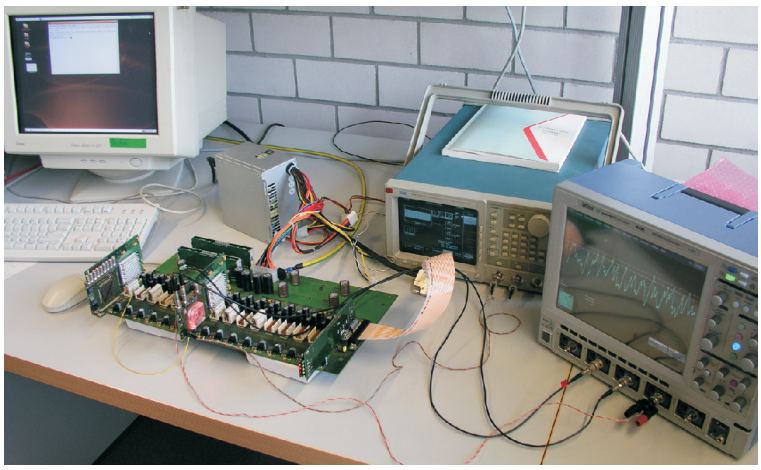


Figure 5.1: The setup currently used for the test and operation of the Spikey chip. Two Nathan modules, each hosting one chip, are plugged into the backplane, which establishes the connection to the control PC. A waveform generator is used for the generation of clock signals with variable frequencies. The oscilloscope on the right is used to record analog output signals.

available on the Nathan module. Figure 5.2 shows a photograph of the top side of the Nathan module. The left half is covered by the Recha PCB hosting the Spikey chip (see section 5.1.2). It is plugged into the according SMT connectors on the Nathan module. A block diagram illustrating the functionality of the Nathan module has been shown in figure 3.2. The design of the Nathan module is described in detail in [Grü03]. In the following, the facts relevant for the operating environment of the Spikey chip are shortly outlined.

FPGA Each Nathan module contains a Virtex II-pro FPGA of type XC2VP7 as the central element to the module [Xil02b]. It provides 11,088 logic cells, a PowerPC processor core, 396 configurable user I/O pins, eight MGTs and four digital clock managers (DCMs), which can be used for clock frequency multiplication and division, and for clock phase shifting. The PowerPC is not used within this setup. However, a Linux kernel has been ported to run on this processor [Sin01], thereby in principle enabling the local execution of neural network experiments including the necessary control software [Sch05].

Memory Two types of memory resources are available on the Nathan module. Two SRAM chips [Int00] with an access latency of two clock cycles are directly soldered on the board allowing the parallel access to 2×512 KB of memory over a 2×32 bit bus at up to 200 MHz. For applications requiring larger memory resources, a socket for one DDR-SDRAM module is available on the back side of the module[59]. The interface is implemented such that memory modules with a capacity of up to 2 GB are addressable.

Analog Circuitry One four-channel 10 bit DAC with a serial interface is available on the Nathan module [Max96]. The serial interface operates at clock frequencies in the order of magnitude of 1 MHz and the DAC is thus not suited to produce fast transients, but rather to supply constant bias voltages to the neural network ASIC. It operates from a 3.3 V dedicated

[59] The socket physically fits to standard 200 pin small outline dual inline memory modules (SODIMM), which are commonly used in laptop computers [Mic02].

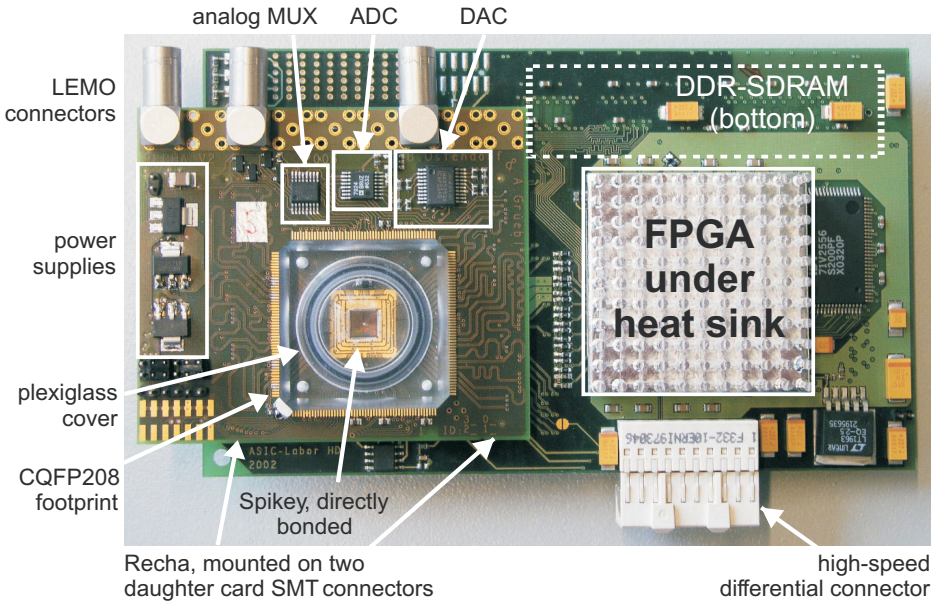


Figure 5.2: Photograph of the Nathan module with the attached Recha carrier board hosting the Spikey chip. The Recha board contains additional active analog circuitry, which is not available on the Nathan module. The leftmost LEMO connector is connected to the `IBTEST` pin of the Spikey chip. Two membrane potential monitor output pins are equipped with the connectors.

power supply and a reference voltage of 1.25 V. Three of its buffered outputs are connected such that they can be set from 0 V to 3.3 V with 10 bit resolution and the fourth output controls a current sink [Grü03]. Furthermore, a temperature sensor is available, which is placed inside of the ANN ASIC socket. It is capable of measuring the FPGA temperature and its own temperature. The local temperature may be used as a clue for the neural network ASIC temperature in thermal equilibrium.

Connectivity The Nathan module is connected to a Slow-Control token ring network (cf. section 5.2.1) and to the MGT network on the backplane by means of a high-speed differential connector. Regarding on-board connectivity, about 2/3 of the I/O pins available at the FPGA are used for the memory connections. The remaining I/O signals are available for the interface to the ANN ASIC and are completely routed to the SMT connectors while a fraction is connected to a local ANN ASIC socket (which is not used for this setup) in parallel. All signals available at the SMT connectors are routed pairwise differentially on the Nathan module with a differential impedance of $Z_{\text{diff}} = 100\,\Omega$, thereby supporting the implementation of high-speed digital interfaces using the LVDS signaling standard. Altogether, the following signals are available at the SMT connectors:

- Overall 64 differential pairs. Besides some unassigned pairs, these differential pairs are grouped into five busses comprising nine pairs each that are routed with equal lengths to the connectors to support the implementation of 8 bit data links plus clock signal.

- The three DAC output voltages and the input to the current sink are available at the connectors. Furthermore, the serial interface of the DAC is forwarded to the connectors to connect further DACs in a daisy chain topology.

- Two dedicated 3.3 V low-dropout linear voltage regulators [Nat01] on the backplane exclusively supply analog and digital supply voltages to the ANN ASIC and these voltages are available at the connectors. The regulators are capable of delivering up to 1.5 A of supply current.

5.1.2 The Recha PCB

The Recha PCB serves as a carrier board for one Spikey chip. Two revisions have been developed: the first version implements all basic functionality for the testing and operation of the chip. The second version (will be called V2) was a conjoint work with B. Ostendorf who added analog readout functionality. The generic functionality of both versions is identical and as the first version is deprecated, the following text refers to the V2 version. A photograph of Recha V2 attached to the Nathan module is shown in figure 5.2. The Spikey chip can be mounted on the board in two ways: a footprint for a 208-pin ceramic quad flat pack package is available to solder a packaged version of the chip onto the board. Furthermore, all signals connected to the package pins are continued to the center of the board, where the chip may be directly wire bonded onto the board (cf. figure 5.3). The data flow direction through the chip is from left to right. This is reflected by the input links to the chip being connected via the SMT connector on the right, and the signals of the output links being routed to the SMT connector on the left.

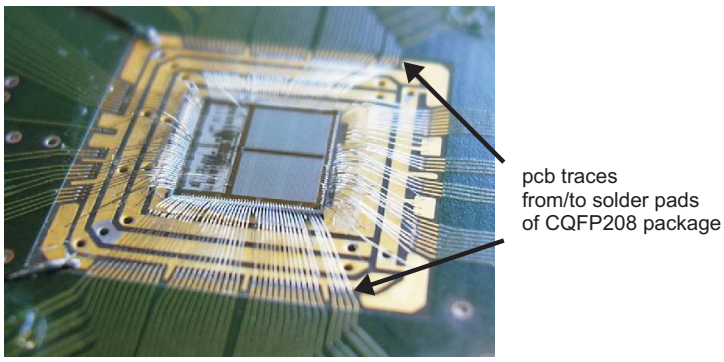


Figure 5.3: Photograph of the Spikey chip being directly bonded on the Recha carrier board.

Features and Functionality

Detailed schematics as well as the layout drawings of the Recha board can be found in appendix F, while figure 5.2 shows a photograph of the top view. The mapping of the Spikey pins to the signal names used on Nathan is given in appendix C.2. Relevant functionality of the Recha board will be described in the following.

Four different power domains are present on the board. The two 3.3 V supplies delivered by the Nathan module are used for the digital I/O power supply of the Spikey chip and for the (analog) supply of the Spikey-internal DAC respectively (cf. section 4.2.1). Two dedicated low-dropout linear voltage regulators generate the separate 1.8 V supply voltages for the analog and the digital part of the Spikey chip. The 3.3 V analog supply is furthermore used to supply the active analog components present on the board. These include:

- A four-channel 10 bit DAC which is identical to the DAC on the Nathan module. On the one hand, this DAC is required to provide the reference voltage V_{casdac} to the Spikey-internal DAC. On the other hand it generates the analog parameter voltages V_{M}, V_{REST} and V_{START}. These parameter voltages require low-impedance, buffered outputs at voltages below 500 mV and could not be generated on-chip (cf. section 4.2.1). The DAC uses the same reference voltage V_{ref}=1.25 V as the DAC on Nathan and due to the identical interfaces it is possible to keep the existing VHDL code for the DAC control with minor modifications to support the operation of two devices[60]. In addition to the four buffered voltage outputs, each buffer's negative input is available to allow for specific gain adjustments. The according circuit is shown in figure 5.4 a, and the output voltage can easily be calculated as

$$V_{\text{out}} = V_{\text{ref}} \cdot (1 + \frac{R_1}{R_2}) \; . \tag{5.1}$$

With $R_1 = 43\,\text{k}\Omega$ and $R_2 = 100\,\text{k}\Omega$, identical to all four DACs, the outputs can be set from 0 V to 1.79 V which covers the full range of the connected Spikey inputs.

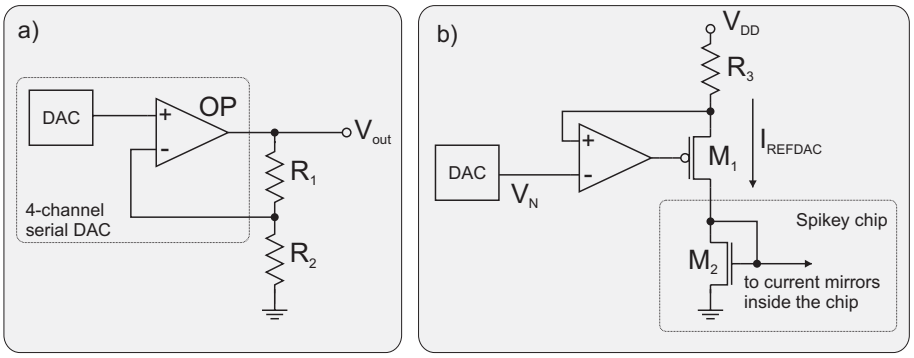


Figure 5.4: a) non-inverting amplifier configuration used at the output of the DAC to adjust the output voltage swing. b) schematic diagram of the current source circuit generating the reference curren I_{refdac} for the Spikey chip.

- A current source to provide the reference current I_{refdac} to the internal DAC of the Spikey chip. The circuit is shown in figure 5.4 b. The OP is controlled by the DAC located on Nathan, which can be set from $V_{\text{N}} = 0 \sim 3.3\,\text{V}$. It adjusts its output such that the voltage at the node connected to its negative input equals V_{N} and consequently I_{refdac} writes

$$I_{\text{refdac}} = \frac{V_{\text{DD}} - V_{\text{N}}}{R_3} \; . \tag{5.2}$$

The maximum achievable current depends on the threshold voltage V_{th} of the transistor M_2, the on resistance R_{on} of M_1 and the value of R_3 and can be calculated as

$$I_{\text{refdac,max}} = \frac{V_{\text{DD}} - V_{\text{th}}}{R_3 + R_{\text{on}}} \; . \tag{5.3}$$

With $R_3 = 5.1\,\text{k}\Omega$, $R_{\text{on}} = 50\,\Omega$ (conservative estimate based on the output characteristics of the transistor) and V_{th} =0.51 V [UMC03], this sums up to $I_{\text{refdac,max}} = 25.0\,\mu\text{A}$ and consequently a maximum parameter current of 2.5 μA on the Spikey chip.

[60] The serial interfaces of the DACs are wired together in a daisy chain. The implemented VHDL code follows the MicroWire protocol; a description of this protocol can be found in [Max96].

- A four-channel 10 bit serial ADC to facilitate the automatic monitoring of selected output pins of the Spikey chip. Three of the ADC inputs are connected to the `IBTEST` pin of the Spikey chip and the first membrane potential monitor output of each *network_block* respectively, thus enabling the automated readout of all on-chip parameter voltages (via IBTEST) and the membrane voltages of 2×48 neurons located on each of the *network_blocks*. In particular, this feature is used for the calibration of the different parameter voltage generators present on the chip [Ost07].

Other features include an analog multiplexer which either selects the `IBTEST` pin or one of the eight membrane potential outputs to one common output for global monitoring purposes, a set of four jumpers to set the `CHIP_ID` pins of the Spikey chip, and a differential clock input, which is connected to a global clock input of the FPGA via the SMT connectors.

Layout Details

The high-speed digital signals of the digital interface to the chip require a careful board layout. On the one hand, it is necessary to layout the differential traces with a differential impedance of $100\,\Omega$ to avoid impedance discontinuities, which would cause signal reflections, and thus a degraded signal integrity [JG93]. On the other hand, it is necessary to compensate for the differing routing delays between the signals of each link. Both requirements are fulfilled by means of the layout tool[61] that has been used.

Nevertheless, the electrical parameters of the PCB substrate and the geometries required to achieve the differential impedance of $100\,\Omega$ have been obtained in cooperation with the manufacturer [wue] and are listed in appendix F.3. To achieve this controlled impedance on the whole area, the stack-up of the Recha board has been chosen such that the two internal layers are realized as massive copper planes used for power and ground network distribution. On the one hand, this ensures a low-inductance power network, and on the other hand, the continuous copper planes serve as reference planes to the differential microstrip lines on the top and the bottom side of the Recha PCB.

The routing of the differential signals has been realized with the aid of board- and system level simulations that are available within the layout tool[62]. The setup of these simulations has been described in [Grü03] for the DDR-SDRAM bus on the Nathan PCB. They allow to interactively check the propagation delays of the signals during manual routing. As a result, the delay of every single signal could be optimally set by its routing length, which is noticeable in the meander-like routing of many of the signals (cf. figure 5.2).

The membrane monitor outputs require a controlled single-ended trace impedance of $50\,\Omega$ to match the output impedance of Spikey's on-chip drivers. These are routed on the bottom side of Recha and are connected to eight LEMO connectors at the top edge of the board. The ninth Lemo connector on the left hand side of the board is connected to the `IBTEST` signal.

[61] Cadence Allegro PCB Designer suite in conjunction with the SpecctraQuest signal integrity suite have been used [SQ 04].

[62] IBIS (Input/Output Buffer Information Specification) models are required for these simulations. While these models are available from the manufacturer for the I/O drivers/receivers of the FPGA, generic models provided by the software have been used to model the LVDS pads in the Spikey chip. As the simulation is mainly targeted at the signal propagation delay and only secondary on the signal integrity, this has been chosen to be sufficient.

5.2 Programmable Logic Design

During the design of the hardware platform consisting of the backplane and the Nathan modules, a modular approach has been followed which allows for the distributed operation of ANN ASICs of different types [Sch05, Grü03]. Central to this concept is the FPGA, which can be configured to control all connected components. To provide reusable components and a common infrastructure to all functional modules realized in the programmable logic of the FPGA, a modular framework has been developed in [Sch06] and [Phi07]. Within the programmable logic design, three communication channels are of importance for the operation of the whole system: the local memory access, the external communication via the token ring network on the backplane, and the high-speed communication via the MGTs of the FPGA. For the operation of the Spikey chip, the components for the external communication and for the memory access are used and the following section shortly describes these entities, namely the *memory controller* and the *Slow-Control*.

5.2.1 Overview

To illustrate the topology of the existing framework and the integration of the programmable logic needed for the Spikey chip into this framework, an overview on the logical components present on one network module is given in figure 5.5. The modules *spikey_control* and *spikey_sei* encapsulate all functionality related to the chip and process both event and control interface data. Remote access to the software is established via the Slow-Control while the local DDR-SDRAM severs as a mass storage device for experimental and configuration data. The MGT connections required for inter-module communication are omitted for clarity. They are not used for testing and single chip experiments.

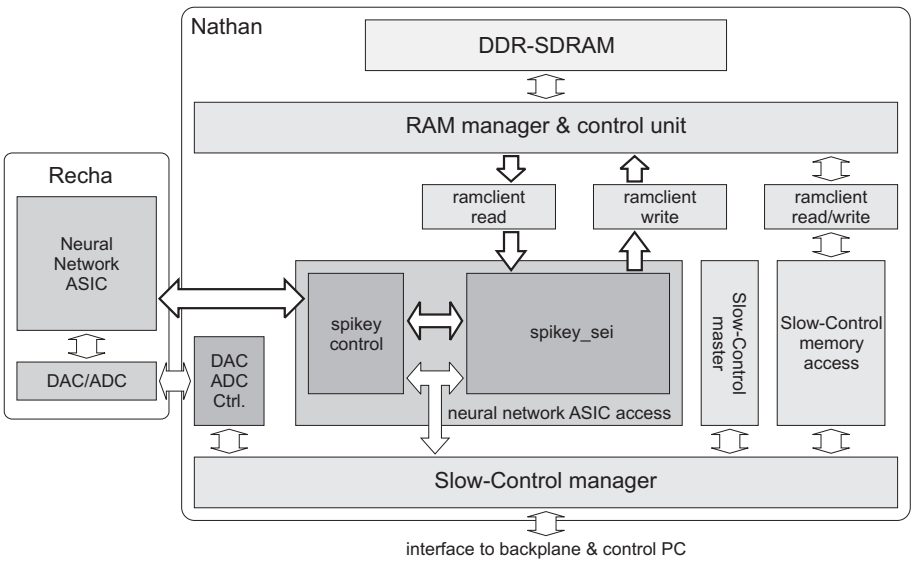


Figure 5.5: Block diagram of the functional modules within the programmable logic design. The light grey boxes denote components of the already existing framework. The modules needed for the neural network ASIC access have been developed within this thesis in collaboration with Dr. J. Schemmel. The bold arrows denote communication channels that are relevant to the communication of data packets with the neural network ASIC.

Memory Access: The Memory Control

Several components within the FPGA may simultaneously require access to the local memory. These components will be called *ramuser* in the following. The memory control realizes the interface between the DDR-SDRAM module and all ramusers inside the FPGA. It has been developed by T. Schmitz [Sch06] within the Electronic Vision(s) group and is comprised of three independent modules:

- The *ramclient* provides an interface with a common protocol to all ramusers. A ramclient buffers requests issued by the ramuser within a FIFO memory of parameterizable depth and forwards them to the manager.
- The *manager* receives requests by the ramclients and grants access to specific ramclients based upon a priority scheme. Requests are forwarded to the memory control unit.
- The *memory control unit* encapsulates the physical interface to the DDR-SDRAM and implements the according access protocol.

Relevant to the ramuser is that the memory controller does not guarantee fixed response latencies. The ramuser issues write requests to the ramclient which buffers the request and forwards it to the manager without further notice to the ramuser. In case of a read request, the ramuser receives the requested data an undefined number of cycles after the actual request. Read data is delivered in the order it was requested. The access latencies strongly depend on the number of ramusers in the system and the traffic that they produce—the overall bandwidth of the DDR-SDRAM interface is shared among all ramclients.

Interface Specification and Clocking The interface to the ramclient consists of a 64 bit data bus, an according address bus and the necessary handshake signals. To accommodate different clock speeds, the ramclient can be configured to use asynchronous FIFO control logic. On this account, the clock frequency of the ramuser can be selected independently from the memory clock. The memory has been verified to work with frequencies of up to 140 MHz (280 MHz DDR). For a concise description, performance measures and the resource consumption of the memory controller the reader may refer to [Sch06].

Register and Control Access: Slow-Control

The Slow-Control is originally developed to allow for backdoor and configuration access to the functional modules. It is organized similar to the memory control: several clients can be connected to a central manager that, on the one hand, distributes and collects the clients' requests (cf. figure 5.5) and on the other hand, encapsulates the physical interface to the token ring network on the backplane. One Slow-Control master, which is capable of initiating transfers to the connected clients and to the token ring network, can be instantiated per module.

Implementation Each Slow-Control client has a 32 bit address space with a data width of 32 bit. The serial interface to the control PC can be operated with frequencies of about 40–80 MHz and the protocol is kept simple to limit the resource consumption of the Slow-Control components. The achievable bandwidth for an atomic access from the control PC to the Nathan module at a clock frequency of 78 MHz is 1.38 Mbyte/s for write accesses and reaches 0.83 Mbyte/s for read accesses. Data on the resource consumption of implementations with different numbers of clients can be found in [Sch06] and a final description of the functionality will be given in [Phi07].

Slow-Control Clients Three Slow-Control modules shall be described, that are of importance to the infrastructure inside the FPGA and are displayed in figure 5.5. One client is used to connect the DDR-SDRAM to the Slow-Control. By accessing this client, the control PC can access the external memory module on the Nathan module. This path will be used to transfer experimental data to the Nathan modules and read out experimental results stored there.

Both the DACs and the ADC, and the temperature sensor are connected to a common interface and are accessed by one Slow-Control client to allow for convenient access via the control PC.

The Slow-Control master is not part of the physical FPGA implementation, but is rather implemented as a behavioral module to initiate Slow-Control transfers during simulation of the control logic. The behavioral description of this module allows to read and write test vectors from and to text files, thereby enabling the simulation of the whole system—including the Slow-Control access from a control PC and the full functionality of the Spikey chip (cf. section 5.3).

Clock Generation Resources

The four DCMs provided by the FPGA are of importance for the internal clock generation and for the synchronization with the neural network ASIC and the memory module. For reasons of simplicity, the clocking is not graphically illustrated but the partitioning of the DCMs is nevertheless of interest for the further reading:

- DCM1 receives an external clock signal as an input. It derives all required clock signals for the Slow-Control, the memory controller and the Spikey controller from this clock.
- DCM2 is reserved for the synchronization of the external DDR-SDRAM to the internal clocks [Sch06].
- DCM3 is used to introduce the required phase shift on the link clocks to the Spikey chip.
- DCM4 generates the external clock to the PLL of the Spikey chip.

Two external clock sources are available and can be selected as an input to DCM1: first, the 156 MHz clock signal, which is globally available on the backplane. Second, a clock signal that is fed to the FPGA from the connectors available on the Recha board. Clock signals can be injected by an oscillator facilitating the verification of the chip for different clock frequencies.

5.2.2 Transfer Models and Organization of the Data Paths

The basic setup for the communication with the presented chip is illustrated in figure 5.5. The Spikey chip is hosted by the Recha board and is connected to the controller and the DAC and ADC respectively. The access to the Spikey chip is performed by two modules: first, the *spikey_control* module, which accounts for the low-level hardware access and contains the functionality of the physical layer and the link layer of the communication protocol. Second, the *spikey_sei* module acting as a data source and sink to the application layer functionality. For this reason, the application layer functionality is distributed among both modules. Thereby, no data sources or sinks are present within *spikey_control* but it rather implements two different transfer models for the control interface data, merges event data streams with control interface data (cf. chapter 3) and contains the required functionality for the synchronization of the Spikey chip with a local system time counter. The *spikey_sei* module serves

both as a source and a sink of data to *spikey_control* and it processes data depending on the selected transfer model, which is described in the following.

Transfer Models

The term "transfer model" refers to the way, in which control interface data is communicated with the chip. To clarify the functionality, the communication protocol is shortly recapitulated (cf. section 4.3.2): each control interface access is directly acknowledged by the chip with an according packet. In case of a read access, this is the result of the previous read command and in case of a write command, this is an empty event packet. If the command execution was unsuccessful, because the command decoder is busy executing the preceding command, an error packet is sent.

Two types of transfer models are distinguished by the controller: non-posted and posted requests.

- Non-posted requests: Data is sent to the chip with no instantaneous verification of the success of the different commands. As the answer of the chip is not evaluated, the data source has to generate the data packets (commands) within the correct order and with appropriate intervals according to the time required for the command execution within the chip. Furthermore, the controller assumes an always-ready data sink where the answer packets of the chip can be stored. This transfer model is realized with the playback memory, which is described in section 5.2.5.

- Posted requests: A command is sent to the chip by the controller and no further action is allowed until completion of this command. The *spikey_control* module cares for the correct completion. This transfer model is implemented to allow for direct Slow-Control access to the digital core modules of the Spikey chip. The timing of this transfer model is non-deterministic, especially because the access via the Slow-Control has no guaranteed or fixed latency [Sch06]. Posted requests effectively block the access to control interface data during execution and are mainly implemented for initial testing purposes that do not require the functionality of the playback memory.

Event data is processed separately from the control interface data (cf. chapter 3). The implementation of the transport network is not yet included within the test setup described in this chapter. Therefore, the only source and sink of event data is the local playback memory.

Clocking and Data Paths

All logic related to the Spikey chip, along with the future implementation of the transport network, is operated synchronous with the externally supplied clock. This setup is selected for two reasons: first, the synchronous operation of all components reduces the number of asynchronous elements which are needed to cross different clock domains. Second, the Spikey does not reach a maximum operating frequency of 200 MHz (cf. chapter 4) and on this account, the 156 MHz provided on the backplane is sufficient. The external clock signal is called `chip_clk` in the following; for the DDR operation of the physical layer to the Spikey chip, `clkhi` is available with double frequency. The memory interface needs to be operated at lower clock frequencies, since the controller implementation only works for up to 140 MHz [Sch06]. In fact, it is operated at half the speed of `chip_clk` and the according clock signal is called `sdram_clk`.

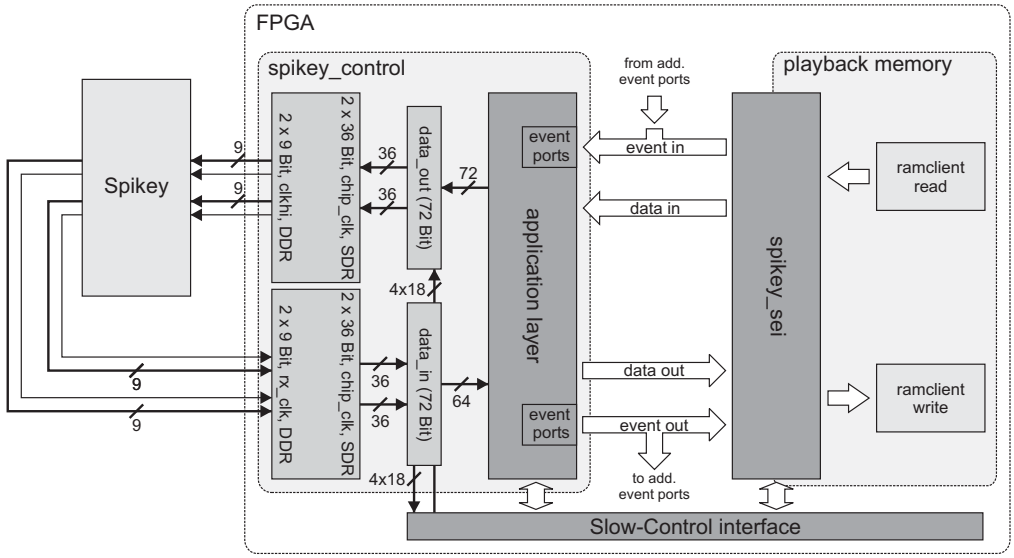


Figure 5.6: Illustration of the data paths within the controller and to/from the Spikey chip. The only source and sink for data on the application layer is the *spikey_sei* module, whereas link layer data is directly accessed through the registers *data_in* and *data_out* in bypass operation.

The data paths related to the communication with the Spikey chip are illustrated in figure 5.6. Data generated within *spikey_sei* is forwarded to *spikey_control* synchronous to `chip_clk` as well as the event streams, either generated by the playback memory or from the transport network. The application layer of *spikey_control* merges data and event streams and forwards a ready formatted 72 bit word to the link layer, which is stored in the register *data_out*. This data already contains the correct values for the frame bits of the packet and is forwarded to the physical layer which sends the data to the Spikey chip via two links. Output data is generated synchronous to `clkhi`. To achieve the required phase shift between the output link clocks and the according data, DCM3 is used, whose output phase shift can be controlled via the Slow-Control.

Data received from the chip is captured by the physical layer, the packet is decoded and stored within the register *data_in* including the frame bits. In this direction, only the 64 bit data payload is forwarded to the application layer, synchronous to `chip_clk`. The module *spikey_sei* is the sink for all control interface data; events may also be forwarded to additional event ports of the transport network.

5.2.3 The Controller of the Chip

The realization of the different protocol layers within the controller module *spikey_control* is described in this section. The structure in conjunction with the playback memory is illustrated in figure 5.6.

Physical Layer

In analogy to the implementation within the Spikey chip, the physical layer is only comprised of the DDR input and output registers. The implementation benefits from the fact that the Virtex-II pro offers native DDR registers within its I/O cells. The I/O cells are located directly

beneath the physical I/O pins of the FPGA and as a result, the routing delay to the registers only slightly differs with the location on the FPGA.

Receive Side The most critical part of the physical layer is the data capture on the input links to the FPGA. On the one hand, this is facilitated by the source synchronous nature of the data transmission and the well-matched phase relation between clock and data signals at the Spikey outputs (cf. section 4.3.1). On the other hand, the implementation suffices from the limited FPGA resources: no more DCM is available to align the input clocks' phases to the routing delay from the I/O cells of the FPGA to its input registers. For this reason, a feature of the FPGA fabric is exploited, which is called *local clocking*. Dedicated resources are used to locally distribute clock signals to input registers located within the input cells [xil05a]. To accomplish this, dedicated input pins have to be used as clock inputs and the related input cells need to be placed in the direct periphery of the input clock pins. This results in a skew of approximately 800 ps between the input signals' propagation delays and the propagation delay of the according clock signal (exact values for each pin are given in appendix C.5). As a result, this remaining skew can be almost eliminated by the *delaylines* on the Spikey chip and the input timing is correctly realized.

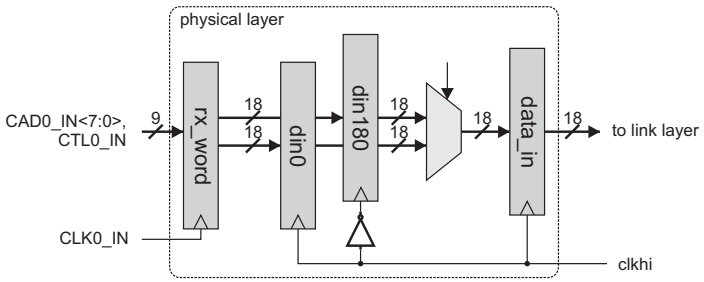


Figure 5.7: Input data capture in the physical layer of the FPGA. The registers *din0* and *din180* sample data at both edges of `clkhi`, but with inverted phase for the upper and the lower half of *rx_word*. A a consequence, at least one of them always samples valid data from *rx_word*.

The implemented logic for one input link is shown in figure 5.7. It benefits from the fact that both the FPGA clocks and the Spikey clocks originate from the same source, thereby having identical frequencies and a fixed phase relation. For this reason, no FIFO logic is required and data can directly be captured from the *rx_word* registers. The only register clocked by the input clock is *rx_word*. Depending on the phase relation between `clkhi` and `CLK0_IN`, either *din0* or *din180* will sample valid data from *rx_word* and the one containing valid data is selected[63] to *data_in*, which is the final output of the physical layer to the link layer.

Transmit Side On the transmit side, the physical layer only consists of the DDR registers within the according output cells. These are connected to the output of the link layer which accounts for the output of the packet stored within *data_out* in the correct order. As mentioned above, the phase of the link clocks relative to the data signals can be tuned using DCM3.

Placement of the I/O Cells As a consequence of the requirement to place the data input cells near the corresponding clock input cell within the FPGA, the grouping into busses of the

[63] This has to be done by a dedicated Slow-Control command.

differential signals on the Nathan module could not be maintained for the Spikey connections. Instead, the input cells are placed as described above and the output cells are placed such that the routing on the Recha board could be done as straight forward as possible. The confinement of the output signals to the SMT connector next to the FPGA requires the output cells to be spread over one complete edge of the FPGA fabric due to the configuration of the signals on this connector. In turn, this fact requires the placement of the link layer output registers clocked with `clkhi` to be spread over the whole FPGA, which complicates the timing closure for these registers. Indeed, these paths within the link layer clocked with `clkhi` limit the maximum achievable clock frequency for the whole design.

Timing closure on the FPGA design is achieved for a clock frequency of 156 MHz by the manual placement of registers directly connected to DDR output registers within the output cells, thereby alleviating the automated placement effort.

Link Layer

In contrast to the Spikey chip, which processes data on the link layer, this very data is generated and read back within the FPGA's link layer. The two modes of operation of the link layer are controlled by the control register corresponding to the `CI_MODE` pin.

Normal Operation During normal operation, the link layer either forwards data from the application layer to the physical layer and vice versa, or generates idle bus packets, if currently no data is available. The output of the link layer can be switched to send idle event packets instead of idle bus packets. This option allows the Spikey chip to transmit its event packets immediately (cf. section 4.3.2) and is the normal mode of operation during experiments.

Bypass Operation The application layer is disconnected from the lower layers in this mode and the link layer constantly sends the content of the register *data_out* while constantly writing received data to *data_in*, regardless of the content. The content of both registers is subdivided into four 18 bit words (cf. figure 5.6), which can directly be accessed via the Slow-Control interface. This mode of operation provides the lowest level access to the Spikey chip. It does neither require the functionality of the application layer, nor the data generation by *spikey_sei* and the playback memory. Thus, it is ideally suited for initial testing of a chip and for optimizing the *delaylines* on the chip[64].

Application Layer

The following tasks are carried out by the application layer: first, requests issued by *spikey_sei* are executed while providing both posted and non-posted operation. Second, the synchronization of the system time counters within the FPGA and the Spikey chip is carried out. The data paths involved are described within this section while the synchronization process will be described in the following section.

The data paths within the application layer are shown in figure 5.8. Both the path on the transmit side and the path on the receive side are controlled by a FSM which handles the data flow based on the packet content and on the requests issued by *spikey_sei*. The flow of event data is decoupled from the control interface data flow and is described in section 5.2.4.

[64] A dedicated Slow-Control address is available for the configuration of the *delaylines*. An FSM automatically applies the configuration data to the outputs and activates the pin `C_DELAY` to load the new delay value into the addressed *delayline*.

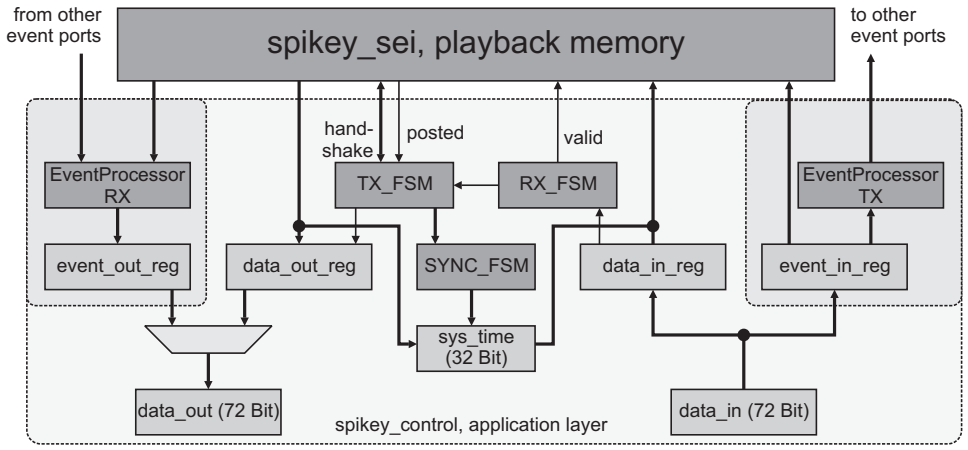


Figure 5.8: Block diagram of the application layer implementation within the *spikey_control module*.

Posted requests are indicated by the signal `posted`, which stays active until completion of the request. All requests are processed by `TX_FSM`, which solely transmits data to the chip during non-posted requests while it needs to evaluate the state of `RX_FSM` during posted requests. Therefore, the functionality on the receive side is now described prior to the transmit side.

The receive FSM has three states:

- `rx_idle`: Always reached upon reception of bus or event idle packets. Valid data triggers one of the two following states.

- `rx_data`: Incoming data is stored in the register *data_in_reg*. The state of `posted` is evaluated, and valid data is flagged with the signal `valid` during non-posted requests.

- `rx_event`: Events are stored in the register *event_in_reg*. On the one hand, the content of this register is connected to the input of the *EventProcessor*. On the other hand, all incoming events are forwarded to *spikey_sei* and thus to the playback memory which stores them for evaluation with the control software.

The state diagram of `TX_FSM` is shown in figure 5.9 a. After reset, it starts in `tx_idle` and goes to `tx_send` upon request. In this state, the data provided by *spikey_sei* is sent to the chip regardless of the transfer model or type of access (read/write)—provided that no events are to be transmitted. The automat returns to `tx_idle` in the case of a non-posted request, and goes to `tx_du_read` in case of a posted request. Within this state, a loopback command is issued to the chip acting as a dummy command with the purpose to verify the successful completion of the actual command. Again, the command is only sent if no events are to be transmitted, and the automat reaches the state `tx_check` where it stays until `RX_FSM` is in the state `rx_data`. If the received data would be flagged with the error bit, the command has not been completed within Spikey and the loopback command is re-issued until valid data is received[65].

[65]The command execution within the chip takes at least two clock cycles. By issuing the dummy loopback directly after the actual command, the first answer always contains the error flag. By this means, also the correct generation of errors is/can be verified.

Read requests need to be issued twice to provide the result of the first read command (cf. section 4.3.2). As a consequence, the first valid data that is received does not yet contain the actual read result and the loopback command is re-issued to gather this actual result (the number of successful reads is stored in `read2`, see figure 5.9 a. After completion of the request, the state `tx_complete` is reached which acknowledges the request and subsequently leads back to the idle state.

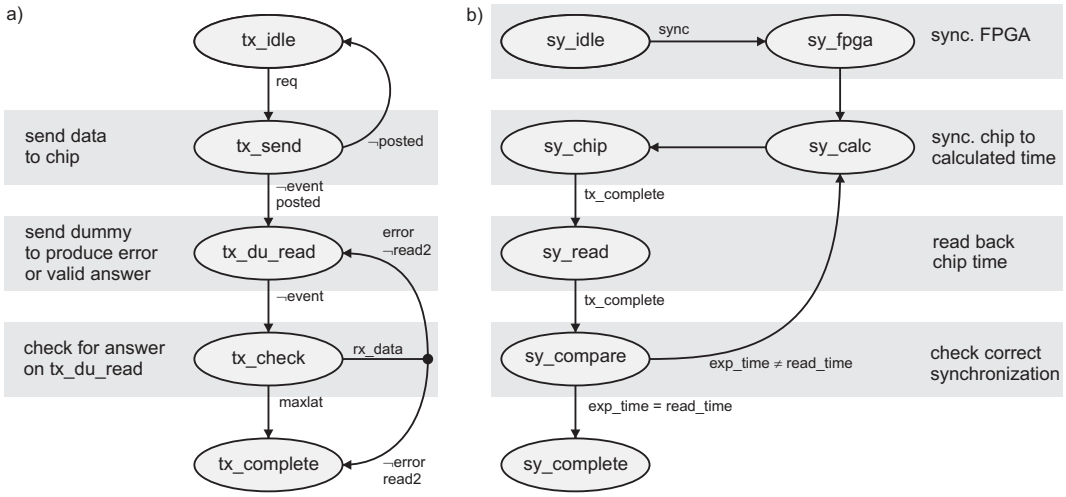


Figure 5.9: State diagrams of: a) `TX_FSM` and b) `SYNC_FSM`.

Note that posted requests are acknowledged to the calling module *spikey_sei*. In contrast to this, non-posted requests are not acknowledged, but incoming data is rather always sent to *spikey_sei* by issuing the `valid` signal.

Sideband Data

In addition to the protocol implementation, the *spikey_control* module contains control registers whose outputs are connected to the various single ended inputs of the Spikey chip, such as `RESET`, `CI_MODE` etc. This data is called *sideband data* and includes besides the mentioned pins the `FIREIN` pins of the Spikey chip and the direct access to the registers `data_out` and `data_in` during bypass operation of the link layer (see below). Sideband data is written to and read back from the controller via its Slow-Control interface. A finite state machine accounts for the handshaking on this port and the register access.

5.2.4 Synchronization and Event Processing

The process of synchronizing the system time counters within the FPGA and the Spikey chip is crucial to the correct processing of neural events within the system. Especially, if several Spikey chips are to be operated in parallel, their counters need to run synchronously to ensure correct event generation and digitization. Moreover, the synchronicity to the FPGA's system time is required for the operation of large scale neural networks, as described in chapter 3. This is also essential for single-chip configurations, as the FPGA time serves as an absolute reference to the time stamps of the events received from the chip which only have a width of 8 bit.

Synchronizing the System

Due to the unknown phase relation between the `chip_clk` signal within the FPGA and the `clkhi` signal within the Spikey chip, the actual number of clock cycles $t_{\text{lat,sync}}$, which are required from the dispatch of the synchronization command by the controller until the synchronization of the Spikey chip's counters has an uncertainty of one clock cycle (cf. section 4.3.7). To still ensure correct synchronization, the current value of the system time counter within the Spikey chip is read back after synchronization and the process is repeated iteratively until correct synchronization is achieved. As this verification cannot be done in software[66], an additional FSM performs the synchronization. The state diagram is shown in figure 5.9 b.

The FSM captures synchronization commands issued by *spikey_sei* and leaves its idle state `sy_idle`. The time to synchronize to is loaded into the FPGA's system time counter in the state `sy_fpga` which is immediately left for the next state `sy_calc`. In this state, the value to synchronize the Spikey chip to is calculated using the following figures:

$$t_{\text{sync}} = t_{\text{sys}} + t_{\text{lat,sync}} \ , \tag{5.4}$$

where t_{sys} is the current FPGA's system time and for $t_{\text{lat,sync}}$ the minimum delay until synchronization of the Spikey chip is assumed. Following this state, the synchronization command is sent to the chip within `sy_chip`, which is done by issuing a posted request to `TX_FSM`. After completion of this command, the current value t_{spi} of the chip's system time counter is read back by another posted request. The value of t_{spi} and the expected value t_{exp} are compared within the state `sy_compare`, while the the expected value is calculated during the `sy_chip` state using the following numbers:

$$t_{\text{exp}} = t_{\text{sys}} + t_{\text{lat,read}} \ , \tag{5.5}$$

where t_{sys} is the system time value during dispatch of the read command and $t_{\text{lat,read}}$ is the latency elapsing from the dispatch until the actual execution of the read command within the Spikey chip.

In case t_{exp} equals t_{spi}, the first synchronization has been successful and the synchronization process is finished by going to `sy_complete` and back to idle, subsequently. In the other case, the phase shift has led to a latency of $t_{\text{lat,sync}} + 1$ and the synchronization process is repeated while using $t_{\text{sync}} = t_{\text{sys}} + t_{\text{lat,sync}} + 1$ in the state `sy_chip`. After the following completion of the previously described steps, the synchronization is finally complete. This way, it is achieved that the maximum difference between the FPGA system time and the chip system time equals half a `clkhi` cycle.

In order to facilitate precise analog recordings of the membrane potentials using an oscilloscope, it is desirable to provide a trigger signal to the oscilloscope, which defines the point in time of synchronization relative to the subsequently generated events on the membrane. For this reason, the FSM performing the synchronization additionally generates an output signal which is active during the synchronization. It is mapped to a signal on the Nathan module which can consequently be used as a trigger signal for the recordings.

Event Processing

The data path for event data is also shown in figure 5.6. Events that are to be sent to the chip originate either from the interface to *spikey_sei* and thus from the playback memory,

[66] The only possibility to access the Nathan module is the Slow-Control. It does not provide a guaranteed access latency which is required in this case.

or from one or more ports of the transport network (cf. section 3.2.3). As the latter is not yet implemented, the *EventProcessor* module only forwards the data received by the ports to *spikey_sei*. Three ports are realized and events that are available at these ports are packed into an event packet, regardless of their content. Therefore, it has to be assured that events delivered simultaneously by *spikey_sei* together fit into one packet. This is done by the software which generates the content of the playback memory (cf. section 5.3). Events are always transmitted immediately and are thereby prioritized over control interface data. To assure correct delivery of control interface packets, event processing can be disabled by means of a control register.

The data path for the receive direction is simple for events that are forwarded to the *EventProcessor* (cf. section 3.3.1). However, to record events in the playback memory, some additional functionality is required to catch the case, where only sparse events occur and the difference in their time stamps becomes larger than the maximum value of the Spikey system time counter (which has a width of 8 bit). In this case, the software that evaluates the received events would not be able to correctly track the absolute time of the events' generation without further information.

The absolute time stamp of received events is calculated by the software by adding the tracked system time to the time stamps of the received events (see also section 5.3). Thereby, the tracked system time counts in multiples of the maximum value of the Spikey chip's system time value, precisely in multiples of 256 clock cycles. During the processing of received events, the high nibble of the currently processed event is compared to that of the previous one. If the current high nibble is smaller than the previous, the Spikey counter has wrapped around and the system time is increased by one. This only works if the distance between the arrival of two events is smaller than 128 clock cycles (the seventh bit of the system time wraps around) because otherwise, the MSB between both time stamps necessarily has flipped. In this case, it is not clear how often this flip has occurred, e.g. by how much to increment the absolute system time.

To solve this issue, the current value of the system time counter of the FPGA is written to the playback memory directly before the according event packet if the previous event packet has been received at a point in time $t_{\text{last}} < t_{\text{sys}} - t_{\text{diff}}$ with t_{sys} being the current system time and t_{diff} being the maximum time difference between two events (128 clock cycles).

5.2.5 Communication with the Controller and the Playback Memory

The module *spikey_sei* acts as source and sink of data to *spikey_control*. It is capable of generating posted or non-posted requests and event data. Posted requests are used to communicate with the Spikey chip via the Slow-Control interface to the Nathan module. Two Slow-Control write accesses are necessary to transmit the content of one 64 bit packet to *spikey_sei*. After these have been received by *spikey_sei*, the posted request is automatically initiated and executed by *spikey_control* as described above. In case of a read request, the result delivered by *spikey_control* is stored and needs to be read back by means of two additional Slow-Control read accesses.

Non-Posted Requests: The Playback Memory

Consecutive non-posted requests are required to be sent with sufficient intervals, since their completion is not verified as in the case of posted requests. Moreover, the generation of external (artificial) event input to the Spikey chip requires the event packet generation with deterministic timing to ensure correct event delivery. To fulfill these needs, the *playback memory*

has been implemented. It consists of a controlling FSM and according memory resources. The FSM executes (plays back) a sequence of commands that is consecutively stored within the memory (the DDR-SDRAM) and is capable of simultaneously recording received data into that same memory. As a consequence, no user interaction is possible during the execution of the playback memory. Instead, the data to be sent has to be stored in memory prior to an experiment while results can be read back at once after completion of the experiment. The memory access is realized using ramclients as illustrated in figure 5.6.

Four types of commands are implemented for the playback memory FSM to generate the data stream to the Spikey chip, with the following functionality:

- Control interface transfer: Directly send the data read from memory to *spikey_control* as a non-posted request. The according data is contained within the command.
- Event command with delay time: The playback memory sends a number of event packets, which follow the command in the memory content, to the event ports and thereupon waits for a programmable period of time (number of clock cycles) until execution of the next command.
- Event packet: Is not recognized as command, but rather as payload to the preceding event command.
- Delay command: Only wait for a certain number of clock cycles contained within the command.

The execution of one command takes one clock cycle plus the programmed delay for the event commands. The software, which generates the data to be sent to the chip accounts for the correct order of commands, the correct insertion of delay commands between control interface transfers and the packing of event data into packets preceded by the appropriate command.

On the receive side, non-posted request answers and incoming event packets are written to the memory and are read out by the controlling software after execution of the complete playback memory content.

In the simple setup with no additional event processing logic present, collisions between control interface data and event packets are not possible as the playback memory only generates one after another. The same holds true for the receive side: data is sequentially received from the Spikey chip and forwarded to either the data interface or the event ports within *spikey_control*. As both paths have the same latency to the playback memory interface, both data can be recorded without collisions.

Implementation The DDR-SDRAM interface operates at half the clock speed of the FPGA's `chip_clk` and is thus not capable of delivering 64 bit data packets to the Spikey chip at the rate of `chip_clk`. To at least provide the possibility of providing the full rate over a certain period of time, the FIFO memory of the two ramclients has been parameterized to use 10 of the FPGA's BlockSelect RAM memory blocks each. This results in a depth of 2048 entries for each ramclient, which are available to buffer the data.

On the transmit side, this is used as follows: when the playback memory is started, the command decoder first goes into a "prefetch" state and issues read commands to the ramclient until either the FIFO of the ramclient is full or no more data is to be played back. Following this, the content is played back while having the maximum buffer capacity of the ramclient's FIFO available.

It is important to keep this fact in mind during the setup of experiments with the system. If continuous data flow at an average of more than half the maximum rate is to be transferred, the FIFOs within the ramclients will eventually starve and the command timing can no longer be deterministic. The command decoder in this case cancels the execution and sets an error flag, which can be read back by the control software. Consequently, the maximum average event rate that can be achieved using the playback memory equals half the theoretical maximum rate obtainable on the interface to the chip (c. section 4.3.7):

$$r_{\text{ev,max,pb}} = 0.5 \cdot r_{\text{ev,max,theo}} = 0.75 \ \frac{\text{events}}{\text{clock cycle}} \ . \tag{5.6}$$

5.3 Control Software

5.3.1 Basic Concepts

A very important part in an ASIC test system are interactive monitoring capabilities of the physical chip as well as the off-line generation of test vectors used for the simulation and verification of the design prior to fabrication. The control software can be executed on a standard PC, thereby allowing a high flexibility since high-level programming languages and according functional libraries, e.g. for the graphical user interaction, are readily available. This also enables the development of high-level interfaces to software frameworks that are used to set up complex neural network experiments, thereby exploiting the full functionality of the Spikey chip and using the connectivity provided by the previously described hardware. Furthermore, the generation of test vectors and their verification is an essential task of the control software and it allows for a seamless migration from the simulation environment to the actual test system.

As for now, a complete hardware abstraction layer (HAL) of the Spikey chip including its operating environment is available which provides high level functionality for the integration of the system into a software framework, which has been developed within the Electronic Vision(s) group. The functionality of this framework will be outlined shortly at the end of this section.

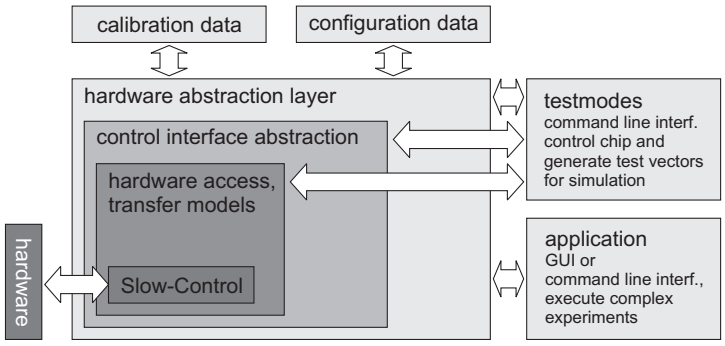


Figure 5.10: Hierarchy of abstraction layers in the software. The hardware abstraction layer loads calibration and configuration data from text files. Software accessing the chip is classified into the two boxes on the right. The software is executed on the control PC and low level hardware access is always executed by the Slow-Control. This access can be replaced by writing/reading to/from text files, which are used for the control of behavioral simulations of the entire setup.

Figure 5.10 shows a schematic overview of the control software. It is implemented using C++ [Str97] and object-oriented programming. The initial class structure of the abstraction layers originated from Dr. J. Schemmel and the current functionality is the result of the collaboration with the author. In the following, some of the classes present at the different abstraction layers are introduced.

The direct access to the chip is encapsulated by an abstract base class `SpikenetComm`. It serves as a base class for the realization of the different transfer models and provides virtual methods to send and receive data. These methods are actually implemented depending on the specific transfer model by the descendant class[67]. These descendent classes are:

- `SC_Sctrl` implements posted requests using direct access via the Slow-Control in two ways: either direct communication with the Slow-Control or the generation of a text file that is read by the Slow-Control master module during simulation of the FPGA code and serves as an input test vector to the simulation. Furthermore, this class exclusively provides the access to sideband data (cf. section 5.2.3).

- `SC_PbMem` implements non-posted requests and the generation of event data, hence, the communication using the playback memory. The playback memory content is automatically assembled depending on the requests issued by the higher-level parts of the software. Consecutive accesses including events are collected by the software and are not sent to the chip until the first request to actually receive data from the chip. The collected requests are then transmitted to the DDR-SDRAM and the playback memory program is started. The obtained results are read back from the DDR-SDRAM and are evaluated by the software.

- `SC_Trans` implements a transfer model that has not yet been mentioned. It models the functionality of a behavioral controller that has been realized in Verilog and allows for an efficient simulation of the Spikey chip while omitting the simulation of the full FPGA code[68]. The communication with the simulator is also done based on text files.

The control interface communication with the Spikey chip is abstracted by the base class `ControlInterface` which provides virtual functions to transmit and retrieve data to and from the core modules within the digital part of the Spikey chip respectively. Using these functions, the descendant classes implement the functionality of the different core modules, such as the access to the parameter memory, the synapse memory controller, or the control register. All descendant classes are instantiated by the class `Spikenet` which as a result encapsulates the complete functionality of the digital part.

The very top level of the HAL is represented by the class `Spikey` which combines the functionality provided by `Spikenet` with a convenient set of functions for the high-level access. Configuration data for one chip including parameter data, synapse configurations and weights, neuron configuration, etc. can be sent to the chip by calling one single method `config()`. Furthermore, two methods for the transmission and reception of spike trains are implemented: `sendSpikeTrain()` and `receiveSpikeTrain()`. Details regarding event (spike) transmission will be explained in the following section.

[67] The actual access to the Nathan modules is always realized via the Slow-Control (cf. figure 5.10). Access functions to the Slow-Control have been developed by S. Philipp [Phi07] and are included in the form of a link library. At the lowest abstraction level, all communication is carried out using this library regardless of the transfer model.

[68] Simulation of the full FPGA code significantly slows down simulation speed, as the complete framework needs to be simulated including behavioral models of the DDR-SDRAM module.

Communication with the Chip: Test Modes

The communication with the chip can be established on all levels of abstraction using the command line based access illustrated in figure 5.10. The command line based software accepts the bus model to be used and any number of *test modes* that are to be executed as input arguments. The test modes are realized as separate classes implementing a method `test()`, which is called during runtime for each specified test mode.

At runtime, the desired transfer model is selected by the user and during initialization, the descendant of `SpikenetComm` implementing this transfer model is constructed, thereby providing read/write functionality to the hardware. In case of the playback memory transfer model, the access to the DDR-SDRAM is fully transparent and the user does not have to care for this.

Based on the representing descendant of `SpikenetComm`, the interface to the digital part of the chip is instantiated by constructing `Spikenet` which itself instantiates the classes representing the digital core modules and the afore constructed bus model class. The user can access all layers of abstraction from within the test mode which is desirable especially for the precise generation of test vectors and for initial testing of the chip.

5.3.2 Event Processing

Time Tracking and Synchronization

To correctly accomplish the system synchronization and event processing, the software tracks the system time during the generation of the playback memory content, off-line[69]. The system time is initialized to 0 at start up and the delay of each command is added to the system time. The system time counters within the Spikey chip and the FPGA remain uninitialized until the first synchronization command which synchronizes them to the current system time (cf. section 5.2.4). From this point in time onwards, events can be generated by the playback memory. This process will be described in the following section.

Generating an Event Playback Memory Program

The generation of event data in a playback memory program is somewhat detached from the generation of control interface data because additional processing steps are required. As the programmable logic does not yet support the packing of events into event packets with three events each, this needs to be realized in software. Furthermore, the software has to keep track of event and system time and decide, whether to transmit or discard an event because it would arrive on the chip, too late. Another issue to be solved in software is the fact that simultaneous events within one *event_buffer_in* module would block event generation by this module for one Spikey system time counter wrap around (cf. section 4.3.5). All of this basically is the functionality that is realized in the *EventPctGen* module for the implementation of large scale neural networks described in chapter 3. The slightly differing artificial event generation works as described in the following.

The method `sendSpikeTrain` accepts an STL vector of events as an argument. This vector contains events for a certain number of synapses with their time stamps sorted in ascending order. After dispatch of a synchronization command, this vector is passed to the method `pbEvt()` provided by `SC_Sctrl` which generates the playback memory program.

[69] This is done in the same way for the `SC_Trans` transfer model. The `SC_Sctrl` transfer model does not provide deterministic timing and timing is not tracked, there.

Since the playback memory is the only source of events in the current operating environment, the method `pbEvt()` does not have to care for events arriving late from remote sources. This would be required with the transport network present (cf. section 3.3). It is therefore programmed to optimally utilize the bandwidth of the Spikey chip's physical interface. This is achieved by packing as much event packets as possible into one playback memory event command, thereby reducing the overhead introduced by too many event commands. This practice is illustrated in figure 5.11. It is desirable to send the events to the chip as early as possible to gain headroom for potentially following peak event rates. This is taken into account by the packing algorithm which is explained by means of algorithm 2 at the end of this chapter.

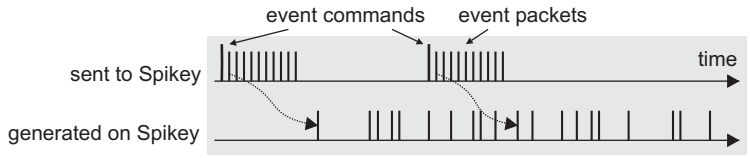


Figure 5.11: Relation of playback memory event commands to events generated on Spikey. Each event command is immediately followed by a certain number of event packets. To gain maximum event throughput, the number of event commands needs to be minimized. This is achieved by the early transmission of as many events as possible. These are stored within the according *event_buffer_in* on Spikey until being generated.

5.3.3 Higher Level Software

As illustrated in figure 5.10, the HAL hides hardware specific details from the application software. To use the Spikey chip in a biologically sensible way, all parameters and values are translated to their biological quantities by a higher level software, which is developed as a joint project within the Electronic Vision(s) group [BGM+07].

Two branches are followed for the high-level software development. On the one hand, a C++ based environment with a graphical user interface (GUI) is provided allowing for visual configuration of the chip and the setup of experiments with an immediate visual feedback. On the other hand, an interpreter-based interface is available which allows for the script-based setup of experiments and thus supports the methodology commonly used by neuroscience modelers.

To bring both worlds together, Python [The07b] has been chosen as the top level glue language motivated by its large flexibility and the possible benefit from an active community developing Python applications. A Python-based software module has been developed [BGM+07] to interface to the afore mentioned interpreter. This tool together with the existing Python interface to the NEST simulator provides the framework for a unified processing of the data from both domains.

Graphical User Interface The GUI allows to set up and interactively operate smaller networks in a modeler's terminology. The configuration of neurons, synapses, their connectivity and all relevant analog parameters are displayed either in biological or technical terms and can be manipulated. In figure 5.12, the network editor is shown providing visualization and configurability for all synaptic connections in the network. In the left pane, input activity can be generated either manually or by modularly implementable routines, whereas the generated output spikes are immediately displayed in the right pane after execution of an experiment.

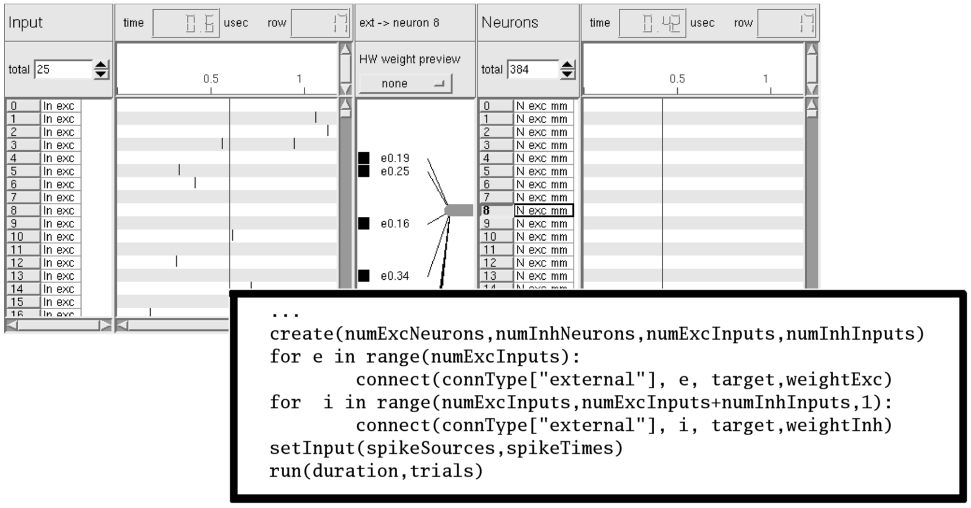


Figure 5.12: Upper left: screen shot of the C++ based graphical network editor. Networks can be set up manually using biological terminology and parameters, but with direct feedback regarding the hardware constraints. Lower right: example code snippet for the unified Python interface, executable on both the Spikey chip and the NEST simulator. Figure taken out of [BGM+07].

The group-wise assignment of parameter voltages as well as constraints regarding connectivity and local feedback are governed by the software.

To intuitively explore different parameters, the GUI provides a loop functionality for a loaded experiment. The response of the chip is immediately displayed after each run and all parameters can be varied during looping. The GUI-based hardware interface is integrated into the object-oriented C++ based HANNEE[70] framework, which has been developed within the Electronic Vision(s) group and is described in [Hoh05].

Python Interface Wrapper code for the `Spikey` class utilizing the open source C++ library `Boost.Python` accesses the abstraction layer described in section 5.3.1 via Python. This Python application programming interface (API) is connected to a pure Python class hierarchy which aims to integrate the hardware into a meta language developed within the FACETS project. It allows to operate the hardware and read out all data provided by the HAL in a biological context. A simple experiment setup is illustrated by the code snippet in the lower right of figure 5.12.

PyScope While in accordance to the neural network model, the afore generated information is all digital (digitized spike times and digital values for the parameters), the analog membrane voltages can be recorded by connecting an oscilloscope to the membrane potential monitor outputs of the Spikey chip. A Python front-end for digital oscilloscopes with a network connection integrates this analog information into the Python framework. The so-called PyScope software provides access to the oscilloscope via Ethernet and in addition to the acquisition of the raw data, the data can be visualized and some basic operations for trace manipulation are available. The PyScope software can be integrated into the scripts and is used for the event-related measurements presented in the following chapter.

[70] HANNEE: Heidelberg Analog Neural Network Evolution Environment

Algorithm 2 Pseudo code to describe the implemented algorithm to convert a vector of events into a playback memory program. The outer loop runs over all events within the vector. Within the while loop it is checked whether the currently processed event fits into an event command together with previously processed events. If yes, it is stored in a temporary vector together with the previous events. If not, an event command is generated sending the previously stored events and the current event will then be the first one in the temporary vector. The first *if* statement within the while block checks for deadlock conditions within *event_buffer_in*.

```
sync:
send synchronization command
lasttstamp[15 : 0] = 0 : stores last time stamp within each event_buffer_in

procedure SC_SCTRL::PBEVT(vector of events)
    for i ← 1 to sizeofvector do
        buf ← event_buffer_in corresponding to event[i]
        tstamp ← time stamp of event[i]

        if systime < tstamp then
            EXIT: "event time overtakes system time!"
        end if

        while true do
            if tstamp ≠ lasttstamp[buf] then
                gencmd ← false
                while tstamp > earliest possible delivery do
                    insert delay commands
                    increment systime by inserted delay
                end while

                if no wraparound since last event and this one then
                    check whether event fits into currently assembled packet
                    this includes checking for same buffer collision
                    and the time stamp's upper nibble
                    if yes then
                        store event in temporary vector, gencmd stays false
                    else
                        gencmd ← true
                    end if
                else
                    gencmd ← true
                end if

            else
                increase tstamp to avoid deadlock in event_buffer_in
                continue while loop
            end if

            if gencmd ← true then
                generate event command from previously stored events
                continue while loop
            else
                break while loop - event is stored within temporary
                vector and will be sent with the next event command
            end if
        end while
    end for
end procedure
```

Chapter 6

Experimental Results

Measurements performed for the characterization of the implemented chip are presented in this chapter. The strategies for the implementation of the physical interface of the chip are verified by means of a comparison of the results expected from the timing closure with the measured performance of the physical interface. The results of the timing closure are furthermore verified by determining the maximum operating frequency of the digital core logic. The successful system integration is demonstrated and the system is characterized by testing the event generation and digitization on the chip with different setups, thereby determining the maximum achievable event rate using the operating environment described in chapter 5. Additional results obtained with more advanced setups of the neural network prove the functionality of the analog part and the entire framework.

The first version of the Spikey chip was taped out in August 2005 in a MPW run and the samples were obtained four months later. To reduce the risk of errors introduced by direct bonding of the chip onto the Recha board, 10 dies were ordered to be packaged into a 208-pin ceramic quad flat pack. This package has been chosen due to its low parasitic pin capacitance and its homogeneous layout[71]. The same holds true for Spikey 2 which was taped out in August 2006 and was delivered in December 2006. The controller code for the chip was developed during the testing of the first version based on the behavioral Verilog code (which implements the `SC_Trans` transfer model) that was used for the verification of the first design. During this testing, the whole operating environment developed to its current state. Due to the seamless interchangeability of the software simulation used for verification and the operation of the physical hardware, the verification as well as the tests of the second version of the chip were drastically eased.

[71] In contrast to ceramic quad flat pack (CQFP) packages, where the pins are distributed over the edge of the package, e.g. pin grid array (PGA) packages have their pins distributed over the area of the package which results in different routing lengths and skew to the pins.

6.1 Test Procedure

Following the step of verification of the successful soldering and bonding of the chip, the system is powered up. The FPGA is configured and the connection between the Slow-Control and the control software is established. In the following, the generic procedure for the initial operation of a new chip is described:

1. Verification of the physical layer: The input clock signals are checked and the correctness of internal clock generation is verified by measuring the output link clocks. The signal integrity of the LVDS signals generated by the FPGA and the Spikey chip is investigated. While verifying the functionality of the delay elements, the process corner is extrapolated based on these results (see section 6.2).

2. Verification and setup of the link layer: The basic functionality in bypass operation is verified and the delay elements are tuned to gain a maximum data valid window, thus, maximize the operating clock frequency (see section 6.2.3).

3. Verification of the application layer: The digital core modules are thoroughly tested with random data. The results from timing closure are verified by determining the maximum error-free operating frequency (see section 6.4).

After successful completion of these tests, more advanced features of the chip are tested:

4. Event generation and digitization: By sending and producing single events/spikes, the synchronization process and the basic functionality is proven. Following this, more complex and biologically realistic experiments are set up (see section 6.5).

Additional measurements characterizing the power consumption are performed. Unless otherwise noted, the presented results are obtained with two chips of the second version: the packaged Spikey # 2, which will be referred to as chip 2, and the directly bonded Spikey # 5, which will be referred to as chip 5.

Besides the operation environment and equipment described in the preceding chapter, the following measurement equipment has been used:

- The LVDS signals on the physical interface are measured using the Tektronix oscilloscope TDS 7254 which has an analog bandwidth of 2.5 GHz and a maximum sample rate of 20 GSamples per second. The according active differential Tektronix probes P7330 have a bandwidth of 3.5 GHz and are suited to characterize the LVDS signals with a clock frequency of up to 400 MHz and expected rise and fall times in the range of a few hundreds of picoseconds.

- All membrane potential traces shown are recorded using the LeCroy oscilloscope waveRunner 44Xi with a bandwidth of 400 MHz and a maximum sample rate per channel of 5 Gsamples per second. The monitored membrane outputs are terminated by internal 50 Ω termination resistors to ground. Compared with the Tektronix model, this oscilloscope features larger memory on each channel to record signals over time with the maximum sample rate. This is especially required for the recording and automated read out of membrane voltage traces over several hundreds of microseconds.

6.2 Performance of the Physical Layer

6.2.1 Clock Generation

The correct functionality of the internal PLL, the clock generation circuitry, and the clock tree is verified by solely applying the clock signal `EXT_CLK` to the chip. The `RESET` pin is kept active and `CI_MODE`, `PLL_BYPASS` and `PLL_RESET` are successively turned off in the stated order. By keeping the reset active and `CI_MODE` inactive, it is ensured that none of the FIFOs within the link layer is active and the only activity produced is the bus idle patterns on the physical output layer.

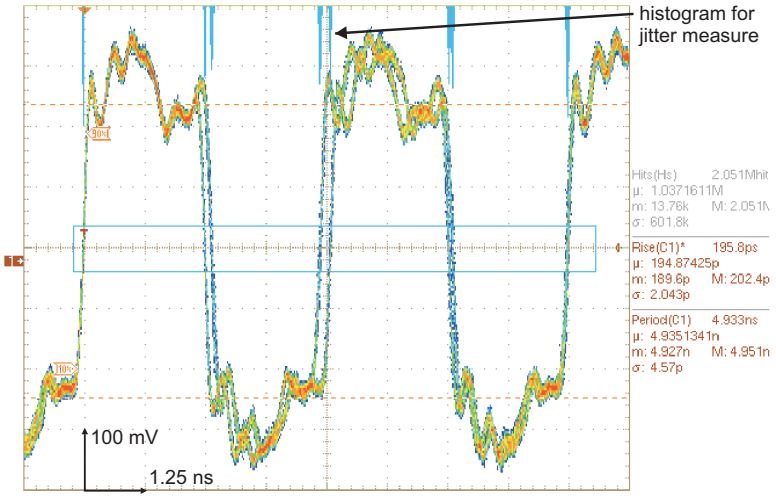


Figure 6.1: Output clock on link 0 of Spikey 1, 200 MHz link clock frequency. The jitter described in section 4.5 can be observed introducing a period error of up to 250 ps.

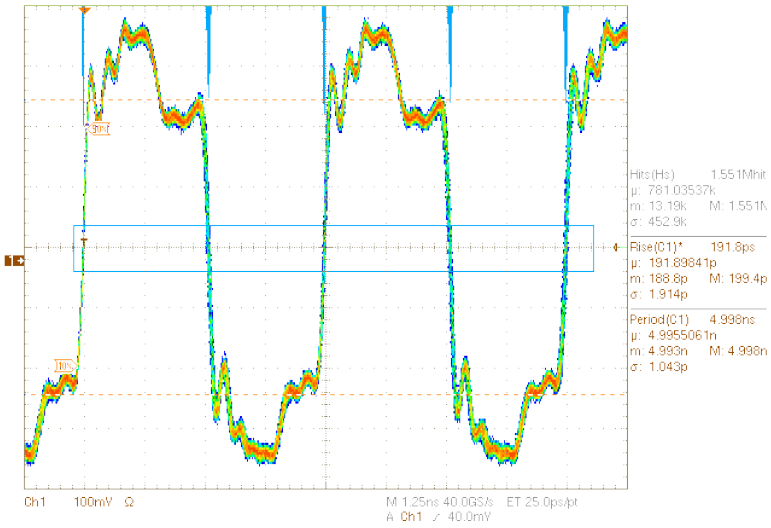


Figure 6.2: Output clock on link 0 of Spikey 2, 200 MHz link clock frequency. The jitter has vanished, proving the functionality of the improved clock generation circuits.

The output clock signals of link 0 on version 1 and 2 of the Spikey chip are shown in figure 6.1 and figure 6.2 respectively. In order to measure the peak-to-peak jitter, data acquisition is performed in fast acquisition mode[72] and a histogram is generated for the number of data points within the box around the x-axis. It is displayed at the top corner of each plot. The first conclusion is that the duty cycle distortion present on the first version has vanished in the second version which proves the successful implementation of the changed clock generation circuits (cf. section 4.5). Second, a jitter of about 100 ps is observed in figure 6.2, which is above the PLL's specification of ± 93 ps [Vir04a]. As it is not possible to test the clock without the idle activity on the output link and the signal is not distorted by heavy reflections, this increased jitter value is considered to be caused by internal crosstalk from the data signals on the link. The measured jitter reduces the required times for signals within the `clkhi` domain by approximately 1.6 % at a period of 3.2 ns. Since an uncertainty of the clock of 100 ps, thus, a reduction of the required time for all signals to 3.1 ns has been considered while constraining the design (cf. section 4.4.1), this deviation has already been included during timing closure and the observed jitter is not supposed to significantly reduce the achievable operating frequencies.

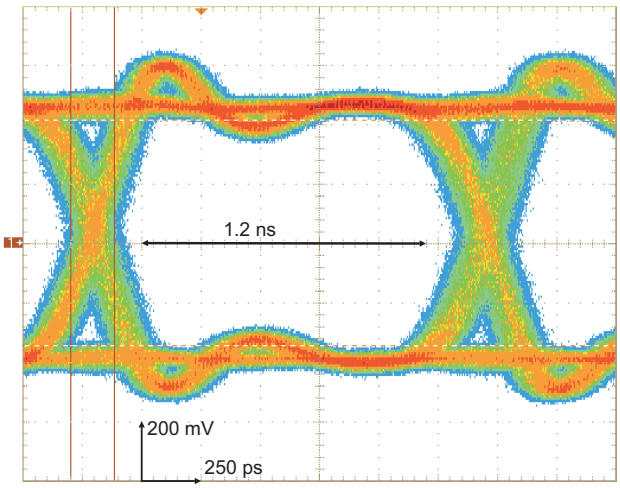


Figure 6.3: Eye diagram measured on input data bit 7 on link 0 of chip 2 at 312 MHz clock frequency. The measurement covers one bit time with a duration of 1.6 ns.

6.2.2 Signal Integrity: Eye Diagram Measurements

A measure that allows the estimation of the achievable performance of the physical interface is the integrity of the signals on the PCBs. In order to verify the system-level simulations and the impedance calculations for the PCB traces, eye diagram measurements are performed. A differential probe was attached to the link clock which was used to trigger the eye diagram recording. Selected data signals were recorded differentially as well. The recording was done with the chip being in bypass mode for $2.5 \cdot 10^5$ random sets of data for the 72 bit *data_out*

[72] In fast acquisition mode, the waveform capture rate is increased up to 400.000 waveforms per second. The points of highest sample density appear in warmer colors.

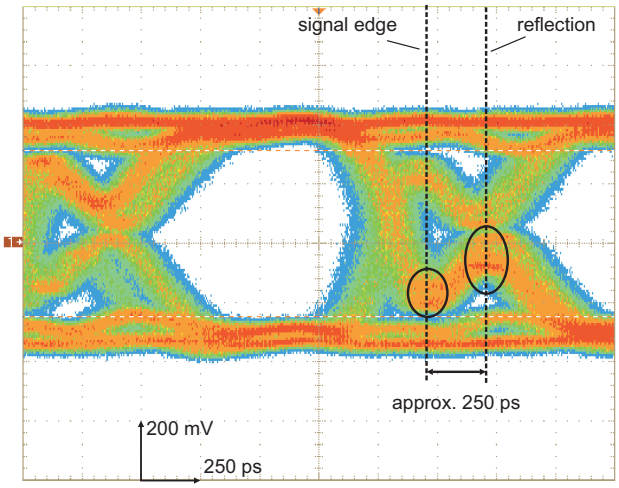


Figure 6.4: Eye diagram measured on output data bit 5 on link 1 of chip 2 at 312 MHz clock frequency. The measurement covers one bit time with a duration of 1.6 ns.

register within the FPGA, thereby assuring the occurrence of the 16 possible patterns on the measured interface signal itself[73] and random crosstalk scenarios on the neighboring signals.

Input signals The eye diagram for one bit time of the input signal `CAD0_IN<7>` on chip 2 is shown in figure 6.3. It is recorded with the probe directly attached to the package pin and at a link clock frequency of 312 MHz, which corresponds to a bit time of 1.6 ns. The specified DC levels of ±440 mV [Xil02b] are reached for static signals and a slight ringing is present for the 0-1 or 1-0 transitions. As no further distortion of the signal is present and the ringing level will not affect input switching behavior, good signal integrity is achieved at the input, resulting in a data valid window with a width of $\Delta t_{\mathrm{i}} = 1.2$ ns at 312 MHz.

Output signals To determine the signal integrity on the transmit side, the signal should be measured at the FPGA's receiver input pins. However, these are not accessible due to the high routing density on the Nathan module. The measurement of the selected signal `CAD1_OUT<5>` (see figure 6.4) is therefore conducted at a via pair with a distance of approximately 2 cm to the die of the FPGA. A strong degradation of the signal level about 250 ps after the signal edge can be observed which is due to a reflection with phase jump of 180° at the FPGA's input pin pair. This is supported by the time difference of approximately 250 ps between the signal edge and the reflection. Given a signal propagation delay of 60 ps/cm on the PCB, this time delay almost matches the total length of the signal return delay to the probed via pair, which is expected to be 240 ps.

These reflections are observed on all output signals and due to the proximity to the FPGA pins, it is assumed that the data valid window at the inputs of the FPGA is reduced to approximately $\Delta t_{\mathrm{o}} = 0.6$ ns at 312 MHz. Increasing the operating frequency will further decrease this value and thus, the maximum achievable clock frequency is limited by these signals. In particular, a period value of 2.5 ns (400 MHz) for the link clock could not be reached, as the lower limit of this clock period is determined by the data valid window which closes at

[73] The content of the *data_out* register is continuously being transmitted in bypass mode. As a consequence, four bit times are repeated on each data signal resulting in 16 possible bit patterns on each interface signal.

$T_{\text{limit}} = 3.2\,\text{ns} - \Delta t_{\text{o}} = 2.6\,\text{ns}$. The achievable interface clock frequency is therefore limited to about 385 MHz.

The reason for the reflections with phase jump is the FPGA's receiver impedances being smaller than the impedance of the differential transmission line on the PCB (according to [DP01], pp. 95). On the one hand, the internal termination resistors of the FPGA are specified to a value of 100 Ω and the correct DC level of 440 mV is not fully reached on the signals (cf. figure 6.4) which indicates a smaller termination resistance. On the other hand, the differential traces on the Nathan module have not yet been characterized, and the trace impedance of the Nathan module is possibly above 100 Ω. Therefore, for future PCB developments, it is suggested to measure this trace impedance using a network analyzer.

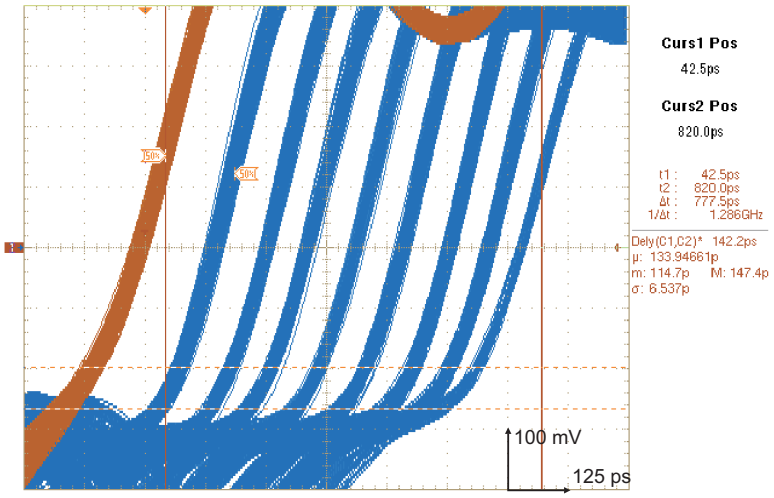


Figure 6.5: Sweep of the delay values on output signal 0 on link 0 of chip 2 (blue traces). The oscilloscope records in infinite persistence mode showing all recorded traces; the link clock signal serves as a reference for the delay measurements (brown trace).

6.2.3 Accuracy of the Delay Elements and Estimation of the Process Corner

The functionality of two *delaylines* at the output is verified together with the mutual timing of clock and data signals. The measured delay values are compared with the back annotated simulation. On the one hand, this allows for the verification of the accuracy of the *RC*-extractor that is used for STA. On the other hand, the process corner of the fabricated chip can be estimated.

Measurement Setup The output signals with the shortest default delay `CAD0_OUT<0>` and the longest default delay `CAD1_OUT<5>` are measured. This gives the largest distance in reference values for the comparison with the *RC*-extraction results. The rising edge waveforms at a clock frequency of 312 MHz are shown in figures 6.5 and 6.6 respectively. The according link clock signal is used as the trigger signal and the chip is in reset state during measurements to minimize side effects due to internal activity. Time differences are measured as the difference between the rising edge of the clock signal and the data signal by the oscilloscope for a period of 10 s for each delay value while the mean value and its standard deviation is calculated automatically by the oscilloscope.

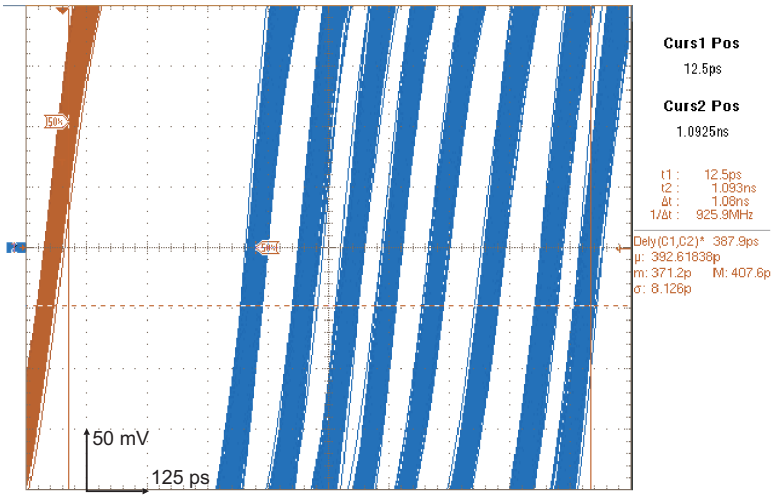


Figure 6.6: Sweep of the delay values on output signal 5 on link 1 of chip 2 (blue traces). The oscilloscope records in infinite persistence mode showing all recorded traces; the link clock signal serves as a reference for the delay measurements (brown trace).

Signal Integrity The standard deviation for all delay values varies between 4–7 ps. It can be seen that the width of the superposed clock signals adds up to about 40 ps, which is considered to be the trigger uncertainty of the oscilloscope. The width of the superposed data signal traces is caused by this uncertainty; this argument is supported by the small value of the standard deviation of the relative delay values. The signal edges for all delay values are monotonic with no distortion and a delay range of approximately 650 ps is covered in both cases. This proves the successful implementation of the *delaylines* with a delay of 80 ps per delay stage and an entire delay range of 640 ps for the typical mean process corner.

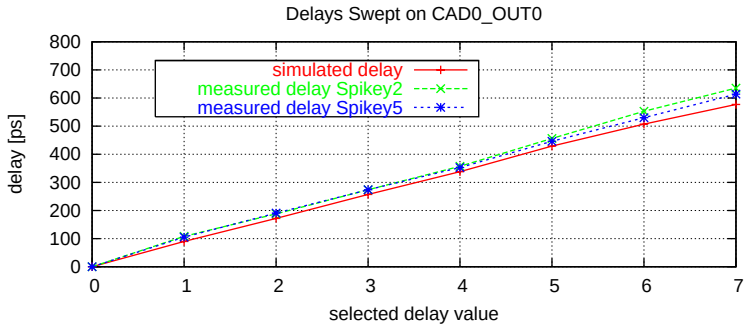


Figure 6.7: Comparison of simulated output delay vs. measured output delay of the signal `CAD0_OUT<0>` against the selected delay value.

Accuracy of the Delay Elements To quantify the two *delaylines*, this measurement is performed for chip 2 and chip 5 and the results compared with the back annotated simulation are shown in figures 6.7 and 6.8 respectively. In both plots, the delay values are normed on the zero delay value to quantify the spread for larger delays. Lines have been drawn for better visibility—no data is present between the data points. Error bars are omitted due to the small

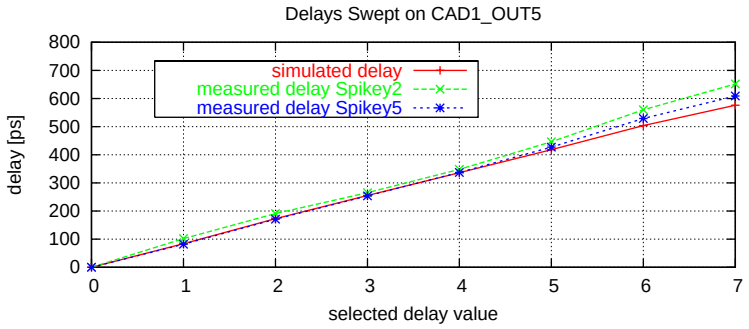


Figure 6.8: Comparison of simulated output delay vs. measured output delay of the signal `CAD1_OUT<5>` against the selected delay value.

standard deviation of the single delay values. For delay values up to 4 only small deviations of 15 ps at maximum from the typical mean simulation are present for both chips. For delay values above 4, the deviation increases to up to 75 ps for `CAD1_OUT<5>` on chip 2. Furthermore, the deviation of the measured delay values from the simulation results increases slightly stronger than linearly with increasing delay value. Since the simulated delay is the sum of the delay through the inverter chain of the *delayline* plus the extracted interconnect delays, this suggests that the gate delay is correctly calculated in simulation, but the interconnect delays (whose portion of the overall delay increase with the selected delay value) are calculated to be too short. This leads to too optimistic results in STA and is considered for the following discussions.

A reason for the deviation of the interconnect delays is the inaccuracy of the *RC*-extraction software used for the delay calculation. The employed extraction software is shipped with First Encounter and operates based on a table-lookup *RC*-model (cf. section 2.5.1). It has been executed in high accuracy mode using *extended capacitance tables* [Cad06b]. Based on the presented results it is suggested to verify the extracted values by means of a stand-alone extraction software providing more accurate extraction results with a self-contained field solver. For reasons of data compatibility this could be done with Cadence Fire & Ice[74].

Estimation of the Process Corner Both chips perform slightly worse than the simulated typical mean which is manifested in a slightly increased slope in figures 6.7 and 6.8. The deviation of chip 2 is used for a worst-case estimation and by using the values for the signal `CAD1_OUT<5>` the deviation results in 12 %. This value will be used as a basis for the performance estimates throughout this chapter.

Due to package parasitics and thereby an increased *RC*-delay, the LVDS output drivers of chip 2 are expected to exhibit a constant, larger delay. According to the raw data this delay equals 10–15 ps comparing chip 2 and chip 5 (data not shown, as the plots are normed to zero delay). Nevertheless, the still deviating measured delay values indicate a small variation of the process corner between the two chips which will be neglected for the following discussions.

Output signal timing To demonstrate the successful implementation of the required phase relation at the output by tuning the according *delaylines*, the delay values of the previously

[74] This step is originally included in the design methodology described in chapter 2. It has not been performed due to missing Fire & Ice licenses. This problem is presently solved.

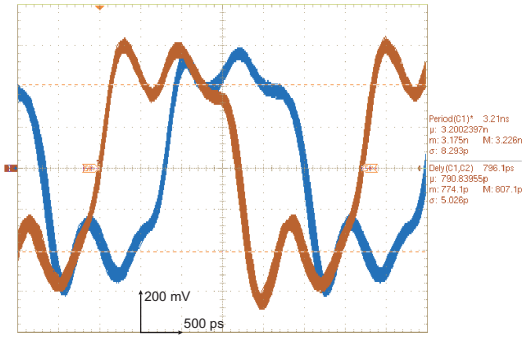


Figure 6.9: Relative timing of the link 0 output clock and the according data signal 0 at 156 MHz. Optimum relative timing is achieved with a delay value of 7.

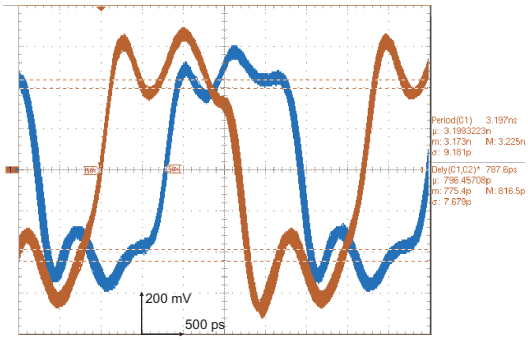


Figure 6.10: Relative timing of the link 1 output clock and the according data signal 5 at 156 MHz. Optimum relative timing is achieved with a delay value of 4.

characterized data signals `CAD0_OUT<0>` and `CAD1_OUT<5>` are set to 7 and 4 respectively. This results in a delay between clock and data signals of 791 ± 5 ps (cf. figure 6.9) and 796 ± 8 ps (cf. figure 6.10) which corresponds to the desired phase shift of $-90°$ at the applied interface clock frequency of 312 MHz.

6.3 Verification of the Link Layer and Maximum Data Rate

The correct functionality of the link layer implementation within the chip is first investigated in bypass operation. Random patterns are written to the *data_out* register of the FPGA, thereby testing both the physical and the link layer of the Spikey chip and the FPGA as well as the clock generation circuitry within the Spikey chip. To get an optimum data valid window at both receive and transmit side, the following steps are carried out:

- Based on the assumption of identical routing delays on the PCBs (see section 5.1.2) optimum values for each *delayline* are calculated based on the back annotated signal delays within the FPGA and the Spikey chip. The results are given in appendix C.5 and are used for the initial setup of the *delaylines*.

- In order to optimize the receive side of the Spikey chip, a test bench has been developed that automatically sweeps the phase of the link clocks relative to the link data while checking for correctness of the data in bypass operation. The phase shift values at the first occurrence of an error is recorded for each data signal which results in a distribution of the first point of failure over the phase shift value for all signals. The *delaylines* are then modified towards the mean point of failure according to the distance to the mean value and the link clock is shifted away from this point by 90° to set the optimum phase relation. The optimum result of this algorithm would be all signals failing at one phase shift value which would yield a maximum data valid window. However, the quality of this result varies slightly with the chip under test, and yields better (smaller) delay distributions for directly bonded chips compared to packaged versions. Therefore, the optimum setting needs to be separately determined for each chip. It can be automatically loaded during initialization of the chip when using the constructor of the hardware abstraction layer (cf. section 5.3.3).

- After optimum delay values have been set, the phase of the link clocks is shifted into the middle of the data valid window to yield maximum headroom for both setup and hold times.

To determine the latter phase and the size of the data valid window, the correct functionality of the link layer in bypass operation is measured in dependence of the design variables link clock frequency f_{link} and link clock phase shift ϕ_{link} relative to the link data signals. The expected results for this measurement are determined using the example simulation shown in figure 6.11 which yields the expected clock-to-data delay at the input flip-flops of the physical layer: at a clock frequency of 200 MHz data should be valid for absolute clock shifts of 572 ps to 2841 ps relative to the link data. For higher frequencies, the value of 'TimeB' in figure 6.11 decreases with the clock period, whereas 'TimeA' is frequency independent. Therefore, the size of the data valid window decreases linearly with the clock period.

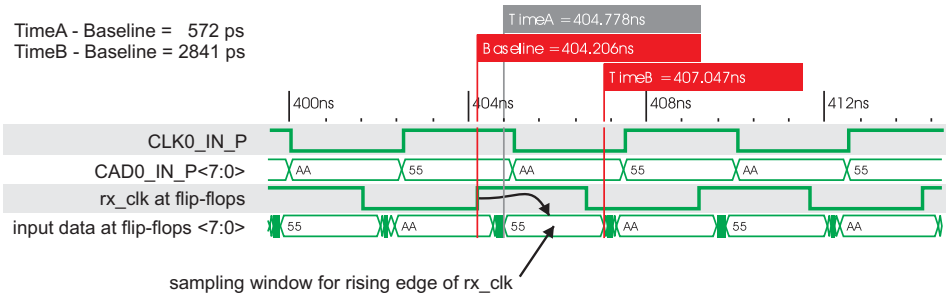


Figure 6.11: Back annotated simulation of input link 0 at 200 MHz link clock frequency for the typical mean process corner. The signals at the chip boundary along with the signals at the clock input and D-input pins of the input flip-flops are shown. The phase shift between input clock and data signals is 0° which theoretically yields valid data for clock shifts reaching from 572 ps to 2841 ps.

The measured functionality of the input links with respect to the afore mentioned variables is shown by means of the schmoo plots in figure 6.12 for chip 2 and in figure 6.13 for chip 5 respectively. For each data point both links were tested with 10^6 random patterns; the light gray points denote at least one error on each link and thus failure. The solid colored points denote pass and thus a bit error rate (BER) $< 2.78^{-8}$ for each link.

For both chips the data valid window of link 1 is about 250 ps larger than that of link 0 which is contrary to the expectation, since inside the chip more crosstalk would be expected on the signals of link 1 running in parallel along the edge of the chip to the digital part. It can still be explained by the external routing on the PCBs: the link 1 signals are taken out of a parallel routed bus on the Nathan module, whereas the link 0 signals are patched together from non-corresponding signals on the PCB. In particular, some of the signals are additionally connected to the surplus PGA-socket on the Nathan module which degrades the integrity of these signals compared to the directly connected signals[75].

The upper border of the data valid window of chip 5 is close to the expected 2841 ps. Furthermore, at 320 MHz it matches the expected value of 1900 ps. This value results out of subtracting the period difference from the original value at 200 MHz.

[75] Note that the presence of this socket is not correctly covered by the system-level simulations due to the lack of correct simulation models provided by the vendor.

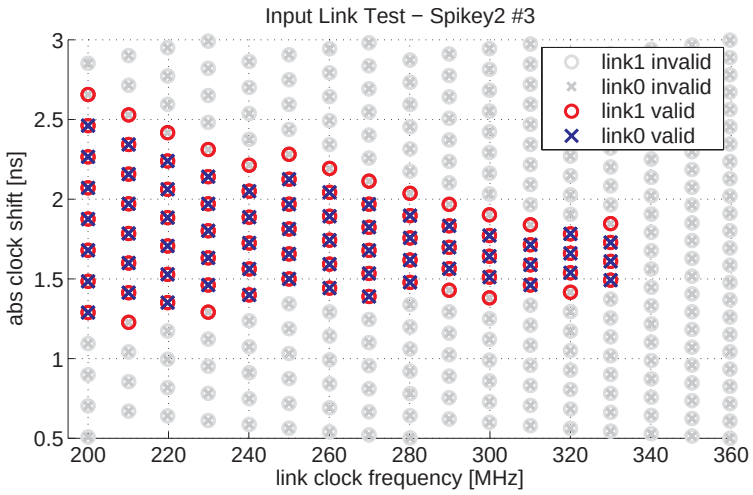


Figure 6.12: Schmoo plot of the link functionality of Spikey #3 in dependence on the design variables clock frequency and shift of the link clock relative to the link data.

Potential reason for shift of lower border The lower border of both measured data valid windows misses the expectations by approximately 700 ps for chip 2 and 900 ps for chip 5. As the upper border could be verified, this behavior indeed suggests that data is not valid inside the chip before this point in time. For the following reason this could be due to metastable states of the input registers: the buffers inserted into the input data signals by CTS do all have considerably large drive strengths and are concentrated on the top right edge of the digital part, along with the actual clock tree buffers of the link clocks and the according input registers. For this reason, the current required during simultaneous switching of these buffers potentially leads to a voltage drop which could produce meta stable states within the input registers. These would result in false input data. This peak current decreases when the switching point of the clock signal is moved away from the data signals' switching point, and data is sampled correctly. The observed behavior supports this assumption.

To get a more quantitative result, the power distribution system could be analyzed with the methods provided by First Encounter [Cad06b]. Due to incomplete library data, this analysis has not been set up so far and this analysis is subject to future work. A positive result is that for both chips the link layer has been successfully tested at 156 MHz with $160 \cdot 10^6$ random 72 bit patterns. To conclude, the link layer reliably works at the desired system clock frequency of 156 MHz and a link clock frequency of 312 MHz with a BER of $\leq 8.7 \cdot 10^{-11}$. In spite of being smaller than the originally specified system clock frequency of 200 MHz, this matches the maximum operating frequency of a system consisting of several Spikey chips interconnected via the MGT network on the backplane. As described in section 5.2.2, the maximum operating frequency for this system will presumably be 156 MHz, provided that all components are operated synchronous to the same clock.

The following sections will verify this operating frequency for application layer of the Spikey chip.

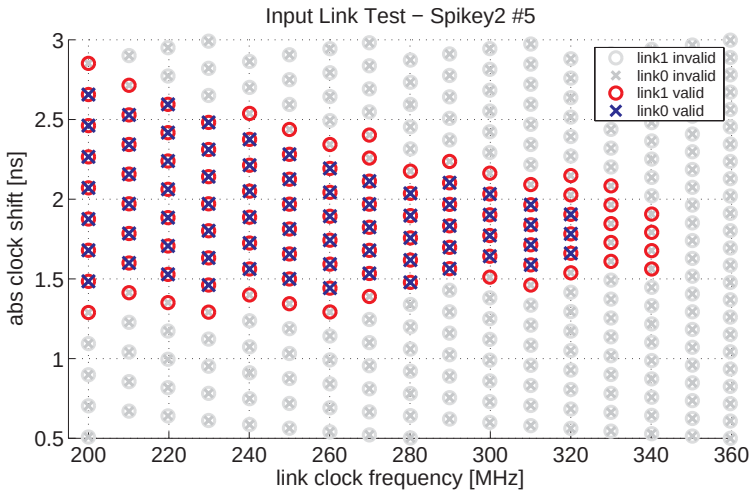


Figure 6.13: Schmoo plot of the link functionality of Spikey #5 in dependence on the design variables clock frequency and shift of the link clock relative to the link data.

6.4 Verification of the Application Layer

After the correct functionality of the physical and the link layer is verified by the above tests, the digital core modules and the event processing logic in the application layer is tested. The success of these tests requires communication with the chip on the control interface abstraction layer.

6.4.1 Basic Functionality

In order to provide basic test functionality of the command decoder and the packet processing within the application layer, the loopback register is implemented on the Spikey chip (cf. section 4.3.6). The access to the loopback register transports data on all available bits of the 64 bit packet. Hence, it is well suited for thorough testing of the chip's internal data paths with randomly generated data. The test mode performing this test sends random data to the chip which is answered by the chip with the inverted version of this data. The test mode compares this data with the expected result; it is run both for posted requests and non-posted requests. The initial testing of the link layer and the loopback test have been successfully completed with all, but one (see section 6.6) chips tested so far. The results of these tests are summarized in section 6.4.3. With the implications on the data valid window this demonstrates:

- The successful implementation of the physical and the link layer in the Spikey chip as well as in the FPGA. Thereby, the correct asynchronous operation between the link clocks and the core clocks is demonstrated.

- The functionality of the top level command decoder and the correct protocol implementation is verified. This functionality is required to proceed with the measurements described in the following, which include the remaining digital core logic, the event processing logic, and the mixed-signal interface.

6.4.2 The Different Core Modules

The core modules within the application layer include the status and control registers, the parameter storage, the synapse memory and STDP controller, the analog readout controller, the event loopback module and the event processing modules *event_buffer_in* and *event_buffer_out* (cf. figure 4.6). The digital logic within these modules is almost fully synchronous, with the exception of the FIFOs within the modules *event_buffer_out*. Furthermore, no complex arithmetic functionality is present within the digital core modules. For these reasons, most of the functionality contained within the application layer can be verified by testing the access to registers or static memory resources.

In contrast to the loopback command, test data first has to be written to the chip in order to perform these tests; it is then verified after read back. The available memory and register resources have been thoroughly tested with random data and random intervals between the commands. During testing for bit errors, these intervals have only been chosen with reasonable values greater or equal than the minimum required delay.

Mixed-Signal Interface to the Synapse Memory

The access to the distributed static memory resources within the *network_blocks* is carried out by a controller FSM within the module *synapse_control*. This module plays a special role, insofar as it operates a large part of the mixed-signal interface between digital and analog part. The correct functionality of the module itself as well as the access to the memory resources has been verified for the first version of the chip. Moreover, initial experiments prove the correct functionality of the correlation measurement circuitry located within each synapse and have been published in [SGMM06]. The readout of these circuits is also part of the functionality of the *synapse_control* module as described in section 4.3.6.

During the tests performed with Spikey 2, intermittent errors were observed for accesses to these memory resources. The errors occur at a low rate: on average, 20–30 single-bit errors are encountered for one test covering the complete memory including the row and column configuration memories. Corresponding to a total number of approximately 10^5 4 bit synapses, this yields a BER of about $8 \cdot 10^{-5}$. Unfortunately, no more exact data can be given at the moment. Nevertheless it turns out that these rare errors do not hinder the setup of experiments with the chip. The correct access to the resources used for the measurements presented in section 6.5 proves the basic functionality of the interface. Further measurements regarding the synapse memory will not be presented.

As stated above, the reason for this behavior is not clear at the point in time of writing the present thesis. It could be verified that it is neither clock frequency dependent (within the accessible range of frequencies, see section 6.4.3), nor could a definite dependency on the supply voltage be observed. Since neither the controller design, nor the design of the *network_block* has changed in the second version, no obvious reason for the errors could be identified. Furthermore, the drive strengths of the standard cells operating the interface is sufficient for all involved signals and the power distribution system has not been changed in the second version.

To conclude, the errors occurring within the memory resources do not generally hinder the setup of (biologically realistic) experiments with the chip. Intermittent errors could lead to erroneously configured synapse weights which alter the neural network's behavior. This has to be taken into account during the experiment setup and the utilized memory resources have to be verified before an experiment is initiated. Finding the cause of these errors is subject to future work and on the one hand, requires more intensive testing of the memory resources.

On the other hand, mixed-signal simulations of the back annotated digital netlist together with the analog circuits including parasitics could possibly reveal timing issues that were not encountered during the digital simulations performed prior to submission of the chip[76].

6.4.3 Maximum Operating Frequency

The previously described verifications are carried out for various clock frequencies. Thereby, potential resonance scenarios within *LC*-circuits of the power distribution system could be identified, which are not covered by the performed system simulations. No such effect was observed during testing. However, the main goal is to determine the maximum operating frequency for each part of the chip, thereby verifying the results of the timing closure given in section 4.4.4. Table 6.1 summarizes the obtained values, together with the maximum BER of the according test and the expected frequency value. The expected value for the core modules equals half the link clock frequency because it is limited by the expected value of the link clock frequency—the actual values obtained by STA are slightly larger (cf. section 4.4.4).

Mode	Expected [MHz]	chip 2 [MHz]	chip 5 [MHz]	max BER
Link	324	330	320	$8.7 \cdot 10^{-11}$
internal loopback	162	156	150	$1.9 \cdot 10^{-9}$
event loopback	162	160	160	$9.5 \cdot 10^{-11}$
parameter memory	162	156	156	$6.5 \cdot 10^{-10}$
synapse memory	162	–	–	–

Table 6.1: Maximum core clock frequency in MHz that has been measured for tests of the respective module with random data and the according maximum BER, in comparison to the expected clock frequency values. The expected values are set to be 12 % worse than the typical mean result of the timing closure (cf. section 6.2.3). Results for the synapse memory access cannot be given due to intermittent errors at all frequencies (cf. section 6.4.2). The operating frequency was swept with step sizes of 5 MHz in an interval of $\{100, 170\}$ MHz on `chip_clk` for each test with one extra measure at 156 MHz.

The performance of the link layer matches the expectation within an error of 2 %. The remaining results for the application layer match within 4 %, with the exception of the internal loopback on chip 5. With the implication of intermittent errors during synapse memory accesses, these results prove the functionality of all circuits relevant for the event transport at an operating frequency of 156 MHz.

6.5 Verification of the Event Processing

The correct processing of events, e.g. the correct synchronization of the chip time to the system time, the accurate generation of events as well as the accurate digitization, are essential for the setup of single chip experiments and for the implementation of multi-chip large-scale neural networks. This section describes the measurements performed to verify this functionality.

Measurement Setup Unless otherwise noted, the presented results were obtained with chip 5 and the LeCroy oscilloscope. All measurements involve the processing of either absolute or

[76] Due to the large number of analog devices within the *network_block*, it was not yet possible to set up this simulation.

relative time differences which have been obtained with the following technical setup: the oscilloscope is triggered by the dedicated FPGA output indicating the start of the synchronization process (cf. section 5.2.4). As the time the system is synchronized to is known, this serves as an absolute time reference. Furthermore, the oscilloscope records the output of the `IBTEST` pin in order to directly measure the point in time of spike generation or digitization. The `IBTEST` pin is configured to output the monitoring signal available for synapse # 1 on the left *network_block* (cf. section 4.2.4, figure 4.5). By appropriately configuring the according synapse input multiplexer, it is possible to record either the generated presynaptic input signal coming from the DTC, or the spike generated by the associated neuron which is in addition digitized by the TDC[77].

Using the PyScope software (cf. section 5.3.3), the `IBTEST`-signal is read from the oscilloscope and after low-pass filtering the data by averaging each data point over a window of 15 samples, the onset of the digital pulses is determined by the point in time where the signal crosses a threshold. Following this, either the relative distance of two pulses or the absolute distance to the trigger signal is compared to the digital input or output of the chip.

The analog parameters used for the Synapses and neurons are set such that the neuron under test would produce exactly one output spike on the simultaneous input of four synapses. The exact parameters can be found in appendix A.

6.5.1 Synchronization of the Chip

The synchronization process is investigated, in order to ensure the correct temporal processing of events. The test involves the verification of the correct initialization of the chip's counters which are clocked with an unknown phase relation to the FPGA controller clock signal. Furthermore, during synchronization it is decided which group of eight *event_buffer_ins* to associate to one of the 200 MHz event clocks, `anaclk` and `anaclkb`. The correct interplay of the synchronization logic implemented in the Spikey chip and the FPGA is verified with the following setup:

The chip is synchronized to time t_{sync} and only one event is sent to synapse # 1 at time $t_{\text{event}} = t_{\text{sync}} + \Delta t$. Therefore, the theoretical distance between the trigger time and the measured event generation is Δt. The times are chosen in a way that the according time stamps lie in time bin 0 of a clock period. By keeping t_{event} and incrementing t_{sync} by one clock cycle (thus decrementing Δt by one clock cycle), on the one hand, the two possible scenarios for synchronization are covered and on the other hand, the event needs to be stored in two different buffers, thereby testing both functionalities. The measurement is repeated for $t_{\text{event}} + 384$ clock cycles to include one Spikey counter wrap around and simultaneously verify the correct transmission of the events by the playback memory. The difference between the measured distance Δt_{OSC} and the expected distance Δt_{OSC} between synchronization start and event generation is plotted in figure 6.14, with the experiment repeated 100 times for each data point. The constant offset of approximately 7.8 time bins is due to the delay introduced by the

[77] The oscilloscope is limited to a minimum signal rise time of 875 ps at a sampling rate of 5 GSamples/s. On the one hand, the rise time limitation will flatten the response to the originally digital pulse coming from `IBTEST`. However, as this modifies the shape of all measured pulses equally, this is considered to not add as much of an error as the comparably low sampling rate. At a `clkhi` frequency of 200 MHz, which has been chosen for the experiments, one time bin of the DTCs/TDCs comes up to 312 ps, whereas the resolution of the oscilloscope is 200 ps. Thus, the digitization error of the oscilloscope is 2/3 LSB of the DTC/TDC circuits and will dominate the measured error for fluctuations smaller than this value. Nevertheless, as the main objective of the presented measurements is the verification of the digital functionality at a resolution of 5 ns, this resolution was chosen to be sufficient for these measurements.

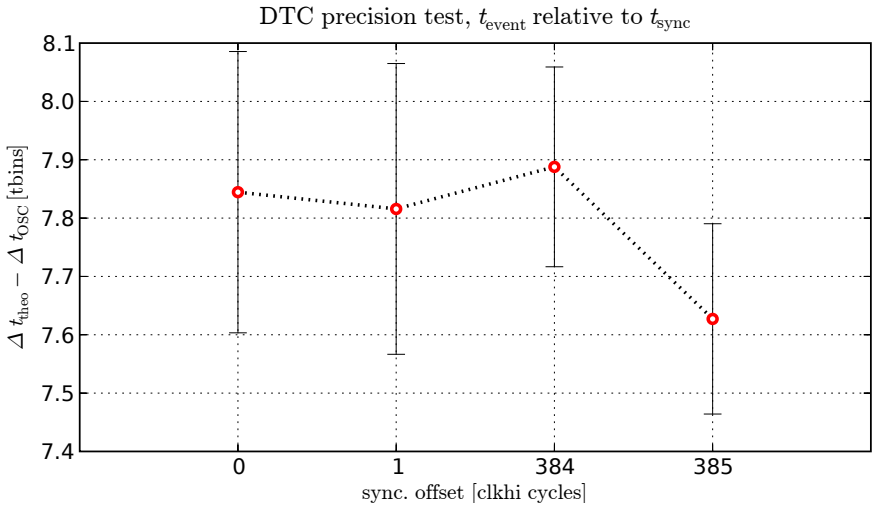


Figure 6.14: Comparison of the measured time differences Δt_{OSC} between a synchronization command and one single event with a fixed time stamp against the expected theoretical distance Δt_{theo} for different synchronization times. By increasing the time of synchronization (values on the x-axis), the difference between synchronization and the event is effectively reduced. The constant offset between the theoretical distance and the measured distance is below half a `clkhi` cycle (8 time bins) which proves the correct synchronization of the chip with even and odd time values (see main text).

analog readout chain, from the multiplexer at the synapse driver through the `IBTEST` pin of the chip, to the oscilloscope.

The fact that the difference stays constant within the error margin (which is well below one `clkhi` period) proves the correct synchronization and event generation. Furthermore, the basic communication between the *event_buffer_ins* located within the digital part and the DTCs within the analog part is verified successfully.

6.5.2 Verification of the Digital Event Transport

The digital event transport inside the chip is verified by using the *event_loopback* module which basically acts as a multi-stage pipeline register for incoming events. which are captured at its output after a certain delay n_{el}. Neither analog functionality, nor the event related logic within the analog part are required for this test. It is therefore ideally suited to fully verify the functionality of the event-related part of the application layer within the digital part. The successful verification of the chip using the *event_loopback* module with random events yields three results:

- Correct functionality of the event generation and capture logic within the application layer including the static memory blocks used for event buffering.
- Correct functionality of the system time tracking algorithm described in section 5.2.4.
- Maximum achievable event rate using the playback memory with random events. This rate is hard to predict because on the one hand, it depends on the known depth of the playback memory ramclients and the occurrence of refresh cycles of the DDR-SDRAM (which slows down memory access and has no temporal correlation to the playback memory). On the other hand, it depends on the compressibility of the random events

into event packets, which in turn depends on the high nibbles of the according time stamps and the addresses of the target synapses (only one event per *event_buffer_in* can be transferred per packet). T

The test using the *event_loopback* module uses the number n_{el} of `clkhi` cycles that are introduced between *event_buffer_in* and *event_buffer_out* by the *event_loopback* module. Each event with a time stamp t, that is sent to the chip and is correctly generated, is captured at time $t + n_{\text{el}}$ and subsequently sent off chip. The test algorithm compares sent and received events against each other, searches for matching pairs and succeeds, if every received event can be matched with a sent event.

The activation of the event loopback module separates the event-related logic of the digital part from the analog part and connects it to the pipeline registers. As the timing of the according multiplexer and clock enable pins is unconstrained for the implementation, this results in potentially invalid data at the output of the event loopback module for up to n_{el} `clkhi` cycles. Thus, to prevent the generation or capture of invalid event data, the reset signal of all *event_buffer_in* and *event_buffer_out* modules is held active during the appropriate time by means of sending the according command to the control register. Following this procedure, the chip is synchronized and the test is performed.

Both, chip 2 and chip 5 have been successfully tested at a `chip_clk` frequency of 156 MHz (`clkhi` and link clock frequency: 312 MHz). One test run consisted of $5 \cdot 10^6$ randomly generated events on 384 input synapses[78] and is repeated 100 times for each chip. This results in an event error rate of $\leq 2 \cdot 10^{-9}$, and a BER of $\leq 9.5 \cdot 10^{-11}$ respectively on the interface. This result is obtained with an average event rate of $0.1 \cdot r_{\text{ev,max,theo}} = 0.15$ events/clock cycle to prevent the read and write ramclient from being starved or overloaded respectively. The maximum achievable rate is determined in the following section.

6.5.3 Maximum Event Rate Using the Playback Memory

To determine the maximum achievable event rate with the playback memory, the above test has been performed in a loop with 10^5 events per loop while increasing the event rate on all input synapses within each loop until one of the ramclients flagged an error during execution of the playback memory cycle[79]. The test was then repeated 10 times for the last error-free event rate with an event count of $2 \cdot 10^6$. The result is given in events per clock cycle and is clock frequency independent due to the fixed frequency relation between DDR-SDRAM and *spikey_control* and the fixed number of refresh cycles for the DDR-SDRAM. The test yields

$$r_{\text{ev,max}} = \frac{0.36}{N_{\text{i,max}}} \text{ [events per clock cycle]} \tag{6.1}$$

for the maximum number $N_{\text{i,max}} = 384$ of input synapses which can simultaneously be tested. With the speed-up factor $s = 10^5$ and the clock frequency $f_{\text{c}} = 156$ MHz, this can be transformed to an average rate in biological real time:

$$\begin{aligned} r_{\text{ev,max,biol}} &= r_{\text{ev,max}} \cdot \frac{f_{\text{c}}}{s} \text{ [Hz]} && (6.2) \\ &= 1.5 \text{ Hz per input synapse} && (6.3) \end{aligned}$$

[78] This is the maximum number of input synapses, since the *event_loopback* module introduces a one-to-one mapping from input synapses to output neurons.

[79] Either the DDR-SDRAM delivers data too slow and the read ramclient starves or the write ramclient overflows due to the insufficient write speed of the DDR-SDRAM (cf. section 5.2.5).

for $N_{\mathrm{i,max}} = 384$ input synapses.

The theoretical maximum average rate using the playback memory is (measured in events per clock cycle)

$$0.5 \cdot r_{\mathrm{ev,max,theo}} = r_{\mathrm{ev,max,pb}} = \frac{0.75}{N_{\mathrm{i,max}}} , \tag{6.4}$$

since the DDR-SDRAM is operated at half the clock frequency of the Spikey chip (cf. section 5.2.5). Only about 50 % of this rate is reached which is due to the following reasons: on the one hand, communication overhead is introduced by the necessary event commands preceding the event data. On the other hand, it is not always possible to pack three consecutive events into one packet due to the random nature of the data. Therefore, event packets containing only one or two valid events additionally reduce the net throughput of events.

The packing algorithm is supposed to perform better at increased event rates due to the increased probability for three events having identical high nibbles in the time stamp. To further investigate this, the obtainable maximum event rate is determined with spike trains of no more than 2048 events which definitively fit into the FIFO memory of the ramclients. By this means, all event event data can be prefetched in the beginning of the playback memory program and events can be generated at the full clock rate (maximum interface rate) without starving the read ramclient (cf. section 5.2.5).

While using the algorithm described above with the event count of 2048, the test yields

$$r_{\mathrm{ev,peak}} = \frac{1.2}{N_{\mathrm{i,max}}} \text{ [events per clock cycle]} \tag{6.5}$$

for $N_{\mathrm{i,max}} = 384$ input synapses. Using the above transformation, this results in an average peak rate in biological real time:

$$\begin{aligned} r_{\mathrm{ev,peak,biol}} &= r_{\mathrm{ev,peak}} \cdot \frac{f_{\mathrm{c}}}{s} \text{ [Hz]} && (6.6) \\ &= 4.9 \text{ Hz per input synapse} && (6.7) \end{aligned}$$

for $N_{\mathrm{i,max}} = 384$ input synapses and $f_{\mathrm{c}} = 156$ MHz. In relation to the theoretically achievable maximum interface rate of $r_{\mathrm{ev,max,theo}} = 1.5$ events per clock cycle, this yields a maximum interface utilization of 80 % using random event data with the playback memory.

6.5.4 Event Generation: Digital-To-Time

As the DTCs convert a digital to an analog value, just as in the case of a voltage DAC, a common measure for the quality of a DAC, the differential nonlinearity (DNL) is determined for the DTCs. It is defined as the maximum deviation of the analog value changes caused by an LSB change from its ideal size of 1 LSB [GAS90]. To include both clocks used for event generation within the measurement, the DNL is measured over four clock cycles, or 64 LSBs of time bin resolution. Two events are generated on the input synapse # 1 with a time difference of 6401 time bins which equals 400 `clkhi` cycles plus 1 LSB of the DTC. The `chip_clk` frequency is set to 100 MHz for reasons given at the end of this section. These two events are shifted in time by one LSB for 64 times. The difference between the analog recordings Δt_{OSC} and the theoretical distance $\Delta t_{\mathrm{theo}} = 6401$ time bins is plotted in figure 6.15.

Each data point represents the mean of 100 measurements and the error bars give the standard error of the mean value. The statistical error is below the systematic error of 2/3 LSB introduced by the oscilloscope and as a consequence, the maximum DNL is estimated as $\mathrm{DNL}_{\mathrm{max}} = \pm 0.6$ LSB. In the first instance, it can be stated that the functionality of the event

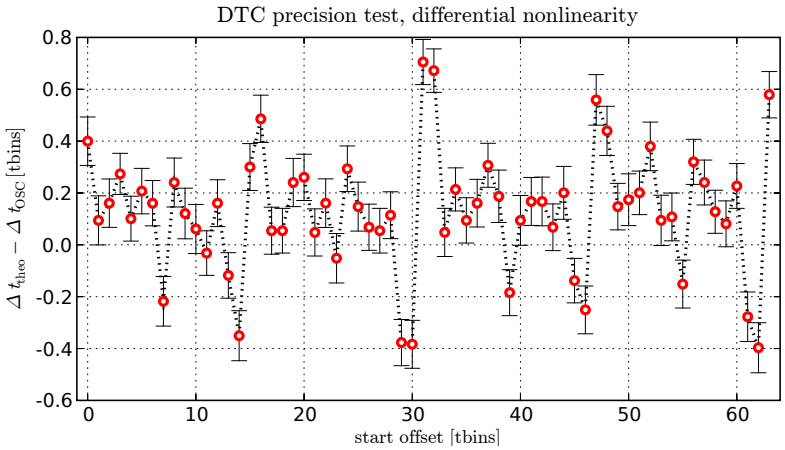


Figure 6.15: Measurement of the differential nonlinearity of the DTCs. The measured time difference Δt_{OSC} between two events generated with a distance of 6401 time bins is subtracted from this theoretical value, while the start event is shifted by one time bin for each data point. The subtraction yields the deviation from the theoretical LSB change for each time bin value.

generation from playback memory down to the synapse drivers works correctly on a cycle basis, since $\mathrm{DNL}_{\mathrm{max}}$ is well below 16 LSB which represents one clock cycle.

The maximum DNL value is caused by peak values occurring with a periodicity of 16 time bins, first in negative, then in positive direction. To find reasons for this systematic behavior, the analog trace of a special event pattern is investigated, as shown in figure 6.16.

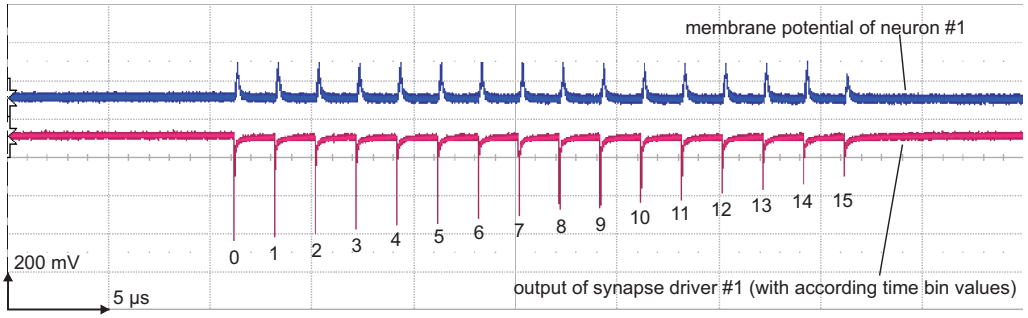


Figure 6.16: Measuring the signal generated by the digital output of the monitored synapse driver (lower trace). The upper trace shows the analog membrane potential of the neuron stimulated by this synapse. Sixteen events are generated with increasing time bin value, starting at zero. The output signal of the synapse driver becomes smaller with increased time stamp which is due to the shortened digital pulse within the chip (see main text). The peak on the membrane voltage corresponding to the last event is smaller than the previous ones which indicates that the synapse driver does not correctly operate due to the shortened pulse length.

Sixteen events are generated with a distance of 6401 time bins starting with time bin zero for the first event and incrementing the time bin value by one with each event. It can be seen that the peak becomes smaller—which is expected—as the event enable signal is only active from the beginning of the time bin of event generation until the end of the clock cycle. Therefore, it becomes shorter resulting in a smaller peak recorded by the oscilloscope after being low-pass filtered by the readout *RC*-network as the time bin is increased. Additionally, the slope of the

pulse becomes smaller and threshold crossing is accordingly detected by the readout software a bit later. This difference becomes larger with increasing time bins.

For this reason, a discontinuity at time bins 15 to 16 is observed in figure 6.15 indicating a wrap around of the 4 bit time bin value to 0. Identical behavior is observed at time bins 31 and 47. This inherent measurement error cannot be compensated with the current setup. As a consequence, more precise measurements are not possible with the current setup, and the DNL has to be assumed to $\mathrm{DNL}_{\mathrm{max}} = \pm 0.6\,\mathrm{LSB}$. Note that this is measured at $f_{\mathrm{c}} = 100\,\mathrm{MHz}$ which yields an achievable temporal resolution of $312.5 \pm 187.5\,\mathrm{ps}$. For higher frequencies, no data could be taken. The reason is explained in the following.

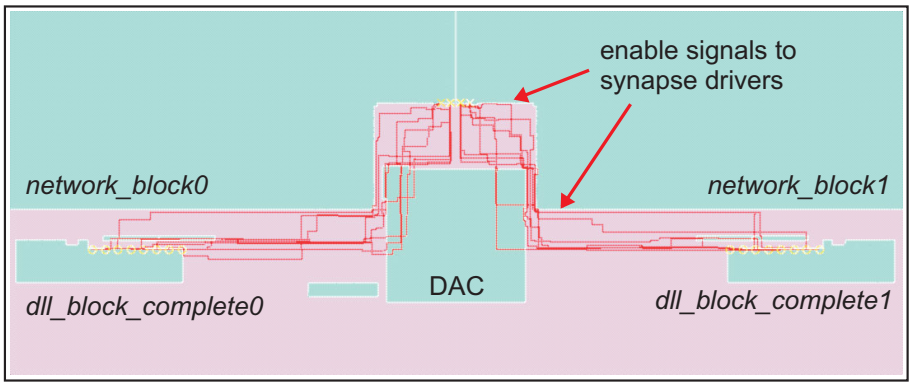


Figure 6.17: Illustration of the automated routing of the event enable signals between *dll_block_complete* and *network_block*. Each corner requires a via adding 6.5 Ω of resistance to the connection.

Shortcoming in the Implementation

In spite of the expected behavior, a problem is encountered during these measurements: the output driver of the DTC seems to be a bit too weak to drive the combined output load of the synapse address lines, the synapse driver gate capacitance within the *network_block*, and the *RC*-network added by NanoRoute during automated routing. As a result, the input of the synapse driver may not be driven over its threshold and consequently no event is generated. The reason for this error is that the driver has originally been designed to drive the load introduced by the *network_block* neglecting the *RC*-network added by automated routing. As no IPO operations are performed during automated routing of the analog part, also the timing checks are not performed and these violated paths are not discovered by the tools in spite of the correct definition of the according technology libraries. An illustration of the implementation result showing the automated routing is given in figure 6.17. A mixed-signal simulation confirming these conclusions is provided in appendix C.6. This shortcoming is already present on the first version of the chip. It was not observed until the testing of the second version because the interplay of the digital part and the DTCs was only tested up to $f_{\mathrm{c}} = 100\,\mathrm{MHz}$ at this point in time.

Since the event generation, in principle, is operational, correct event generation is possible at $f_{\mathrm{c}} = 100\,\mathrm{MHz}$ resulting in the temporal resolution determined above. The achievable event rates scale accordingly with a factor of 1/1.56 compared to the results obtained in section 6.5.3. Different workarounds are possible to obtain higher operating frequencies: first, the time bin information could be neglected and all events be generated at time bin 0 which results

in a temporal resolution of $(2 \cdot f_c)^{-1}$. Second, the maximum achievable time bin $t_{bin,max}$ could be determined depending on f_c. Thereupon, the software generating the playback memory content or the *EventRouter* module could cut off time bin values to this maximum which would introduce a maximum error of $t_{bin,max}$ time bins.

6.5.5 Event Digitization: Time-To-Digital

The following setup has been used to measure the accuracy of event digitization. The digital spike output of neuron # 1 in the left *network_block* is fed back as an input to synapse # 1 and is recorded as a presynaptic signal in the same manner as the external event input signal. Analog parameters are set as described in appendix A. Four excitatory synapses are used to generate one spike: Synapses # 0, # 64, # 128, and # 192. The error in the event capture logic renders a time bin accurate measurement of the TDCs DNL impossible because it cannot be deterministically said, whether the read back time stamp is correct, or not[80]. On this reason, the functionality of the mixed-signal interface between TDCs and the *event_buffer_out* modules is measured on a clock cycle basis in order to at least verify the correct functionality of the readout chain from TDCs to playback memory and software processing.

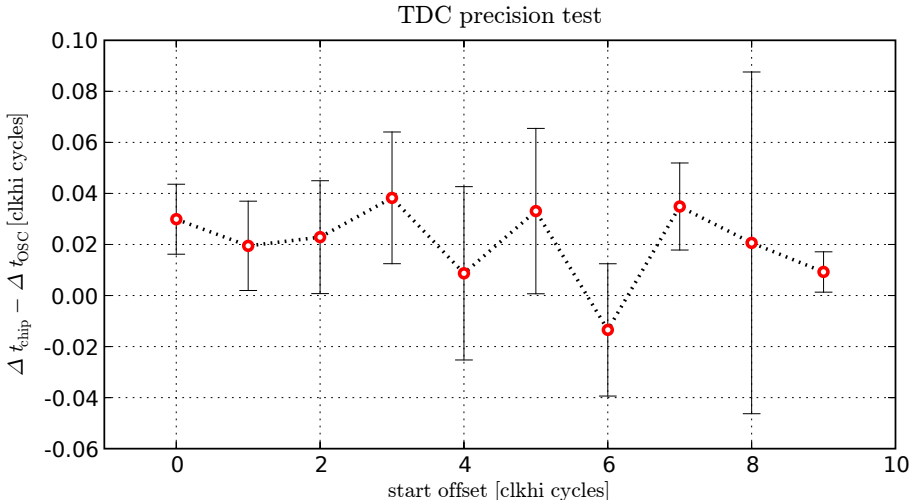


Figure 6.18: Measurement of the TDC precision on a clock cycle basis. Two events are induced by external input with a distance of 400 clock cycles. Both events are shifted 10 times by one clock cycle. The difference between the time difference measured by the chip and the time difference measured by the oscilloscope is plotted against this offset. The correct tracking of the system time during event digitization is proven by this result.

In this case, two events have been generated with a time distance of 6400 time bins. These events have been shifted by 16 time-bins 10 times (each shift equalling one `clkhi` cycle); the result is shown in figure 6.18, where the axes are now labeled in terms of `clkhi` cycles and error bars give the standard deviation of the mean. Potential errors due to false time stamps are supposed to be stochastically eliminated in this measurement because the input spikes are

[80] As described in section 4.3.5, the captured time stamp is too high by one `clkhi` cycle with a certain probability. This probability depends on the clock frequency, the process corner and the environmental conditions. To reliably work around this issue, the distribution of read back time stamps could be measured, and the erroneous time bins could be identified. However, this measurement requires an elaborate algorithm which has not been implemented so far. Therefore, no precise digitization of events is possible with the current setup.

generated with equal time stamps and thus the difference at the output will on average either contain the error for both events, or none. The results show an error well below one clock cycle with a maximum error of 0.07 clock cycles which equals 0.9 time bins. It is concluded, that the mixed-signal interface of the event capture logic on a clock cycle basis is fully functional. The functionality of the purely digital event capture under heavy load conditions has already been verified in section 6.5.2 with random event data using the event loopback test functionality. The event digitization is characterized on a time bin basis in the following measurement. It has been performed with the first version of the chip, Spikey 1, but the results are nevertheless valid for the second version, since neither the TDC implementation, nor the *event_buffer_out* module have been changed.

Checking for the Known Event Capture Error

The presence of the event capture error is demonstrated by a measurement performed with the first version of the chip. In this case, a different setup was used, since direct monitoring of the synapse driver's input is not available on Spikey 1. Two neurons N1 and N2 associated to different TDCs on the right *network_block* with physical addresses 321 and 387 are activated to measure the time distance between events generated by both of them. One excitatory synapse per neuron is used to generate exactly one output spike per input spike: synapse #510 is used for neuron N1 and serves as a reference for the time measurements. Synapse #446 is connected to neuron N2 and is used for the measurement event. The measurement starts with the two input events being generated simultaneously within time bin 0 of an arbitrary clock cycle. To cover two `clkhi` periods, the time stamp of the measurement event is increased 32 times by one time bin.

It is expected that digitized events having a time bin of approximately $13 \lesssim t_{\text{tbin,th}} < 16$ get a time stamp too large by one `clkhi` cycle. As a result, the expected measured time difference writes:

$$\Delta t_{\text{tbin,out}} = \Delta t_{\text{offset}} + \begin{cases} t_{\text{N1}} - (t_{\text{N2}} + 16) & \text{if } t_{\text{tbin,N2}} \gtrsim t_{\text{tbin,th}} \\ t_{\text{N1}} - t_{\text{N2}} & \text{else.} \end{cases} \tag{6.8}$$

The value of Δt_{offset} depends on the actual firing times of the two neurons, which slightly differ since no calibration routine for the different neuron circuits is yet available. It could without loss of generality be set to zero. The measured time distance of the digitized events is plotted against the theoretical input distance in figure 6.19. The error bars represent the standard deviation of the mean of the measured values; approximately 800 event pairs have been measured per data point[81]. A linear fit with unity slope has been placed through the data points proving the validity of the second row in equation 6.8.

Two points differ from the fit by several time bins. As indicated in the figure, the deviation equals 16 time bins within the error margin. The raw event data read back from the chip was not stored for this measurement, but the mean values were rather calculated by the software and recorded. Therefore, it is not possible to trace the exact time bins of t_{N1} and t_{N2} and on that account the location of the expected dips is random and cannot be predicted. Nevertheless, the existence of the error is supported by the good compliance with equation 6.8, when setting $t_{\text{tbin,th}} = 15$, since the distance of the dips equals 16 time bins. The data points directly following the dip do not fit to the unity slope; this is most likely caused by fluctuating events that are also digitized within time bin 15 for the respective data point.

[81] 1000 runs have been performed per point, and not always both neurons produced an output spike.

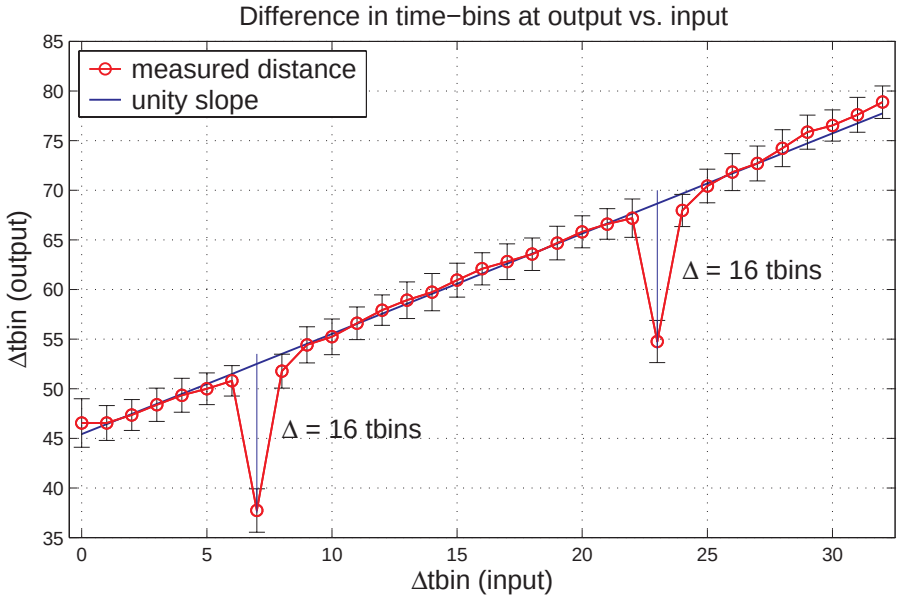


Figure 6.19: Producing the event capture error present in both versions of the Spikey chip. Two events are generated by external input and the difference between the digitized events is plotted while their difference is increased by one time bin for each data point. The occurrence of both dips with a distance of 16 time bins proves erroneously digitized events for one time bin value of $t_{\text{tbin,th}} = 15$.

To further investigate the occurrence of the error and determine an exact value for $t_{\text{tbin,th}}$, it would be required to record the raw event data and plot the observed frequency of the different clock cycle values in a histogram. Two peaks would be expected in this histogram with the peak of erroneously captured events begin shifted relative to the other, thereby determining all time bins above $t_{\text{tbin,th}}$.

The knowledge of the erroneous time stamps enables a workaround of this error: the time stamps can be corrected by subtracting one LSB of the clock cycle time stamp from every digitized event with a time bin value greater or equal $t_{\text{tbin,th}}$. This can either be done within the data path of incoming events within the *spikey_control* module when operating several chips within a system. For single chip experiments, this subtraction can automatically be done by the software tracking the global time of events read back from the playback memory. The accurate implementation first requires an automated determination of the faulty time bins for the desired clock frequency. It is subject to future work and with the current setup, a value of $t_{\text{tbin,th}} = 15$ can be assumed at a `chip_clk` frequency of 100 MHz.

6.6 Process Variation and Yield

The comparison of the measured delay of the *delaylines* against the simulated values already has shown that on the one hand, the delivered samples perform only slightly worse than it would be expected for the typical mean process corner—with the restrictions stated in section 6.2.3. A total of six out of the 60 obtained chips of the second version have been tested so far, and besides the thoroughly tested chips 2 and 5, the chips 3 and 4 exhibit the same performance. Because the dies were taken from random locations out of different covering boxes (waffle packs) these six samples at least give a glance on the process variations and

yield which are to be expected for the entire lot. This leads to the conclusion that the majority of the samples will be in the typical range.

Nevertheless, the first tested chip had a defect within the digital part. It was not possible to even get the link layer in bypass mode to work; the data transmitted by the chip was not correlated to the data present at its inputs. This behavior indicates a defect within the FIFO control logic which could not be investigated more precisely, since no direct access to this logic is possible. Another possible defect has been observed within chip 6 which deterministically produces false digitized output time stamps on bit # 3 of neurons <63:0> and <255:192>. This behavior is clock frequency independent and is therefore likewise interpreted as a defect. Neglecting the problems encountered with the synapse memory, this results in a yield of 67 % based on six samples.

Common to all functional samples are the intermittent errors encountered during testing of the synapse memory. As discussed in section 6.4.2, the reason is not yet clear. The behavior is frequency independent and only slightly improves with increased supply voltages. This also indicates problems with the power distribution to the according drivers. However, this cannot be said for sure, until a precise analysis of the power distribution network, along with a mixed-signal simulation of the interface have been performed. The intensive analysis of these intermittent errors is therefore required before considering a redesign of the chip, together with a review of the power distribution system.

Yield of the first version The measurements described above have likewise been conducted with the first version of the Spikey chip of which four samples have been tested. All of them were fully functional up to 160 MHz core and 320 MHz link clock frequency. The fact that the power distribution system as well as the average drive strength of the standard cells involved in the mixed-signal interface to the synapse memory did not change between the two versions is, on the one hand, in contrast to the assumptions made regarding insufficient power distribution. On the other hand, the power consumption introduced by the additional buffers within the interface data signals is to be considered. This power consumption could possibly lead to a voltage drop within the power distribution system which subsequently could cause the malfunction of the synapse memory.

6.7 Power Consumption

Digital Part

To verify the assumed power consumption of the digital part, which is estimated in section 4.4.3, it is measured under various conditions using chip 2 which are listed in table 6.2. The core clock frequency has been set to 156 MHz, and the link operates at 312 MHz double data rate. The voltage developed at a 1 Ω resistor is measured with a root mean square (rms) voltmeter[82] which has previously been calibrated by a test current through the resistor. This results in an error below 0.1 mA for all DC current measurements.

Immediately after power on, no termination resistors are active inside the FPGA, and only quiescent current is drawn by the chip. The 20 LVDS outputs of the Spikey chip get terminated after the FPGA has been configured and the current is raised by 75 mA, as expected based on the specified 3.5 mA per differential pair [ANS96] and the estimations given in table 4.17.

[82] The Philips PM 2525 digital multimeter has been used with an averaging period of 2 sec and an accuracy of ±0.01 mV for all but the *all off* measurement.

Mode	Current Drain [mA]
all off	$-0.03\mu A$
power on	16
FPGA configured (terminations on)	91
FPGA clocked, Spikey not clocked	137
Spikey clocked, reset off, bus idle	186
`CI_MODE` on, static data	171
`CI_MODE` on, random data	197
`CI_MODE` on, worst toggle rate	205
event loopback, bus idle packets	224
event loopback, event idle packets	218

Table 6.2: Current consumption of the digital part. Core clock frequency: 156 MHz, link clock frequency: 312 MHz. The different modes correspond to the power-up process for the chip.

When only clocking the FPGA while resetting the internal PLL of the chip, the current drawn solely by the input data lines and the link clock trees sums up to 46 mA. This supports the assumptions made on the large power consumptions of those elements and on the potential power distribution issues in section 6.3.

Within bypass mode (`CI_MODE` on) a strong dependence on the data pattern is observed. The current consumption varies by up to 34 mA depending on the activity on the data signals. The worst toggle rate is achieved by applying the bus idle pattern in bypass mode (every data signal toggles twice within one link clock cycle). Comparing to the case *Spikey clocked, reset off, bus idle* it can be concluded that the internal flip-flops contribute a current consumption of 19 mA.

The event loopback measurements are performed to estimate the maximum power consumption of the digital part, as this mode provides activity on all pipeline registers of the *event_loopack* module. No other access to the chip using random data makes more registers toggle simultaneously. The test is performed with the maximum achievable peak event rate (cf. section 6.5.2) and a playback memory setup that continuously looped over one cycle that was stored in memory. This results in a link utilization of approximately 80 %. The two measurements show the current consumed with bus idle packets or event idle packets in the remaining 15 % of the packet slots. The overall current consumption does not exceed 224 mA (403 mW) which is in accordance to the assumptions made in section 4.4.3. Compared to the bus idle packets, event idle packets introduce a slightly decreased toggle rate, which is the reason for the difference in current consumption.

To summarize: the power consumption of the digital part is dominated by the quiescent current of the LVDS transmitters and the dynamic power required by the large number of buffers required for the interface implementation. While the termination power is a matter of fact, the latter is a drawback from the source synchronous interface implementation. Since this implementation has proven to be a viable way of implementing a high-speed source synchronous interface it should be kept for further designs, but its power consumption could possibly be dropped by constraining the clock tree synthesizer to use smaller buffer cells than the currently used ones. In case a redesign is considered, also the power structures supplying these buffers should be revised to eliminate the potential problems described in section 6.3.

Analog Part

Regarding the analog power consumption, two parameters are of major interest: on the one hand, the quiescent current drawn by the buffers within the *curmem_vout* blocks[83]. On the other hand, there is the current drawn by the circuits connected to the current memory cells which drift to high values when not being refreshed [Scha].

Mode	V_{casdac} [V]	I_{refdac} [μA]	Current Drain [mA]
reset inactive, autom. refresh to 0 μA	0	0	32
parameters correctly configured	1.6	25	78

Table 6.3: Current consumption of the analog part.

The measured values are listed in table 6.3. The DAC has been biased with the values also used for regular operation, $V_{\text{casdac}} = 1.6\,\text{V}$ and $I_{\text{refdac}} = 25\,\mu\text{A}$. After releasing the reset, all current memories are automatically refreshed to $0\,\mu\text{A}$ by the digital part and the current consumption sums up to 32 mA with the internal DAC being switched off. This quiescent current is caused by the output buffers within *curmem_vout* and could be reduced by an optimized design. The current obtained for *correct* configuration has been measured after configuring the chip for the experiments described above with no network activity present. In this case, the quiescent current through the buffers could rise to up to 1.1 mA per buffer depending on the input voltage (cf. section 4.2.1). Moreover, the quiescent current drawn by the $50\,\Omega$ membrane potential output buffers reaches 2.8 mA when biased with $0.5\,\mu\text{A}$ and no external termination resistors present (data not shown). A total of 46 voltage output buffers and eight off-chip drivers are active—assuming a quiescent power of $10\,\mu\text{W}$ per neuron, the analog power consumption is obviously dominated by the buffers present on the chip.

Remark When configuring the *parameter_ram* module within the digital part in a way that it only refreshes a subset of the available current memories, one has to be aware that the remaining current memories will exhibit drifting behavior, which results in an unwanted power consumption and resulting self-heating of the chip. Therefore, care should be taken to always correctly—and completely—set up the parameter memory.

6.8 An Initial Biologically Realistic Experiment

The preceding sections dealt with the test and the verification of the Spikey chip in conjunction with its operating environment. In this section, a first experiment demonstrating the desired functionality of the neural network model is presented. The properties of a single neuron under Poisson distributed input bombardment are explored by comparing the membrane potentials generated by the chip and obtained by a simulation using NEST as a simulator under identical input. The setup of this experiment is a conjoint work with D. Brüderle, and the results have already been published in [BGM+07].

The Python interface described in section 5.3.3 has been utilized to configure both hardware and software, each with the same script. To fully exploit the capabilities of the hardware, the neuron has been set up to be connected to 200 excitatory and 50 inhibitory input synapses, and a randomly generated Poisson process spike train is applied to each synapse for a duration of two seconds with a fixed average rate of 3 Hz. These quantities are given in biological time.

[83] They cannot be biased to consume zero quiescent current, see section 4.2.1.

The synapses have been tuned such that the output firing rate of the neuron under bombardment approximately fits the 3 Hz input firing rate of one single input synapse. To compare the results obtained by the hardware recordings against the membrane trace generated by the NEST reference simulation, the distance between the output spike times of both models has been chosen as a measure as proposed in [vR01].

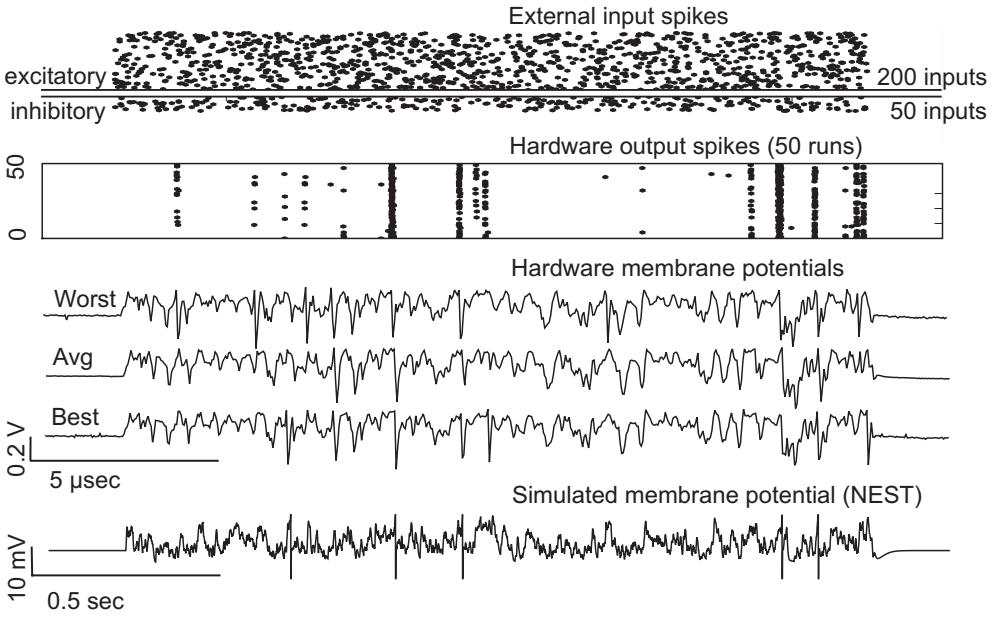


Figure 6.20: Poisson process input to one neuron and its resulting output. The output spike times of 50 identical experiments run on the hardware are compared to the output of the NEST software (see main text). Figure taken out of [BGM+07].

Figure 6.20 illustrates the results, thereby exposing the benefits of the high-level interface allowing for data acquisition from both the software and the hardware domain. The input data, as applied to the synapses is shown at the top, followed by the digitized and automatically recorded output spikes generated by the bombarded neuron on the Spikey chip over 50 runs with identical setup. Below, the analog membrane traces recorded with the PyScope software during the worst run, averaged over all runs, and during the best run are shown. The last trace illustrates the simulation result obtained with NEST, where the neuron was fed with the same input. An obvious correspondence between the membrane potential traces can be seen which is further supported by the plotted spike times.

This is a very promising result, as it shows, that the Spikey chip can be operated in a biologically realistic regime. More complex experiments will be conducted in the near future, in spite of the issues that have been unraveled by the measurements presented in the preceding sections.

Summary and Outlook

Within the scope of this thesis, a digital communication interface to the analog VLSI implementation of a spiking neural network model has been developed and integrated in a mixed-signal ASIC. Moreover, a hardware system enabling the transport of neural events between several neural network ASICs has been presented. While the inter-chip event transport has been verified by means of simulations, two versions of the ASIC were actually fabricated and their functionality has been successfully demonstrated.

The representation of the model within a single ASIC consists of 384 leaky integrate-and-fire neurons connected to 256 conductance based synapses each. Due to a large variety of adjustable parameters, the neuron model can be tuned to reflect the behavior of most cortical neuron types. Furthermore, the synapse models include an implementation of spike timing dependent plasticity (STDP) and short term synaptic depression and facilitation. Large scale neural networks composed of these elements are supposed to reach a sufficient complexity to enable the investigation of information processing in real brains, and especially the investigation of adaption mechanisms, or even learning processes within cortical areas. The emergence of these processes not only requires the correct modeling of the neural constituents, namely neurons and synapses, but also the correct modeling of the axonal interconnections between neurons and synapses.

On the one hand, the model equations given in section 1.2.1 cover the behavior of the synapses, the dendritic tree, and the neuron itself. They are realized by the analog VLSI implementation of the model. On the other hand, axonal connections are modeled by treating output spikes as stereotyped digital pulses. These can either be locally fed back to synaptic inputs, thereby providing constant axonal delay, or they can be digitized in time and be processed by digital logic. The need for arbitrary axonal connections and—even more important—arbitrary axonal delays required the development of appropriate routing techniques which account for the correct spatio-temporal processing of the events within the modeled neural network.

Thereby, one advantage of the VLSI model from biology especially challenges the realization of large scale neural networks: its operating speed. Since silicon devices like capacitors and transistors, which serve as a basis for the physical model, are realized close to the minimum feature size of the utilized 180 nm CMOS process, time constants of the model are shrunk resulting in a speed-up factor of 10^5 for its continuous-time operation. Asynchronous digital communication techniques like the address event representation (AER) protocol do not provide the required temporal accuracy of 0.1 ms biological time or 1 ns model time respectively. For these reasons, events get a time stamp along with their physical address facilitating the temporal processing of events relative to a global system time.

The presented achievements are twofold covering the transport of events between several chips and the implementation of the mixed-signal ASIC Spikey. Both aspects will be summarized in the following.

Event Routing The isochronous transport network utilized for the event transport provides fixed-delay connections with a guaranteed latency. Since these connections are set up prior to the network operation, no handshaking is required and event data is directly forwarded by the transport network with constant throughput. Based upon this connectivity *virtual connections* have been introduced between different neural network chips in order to model axonal connectivity. All connections between two chips sharing identical axonal delays are bundled into one virtual connection. This bundling has to be performed by a suitable algorithm which maps the neural network to be simulated to the hardware topology. For these purposes, a mapping algorithm is currently being developed [EME+06, Phi07]. A drawback of this realization is the limitation of the diversity of axonal delays to the number of feasible virtual connections in the entire network. Four major features of this algorithm can be summarized:

- Optimum Bandwidth Utilization:
  A customized version of the leaky bucket algorithm has been implemented at the source of each virtual connection. For each event the algorithm determines, whether it will be delivered in time or not and drops the event accordingly. By this means unnecessary event transmissions are avoided during overload scenarios and the bandwidth of the transport network is optimally utilized (cf. section 3.5.1).

- Temporal Sorting:
  The problem of temporal sorting the event streams of multiple virtual connections at the destination is partly already solved at the source: the dispatch of events is delayed until the latest possible point in time, thereby assuring arrival of events at the destination with only slightly differing time stamps. It has been shown that temporal sorting within a small window can be accomplished using only few programmable logic resources and event drop rates below 1 % are achieved at the destination even under maximum load conditions (cf. section 3.5.1).

- Scalability:
  Connections can be established between chips within a daisy chain and between chips connected to different daisy chains. Thereby, several daisy chains of chips may be connected to a single switch of the transport network and to switches located on different Nathan modules interconnected by the MGT network. The maximum number of virtual connections is only limited by the available logic resources and the feasibility of a first experiment comprising four chips has been demonstrated by simulations and a resource estimation.

- Axonal Delay:
  While the maximum axonal delay is only limited by the memory resources required for delaying the events, the minimum axonal delay $t_{\mathrm{d,min}}$ depends on the topology of the transport network, the clock frequency and the speed-up factor of the model. At a clock frequency of 156 MHz and a speed-up of 10^5, minimum biological delays in the range $5.76\,\mathrm{ms} \leq t_{\mathrm{d,min}} \leq 23.68\,\mathrm{ms}$ are feasible. More biologically viable values can be obtained by reducing the speed-up to 10^4: $0.58\,\mathrm{ms} \leq t_{\mathrm{d,min}} \leq 2.37\,\mathrm{ms}$.

In section 3.5.2 it has been shown that the algorithm performs stable and handles peak event rates expected during synchronized or bursting network activity. The simulated network comprises 1000 neurons spread over four chips interconnected by the MGT network. Average event rates of 3 Hz and 30 Hz per neuron are handled by the network for speed-up factors of 10^5 and 10^4 respectively. This is a drawback arising from the bandwidth currently available

from the MGT network: the speed-up factor of the model operation has to be reduced to 10^4 by means of tuning analog parameters, since the event rates during synchronized activity could not be transported at a speed-up of 10^5.

Mixed-Signal Chip Design The digital event processing logic and the physical interface of the Spikey chip, along with auxiliary digital control functionality, have been developed to meet the specifications from the analog VLSI model. The digital functionality was successfully verified in section 6.4 and the according specifications are met with the following restriction: due to a critical path within the event capture logic, the speed of the entire design is limited to operating frequencies of 156 MHz for the digital part and 312 MHz for the event generation and digitization logic. This is below the actual specification of 200 MHz and 400 MHz respectively. As explained in section 4.4.4, no additional registers were inserted into the critical path in order to achieve a low latency for the event capture in terms of number of clock cycles. A positive result is that the chip can still be operated at the maximum operating frequency of the MGT network which equals 156 MHz.

Regarding the interface to the chip itself, peak event rates of 24.4 Hz per synapse/neuron are feasible utilizing all synapses and neurons present on the chip (cf. section 4.3.7). Average event rates of 9.14 Hz and 12.2 Hz can be achieved per synapse and per neuron respectively when using all synapses/neurons in parallel with identical rates (all in biological time considering a speed-up of 10^5). Assumed that a fraction of the synapses will be reserved for local feedback, this bandwidth can be shared among fewer synapses. As a satisfying result, the specification to allow for an average biological firing rate of 10 Hz has been met. The following three items have been emphasized during development:

- Mixed-Signal Design Flow:
  A design flow for purely digital ASICs has been customized. Using a convenient and flexible set of scripts, the entire mixed-signal chip assembly is carried out automatically and reproducible resulting in a “Make Chip” design flow. It comprises the integration of the analog VLSI design data, digital front and back end, and the physical as well as the functional verification. The design flow yielded functional ASICs during two design cycles while greatly facilitating the second tape-out due to the gained experience. Minor improvements to the design flow will be suggested below.

- Physical Interface Design:
  A new method for the implementation of high-speed source synchronous interfaces has been proposed and tested in silicon. Without phase adjusting elements like a PLL or DLL, bit rates of 640 Mbit/s could be achieved (cf. section 6.3). In order to deskew the interface data signals, delay elements have been realized that could automatically be realized using the automated design flow. A simple standard cell based approach has been chosen and its functionality has been verified in section 6.2.3.

- Event Transport Logic and Digital Design:
  A protocol has been introduced accounting for the packet based communication of event and control interface data with the chip. The chip can be synchronized to a global system time and correct event delivery was verified up to the DTCs within the analog part. Unfortunately, the incorrect implementation of the interface between TDCs and digital part potentially causes the digitized events to have a temporal error of one clock cycle (cf. section 6.5.5). A possible workaround has been proposed. By means of this

workaround correct event generation and digitization is assured and the temporal error will be reduced to the nonlinearities of the DTCs and TDCs.

Operating Environment In order to operate the chip a controller has been implemented in programmable logic and the according hardware substrates have been described in chapter 5. Low level software has been presented encapsulating the access to the hardware, thereby providing a convenient high-level interface to the user. In section 6.5.1 it has been demonstrated that time tracking for the entire system is successfully implemented and the generation and recording of neural events is possible by means of the introduced playback memory. Along with a high-level software interface, it has been shown in section 6.8 that the chip can be operated in a biologically realistic regime.

Outlook Based upon the obtained results, it can be stated that the Spikey chip as well as the entire system provides now the possibility to tackle first single chip experiments exploiting the full potential of the Spikey chip. Nevertheless, a total of three issues have been encountered which would suggest the design of a third version of the chip. First, there is the error within the event capture logic; it is easily resolved by correcting the according Verilog source. Second, there are the errors encountered sporadically during synapse memory testing. These require further testing and are potentially related to the third issue which is the reduced width of the data valid window at the input of the Spikey chip. In section 6.3 it has been supposed that a reason for this behavior could be an undersized PDN.

Apart from the logic error, these issues are due to the automated design flow being imperfect in two aspects: first, the mixed-signal interfaces between analog and digital part could not automatically be tested. Second, the PDN of the chip could not be verified for dynamic power dissipation. In the opinion of the author, both have to be integrated into the design flow before starting a redesign. Moreover, STA results should be verified using a third party *RC*-extraction software (cf. section 6.2.3).

The most important step towards large-scale spiking neural networks will be the actual implementation of the event routing algorithm within programmable logic. During the writing of this thesis parts of the algorithm have been realized in VHDL [Schb]. Timing closure could be achieved on the *EventOutQueue* module and it has been nice to see that the resource consumption was over-estimated. Therefore, even larger network topologies can possibly be realized on the existing hardware platform. However, two approaches are conceivable to scale the network to more than presumably six chips: first, different neural network topologies could be selected requiring fewer global connections resulting in fewer virtual connections. Second, larger FPGAs would provide the required logic resources, which could for example be implemented on a future daughter card to the Nathan module carrying multiple Spikey chips hosted by an additional FPGA.

Both approaches will enable the setup of neural networks consisting of several thousands of neurons and more than a million of synapses. Consequently, the presented system will eventually become a tool to complement software simulations of moderately sized cortical microcircuits including synaptic plasticity and cellular diversity. Comparably new aspects of modeling recurrent neural networks like *liquid state machines*, as proposed for example in [MNM02], could be realized in real neuromorphic hardware. Moreover, the system constitutes the *Stage I* hardware of the FACETS project, an interdisciplinary research project aiming to model neural microcircuits from the primary visual cortex in analog VLSI as well as in

numerical simulations. Thereby, the goal of the FACETS project is to create a theoretical and experimental foundation for the realization of novel computing paradigms which exploit the concepts experimentally observed in biological nervous systems [Mea05].

However, the size of neural microcircuits that can be realized with the FACETS *Stage I* hardware will still not exceed approximately 10^4 neurons, even when using huge FPGAs, due to the resource consumption of the required virtual connections. Therefore, a different approach is followed for the *Stage II* hardware: in order to obtain the necessary high connectivity between the analog VLSI models, the crossover of the digital signals from the die to a PCB is avoided by performing *wafer-scale* integration of the circuits. The wafer is not cut into single dies but the circuits on the wafer are rather directly interconnected by postprocessing the wafer as a whole. A pitch of 4 μm is feasible providing the required high connectivity, even for local routing of the network traffic. Parts of the presented system will be reused for the *Stage II* hardware. Besides the analog circuitry, it is especially the experiences gained regarding the design flow that will be of benefit for the wafer-scale integration techniques. Furthermore, parts of the event routing algorithm will be used for the long-range communication between wafers and peripheral components required for monitoring the entire system.

New challenges and possibilities will arise from the *Stage II* system: one wafer will presumably integrate more than 10^5 neurons and 10^8 synapses and it is even planned to interconnect several wafers. Fault-tolerance mechanisms have to be developed and new techniques for the postprocessing will have to be explored. Together with the speed-up of 10^4 which is aimed for the operation, this system will be a powerful information processing device. It will enable the biologists to perform experiments at great speed. For example could the recorded activity of the moving fovia be taken as an input to the system providing the possibility to investigate the emerging adaption processes within the simulated cortical area. In contrast to software simulations that will likely run slower than real time, the wafer-scale system will enable fast iterations and parameter searches. Apart from the benefits this system will bring to neuroscience, it will possibly fulfill another goal of the FACETS project: finding ways of information processing beyond the Turing paradigm.

Acronyms

ADC analog-to-digital converter

AER address event representation

ANN artificial neural network

ASIC application specific integrated circuit

BER bit error rate

CMOS complementary metal oxide semiconductor

CTS clock tree synthesis

CQFP ceramic quad flat pack

DAC digital-to-analog converter

DC direct current

DCM digital clock manager

DDR double data rate

DEF data exchange format

DLL delay-locked loop

DNL differential nonlinearity

DRC design rule check

DTC digital to time converter

ESD electrostatic discharge

FIFO first-in first-out

FPGA field programmable gate array

FSM finite state machine

HAL hardware abstraction layer

HDL hardware description language

IPO in-place optimization

JTAG joint action test group

LEF library exchange format

LSB least significant bit

LUT look-up table

LVDS low voltage differential signaling

LVS layout versus schematic

MGT multi-gigabit transceiver

MPW multi-project wafer

MSB most significant bit

NMOS n-type metal oxide semiconductor

PCB printed circuit board

PDN power distribution network

PGA pin grid array

PLL phase-locked loop

PSP postsynaptic potential

QoS quality of service

rms root mean square

SI signal integrity

SMT surface mount technology

SoC system on chip

SRAM static random access memory

STA static timing analysis

STDP spike timing dependent plasticity

STL standard template library

TCL tool command language

TDC time to digital converter

UMC United Microelectronics Corporation

VHDL VHSIC hardware description language

VHSIC very high speed integrated circuit

VLSI very large scale integration

VST Virtual Silicon Technology

Appendix A

Model Parameters

Available Model Parameters

This section lists all model parameters that can be adjusted on the chip together with a short explanation of their meaning. Table A.2 lists the available bias currents. Table A.4 lists the available voltage paremeters including voltages that need to be supplied externally. Table A.6 lists all parameters with their physical address on the Spikey chip. These are the addresses that need to be written to the parameter memory during setup.

Parameter	Address Offset	Meaning
V_{dtc0}, V_{dtc1}, V_{dtc2}, V_{dtc3}	0..3	short term plasticity time constant for spike history, higher current ⇒ shorter averaging windowtime, connected to even and odd neurons on both halves.
V_{cb0}, V_{cb1}, V_{cb2}, V_{cb3}	4..7	spike driver comparator bias current, connected to even and odd neurons on both halves.
V_{plb0}, V_{plb1}, V_{plb2}, V_{plb3}	8..12	spike driver pulse length bias, higher current ⇒ shorter internal pulse, important for short term plasticity, connected to even and odd neurons on both halves.
$I_{\text{bnoutampba}}$, $I_{\text{bnoutampbb}}$	13..14	both add together to the neuronoutampbias, global for the chip.
$I_{\text{bcorrreadb}}$	15	correlation read out bias, global for the chip.

Table A.2: List of current parameters. The address offset gives the offset relative to the starting address of the according *vout_block* in table A.6.

Parameter	Address Offset	Meaning
E_{i0}, E_{i1}	0, 2	Inhibitory reversal potential, even and odd neuron addresses.
E_{l0}, E_{l1}	4, 6	Leakage reversal potential, even and odd neuron addresses.
E_{r0}, E_{r1}	8, 10	Reset potential, even and odd neuron addresses.
E_{x0}, E_{x1}	12, 14	Excitatory reversal potential, even and odd neuron addresses.
V_{clra}	16	Storage clear bias for entire synapse array, acausal (higher bias $\Rightarrow$ smaller amount stored on cap)
V_{clra}	18	As above, causal
V_{cthigh}	20	Correlation readout threshold high value
V_{ctlow}	22	Correlation readout threshold low value
V_{fac0}, V_{fac1}	24, 26	Short term facilitation reference voltage, even and odd synapse columns.
V_{stdf0}, V_{stdf1}	28, 30	Short term capacitor high potential, even and odd synapse columns.
V_{th0}, V_{th1}	32, 34	Neuron threshold voltage, even and odd neuron addresses.
$V_{casneuron}$	36	Neuron input cascode gate voltage
$V_{resetdll}$	38	DLL reset voltage
$ARO_{dllvctrl}$	16	DLL control voltage readout
ARO_{pre1b}	16	Directly monitor presynaptic signal after delay element of synapse # 1
$ARO_{selout1hb}$	16	Directly monitor presynaptic signal after multiplexer of synapse # 1
VM	ext	externally supplied, synapse array correlation measurement parameter (precharge voltage of measurement cap.)
VREST	ext	externally supplied, synapse driver resting potential
VSTART	ext	externally supplied, synapse driver start potential # 1

Table A.4: List of voltage parameters. The address offset gives the offset relative to the starting address of the according *vout_block* in table A.6. The last three parameter voltages need to be supplied externally.

Location	Physical Address	Parameter Name	Meaning
left synapse drivers	0	drviout	Max. synapse driver output current. Scales the shape of the postsynaptic pulse.
	1	adjdel	Delay for presynaptic correlation pulse. Can be used to calibrate STDP measurements for signal propagation delay within the synapse array.
	2	drvifall	Sets slew rate for falling edge of presynaptic signal.
	3	drvirise	Sets slew rate for rising edge of presynaptic signal.
	...	...	...
	1020 1021 1022 1023	drviout adjdel drvifall drvirise	see above
right synapse drivers	1024 ... 2047	...	right —same as for left side.
left neuron circuits	2048	neurileak	Bias for neuron leakage circuit. Controls g_{leak}, thus, I_{leak}.
	2049	neuricb	Bias for neuron V_{th} comparator. Controls dependency of spike generation on $\dot{V}_{\text{th}}$
	...	...	...
	2430 2431	neurileak neuricb	see above
right neuron circuits	2432 ... 2815	...	right —same as for left side.
left parameter voltages	2816	$I_{\text{bias},0}$	Bias for output buffer within . Total 23.
	2817	$V_{\text{param},0}$	Parameter voltage. The different voltages are listed in table A.4. Total 20.
	...	...	...
	2860 2861	$I_{\text{bias},22}$ –	see above
right parameter voltages	2880 ... 2927	...	right side—same as for left side.
bias currents	2944 ... 2958	...	Miscellaneous bias currents. Listed in table A.2
bias currents	2959 ... 2966	...	Bias currents for 50 Ω membrane potential output buffers. Total 8.
	2967	IBTEST	Current memory directly connected to IBTEST pin.

Table A.6: List of analog parameters available in each column and row circuit with the according physical addresses. Available voltage parameters are listed separately in table A.4.

Measurement Model Parameters

Table A.8 lists the model parameter settings relevant for the measurements presented in chapter 6. They are used as a standard setup for experiments that require neurons to produce an output spike upon only one or a few input spikes.

Parameter	Value
E_{i0}, E_{i1}	0.6 V, 0.7 V
E_{l0}, E_{l1}	0.6 V, 0.6 V
E_{r0}, E_{r1}	0.3 V, 0.3 V
E_{x0}, E_{x1}	1.7 V, 1.7 V
V_{clra}	1.55 V
V_{clra}	1.55 V
V_{cthigh}	0.89 V
V_{ctlow}	0.8 V
V_{fac0}, V_{fac1}	1.1 V, 1.1 V
V_{stdf0}, V_{stdf1}	1.6 V, 1.6 V
V_{th0}, V_{th1}	1.1 V, 1.15 V
$V_{casneuron}$	1.6 V
$V_{resetdll}$	0.8 V
$V_{dtc0..3}$	0.2 μA
$V_{cb0..3}$	0.5 μA
$V_{plb0..3}$	0.5 μA
$I_{bnoutampba}$	0.1 μA
$I_{bnoutampbb}$	0.4 μA
$I_{bcorrreadb}$	0.6 μA

Table A.8: List of voltage and current parameters that have been set for the measurements presented in chapter 6. The voltage parameters were set identically for both halves of the chip.

Appendix B

Communication Protocol and Data Format

This appendix is dedicated to the description of selected control interface packets that are commonly used for configuration, operation and monitoring of the Spikey chip. The synchronization functionality as well as the specific content of the event packet have been described in detail in section 4.3.2; they are included here for reasons of completeness.

The commands for the access to the parameter memory and the synapse memory control modules contain 4 bit subcommands. These are explained as appropriate, alongside with the according content of the control interface packet.

The content of the packet is displayed in a way that it corresponds to the physical data on the interfaces. This way of illustration has proven to be useful, especially during low-level debugging of the physical interface. It provides an intuitive arrangement of the data in the same order as it is clocked into/out of the chip on the single signals.

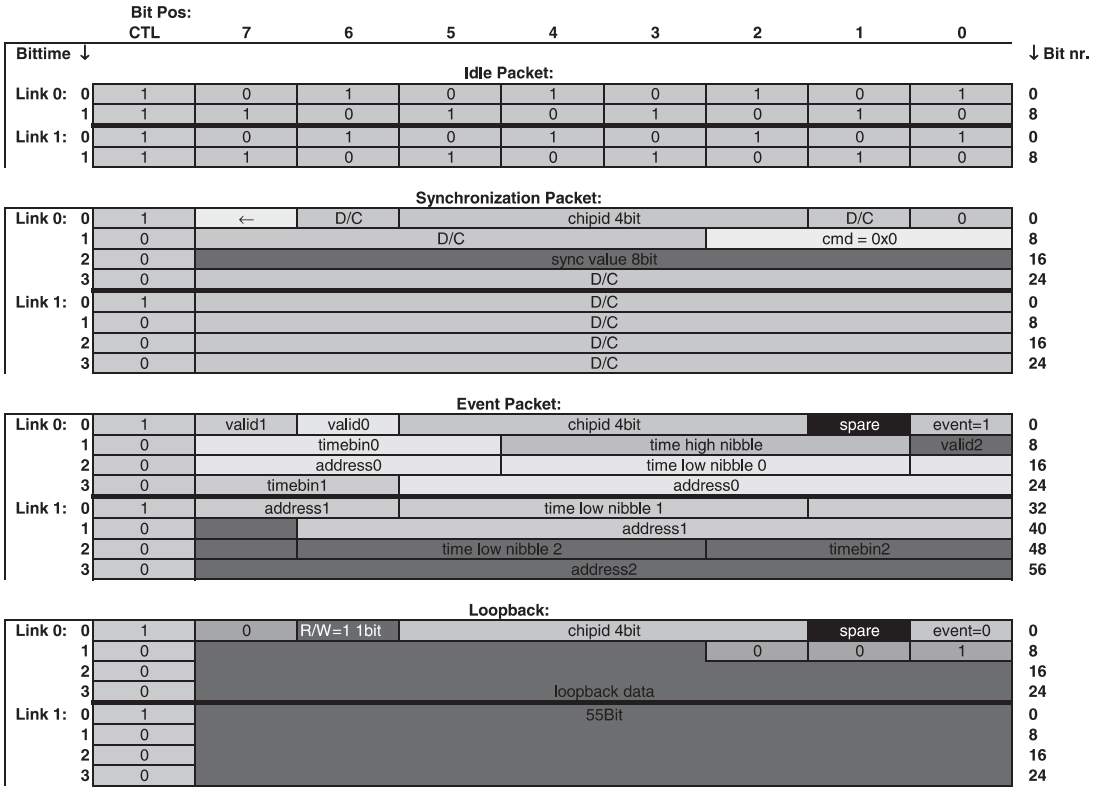
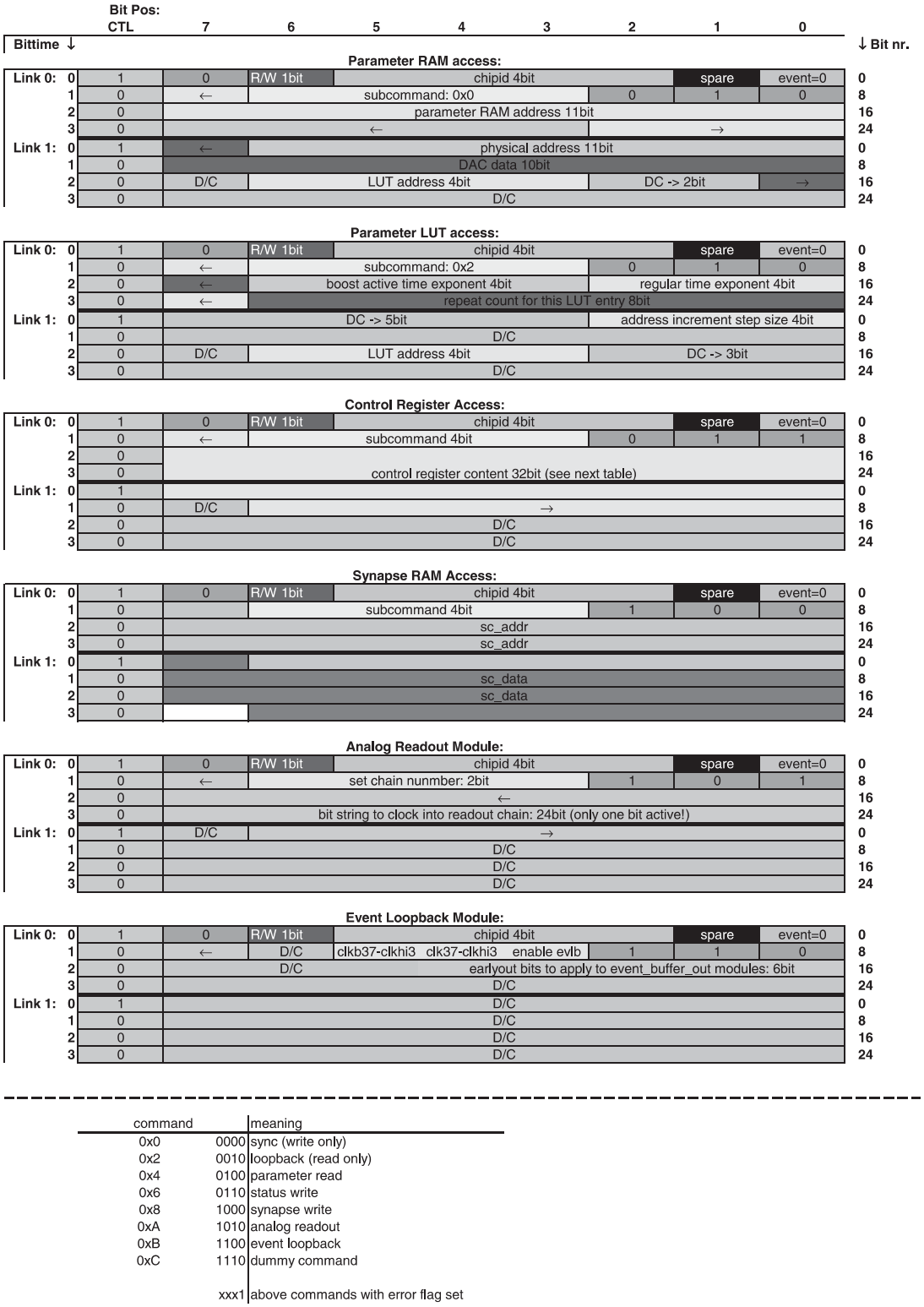


Table B.1: Compilation of the packet content for the listed control interface commands.



command		meaning
0x0	0000	sync (write only)
0x2	0010	loopback (read only)
0x4	0100	parameter read
0x6	0110	status write
0x8	1000	synapse write
0xA	1010	analog readout
0xB	1100	event loopback
0xC	1110	dummy command
	xxx1	above commands with error flag set

Table B.2: Compilation of the packet content for the listed control interface commands (continued).

Control and Status Register Access

subcommand

bit no.	read/write control (0x0)	read chip_clk status (0x1)	read chip_clkb status (0x2)	read clkhi status (0x3)
0	unused	event_buffer_in<0> status: FIFO full	event_buffer_in<8> status: FIFO full	event_buffer_out<0> status: FIFO full
1		dito: FIFO almost full	dito: FIFO almost full	dito: FIFO almost full
2		dito: FIFO half full	dito: FIFO half full	dito: FIFO half full
3		dito: FIFO error	dito: FIFO error	dito: FIFO error
4		event_buffer_in<1> status	event_buffer_in<9> status	event_buffer_out<1> status
5	enable event error packet generation	...	...	...
6	earlyout debug mode			
7	enable anaclk and anaclkb			
8	enable anaclkhi	event_buffer_in<2> status	event_buffer_in<10> status	event_buffer_out<2> status
9	enable parameter memory update	...	...	...
10	event_buffer_in<0> reset			
11				
12	...	event_buffer_in<3> status	event_buffer_in<11> status	event_buffer_out<3> status
13		...	...	...
14				
15				
16		event_buffer_in<4> status	event_buffer_in<12> status	event_buffer_out<4> status
17	event_buffer_in<8> reset	...	...	...
18				
19				
20	...	event_buffer_in<5> status	event_buffer_in<13> status	event_buffer_out<5> status
21		...	...	...
22				
23				
24	event_buffer_in<15> reset	event_buffer_in<6> status	event_buffer_in<14> status	clkhi_pos<7:0>
25	event_buffer_out<0> reset	...	...	
26				
27	...			...
28		event_buffer_in<7> status	event_buffer_in<15> status	
29		...	...	
30				
31	event_buffer_out<5> reset	clk_pos<6:0>	clkb_pos<6:0>	current packet high nibble <3:0>
32				...
33				
34				
35	unused	...	...	
36				PLL_LOCKED
37				unused
38				

Table B.3: Control and status register content in dependence of the four possible subcommands. Only the control register can be written. The status registers are read only.

Appendix C

Implementation Supplements

C.1 Spikey Pinout

This page is intentionally left blank. A graphical illustration of the pinout of the Spikey chip is given on the following page. Subsequently, the pin names are listed together with their meanings.

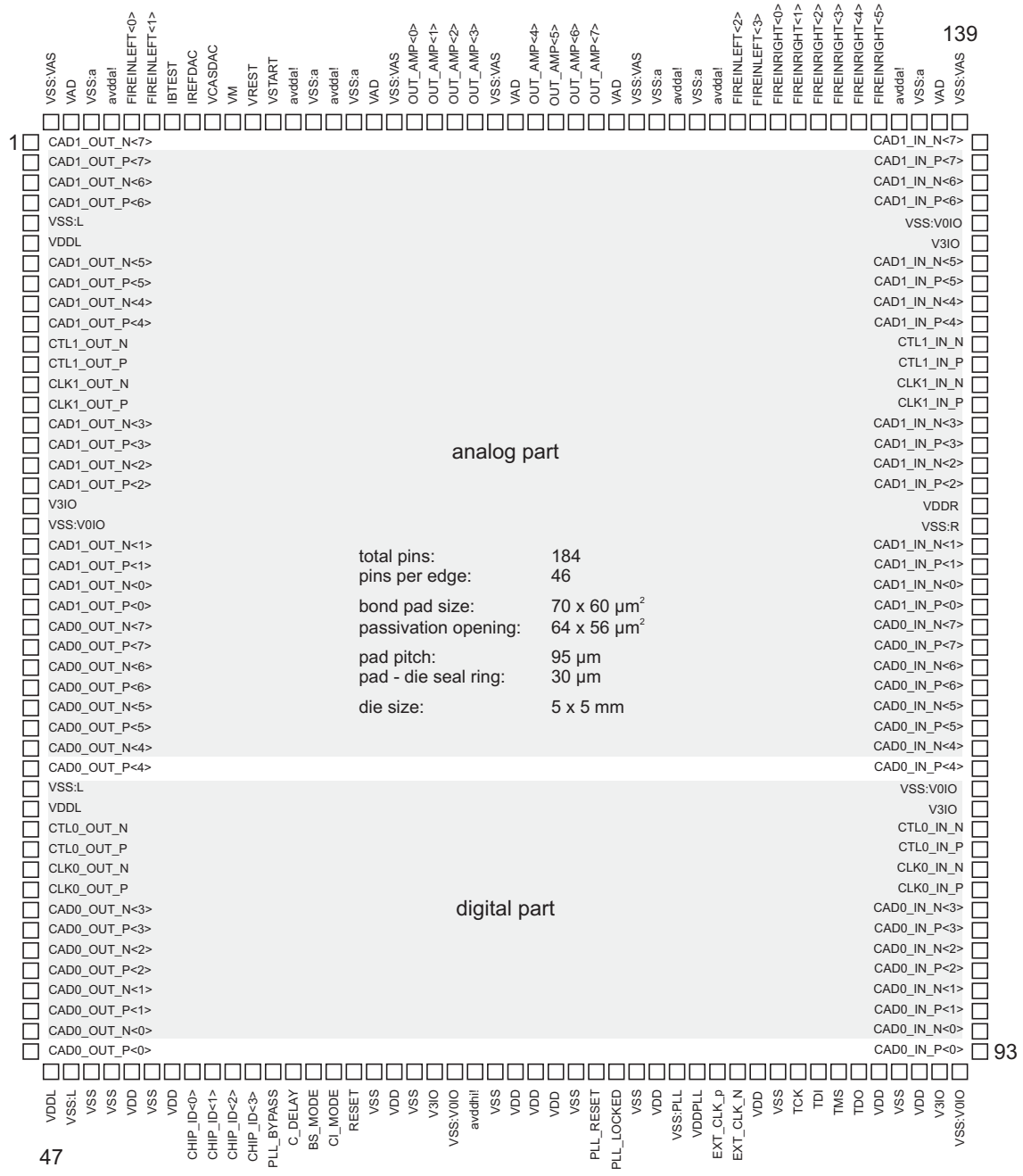


Figure C.1: Graphical illustration of the Spikey chip pinout.

Pad No.	Pin Name	Type	Description
32-25, 46-39	CAD0_OUT_P<7:0>, CAD0_OUT_N<7:0>	output	Output data bus link 0, connect to corresp. input data bus on subseq. chip in chain
36, 35	CTL0_OUT_P, CTL0_OUT_N	output	output frame bit link 0. – see above –
38, 37	CLK0_OUT_P, CLK0_OUT_N	output	output clock signal link 0. – see above –
4-1, 10-7, 18-15, 24-21	CAD1_OUT_P<7:0>, CAD1_OUT_N<7:0>	output	Output data bus link 1. – see above –
12, 11	CTL1_OUT_P, CTL1_OUT_N	output	output frame bit link 1. – see above –
14, 13	CLK1_OUT_P, CLK1_OUT_N	output	output clock signal link 1. – see above –
93-100, 107-114	CAD0_IN_P<7:0>, CAD0_IN_N<7:0>	input	Input data bus link 0. Connect to corresp. output data bus on prev. chip in chain
103, 104	CTL0_IN_P, CTL0_IN_N	input	input frame bit link 0. – see above –
101, 102	CLK0_IN_P, CLK0_IN_N	input	input clock signal link 0. – see above –
115-118, 121-124, 129-132, 135-138	CAD1_IN_P<7:0>, CAD1_IN_N<7:0>	input	Input data bus link 1. – see above –
127, 128	CTL1_IN_P, CTL1_IN_N	input	input frame bit link 1. – see above –
125, 126	CLK1_IN_P, CLK1_IN_N	input	input clock signal link 1. – see above –
54-57	CHIP_ID<3:0>	input	static chip's address in chain.
80, 81	EXT_CLK_P, EXT_CLK_N	input	chip's core clock, needs to be the same for each chip within one chain
84-86	TCK, TDI, TMS	input	JTAG interface.
87	TDO	output	JTAG interface.
60	BS_MODE	input	Boundary scan mode.
61	CI_MODE	input	Command interface mode.
59	C_DELAY	input	Config interface delay lines.
58	PLL_BYPASS	input	Bypass internal PLL.
75	PLL_LOCKED	output	Internal PLL locked state, open drain.
74	PLL_RESET	input	Reset internal PLL.
62	RESET	input	Chip reset.
166-163, 160-157	OUT_AMP<0:7>	analog output	analog signals to monitor membrane voltages.

178	IBTEST	analog output	analog monitor pin, output of *analog_readout*
177	IREFDAC	reference current	DAC reference current, generate externally.
176	VCASDAC	bias voltage	DAC bias voltage, generate externally.
175-173	VM, VREST, VSTART	parameter voltage	reference/parameter voltages, generate externally.
180, 179, 150, 149	FIREINLEFT<0:3>	input	digital inputs directly connected to synapse row drivers.
148-143	FIREINRIGHT<0:5>	input	
51, 53, 64, 70-72, 77, 82, 88, 90	VDD	power	1.8 V digital core supply
49, 50, 52, 63, 65, 69, 73, 76, 83, 89	VSS	ground	digital core ground
6, 34, 47	VDDL	power	1.8 V LVDS logic supply, transmit side (left)
5, 33, 48	VSS:L	ground	LVDS logic ground, transmit side (left)
120	VDDR	power	1.8 V LVDS logic supply, receive side (right)
119	VSS:R	ground	LVDS logic ground, receive side (right)
19, 66, 91, 105, 133	V3IO	power	3.3 V digital I/O supply
20, 67, 92, 106, 134	VSS:V0IO	ground	digital I/O ground
142, 151, 153, 170, 172, 181	avdda!	power	1.8 V Analog core supply
141, 152, 154, 168, 171, 182	VSS:a	ground	analog core ground
68	avddhi!	power	3.3 V analog supply (DAC only)
140, 156, 161, 168, 183	VAD	power	1.8 V analog I/O supply
139, 155, 162, 167, 184	VSS:VAS	ground	analog I/O ground
79	VDDPLL	power	1.8 V PLL supply (analog)
78	VSS:PLL	ground	PLL ground (analog)

Table C.1: Pinout of the Spikey chip. The pin numbers correspond to the numbers in figure C.1. To obtain the pin numbers for the CQFP208 package, the empty pins at the edges have to be considered (cf. figure E.1).

C.2 Pin Mapping Nathan-Spikey

The following listing defines the mapping of the signals on the Nathan PCB to the Spikey chip. The syntax is pseudo-VHDL and the signals left of the arrows are assigned to inputs. The last column defines the I/O standard to use in the FPGA.

```
ANNA_BUS_N(25) <= CAD0_OUT_P(7)   ANNA_BUS_P(25) <= CAD0_OUT_N(7)   LVDS_25_DT
ANNA_BUS_P(24) <= CAD0_OUT_P(6)   ANNA_BUS_N(24) <= CAD0_OUT_N(6)   LVDS_25_DT
ANNA_BUS_N(23) <= CAD0_OUT_P(5)   ANNA_BUS_P(23) <= CAD0_OUT_N(5)   LVDS_25_DT
ANNA_BUS_P(22) <= CAD0_OUT_P(4)   ANNA_BUS_N(22) <= CAD0_OUT_N(4)   LVDS_25_DT
ANNA_BUS_N(13) <= CAD0_OUT_P(3)   ANNA_BUS_P(13) <= CAD0_OUT_N(3)   LVDS_25_DT
ANNA_BUS_N(10) <= CAD0_OUT_P(2)   ANNA_BUS_P(10) <= CAD0_OUT_N(2)   LVDS_25_DT
ANNA_BUS_P( 0) <= CAD0_OUT_P(1)   ANNA_BUS_N( 0) <= CAD0_OUT_N(1)   LVDS_25_DT
ANNA_BUS_P(11) <= CAD0_OUT_P(0)   ANNA_BUS_N(11) <= CAD0_OUT_N(0)   LVDS_25_DT
ANNA_BUS_P(12) <= CTL0_OUT_P      ANNA_BUS_N(12) <= CTL0_OUT_N      LVDS_25_DT
ANNA_BUS_P( 6) <= CLK0_OUT_P      ANNA_BUS_N( 6) <= CLK0_OUT_N      LVDS_25_DT

ANNA_BUS_P(35) <= CAD1_OUT_P(7)   ANNA_BUS_N(35) <= CAD1_OUT_N(7)   LVDS_25_DT
ANNA_BUS_P(32) <= CAD1_OUT_P(6)   ANNA_BUS_N(32) <= CAD1_OUT_N(6)   LVDS_25_DT
ANNA_BUS_P(31) <= CAD1_OUT_P(5)   ANNA_BUS_N(31) <= CAD1_OUT_N(5)   LVDS_25_DT
ANNA_BUS_P(30) <= CAD1_OUT_P(4)   ANNA_BUS_N(30) <= CAD1_OUT_N(4)   LVDS_25_DT
ANNA_BUS_N(29) <= CAD1_OUT_P(3)   ANNA_BUS_P(29) <= CAD1_OUT_N(3)   LVDS_25_DT
ANNA_BUS_P(26) <= CAD1_OUT_P(2)   ANNA_BUS_N(26) <= CAD1_OUT_N(2)   LVDS_25_DT
ANNA_BUS_N( 4) <= CAD1_OUT_P(1)   ANNA_BUS_P( 4) <= CAD1_OUT_N(1)   LVDS_25_DT
ANNA_BUS_N( 3) <= CAD1_OUT_P(0)   ANNA_BUS_P( 3) <= CAD1_OUT_N(0)   LVDS_25_DT
ANNA_BUS_N(28) <= CTL1_OUT_P      ANNA_BUS_P(28) <= CTL1_OUT_N      LVDS_25_DT
ANNA_BUS_P(27) <= CLK1_OUT_P      ANNA_BUS_N(27) <= CLK1_OUT_N      LVDS_25_DT

CAD0_IN_P(7) <= ANNA_CLK2_P       CAD0_IN_N(7) <= ANNA_CLK2_N       LVDS_25
CAD0_IN_P(6) <= ANNA_BUS_N( 8)    CAD0_IN_N(6) <= ANNA_BUS_P( 8)    LVDS_25
CAD0_IN_P(5) <= ANNA_BUS_P( 9)    CAD0_IN_N(5) <= ANNA_BUS_N( 9)    LVDS_25
CAD0_IN_P(4) <= ANNA_BUS_P(37)    CAD0_IN_N(4) <= ANNA_BUS_N(37)    LVDS_25
CAD0_IN_P(3) <= ANNA_BUS_P(36)    CAD0_IN_N(3) <= ANNA_BUS_N(36)    LVDS_25
CAD0_IN_P(2) <= ANNA_BUS_P(41)    CAD0_IN_N(2) <= ANNA_BUS_N(41)    LVDS_25
CAD0_IN_P(1) <= ANNA_BUS_P(40)    CAD0_IN_N(1) <= ANNA_BUS_N(40)    LVDS_25
CAD0_IN_P(0) <= ANNA_BUS_P(42)    CAD0_IN_N(0) <= ANNA_BUS_N(42)    LVDS_25
CTL0_IN_P    <= ANNA_BUS_P(38)    CTL0_IN_N    <= ANNA_BUS_N(38)    LVDS_25
CLK0_IN_P    <= ANNA_BUS_P(39)    CLK0_IN_N    <= ANNA_BUS_N(39)    LVDS_25

CAD1_IN_P(7) <= P7_BUS_N( 3)      CAD1_IN_N(7) <= P7_BUS_P( 3)      LVDS_25
CAD1_IN_P(6) <= P7_BUS_N( 5)      CAD1_IN_N(6) <= P7_BUS_P( 5)      LVDS_25
CAD1_IN_P(5) <= ANNA_BUS_N(21)    CAD1_IN_N(5) <= ANNA_BUS_P(21)    LVDS_25
CAD1_IN_P(4) <= ANNA_BUS_N(19)    CAD1_IN_N(4) <= ANNA_BUS_P(19)    LVDS_25
CAD1_IN_P(3) <= ANNA_BUS_P(17)    CAD1_IN_N(3) <= ANNA_BUS_N(17)    LVDS_25
CAD1_IN_P(2) <= ANNA_BUS_P(16)    CAD1_IN_N(2) <= ANNA_BUS_N(16)    LVDS_25
CAD1_IN_P(1) <= ANNA_BUS_P(15)    CAD1_IN_N(1) <= ANNA_BUS_N(15)    LVDS_25
CAD1_IN_P(0) <= ANNA_BUS_N(14)    CAD1_IN_N(0) <= ANNA_BUS_P(14)    LVDS_25
CTL1_IN_P    <= ANNA_BUS_N(20)    CTL0_IN_N    <= ANNA_BUS_P(20)    LVDS_25
CLK1_IN_P    <= ANNA_BUS_N(18)    CLK0_IN_N    <= ANNA_BUS_P(18)    LVDS_25

EXT_CLK_P <= P7_BUS_N(20)              LVDS_25
EXT_CLK_N <= P7_BUS_P(20)              LVDS_25

P7_BUS_P(10) <= FIREINLEFT(3)          LVCMOS25
P7_BUS_N(10) <= FIREINLEFT(2)          LVCMOS25
P7_BUS_P( 9) <= FIREINLEFT(1)          LVCMOS25
P7_BUS_N( 9) <= FIREINLEFT(0)          LVCMOS25
P7_BUS_P(13) <= FIREINRIGHT(5)         LVCMOS25
P7_BUS_N(13) <= FIREINRIGHT(4)         LVCMOS25
```

```
P7_BUS_P(12) <= FIREINRIGHT(3)          LVCMOS25
P7_BUS_N(12) <= FIREINRIGHT(2)          LVCMOS25
P7_BUS_P(11) <= FIREINRIGHT(1)          LVCMOS25
P7_BUS_N(11) <= FIREINRIGHT(0)          LVCMOS25

BS_MODE      => P7_BUS_P(16)            LVCMOS25
CI_MODE      => P7_BUS_P(15)            LVCMOS25

PLL_BYPASS   => P7_BUS_P(17)            LVCMOS25
PLL_LOCKED   => P7_BUS_N(14)            LVCMOS25
PLL_RESET    => P7_BUS_P(14)            LVCMOS25

C_DELAY      => P7_BUS_N(16)            LVCMOS25
RESET        => P7_BUS_N(15)            LVCMOS25

ANNA_BUS_N( 7) <= sref                  LVCMOS25

P7_BUS_N(1) <= dac12_cs_b_int           LVCMOS25
ANNA_BUS_N(33) <= anamuxselb(0)         LVCMOS25
ANNA_BUS_P(34) <= anamuxselb(1)         LVCMOS25
ANNA_BUS_N(34) <= anamuxselb(2)         LVCMOS25
```

C.3 Synchronization

The following is a listing of the Verilog source code of the *sync* module, which initializes the system time counters upon sync and selects the destination *event_buffer_in* modules depending in the LSB values of the synchronization time value and the incoming events' time stamps.

```
module sync
(   clkhi,     chip_clk,
    evt_clk_select,
    clk_pos,   clkb_pos,
    clkhi_pos, prev_clkhi_pos,
    sync, l    oad_counter,    sync_val, rst);

    input   clkhi, chip_clk;
    output  evt_clk_select;
    output  load_counter;

    output  [`ev_clkpos_width-2 : 0] clk_pos;
    output  [`ev_clkpos_width-2 : 0] clkb_pos;

    output  [`ev_clkpos_width-1 : 0] clkhi_pos;
    output  [`ev_clkpos_width-1 : 0] prev_clkhi_pos;
    input   sync;
    input   [`ev_clkpos_width-1 : 0] sync_val;
    input   rst;

reg     evt_clk_select;
reg     clk200reg, clk400reg;

reg     [`ev_clkpos_width-2 : 0] clk_pos;
reg     [`ev_clkpos_width-2 : 0] clkb_pos;

reg     [`ev_clkpos_width-1 : 0] clkhi_pos;         // actual system time
reg     [`ev_clkpos_width-1 : 0] prev_clkhi_pos;    // needed to account for the earlyout bit

reg     latchsyncr,latchsyncf;   // these registers capture the sync signal
reg     syncreg1, syncreg2;      // syncreg1 and 2 do the synchronization to clkhi

wire rst_ls;       //reset latchsync reg
wire load_counter; //signals other part of the chip that it is now safe to update their counter from sync_val

// to avoid glitches, sync-values are registered in deframer
// -> values should be stable, when sync changes to 1
// (sync works as asynchronous set)

// synchronization scheme:
//  1:  "buffer" sync1 to sync @posedge sync1 -> stays 1 until next reset
//  2:  register sync @negedge of resp. clock
//  3:  negedge clkhi is always first. sync_val[0] and the value of chip_clk decide,
//          how to connect the evt_clk's after following scheme:
//          sync_val[0] |   chip_clk    || evt_clk  | evt_clkb
//          ---------------------------------------------
```

```
//                      0           |           0           ||      clk     |       clkb
//                      0           |           1           ||      clkb    |       clk
//                      1           |           0           ||      clkb    |       clk
//                      1           |           1           ||      clk     |       clkb
//  3:  load sync_cal into each counter @posedge of resp. clock


// one-hot encoded sync states
    parameter sync_statebitsmsb = 4;

// bit meanings
    parameter  sync_idle   = 0,
               sync_select = 1,
               sync_load   = 2,
               sync_wait   = 3,
               sync_synced = 4,

               SNsync_idle    = 5'b1 << sync_idle,
               SNsync_select  = 5'b1 << sync_select,
               SNsync_load    = 5'b1 << sync_load,
               SNsync_wait    = 5'b1 << sync_wait,
               SNsync_synced  = 5'b1 << sync_synced;

reg  [sync_statebitsmsb:0] SNsync_state;


// catch asynchronous sync
// flip-flop clocked with SNsync_state[sync_synced]->cleared
// sync sets flip-flop -> set

assign rst_ls = rst | SNsync_state[sync_select];
assign load_counter = SNsync_state[sync_wait] || SNsync_state[sync_load];

always @(posedge clkhi or posedge rst_ls)
    if ( rst_ls ) latchsyncr <= 0;
    else if(latchsyncr==0)latchsyncr <= sync;

always @(negedge clkhi or posedge rst_ls)
    if ( rst_ls ) latchsyncf <= 0;
    else if(latchsyncf==0)latchsyncf <= sync;

// upon sync, the value of chip_clk has to be determined, to decide, which buffers
// to assign odd and even event times to. This is achieved by the use of the below registers.
// If their values are different (exor = 1) then the 200MHz clock hasn't changed since
// the last 400MHz cycle and we are currently in low cycle of the 200MHz clock.
// the 200MHz register has to be somehow initialized...
initial clk200reg = 0;

always @(posedge chip_clk) clk200reg <= !clk200reg;
always @(posedge clkhi) clk400reg <= !clk200reg;

always @(posedge clkhi or posedge rst)
    if (rst) evt_clk_select <= 0;
    else if (SNsync_state[sync_select]) evt_clk_select <= !(sync_val[0] ^ !(clk200reg ^ clk400reg));

// crossing clock domains !!
// make sure the sync signal gets synchronized properly...
always @(posedge clkhi or posedge rst)
    if (rst) syncreg1 <= 0;
    else syncreg1 <= latchsyncr | latchsyncf;
always @(posedge clkhi or posedge rst)
    if (rst) syncreg2 <= 0;
    else syncreg2 <= syncreg1;


// sync FSM:
always @(posedge clkhi or posedge rst)
begin: sync_fsm

    if (rst) begin
        prev_clkhi_pos <= 0;
        SNsync_state <= SNsync_idle; end

    else begin

        case (SNsync_state)

            SNsync_idle: if (syncreg2) SNsync_state <= SNsync_select;

            SNsync_select: SNsync_state <= SNsync_load;

            SNsync_load: begin
`ifdef SYNTHESIS
`else
                $display("%t: Chip: %0h, received sync to clock #%h", $realtime, chip_id, sync_val);
`endif
                SNsync_state <= SNsync_wait; end

// wait one cycle, then start counters
// delay needed for correct simulation
            SNsync_wait: SNsync_state <= #`tim_ctoff SNsync_synced;
```

```
            SNsync_synced: begin
                prev_clkhi_pos <= clkhi_pos;
                if (syncreg1 && syncreg2) SNsync_state <= SNsync_select; end

            default: SNsync_state <= SNsync_idle;
        endcase
    end
end


// the data for the clock positions loaded into these counters is stored premanently
// by the deframer (synchronous to rx_clk0) 3 cycles before load.
always @(posedge clkhi or posedge rst)
begin
    if (rst) clkhi_pos <= 0;
    else if (SNsync_state[sync_load] || SNsync_state[sync_wait]) clkhi_pos <= sync_val;
    else clkhi_pos <= clkhi_pos + 1'b1;
end

always @(posedge chip_clk or posedge rst)
begin
    if (rst) clk_pos <= 0;
    else if (SNsync_state[sync_load] || SNsync_state[sync_wait])
        if (sync_val[0] && !evt_clk_select) clk_pos <= sync_val[`ev_clkpos_width-1 : 1] + 1'b1;
        else clk_pos <= sync_val[`ev_clkpos_width-1 : 1];
    else clk_pos <= clk_pos + 1'b1;
end

always @(posedge chip_clk or posedge rst)
begin
    if (rst) clkb_pos <= 0;
    else if (SNsync_state[sync_load] || SNsync_state[sync_wait])
        if (!sync_val[0] && !evt_clk_select) clkb_pos <= sync_val[`ev_clkpos_width-1 : 1];
        else clkb_pos <= sync_val[`ev_clkpos_width-1 : 1] + 1'b1;
    else clkb_pos <= clkb_pos + 1'b1;
end


endmodule
```

C.4 Simulated Spread on Delaylines

Within the following four figures, the extracted delay values for all *delaylines* are given. The signals of one link are plotted in one figure each. The x-axis denotes the physical address of the *delayline* and the extracted delay value for the default delay tap is plotted on the y-axis.

It can be seen that the delay values of the input signals are spread over differences of up to 600 ps. Moreover, the spread strongly varies with the process corner. Since the technology data does not contain process corner dependant information on the *RC* parasitics, this variation is due to the buffer cells that have been inserted by clock tree synthesis (CTS).

On the one hand, the spread is still below the adjustable range that can be covered by the *delaylines* (cf. section 6.2.2). Therefore, it is possible to tune the interface timing to an optimal data valid window size. On the other hand,

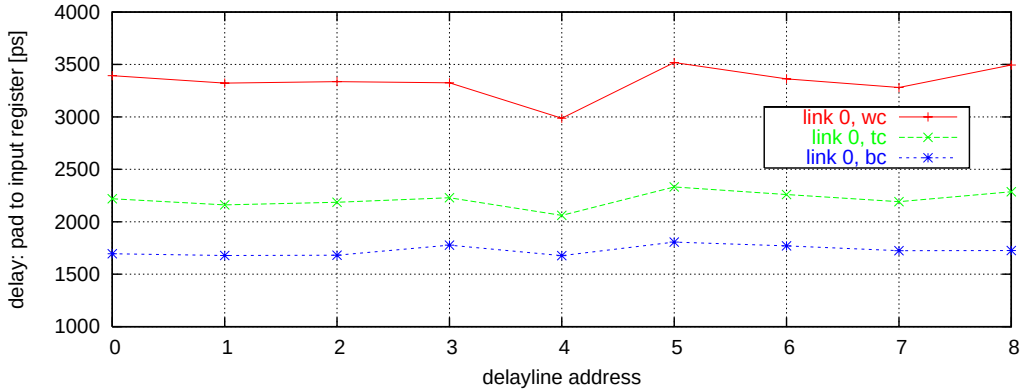


Figure C.2: Delay values determined for the *delaylines* of input link 0 by back annotated simulations. According to the measurements presented in section 6.2.3, the performance of the chips is at worst 12 % worse than the typical case.

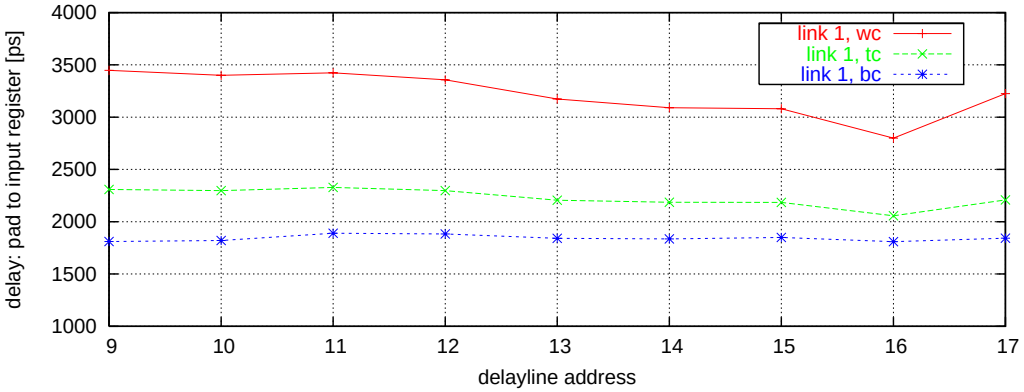


Figure C.3: Delay values determined for the *delaylines* of input link 1 by back annotated simulations. According to the measurements presented in section 6.2.3, the performance of the chips is at worst 12 % worse than the typical case.

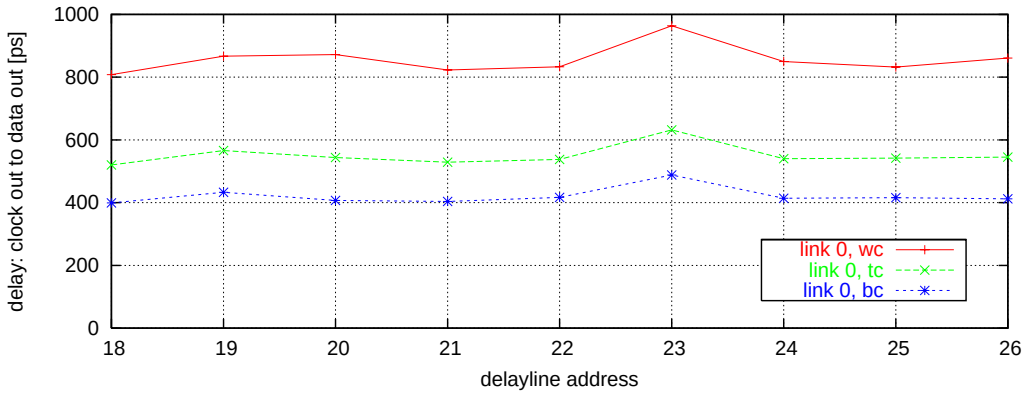


Figure C.4: Delay values determined for the *delaylines* of output link 0 by back annotated simulations. According to the measurements presented in section 6.2.3, the performance of the chips is at worst 12 % worse than the typical case.

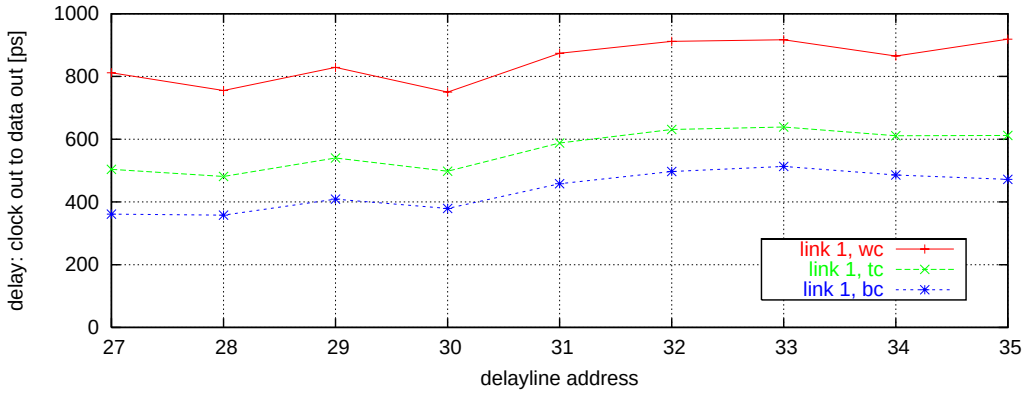


Figure C.5: Delay values determined for the *delaylines* of output link 1 by back annotated simulations. According to the measurements presented in section 6.2.3, the performance of the chips is at worst 12 % worse than the typical case.

C.5 Theoretical Optimum Delay values for the Spikey chip

	Spikey		FPGA			Total Delays		
Signal	T_{PadD}	T_{PadCK}	T_{CO}	T_{CT}	$T_{\text{CO+CT}}$	T_{del}	$T_{\text{del}} - T_{\text{av}}$	delay tap
cad0_o0	3.39	2.76	1.72	1.23	2.95	-0.69	0.27	**0**
cad0_o1	3.23	2.76	1.70	1.22	2.93	-0.51	0.09	**2**
cad0_o2	3.34	2.73	1.73	1.23	2.96	-0.67	0.26	**0**
cad0_o3	3.25	2.76	1.73	1.23	2.96	-0.56	0.14	**2**
cad0_o4	2.99	2.76	1.71	1.28	2.99	-0.33	-0.09	**4**
cad0_o5	3.39	2.77	1.71	1.25	2.96	-0.68	0.27	**0**
cad0_o6	3.36	2.67	1.71	1.28	2.99	-0.79	0.37	**0**
cad0_o7	3.17	2.67	1.73	1.18	2.91	-0.51	0.09	**2**
ctl0_o	3.32	2.70	1.75	1.28	3.03	-0.76	0.34	**0**
clk0_o	—		1.71	1.19	2.90	—		
cad1_o0	3.03	2.68	1.75	1.29	3.04	-0.39	-0.03	**3**
cad1_o1	2.95	2.67	1.72	1.28	3.00	-0.26	-0.16	**5**
cad1_o2	3.02	2.67	1.73	1.28	3.01	-0.35	-0.07	**4**
cad1_o3	2.95	2.67	1.70	1.27	2.97	-0.24	-0.18	**5**
cad1_o4	2.90	2.67	1.70	1.29	2.99	-0.20	-0.22	**5**
cad1_o5	2.84	2.67	1.69	1.31	3.00	-0.16	-0.26	**6**
cad1_o6	2.85	2.67	1.70	1.27	2.97	-0.14	-0.28	**6**
cad1_o7	2.74	2.67	1.71	1.27	2.98	-0.04	-0.38	**7**
ctl1_o	2.91	2.61	1.69	1.29	2.98	-0.27	-0.15	**5**
clk1_o	—		1.72	1.30	3.02	—		
			Av. Delay			**-0.42**		

Table C.2: Calculation of the theoretical optimum delay values for the *delaylines* at the Spikey chip's input. The values are calculated for the typical mean process corner of both, the Spikey chip and the FPGA. The last column gives the absolute delay value to set on the respective delay line.

Legend:

$$
\begin{aligned}
T_{\text{PadD}} &= \text{data input pad to D-pin of input flip-flop} \\
T_{\text{PadCK}} &= \text{clock input pad to CK-pin of input flip-flop} \\
T_{\text{CO}} &= \text{clock to output delay up to output pad} \\
T_{\text{CT}} &= \text{clock tree delay from DCM to CK-pin of output flip-flop} \\
T_{\text{del}} &= (T_{\text{PadCK}} + T_{\text{CO+CT}}(clk)) - (T_{\text{PadD}} + T_{\text{CO+CT}}) \\
&= \text{resulting delay between clock and data signal} \\
T_{\text{av}} &= \text{average value of the above for all signals}
\end{aligned}
$$

	Spikey	FPGA		Total Delays		
	T_{CO}	T_{PadCK}	T_{PadD}	T_{del}	$T_{del} - T_{av}$	delay tap
cad0_i0	0.74	2.85	3.87	-1.77	-0.12	**4**
cad0_i1	0.95	2.92	3.89	-1.92	0.03	**3**
cad0_i2	0.82	2.87	3.90	-1.85	-0.04	**3**
cad0_i3	0.92	2.70	3.90	-2.12	0.23	**1**
cad0_i4	0.78	2.86	3.92	-1.83	-0.06	**4**
cad0_i5	1.04	2.86	3.91	-2.09	0.20	**1**
cad0_i6	0.79	2.86	3.91	-1.84	-0.05	**3**
cad0_i7	0.92	2.70	3.89	-2.11	0.22	**1**
cad0_i8	0.86	2.84	3.89	-1.91	0.02	**3**
clk0_i	0.00	—				
cad1_i0	0.76	2.87	3.87	-1.76	-0.14	**4**
cad1_i1	0.83	2.88	3.88	-1.83	-0.06	**4**
cad1_i2	0.78	2.87	3.89	-1.80	-0.09	**4**
cad1_i3	0.83	2.89	3.89	-1.83	-0.06	**4**
cad1_i4	0.82	2.95	3.86	-1.74	-0.15	**5**
cad1_i5	0.98	2.93	3.87	-1.92	0.03	**3**
cad1_i6	0.86	2.71	3.84	-1.99	0.10	**2**
cad1_i7	0.94	2.88	3.89	-1.95	0.06	**2**
cad1_i8	0.95	3.09	3.88	-1.75	-0.14	**4**
clk1_i	0.00	—				
			average delay	**-1.89**		

Table C.3: Calculation of the theoretical optimum delay values for the *delaylines* at the Spikey chip's output. The values are calculated for the typical mean process corner of both, the Spikey chip and the FPGA. The last column gives the absolute delay value to set on the respective delay line.

Legend:

$$
\begin{aligned}
T_{PadD} &= \text{data input pad to D-pin of input flip-flop} \\
T_{PadCK} &= \text{clock input pad to CK-pin of input flip-flop} \\
T_{CO} &= \text{clock to output delay up to output pad} \\
T_{del} &= T_{PadCK} - (T_{CO} + T_{PadD}) \\
&= \text{resulting delay between clock and data signal} \\
T_{av} &= \text{average value of the above for all signals}
\end{aligned}
$$

C.6 Mixed-Signal Simulation of the DTC Output

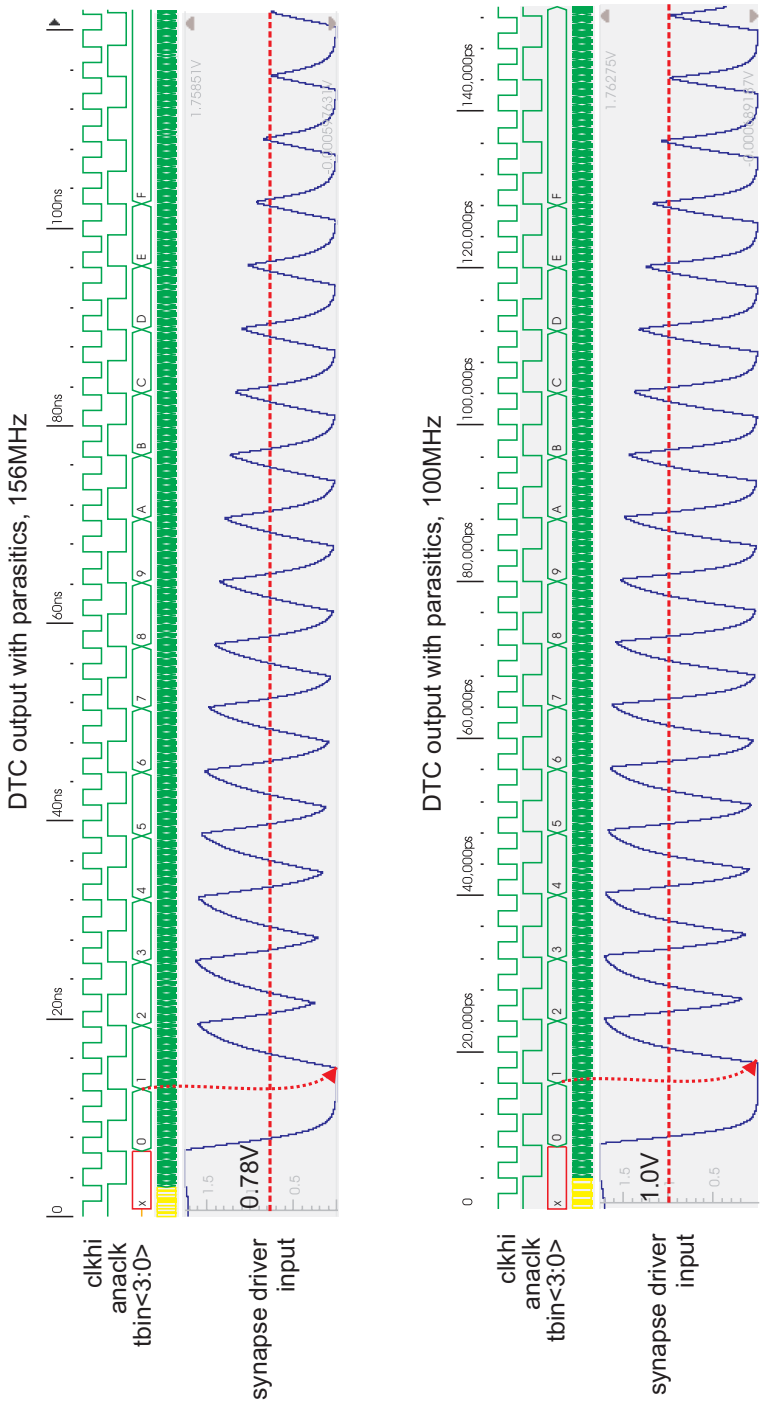


Figure C.6: Mixed-signal simulation of the DTC's output signal, which enables the event generation in a specific synapse block for all time bins {0,15}. The waveform is plotted at the synapse driver's input. Parasitics introduced by automated routing are included in the output load. It can be seen that the input voltage to the driver becomes smaller and won't drive the input over the threshold for large time bin values. Thanks to Sebastian Millner for the plot!

Appendix D

Mixed-Signal Design Flow Supplements

D.1 List of Routing Options

```
setNanoRouteMode  dbSkipAnalog false

setNanoRouteMode  drouteAntennaFactor 0.99
# keep nanoroute from senseless trying...
setNanoRouteMode  drouteAutoStop true
# no time limit for router
setNanoRouteMode  drouteElapsedTimeLimit 0
setNanoRouteMode  drouteEndIteration default
setNanoRouteMode  drouteFixAntenna true
setNanoRouteMode  drouteForbidStackVia false
# do not optimize for via count by default.
# Do NOT specify together with drouteUseMultiCutViaEffort
setNanoRouteMode    drouteMinimizeViaCount false
setNanoRouteMode  drouteNoTaperOnOutputPin false
# may be set to  true after routing and antenna insertion to
#minimize resistance and improve yield:
setNanoRouteMode  drouteUseMultiCutVia false
#setNanoRouteMode  drouteUseMultiCutViaEffort low
setNanoRouteMode  drouteSearchAndRepair true
setNanoRouteMode  drouteStartIteration default
# If you do not specify this option, NanoRoute uses the spacing
# defined in the USEMINSPACING OBS statement in the LEF file:
#setNanoRouteMode  drouteUseMinSpacingForBlockage false
# the default here is 2...
setNanoRouteMode  drouteUseViaOfCut 2
setNanoRouteMode  drouteViaOnGridOnly false
setNanoRouteMode  envNumberFailLimit 10
setNanoRouteMode  envNumberProcessor 6
#setNanoRouteMode  envSuperthreading \
#   "SSH \
#  {algol 2 /cad/products/encounter/SOC52/bin/nanoroute} \
#  {fourofeight 6 /cad/products/encounter/SOC52/bin/nanoroute} \
#   "
# set this one to true only with very high congestion!:
setNanoRouteMode  grouteMinimizeCongestion true
# try setting this to true for analog routing!
setNanoRouteMode  routeAllowPowerGroundPin false
setNanoRouteMode  routeAntennaCellName {"HDANT" "HDANT2"}
setNanoRouteMode  routeAntennaPinLimit 1000 #default
setNanoRouteMode  routeAutoGgrid true #default
setNanoRouteMode  routeBottomRoutingLayer 1 #default
setNanoRouteMode  routeDeleteAntennaReroute true #default
setNanoRouteMode  routeEcoOnlyInLayers "1:6"
setNanoRouteMode  routeExtraViaEnclosure 0 #default
setNanoRouteMode  routeFixTopLayerAntenna true #default
setNanoRouteMode  routeHonorPowerDomain false #default
# Specify false for this parameter when you route a block whose
# top level is going to have diodes inserted:
setNanoRouteMode  routeIgnoreAntennaTopCellPin true #default
# set the following two to true for the last global routing step:
setNanoRouteMode  routeInsertAntennaDiode false #default
setNanoRouteMode  routeInsertDiodeForClockNets false #default
setNanoRouteMode  routeMinShieldViaSpan -1 #default
#!obsolete in 5.2!setNanoRouteMode  routeReInsertFillerCellList ./CFG/corefillers.txt
#setNanoRouteMode  routeReplaceFillerCellList fileName
# just to make sure:
setNanoRouteMode  routeSelectedNetOnly false
setNanoRouteMode  routeSiEffort medium
setNanoRouteMode  routeStrictlyHonorNonDefaultRule false #default
```

```
setNanoRouteMode  routeStripeLayerRange "1:6"
setNanoRouteMode  routeTdrEffort 8
setNanoRouteMode  routeTopRoutingLayer 6
# I assume the cells having detailed blockage information:
setNanoRouteMode  routeUseBlockageForAutoGgrid true
setNanoRouteMode  routeWithEco false #default
setNanoRouteMode  routeWithSiDriven true
# with this crosstalk reduction can be performed - see fetxtcmdref!
setNanoRouteMode  routeWithSiPostRouteFix false
setNanoRouteMode  routeWithTimingDriven true
setNanoRouteMode  routeWithViaInPin false
setNanoRouteMode  routeWithViaOnlyForStandardCellPin false
setNanoRouteMode  timingEngine CTE #default
```

D.2 Applied Timing Constraints

```
################################################################################
# definition of clocks
################################################################################

#clock frequency (defined as clkhi period)
set CLKPER       3.2

# the accumulated phase error / jitter of the PLL will not exceed (according to datasheet)
# 100ps, therefore the clock period is set to 2.4ns
set MAXCLKJITTER   0.1
set CLKHIPERIOD      [expr $CLKPER     ]
set SYSCLKPERIOD     [expr $CLKPER * 2 ]
set CLKHIPERIODQ     [expr $CLKPER / 4 ]
set CONFIGPERIOD   20
set JTAGPERIOD       200

create_clock -name "_RX_CLK0_" -period $CLKHIPERIOD [get_ports "CLK0_IN_P"]
create_clock -name "_RX_CLK1_" -period $CLKHIPERIOD [get_ports "CLK1_IN_P"]

create_clock -name "_EXT_CLK_" -period $CLKHIPERIOD [get_ports "EXT_CLK_P"]

# the clock from the PLL
create_clock -name "_PLL_CLK400_" -period $CLKHIPERIOD [get_pins -hierarchical "*/PLLOUT"]

# clkhi and chip_clk are generated by CLK_BY1BY2
# No phase relation exists to any of the other clocks, so these clocks are
# not treated as generate_clocks but rather as standalone clocks.
create_clock -name "_CHIP_CLK_" -period $SYSCLKPERIOD [get_pins -hierarchical "*/CK_BY2"]
create_clock -name "_CLKHI_"        -period $CLKHIPERIOD [get_pins -hierarchical "*/CK_BY1"]

# the very slow "clock" to clock in the delay value during reset
create_clock -name "_DEL_CONF_" -period $CONFIGPERIOD [get_ports "C_DELAY"]

# the JTAG clock
create_clock -name "_JTAG_CLK_" -period $JTAGPERIOD [get_ports "TCK"]

set_dont_touch_network _RX_CLK0_
set_dont_touch_network _RX_CLK1_
set_dont_touch_network _DEL_CONF_
set_dont_touch_network _PLL_CLK400_
set_dont_touch_network _EXT_CLK_
set_dont_touch_network _JTAG_CLK_
set_dont_touch_network _CHIP_CLK_
set_dont_touch_network _CLKHI_

set_clock_uncertainty $MAXCLKJITTER [get_clocks *]

################################################################################
# definitions for source synchronous LVDS IO
################################################################################

# The interface data and clock lines are processed by CTS to have minimum skew. Omit clock definitions
# for the data lines here as no synchronous paths are affected. The only thing to constrain is the
# routing delay through the delay lines and false paths form clock to input and clock to output ports.
# -> CTS can't trace through the delylines themselves due to the tri-state busses.
# OUTPUT LINK including CTL pins
for {set i 0} {$i<=8} {incr i} {
    set_false_path -from _CLKHI_ -through [get_pins $SPIKENET_TOP*delaylines/tx_del??$i?/di_outbuf/Z] \
        -to [get_ports C*_OUT*]

    # constrain the paths from the ddr flip flops' otuputs with (reasonable) minimum delay as
    # the tool otherwise has no constraints there and unneccessarily loads the multiplexer
    # outputs with long lines.

    # constrain the path form clkhi to the start of the output data pseudo clock lines defined in
    # the clock tree config. this is the part through the select pin of the ddr mulitplexer
    set_max_delay 1.55 -from [get_pins $SPIKENET_TOP*transmitter_tx_framer?__framer_buf2?$i?/A]\
        -to [get_pins $SPIKENET_TOP*delaylines/tx_del??$i?/di_outbuf/Z]

    #flatted for calibre
    set_max_delay 1.3 -from [get_pins $SPIKENET_TOP*transmitter_tx_framer?__framer_dff?$i?/Q]\
        -to [get_pins $SPIKENET_TOP*delaylines/tx_del??$i?/di_outbuf/Z]
```

```
}

#FE skips the whole loop if it encounters an error
#therefore the hierarchical version for synopsys gets an extra loop here
for {set i 0} {$i<=8} {incr i} {
    #this is the path from the output of the flipflops through the ddr multiplexer
    set_max_delay 1.55 -from [get_pins $SPIKENET_TOP*transmitter/tx_framer?__framer_buf2?$i?/A]\
        -to [get_pins $SPIKENET_TOP*delaylines/tx_del??$i?/di_outbuf/Z]
    #this is the path from the output of the flipflops through the ddr multiplexer
    set_max_delay 1.3 -from [get_pins $SPIKENET_TOP*transmitter/tx_framer?/_framer_dff?$i?/Q]\
        -to [get_pins $SPIKENET_TOP*delaylines/tx_del??$i?/di_outbuf/Z]
}


set_false_path -from _CLKHI_ -through [get_pins $SPIKENET_TOP*delaylines/txc?_del/Z] \
    -to [get_ports CLK?_OUT*]


# INPUT LINK
for {set i 0} {$i<=7} {incr i} {
    # first set the path from the input pin to the clock to false. CTS does the balancing later.
    set_false_path -from [get_ports CAD?_IN_*[$i]] -to [get_clocks _RX_CLK?_]

    # set maximum delay constraint on the delaylines to have them with all equal delay
    set_max_delay 1.2 -from [get_pins $SPIKENET_TOP*delaylines/rx_del??$i?/di_a/A] \
        -to [get_cells -hierarchical "*dff_*"]
}
# the ctl signals
set_false_path -from [get_ports CTL?_IN_*] -to [get_clocks _RX_CLK?_]
set_max_delay 1.2 -from [get_pins $SPIKENET_TOP*delaylines/rx_del??8?/di_a/A] \
    -to [get_cells -hierarchical "*dff_*"]

# make the tool calculate input and output timing only through the di_null tristate
# buffers. Disable all other buffers of the delay lines
for {set i 0} {$i<=8} {incr i} {
    set_false_path -through [get_cells $SPIKENET_TOP*delaylines/rx_del??$i?/di_min*]
    set_false_path -through [get_cells $SPIKENET_TOP*delaylines/rx_del??$i?/di_plus*]

    set_false_path -through [get_cells $SPIKENET_TOP*delaylines/tx_del??$i?/di_min*]
    set_false_path -through [get_cells $SPIKENET_TOP*delaylines/tx_del??$i?/di_plus*]
}

##############################################################################

# the current addresses have to be valid directly before cw_xx. This sums up
# to about one clock cycle.
set_max_delay [expr $SYSCLKPERIOD /2] -from [get_cells -hierarchical "*applied_addr_reg*"] \
    -to [get_pin  "$SPIKENET_TOP*cw_*"]
set_max_delay [expr $SYSCLKPERIOD /2] -from [get_cells -hierarchical "*applied_addr_reg*"] \
    -to [get_pin  "$SPIKENET_TOP*ca*"]
# the signals for the analog readout chain
set_max_delay [expr $SYSCLKPERIOD /2] -through  [get_pin "$SPIKENET_TOP*aro_*"]

set_false_path -from [get_ports CHIP_ID*]
set_false_path -from [get_ports CI_MODE]
set_false_path -from [get_ports BS_MODE]
set_false_path -from [get_ports RESET]
set_false_path -from [get_ports PLL_RESET]

set_min_delay -5 -to [get_clocks _DEL_CONF_]
set_max_delay 5 -from [get_clocks _DEL_CONF_]

# this signal only changes during sync... events shouldn' be transferred directly
# after sync!
set_max_delay 10 -from [get_cells -hierarchical "*evt_clk_select_reg*"]

# the following blocks VST event buffer memory WCK --> DOUT paths
set_disable_timing event_in_ram_wc_162V_85C/event_in_ram   -from RCK -to WCK
set_disable_timing event_out_ram_wc_162V_85C/event_out_ram -from RCK -to WCK
set_disable_timing event_in_ram_bc_198V_0C/event_in_ram    -from RCK -to WCK
set_disable_timing event_out_ram_bc_198V_0C/event_out_ram  -from RCK -to WCK


# set false paths between unrelated clock domains in design:
# all paths crossing the following clock domains either have no
# phase relation or are synchronized by two register stages.
set_false_path -from _RX_CLK0_ -to _DEL_CONF_
set_false_path -from _RX_CLK1_ -to _DEL_CONF_

set_false_path -from _CLKHI_ -to _RX_CLK0_
set_false_path -from _CLKHI_ -to _RX_CLK1_
set_false_path -from _CHIP_CLK_ -to _RX_CLK0_
set_false_path -from _CHIP_CLK_ -to _RX_CLK1_

set_false_path -from _RX_CLK0_ -to _CHIP_CLK_
set_false_path -from _RX_CLK1_ -to _CHIP_CLK_

set_false_path -from _RX_CLK0_ -to _CLKHI_
set_false_path -from _RX_CLK1_ -to _CLKHI_

set_false_path -from _DEL_CONF_ -to [get_ports C*_OUT_*]
set_false_path -from _DEL_CONF_ -to _RX_CLK0_
```

```
set_false_path -from _DEL_CONF_ -to _RX_CLK1_

set_false_path -from _EXT_CLK_ -to _CHIP_CLK_
set_false_path -from _EXT_CLK_ -to _CLKHI_

# disable optimization of pll_locked signal
set_false_path -through [get_pin "$SPIKENET_TOP*pll_locked400"]

# set the clock enable signal for the event loopback to false.
# -> Event out fifos must be resetted after switching!
set_false_path -from [get_cells -hierarchical "*el_enable_reg*"]

# the only things clocked by rx_clk* are the input fifos (they have two asynchronous
# clock ports) and the synchronisation logic in deframer0. The sync signal is
# properly synchronized in spikenet_top/sync_inst (2 regs) so it should be ok to only
# set a max_delay constraint on this path.
set_max_delay 1 -from [get_clocks _RX_CLK0_] -to [get_cells -hierarchical "*latchsync?_reg*"]


################################################################################
# FE only timing constraints
################################################################################

# As the anaclks are not really gated (they are switched only once or so for
# a session), the clock gating check may be disabled, here.
set_disable_clock_gating_check [get_pins spikenet_top_pad/spikenet_top/clock_gen_aclkand/A2]
set_disable_clock_gating_check [get_pins spikenet_top_pad/spikenet_top/clock_gen_aclkband/A2]
set_disable_clock_gating_check [get_pins spikenet_top_pad/spikenet_top/clock_gen_aclkhiand/A2]

# disable the clock_Gatin_check for the multiplexer selecting
# between ext_clk and pllout
set_disable_clock_gating_check [get_pins spikenet_top_pad/spikenet_top/clock_gen_U5/A1]
set_disable_clock_gating_check [get_pins spikenet_top_pad/spikenet_top/clock_gen_U5/B1]
```

Appendix E

Bonding Diagram and Packaging

This page is intentionally left blank.

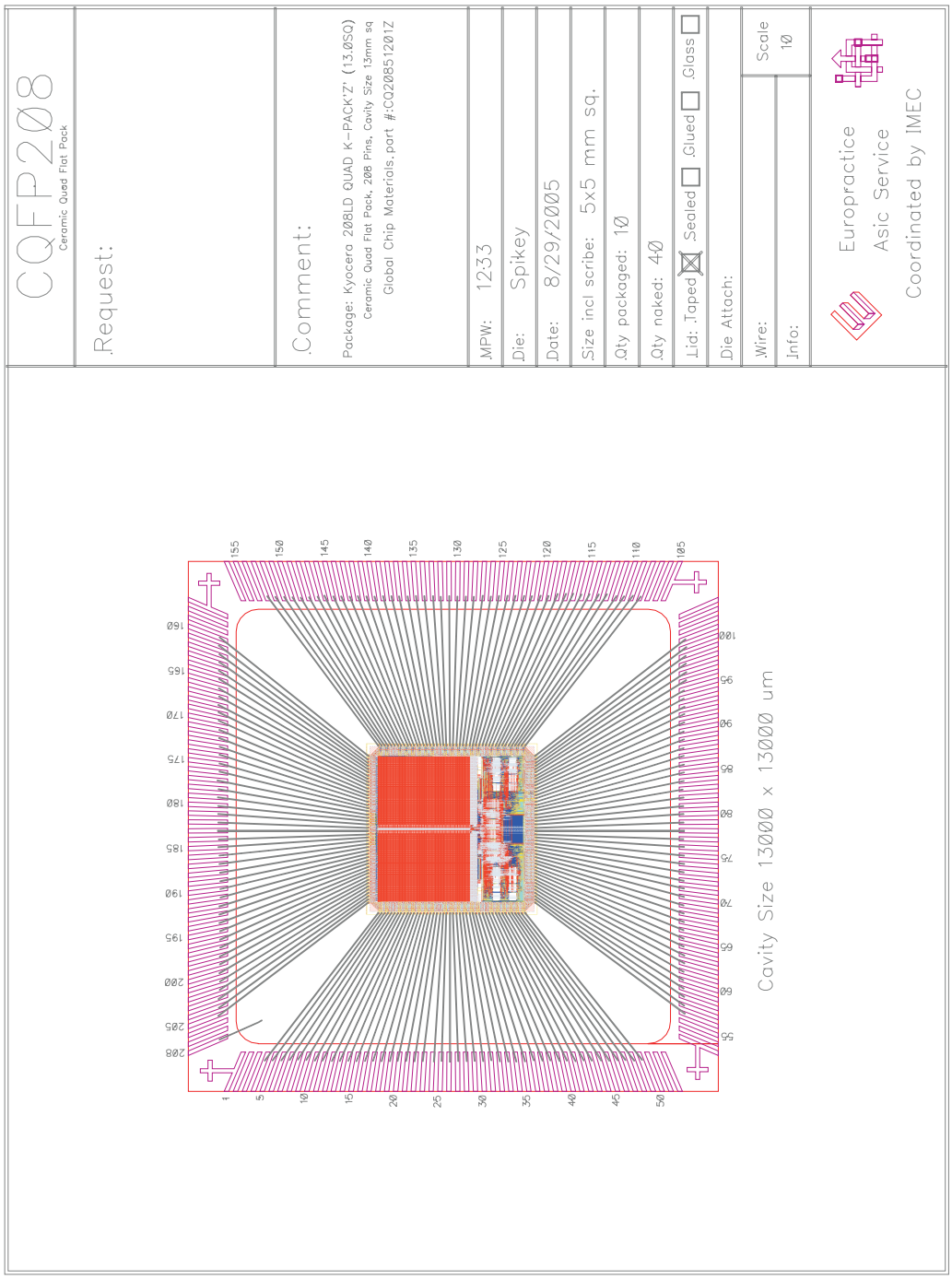


Figure E.1: Bonding diagram for Spikey 1 as sent to Europractice for packaging. The pinout of Spikey 2 is identical to the one of Spikey 1, therefore the same diagram was used for the second version.

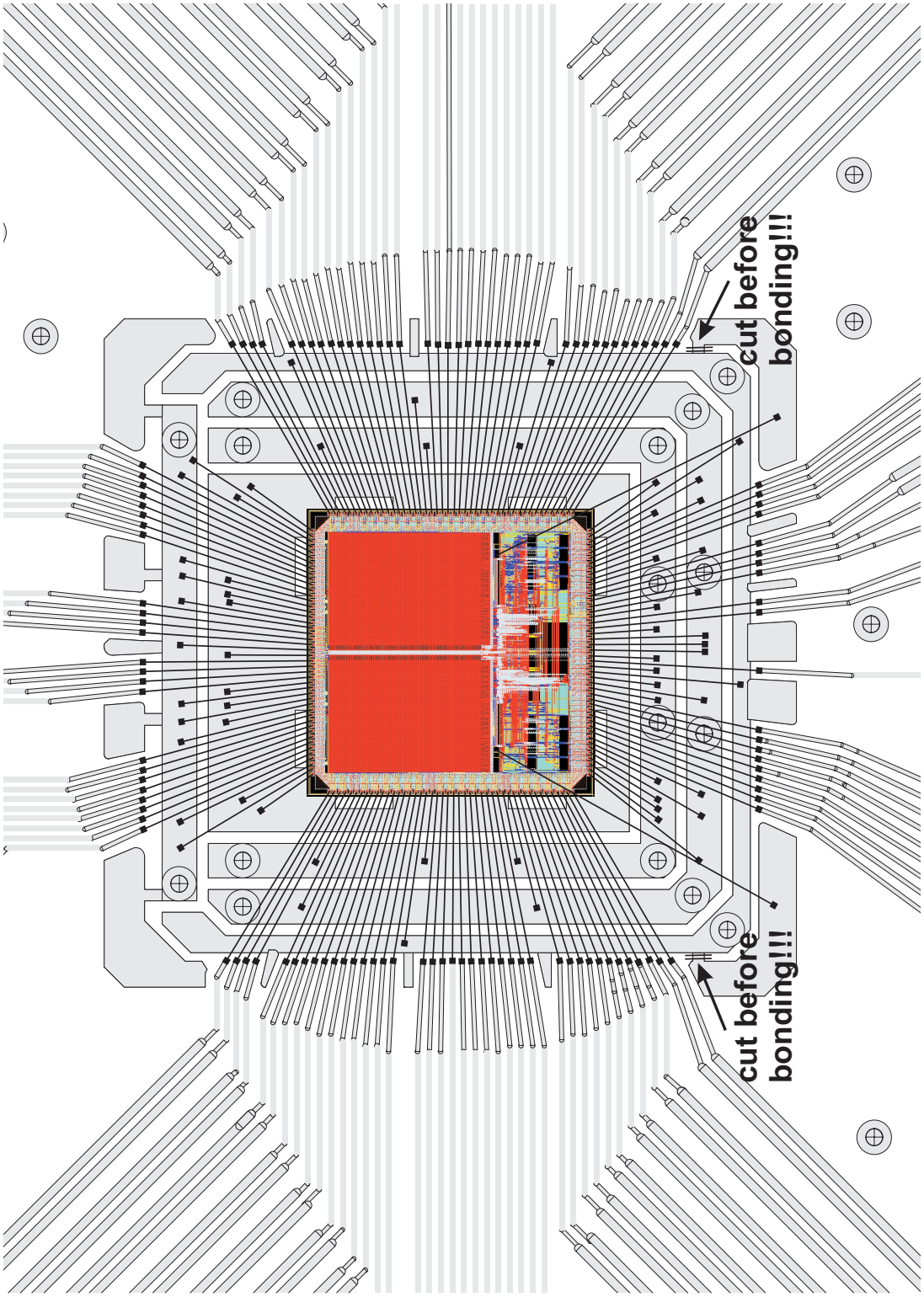


Figure E.2: Bonding diagram for both versions of the Spikey chip for bonding onto the Recha PCB. This footprint also has not changed for the Recha successor PCB, Recha V2 which has been developed by B. Ostendorf.

Appendix F

Recha PCB

F.1 Modifications to the Nathan PCB

The following modifications to the Nathan module are required in order to accommodate the Recha PCB:

- The PGA144 socket has to be soldered off to free up the space between the SMT connectors. This is required for the decoupling capacitors located on the bottom of the Recha PCB. See [Ost07] for a technical solution.
- All 120 Ω resistors `R9` on the top side need to be replaced by 0 Ω resistors to achieve correct LVDS signaling levels.
- All 100 Ω resistors `R8` on the top side need to be completely removed to achieve correct LVDS signaling levels.
- Within the UCF file for the FPGA, the signaling standard of the LVDS pairs relating to the afore listed resistors has to be changed from `BLVDS_25` to `LVDS_25`.

F.2 Schematics

Page one of the Recha schematics will be displayed on the following page.

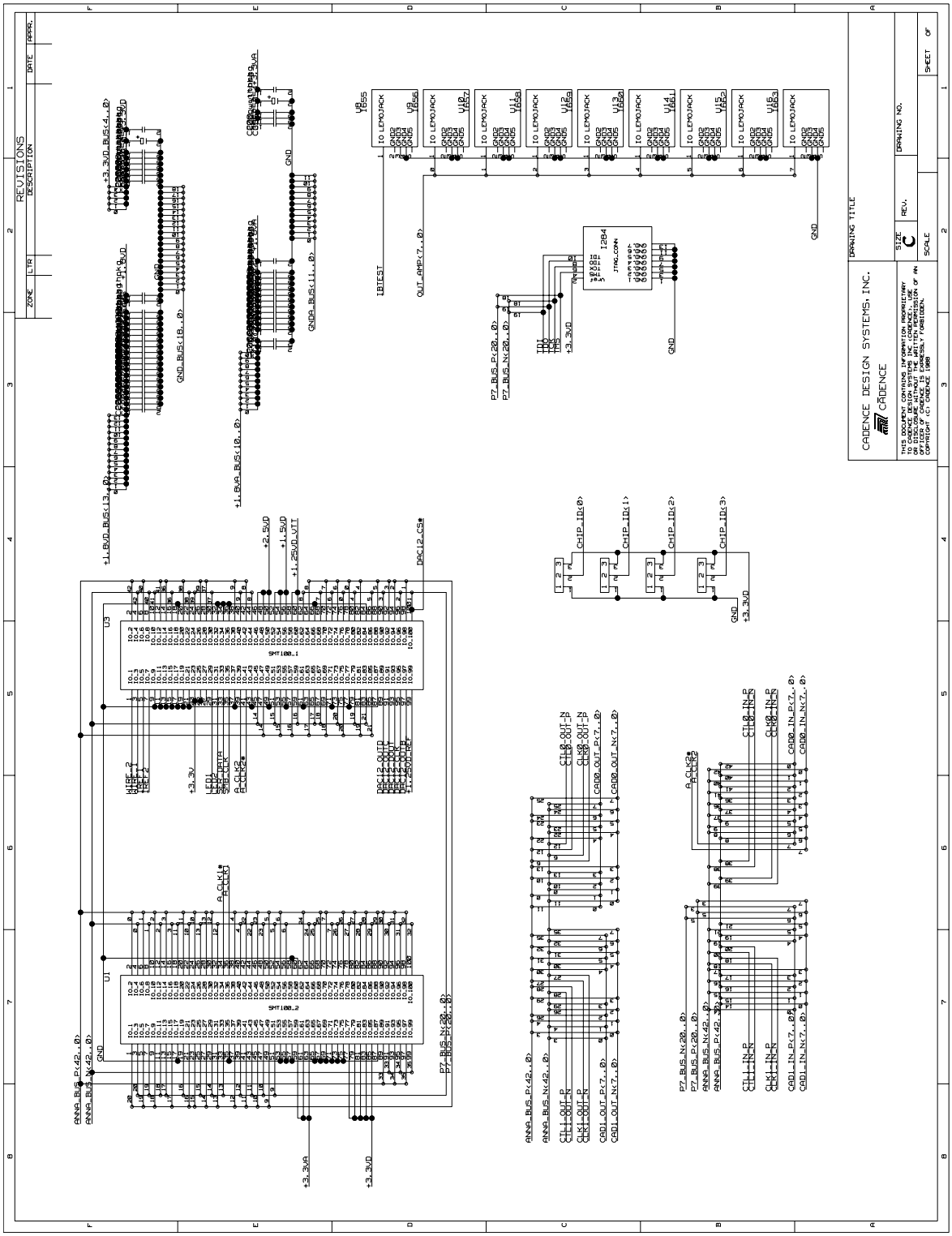


Figure F.1: Recha schematic, page 1 of 2.

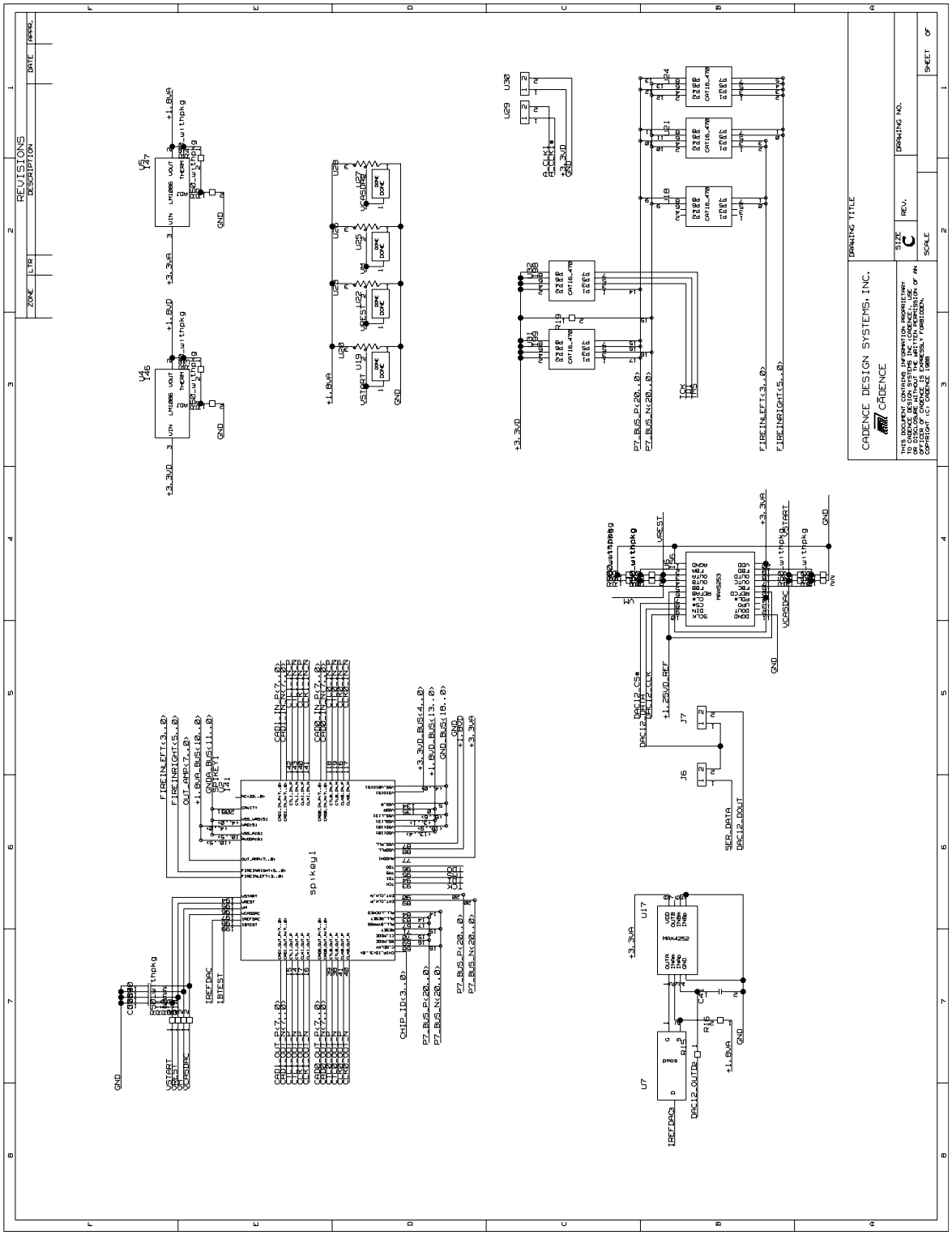


Figure F.2: Recha schematic, page 2 of 2.

F.3 Layouts

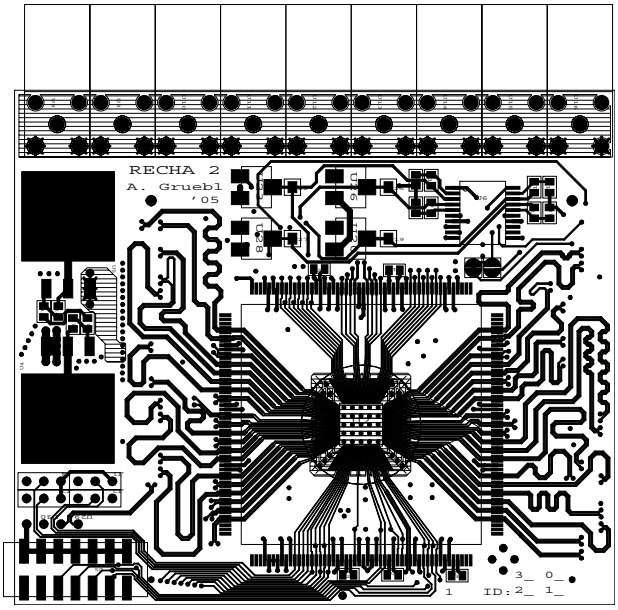


Figure F.3: Top layer of the Recha PCB.

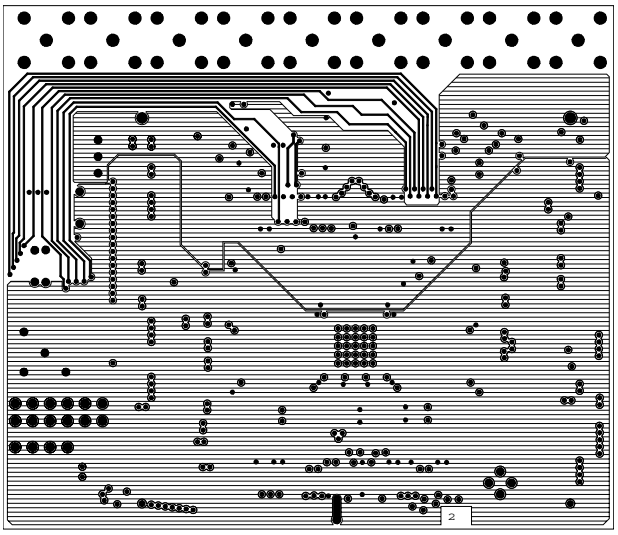


Figure F.4: Layer 2 of the Recha PCB.

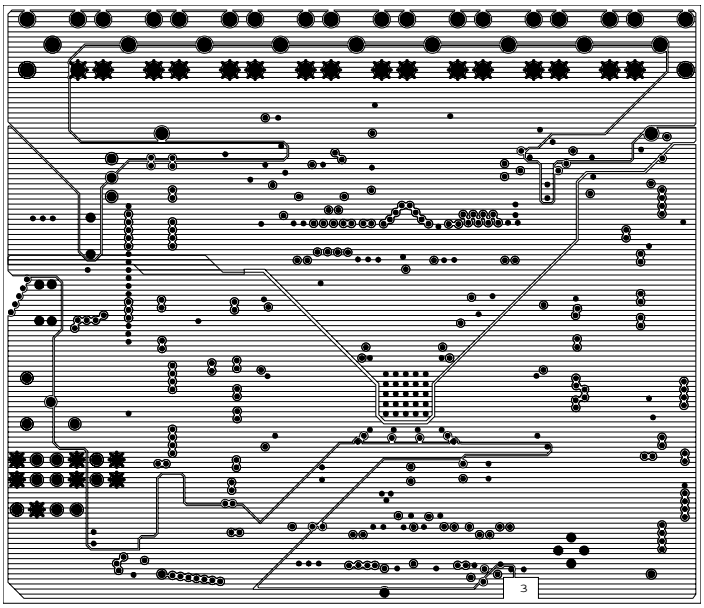


Figure F.5: Layer 3 of the Recha PCB.

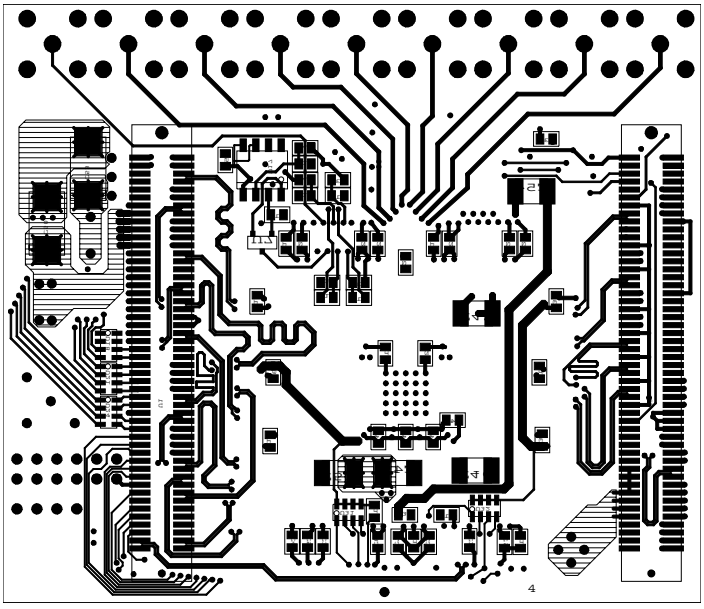


Figure F.6: Bottom layer of the Recha PCB.

Bibliography

[Ach] R. Achenbach. personal communication.

[AGM00] Y. Wang A. Gupta and H. Markram. Organizing principles for a diversity of gabaergic interneurons and synapses in the neocortex. *Science*, 287:273–278, Jan. 2000.

[ANS96] ANSI/TIA/EIA-644. *Electrical Characteristics of Low Voltage Differential Signalling (LVDS)*, March 1996.

[BGM+07] D. Brüderle, A. Grübl, K. Meier, E. Mueller, and J. Schemmel. A software framework for tuning the dynamics of neuromorphic silicon towards biology. In *Proceedings of the 9th International Work-Conference on Artificial Neural Networks (IWANN'2007), accepted for publication*, 2007.

[Bha99] H. Bhatnagar. *Advanced ASIC Chip Synthesis Using Synopsys DesignCompiler and PrimeTime*. Kluwer Academic Publishers, 1999. ISBN 0-7923-8537-3.

[BP97] G. Bi and M. Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Neural Computation*, 9:503–514, 1997.

[Brü04] D. Brüderle. Implementing spike-based computation on a hardware perceptron. Diploma thesis, University of Heidelberg, HD-KIP-04-16, 2004.

[Brü07] D. Brüuderle. *PhD thesis*, University of Heidelberg, in preparation, 2007.

[Cad05a] Cadence Design Systems, Inc. *LEF/DEF Language Reference, Product Version 5.6*, 2005. availbale from Cadence online documentation *cdsdoc*.

[Cad05b] Cadence Design Systems, Inc. *Virtuoso AMS Environment User Guide, Product Version 5.6*, 2005. availbale from Cadence online documentation *cdsdoc*.

[Cad06a] Cadence Design Systems, Inc. *Encounter Timing Closure Guide, Product Version 5.2.1*, February 2006. availbale from Cadence online documentation *cdsdoc*.

[Cad06b] Cadence Design Systems, Inc. *Encounter User Guide, Product Version 5.2.3*, June 2006. availbale from Cadence online documentation *cdsdoc*.

[CBF00] A. P. Chandrakasan, W. J. Bowhill, and F. Fox. *Design of High-Performance Microprocessor Circuits*. Wiley-IEEE Press, 2000.

[Cor] United Microelectronics Corporation. http://www.umc.com.

[DA01] P. Dayan and L. F. Abott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT press, Cambride, Massachusetts, London, England, 2001.

[DGA99] M. Diesmann, M.-O. Gewaltig, and A. Aertsen. Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402:529–532, 1999.

[DLM+99] V. Douence, A. Laflaquiere, S. Le Masson, T. Bal, and G. Le Masson. Analog electronic system for simulating biological neurons. In *Proceedings of the International Work-Conference on Artificial and Natural Neural Networks, IWANN 1999*, pages 188–197, 1999.

[DP01] W. J. Dally and J. W. Poulton. *Digital Systems Engineering*. Cambridge University Press, Cambridge, UK, 2001.

[DRP03] A. Destexhe, M. Rudolph, and D. Pare. The high-conductance state of neocortical neurons in vivo. *Nature Reviews Neuroscience*, 4:739–751, 2003.

[EME+06] M. Ehrlich, C. Mayr, H. Eisenreich, S. Henker, A. Srowig, A. Grübl, J. Schemmel, and R. Schüffny. Wafer-scale vlsi implementations of pulse coupled neural networks. Fourth IEEE International Multi-Conference on Systems, Signals & Devices - paper submitted, December 2006.

[EVg] Kirchhoff Institute for Physics Electronic Vision(s) group. Spikey 2 svn repository. https://www.kip.uni-heidelberg.de/repos/VISION/project/spikey2.

[FGP+04] J. Fieres, A. Grübl, S. Philipp, K. Meier, J. Schemmel, and F. Schürmann. A platform for parallel operation of VLSI neural networks. In *Proc. of the 2004 Brain Inspired Cognitive Systems Conference (BICS2004)*, University of Stirling, Scotland, UK, 2004.

[FS95] D. Ferster and N. Spruston. Cracking the neuronal code. *Science*, 270:756–757, 1995.

[GAS90] R. L. Geiger, P. E. Allen, and N.l R. Strader. *VLSI Design Techniques for Analog and Digital Circuits*. McGraw-Hill, Inc, 1990.

[Ger99] W. Gerstner. Spiking neurons. In W. Maass and C.M. Bishop, editors, *Pulsed Neural Networks*. MIT Press, 1999.

[GK02] W. Gerstner and W. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[Gop05] L. Gopalakrishnan. *FIFOs Using Virtex-II Shift Registers*. Xilinx, Inc, www.xilinx.com, January 2005. XAPP256.

[Grü03] A. Grübl. Eine FPGA-basierte platform für neuronala netze. Diploma thesis (german), University of Heidelberg, HD-KIP-03-2, 2003.

[HC97] M. L. Hines and N. T. Carnevale. The neuron simulation environment. *Neural Computation*, 9:1179–1209, 1997.

[HdO06] D. Husmann de Oliveira. A new backplane for the nathan boards. Technical report, Internal talk given on the meeting of the Electronic Vision(s) group, 2006.

[HH52] A.F. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol. (London)*, 117:500–544, 1952.

[HMW96] Ph. Häflinger, M. Mahowald, and L. Watts. A spike based learning neuron in analog VLSI. *Advances in neural information processing systems*, 9, 1996.

[Hoh05] S. Hohmann. *Stepwise Evolutionary Training Strategies for Hardware Neural Networks*. PhD thesis, University of Heidelberg, 2005.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[HP95] J. Hennessy and D. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1995.

[HY04] B. Hirschl and L. P. Yaroslavsky. Fpga implementations of sorters for non-linear filters. In *Proceedings of the XII. European Signal Processing Conference (EUSIPCO)*, Vienna, Austria, September 2004.

[Hyp06] HyperTransport Technology Consortium. *HyperTransport I/O Link Specification*, revision 3.0a edition, November 2006. Document No. HTC20051222-0046-0017.

[Int00] Integrated Device Technology, Inc. *128K x 36, 256K x 18, 3.3V Synchronous ZBT SRAMs, 2.5V I/O, Burst Counter, Pipelined Outputs*, idt71v2556 edition, October 2000. www.idt.com.

[JG93] H. Johnson and M. Graham. *High-Speed Digital Design - A Handbook of Black Magic*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 1993.

[KA88] J. Krüger and F. Aiple. Multimicroelectrode investigation of monkey striate cortex: Spike train correlations in the infragranular layers. *Journal of Neurophysiology*, 60 (2):798–828, 1988.

[lvd00] National Semiconductor Corporation. *LVDS Owner's manual*, 2.0 edition, 2000. www.national.com.

[Maa97] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10:1659–1671, 1997.

[MAG+02] M. Mazzucco, A. Ananthanarayan, R. L. Grossman, J. Levera, and G. Bhagavantha Rao. Merging multiple data streams on common keys over high performance networks. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–12, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[Max96] Maxim Integrated Products. *+3V, Quad, 12-Bit Voltage-Output DAC with Serial Interface*, max5253 edition, September 1996. www.maxim.com.

[McK99] N. McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, Volume 7, Issue 2:188–201, 1999.

[Mea05] K. Meier et al. FACETS - fast analog computing with emergent transient states (15879). EU FP6-2004-IST-FETPI, 2005.

[Mic02] Micron Technology, Inc., www.micron.com. *Small-Outline DDR SDRAM Module*, Jan 2002.

[MMS04] E. Müller, K. Meier, and J. Schemmel. Methods for simulating high-conductance states in neural microcircuits. In *Proceedings of the Brain Inspired Cognitive Systems (BICS)*, Stirling, UK, 2004.

[MNM02] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[MNM04] W. Maass, T. Natschläger, and H. Markram. *Computational models for generic cortical microcircuits*, chapter 18, pages 575–605. Number ISBN 1-58488-362-6. J. Feng, Boca Raton, 2004.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, pages 127–147, 1943.

[MP69] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[MS95] Z.F. Mainen and T.J. Sejnowski. Reliability of spike timing in neocortical neurons. *Science*, 268:1503–1506, 1995.

[Mue03] E. Mueller. Simulation of high-conductance states in cortical neural networks. Diploma thesis, University of Heidelberg, HD-KIP-03-22, 2003.

[Mul06] E. B. Muller. *Markov Process Models for Neural Ensembles with Spike-Frequency Adaptation*. PhD thesis, Ruprecht-Karls University Heidelberg, 2006.

[Nat01] National Semiconductor Corporation. *1.5A Low Dropout Positive Regulators*, lm1086 edition, August 2001. www.national.com.

[Ost07] B. Ostendorf. Charakterisierung eines neuronalen netzwerk-chips. *Diploma thesis*, University of Heidelberg, in preparation (in german), 2007.

[OWLD05] M. Oster, A. M. Whatley, S-C. Liu, and R. J. Douglas. A hardware/software framework for real-time spiking systems. In *Artificial Neural Networks: Biological Inspirations û ICANN*, pages 161–166. Springer, 2005.

[PGMS07] S. Philipp, A. Grübl, K. Meier, and J. Schemmel. Interconnecting vlsi spiking neural networks using isochronous connections. In *Proceedings of the 9th International Work-Conference on Artificial Neural Networks (IWANN'2007), accepted for publication*, 2007.

[Phi07] S. Philipp. *PhD thesis*, University of Heidelberg, in preparation, 2007.

[Raz96] B. Razavi. *Monolithic Phase-Locked-Loops and Clock Recovery Circuits*. IEEE Press, 1996.

[RD06] D. Shin R. Dömer, A. Gerstlauer. Cycle-accurate rtl modeling with multi-cycled and pipelined components. In *Proceedings of the International System-on-Chip Design Conference*, Seoul, Korea, October 2006.

[Ros60] F. Rosenblatt. Perceptron simulation experiments. In *Proceedings of the IRE*, pages 301–309, 1960.

[SA99] T. Shanley and D. Anderson. *PCI System Architecture*. Addison-Wesley Longman, Amsterdam, 4th edition, 1999.

[SBMO07] J. Schemmel, D. Brüderle, K. Meier, and B. Ostendorf. Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS'07)*. IEEE Press, 2007.

[Scha] J. Schemmel. personal communication.

[Schb] T. Schmitz. personal communication.

[Sch05] F. Schürmann. *Exploring Liquid Computing in a Hardware Adaptation: Construction and Operation of a Neural Network Experiment*. PhD thesis, Ruprecht-Karls University Heidelberg, 2005.

[Sch06] T. Schmitz. *Evolution in Hardware – Eine Experimentierplattform zum parallelen Training analoger neuronaler Netzwerke*. PhD thesis, Ruprecht-Karls-University, Heidelberg, 2006.

[SGMM06] J. Schemmel, A. Grübl, K. Meier, and E. Mueller. Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN'06)*. IEEE Press, 2006.

[SGOL+06] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gomez-Rodriguez, H. K. Riis, T. Delbrück, and S.-C. Liu. AER building blocks for multi-layer multi-chip neuromorphic vision systems. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1217–1224. MIT Press, Cambridge, MA, 2006.

[SH02] J. B. Sulistyo and D. S. Ha. A new characterization method for delay and power dissipation of standard library cells. *VLSI Design*, 15(3):667–678, 2002.

[SHMS04] J. Schemmel, S. Hohmann, K. Meier, and F. Schürmann. A mixed-mode analog neural network using current-steering synapses. *Analog Integrated Circuits and Signal Processing*, 38(2-3):233–244, 2004.

[Sin01] A. Sinsel. Linuxportierung auf einen eingebetteten powerpc 405 zur steuerung eines neuronalen netzwerkes. Diploma thesis (german), University of Heidelberg, HD-KIP-03-14, 2001.

[SKZ+03] D. Schubert, K. Kötter, H. Zilles, H.J. Luhmann, and J.F. Staiger. Cell type-specific circuits of cortical layer iv spiny neurons. *Journal of Neuroscience*, 23:2961–2970, 2003.

[SMA00] S. Song, K. Miller, and L. Abbott. Competitive hebbian learning through spiketiming-dependent synaptic plasticity. *Nat. Neurosci.*, 3:919–926, 2000.

[SMM04] J. Schemmel, K. Meier, and E. Mueller. A new VLSI model of neural microcircuits including spike time dependent plasticity. In *Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN'04)*, pages 1711–1716. IEEE Press, 2004.

[SMS01] J. Schemmel, K. Meier, and F. Schürmann. A VLSI implementation of an analog neural network suited for genetic algorithms. In *Proceedings of the International Conference on Evolvable Systems ICES 2001*, pages 50–61. Springer Verlag, 2001.

[SQ 04] Cadence Design Systems, Inc. *SPECCTRAQuest Simulation and Analysis Reference*, 15.2 edition, 2004. availbale from Cadence online documentation *cdsdoc*.

[Str97] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, Reading, MA, August 1997.

[sup03] *Super-LVDS Receiver/Transmitter Cell Design Manual*, version 6 edition, January 2003. Internal Document available from http://wwwasic.kip.uni-heidelberg.de/asiccc/docu.html.

[Syn04a] Synopsys, Inc. *Design Compiler User Guide*, June 2004. Chapter I-III, Version V-2004.06, available e.g from docserv.kip.uni-heidelberg.de.

[Syn04b] Synopsys, Inc. *Library Compiler User Guide*, 2004. Chapters 1-3.

[Syn04c] Synopsys, Inc. *Synchronous (Dual-Clock) FIFO Controller with Static Flags DW_fifoctl_s2_sf*, 2004. DesignWare Building Block IP, Version 2004.06.

[Syn04d] Synopsys, Inc. *Synchronous (Dual-Clock) FIFO with Static Flags DW_fifo_s2_sf*, 2004. DesignWare Building Block IP, Version 2004.06.

[Syn04e] Synopsys, Inc. *Synchronous (Single-Clock) FIFO Controller with Static Flags DW_fifoctl_s1_sf*, 2004. DesignWare Building Block IP, Version 2004.06.

[Tan03] A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458, fourth edition edition, 2003.

[The07a] The Neural Simulation Technology (NEST) Initiative. Homepage. http://www.nest-initiative.org, 2007.

[The07b] The Python Programming Language. Homepage. http://www.python.org, 2007.

[TMG02] C. Toumazou, G. Moschytz, and B. Gilbert. *Trade-Offs in Anaolg Circuit Design - The Designer's Companion*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[UMC03] UMC, Inc. *0.18um Mixed-Mode/RFCMOS 1.8V/3.3V 1P6M Electrical Design Rule*, ver. 1.2_p2 edition, 2003.

[Vir04a] Virtual Silicon Technology, Inc. *eSi-PLL Phase Locked Loop*, 2004. Version 2.1.

[Vir04b] Virtual Silicon Technology, Inc. *eSi-RAM/1P Single-Port Synchronous SRAM*, 2004. Version UMCL18U420T2_1.2.

[Vir04c] Virtual Silicon Technology, Inc. *eSi-RAM/2P Two-Port Register File SRAM*, 2004. Version UMCL18U420T2_1.3.

[Vir04d] Virtual Silicon Technology (VST). *0.18um VIP Standard Cell Library, Process: UMC Logic 0.18um Generic II Technology:* 0.18μm, July 2004. UMCL18G212T3, Revision 1.0.

[vR01] M. C. W. van Rossum. A novel spike distance. *Neural Computation*, 13(4):751–763, 2001.

[WG78] C.D. Woody and E. Gruen. Characterization of electrophysical properties of intracellularly recorded neurons in the neocortex of awake cats. *Brain Res.*, 158:343–357, 1978.

[wue] Würth elektronik. www.wuerth-elektronik.de.

[Xil02a] Xilinx, Inc., www.xilinx.com. *Virtex-II Pro Platform FPGA Handbook*, 2002.

[Xil02b] Xilinx, Inc. *Virtex-II Pro Platform FPGAs: Datasheet*, ds083 edition, 2002. www.xilinx.com.

[xil05a] Xilinx, Inc, www.xilinx.com. *Local Clocking Resources in Virtex-II Devices*, 2005. XAPP609.

[Xil05b] Xilinx, Inc., www.xilinx.com. *Synthesis and Verification Design Guide*, product revision 7.1i edition, 2005.

[Xil05c] Xilinx, Inc., www.xilinx.com. *Xilinx CORE Generator, Version 7.1i*, 2005.

Danksagung - Acknowledgements

Mein herzlicher Dank gilt allen, die zum Gelingen dieser Arbeit beigetragen haben. Insbesondere möchte ich mich an dieser Stelle bedanken bei

- Herrn Prof. Dr. Karlheinz Meier für die stets positive Unterstützung und die Möglichkeit, nach meiner Diplomarbeit auch im Rahmen dieser Doktorarbeit an den Hardwareaspekten der Electronic Vision(s) forschen zu können.
- Herrn Prof. Dr. René Schüffny für die freundliche Übernahme des Zweitgutachtens und für die angenehme Zusammenarbeit mit ihm und seiner Arbeitsgruppe im Rahmen von FACETS.
- Dr. Johannes Schemmel für die Zusammenarbeit an den beiden Spikeys. Seine Ideen und sein "Analogteil" haben meine Arbeit erst ermöglicht und er hat mir in vielen fachlichen Diskussionen stets kompetent und scharfsinnig weiterhelfen können. Nicht zuletzt möchte ich ihm auch für die (ent)spannenden Bike-Touren auf, über und um die Heidelberger Berge danken.
- Stefan Philipp und Dr. Tillmann Schmitz für die gemeinsame Zeit im Hardware-Zimmer, für bereichernde Diskussionen über VHDL-Design und natürlich für ihr VHDL-Design, das den Spikey-Controller so toll aufgenommen hat. Nochmals vielen Dank an Tillmann für das Korrekturlesen des Manuskripts.
- Dr. Martin Trefzer für seine unendliche Ausdauer beim Korrekturlesen des Manuskripts und für die vielen netten Gespräche - nicht nur im KIP.
- Daniel Brüderle für seine große Hilfe bei den letzten Spikey 2-Messungen.
- Dr. Ina Kyas für die langjährige Freundschaft seit unserem gemeinsamen Studienbeginn. Ihr motivierender Zuspruch, ihr Glücksschweinchen und natürlich auch das Korrekturlesen des Manuskripts haben einen großen Teil zum Gelingen dieser Arbeit beigetragen.
- Den Softies Dr. Johannes Fieres und Dr. Eilif Mueller für die Beantwortung meiner neurowissenschaftlichen Fragen.
- Ralf Achenbach für seine Hilfe im Reinraum und die Kämpfe mit dem (Auto)Bonder, das stets zielsichere Lokalisieren der Gerätschaften des ASIC-Labors und für viele schöne Kaffeekränzchen, die mir den Alltag beim Verfassen dieser Arbeit erleichtert haben.

- Markus Dorn für die zuverlässige Betreuung unserer CAD-Maschinen und die sofortige Hilfe bei irgendwelchen “shared object library not found”- oder sonstigen Katastrophen, die die Arbeit mit den Tools so begleiten.
- Boris Ostendorf für seine produktive Hilfe im Rahmen seiner Diplomarbeit.
- Den Mitarbeitern der Elektronikwerkstatt für ihre Unterstützung bei Bauteilfragen, Lötproblemen und in vielen anderen Belangen.
- Allen aktuellen und ehemaligen Freunden und Kollegen aus der Electronic Vision(s) Gruppe für die angenehme Atmosphäre im KIP und außerhalb des KIP.
- Allen Mitarbeitern des KIP für die freundliche und nette Atmosphäre, für die super funktionierende Infrastruktur, für die Hilfe bei allen bürokratischen Fragen und dafür, dass der Laden läuft.

- Meinen lieben Eltern für ihre moralische Unterstützung in allen Lebenslagen und nicht zuletzt für ihre finanzielle Unterstützung, ohne die weder mein Studium noch diese Doktorarbeit möglich gewesen wären.
- Meiner liebsten Ute für die vielen schönen Momente, die wir zusammen verbringen können, für die Ruhe, die sie mir gibt und dafür, dass sie immer für mich da ist. Aber auch für ihre bedingungslose Unterstützung während des Verfassens dieser Arbeit und für die schöne Aussicht, im Anschluss an meine Disputation unsere Hochzeit feiern zu können.

Erklärung:

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 02. Mai 2007

.....
(Unterschrift)