Dissertation

submitted to the

Combined Faculty of Natural Sciences and Mathematics

at Heidelberg University

for the degree of

Doctor of Natural Sciences

put forward by

Diplom-Informatiker:	Markus Roman Müller
Born in:	Mannheim
Date of oral exam:	....

Digital Centric Multi-Gigabit SerDes Design and Verification

Advisor: Prof. Dr. Ulrich Brüning

Abstract

Advances in semiconductor manufacturing still lead to ever decreasing feature sizes and constantly allow higher degrees of integration in application specific integrated circuits (ASICs). Therefore the bandwidth requirements on the external interfaces of such systems on chips (SoC) are steadily growing. Yet, as the number of pins on these ASICs is not increasing in the same pace - known as pin limitation - the bandwidth per pin has to be increased.

SerDes (Serializer/Deserializer) technology, which allows to transfer data serially at very high data rates of 25Gbps and more is a key technology to overcome pin limitation and exploit the computing power that can be achieved in todays SoCs. As such SerDes blocks together with the digital logic interfacing them form complex mixed signal systems, verification of performance and functional correctness is very challenging.

In this thesis a novel mixed-signal design methodology is proposed, which tightly couples model and implementation in order to ensure consistency throughout the design cycles and hereby accelerate the overall implementation flow. A tool flow that has been developed is presented, which integrates well into state of the art electronic design automation (EDA) environments and enables the usage of this methodology in practice.

Further, the design space of todays high-speed serial links is analyzed and an architecture is proposed, which pushes complexity into the digital domain in order to achieve robustness, portability between manufacturing processes and scaling with advanced node technologies. The all digital phase locked loop (PLL) and clock data recovery (CDR), which have been developed are described in detail.

The developed design flow was used for the implementation of the SerDes architecture in a 28nm silicon process and proved to be indispensable for future projects.

Zusammenfassung

Fortschritte in der Halbleiterfertigung führen weiterhin zur Realisierung immer feinerer Strukturen und erlauben immer höhere Grade der Integration in anwendungsspezifischen integrierten Schaltungen (ASICs). Als Konsequenz hieraus steigen auch die Bandbreitenanforderungen an den externen Schnittstellen solcher 'Systems on Chips' (SoCs) immer weiter an. Da allerdings die Anzahl der Kontakte an solchen Mikrochips nicht im gleichen Maße erhöht werden kann - bekannt als Pin Limitierung -, muss stattdessen die Bandbreite pro Pin gesteigert werden.

Die SerDes (Serializer/Deserializer) Technologie, die es erlaubt Daten seriell mit sehr hohen Raten von 25Gbps und mehr zu übertragen, ist eine Schlüsseltechnologie um der Pin Limitierung entgegenzuwirken und die Rechenleistung welche in heutigen SoCs erreicht werden kann auszuschöpfen. Da solche SerDes Blöcke zusammen mit der digitalen Logik die sie ansteuert komplexe mixed-signal Systeme bilden, ist die Verifikation von Leistungsfähigkeit und funktionaler Korrektheit eine große Herausforderung.

In dieser Arbeit wird eine neue mixed-signal Entwurfsmethodik vorgeschlagen, die Modell und Implementierung eng aneinander koppelt, um die Konsistenz über die einzelnen Entwurfsschritte hinweg sicherzustellen und dadurch den gesamten Implementierungsprozess beschleunigt. Ein Softwarewerkzeug, dass entwickelt wurde und sich gut in aktuelle electronic design automation (EDA) Software integriert und dadurch die Methodik in der Praxis unterstützt, wird vorgestellt.

Weiterhin wird der Entwurfsraum aktueller serieller Hochgeschwindigkeitsverbindungen analysiert und eine Architektur vorgeschlagen, in der die Komplexität in digitale Schaltungen verlagert ist, um ein robustes, zu anderen Fertigungsprozessen portierbares und mit Fortschritten in der Technologie skalierendes Design zu erhalten. Der digitale Phasenregelkreis und die Taktrückgewinnung, die entwickelt wurden, werden im Detail beschrieben.

Der entwickelte Entwurfsprozess wurde zur Implementierung der SerDes Architektur in einem 28nm Silizium Prozess verwendet, in dem es sich bewährte und sich auch für zukünftige Projekte als unverzichtbar erwies.

Contents

1 Introduction

1.1 Motivation

While the transistor count per area in modern system on chips (SoCs) is increasing, more and more data is processed and needs to be moved from and onto chips. As pin pitch and number of contacts is not scaling with the same pace, the bandwidth per pin needs to be increased in order to keep up with the processing power. To solve this problem known as pin limitation, high speed serialization/deserialization (SerDes) technology is necessary in order to drive innovation further.

A serializer samples parallel data and puts it serially on a single transmission channel with a respectively higher clock rate. On the receiving side, the serial data is sampled and deserialized to a lower clock speed parallel data bus again (see figure 1.1). With application specific integrated circuit (ASIC) core clock frequencies of 2 GHz and more and serialization factors of typically 16, extremely high speed circuits are required in these SerDes macros. Line rates of 25Gbps and more, where a single bit time lasts only about 40ps, require immense efforts to counter the effects of the lossy transmission lines.

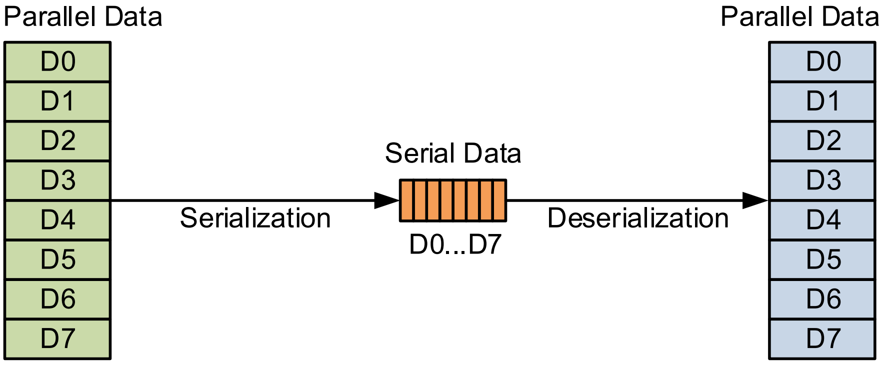


Figure 1.1: SerDes working principle

Because of the increasing importance for the overall system, high-speed serial links have become an extremely important component in todays systems on chips.

Complex protocols like e.g. PCI Express (PCIe) require a lot of interaction between the SerDes macro, which is usually implemented as an analog or mixed signal block in a full custom manner, and the digital logic implementing the actual protocol, which is synthesized from a hardware description language (HDL) using highly automated electronic design automation (EDA) tools.

The different design and verification approaches established over the years in full custom analog and semi custom digital design can lead to serious integration problems during the SoC implementation. This, in turn leads to exceeded deadlines, erroneous designs being submitted and costly chip re-spins. While synthesizable logic written in a hardware description language can be simulated and verified with an event driven simulator, a full custom block like a SerDes has a schematic representation and is usually simulated using circuit simulators like SPICE. Because computation times for SPICE simulations are magnitudes higher than those of event driven HDL simulation, abstract models need to be created for the full custom blocks in order to be able to simulate the whole SoC in a realistic time frame. The creation of fast, accurate models which are kept consistent throughout the overall design cycle is a key aspect for the entire SoC verification to work.

This thesis presents a design methodology, which ensures correctness between model and implementation, accelerates the design cycle and improves mixed signal verification to address the issues mentioned above. The architecture of a complex high-speed multi-gigabit SerDes macro is described and implemented by applying the developed methodology as a proof of concept.

1.2 Outline

In the subsequent chapter, current methodologies for semi and full custom designs are briefly introduced. After this, the design and verification methodology, which was developed in the course of this thesis and the tools assisting it are presented.

In chapter three, first an overview on current SerDes architectures is given. Afterwards the SerDes, which was designed and implemented in the course of this thesis is described. A certain focus is put on the clock data recovery (CDR) in the SerDes receiver, the phase locked loop (PLL) for central clock generation and the overall clocking architecture. The chapter is concluded with a look on testability and the physical coding layer which interfaces a SerDes.

Chapter four presents the physical implementation and the testchip for the SerDes architecture, which was taped out to manufacturing. Further, lessons learned during the actual implementation are stated, it is analyzed how the design did actually benefit from the methodology and what needs to be addressed additionally in the future in order to further improve the design cycle.

Finally a summary and conclusion as well as an outlook on future work conclude this thesis.

2 Design Methodology

2.1 Introduction

The design of a high-speed SerDes is a challenging task. As a complex mixed signal design, which is later on integrated in a larger system and also communicates with the outside world over a transmission channel, there are many topics to be addressed in order to produce a robust design, which is able to deliver a well specified performance.

First, the mixed signal design itself needs a proper verification methodology. For the digital part, which is synthesizable, well developed methodologies like Metric Driven Verification [1] using frameworks like the Unified Verification Methodology (UVM) [2] exist and can be utilized.

Assertions and test coverage databases can be used to ensure a certain level of confidence that everything has been tested exhaustively so the design does not contain any more bugs. These approaches are also well supported by state of the art EDA tools and automation flows. For pure analog circuit verification the primary approach is still often cell and block based testbenches complemented by spreadsheets to collect data if a certain cell meets specification or has been verified against certain manufacturing process corner cases.

This of course makes it very difficult to keep track of the verification status on a larger project. That may have not been a problem in the past when there was not much interaction between an isolated analog block and the rest of the often digital standard cell based design, but this is getting increasingly problematic as more tightly coupled mixed signal designs emerge. The lack of formalized analog and mixed-signal verification approaches can also be viewed by the fact that there have been very few publications on this matter in the past. Though, this is a topic which lately got traction in the EDA industry and methodologies as well as tools to address these problems are coming up [3].

The second topic is about models for system integration. As there are many interactions between analog and digital in the SerDes macro itself, as well as with the rest of the system a simulation of the complete design is necessary. By the usage of schematics and SPICE transistor models, it is almost impossible to simulate more complex scenarios because of the required simulation time intervals. To tackle this problem, modeling languages have been developed and extended to allow the description of analog and mixed signals systems, namely VerilogA, Verilog AMS, AHDL and now SystemVerilog [4]. The major challenges here are that the models need to be precise enough to reflect the actual cell behaviour and still be coarse enough to get a speedup in simulation time. As the models are normally developed separately and after the actual implementation of the cells, it can be hard to keep them consistent with the actual design.

Another aspect is performance prediction and verification. The performance of the system to be designed needs to be predictable - preferably early in the design phase. This can be done by early first order estimations, calculations in MATLAB or simulations in system simulators like ADS or Simulink. Yet, later on it needs to be verified that these simulations and estimations are matched by the actual implementation, which can be hard to do.

Last but not least, testability and design for test (DFT) is of great importance for a successful design. After the completed chip is received back from manufacturing one obviously needs to be able to verify functionality as well as performance. Also, if problems turn up there need to be enough means to be able to identify the root cause inside the circuits to be able to find a workaround or fix in a next iteration. Additional logic or circuits have to be added to the design in advance to be able to accomplish this and testability has to be taken into account during the design phase.

In the following an overview over current developments in the field of mixed signal design methodologies is given. After this, the top-down methodology developed in the course of this thesis is presented.

2.2 State of the Art Design Methodologies

When looking at design methodologies, one has to take the whole process into account. This may span over writing HDL code, creating schematics, implementing a layout and verifying correctness at various stages during the design flow. For all of this, quite different approaches in analog full custom design and semi custom digital design exist.

Whereas a pure digital design is usually written in an HDL and therefore text based, a full custom design is generally created on a schematic level and therefore naturally has a graphical representation. The digital implementation flow is highly automated and constraint driven - EDA tools like HDL synthesis and place & route e.g. receive timing and placement constraints, then the design is implemented with the use of optimization algorithms.

Analog design in contrast is often a very manual process - almost all tasks during schematic and layout generation are directly executed by the designer himself.

Further, for verification of the HDL based digital designs, metric driven verification (MDV) using constraint random stimulus is well established with industry standards like the 'Universal Verification Methodology' (UVM) [2]. This enables automated testing like regression tests together with collection of test coverage in order to be able to assess how good a design was actually verified. In the world of analog design, not much of this kind of automation is present. Design verification is often mostly done manually and often simulation outputs are judged by 'visual inspection' if a design works correctly or not. Further, there is no well established way how test results are eventually collected to asses verification quality.

These different approaches obviously clash in the creation of a mixed signal design. And, though the main portions of a big SoC may be synthesized logic almost every chip needs some analog parts, which may be small in some cases but vital for the overall operation of it. Additionally with shrinking feature sizes in the semiconductor industry and more 'digitally assisted analog' circuits [5], the connections between digital and analog parts tend to be closer and more complex, which increases the requirements and problems.

The EDA industry has noticed this and efforts are undertaken to bring the two worlds closer together, mostly by trying to apply some of the automation and methodology known from the semi custom digital design flow to the analog/mixed-signal world. For example there is the attempt to extend the UVM methodology to metric driven analog verification, namely UVM-MS [6]. Also, tools are coming up to gather analog simulation results, to get a global view on the verification status of a design, which is called ’plan based verification’ [3] by e.g. Cadence Design Systems. To complement this, means for self-checking like assertions, which are a well known technique in programming are brought to SPICE based circuit simulators [7].

Also, the overall design approach is often quite different. Whereas digital logic is usually implemented in a top-down process, analog circuits are often designed in a bottom-up block oriented fashion. This can lead to architectural problems showing up late in the design cycle, which leads to time consuming redesigns [8].

There have been many efforts to bring the top down design process into the analog world.

One approach to increase automation in analog design are circuit generators, which build both layout and schematics for entire blocks like DACs, ADCs, DC-DC converters etc. There are two different classes of circuit generators. The first one is constraint or optimization driven, which means the designer inputs constraints for the circuit he wants to generate, and the design is then synthesized from templates with the help of optimization algorithms. For this, the circuit generator does not need to follow the same procedure that a human designer would do. Though, top-down generated designs are often rejected by designers, because their expectations are not met [9]. This may be for example because the synthesized design does not exhibit the same symmetries or layout that the designer originally envisioned - even if they have no impact on the circuits performance.

The second class are design procedure driven generators [10], [11], [12]. Here a human ’expert’ designer basically codifies the design procedure that he would adhere to. Therefore this approach and the outputs of these generators are more comprehensible and are more likely to be adopted by designers.

All generators have the advantage, that they automate the cell- or block-level implementation and also improve portability to other process nodes.

2.3 Mixed Signal Modeling

Creating simulation models for a mixed-signal design is a very important task to facilitate integration and verification in the context of a complete chip. Without them, the interaction between synthesizable parts and full custom blocks can not be verified. SPICE models, which are used for transistor level simulations are not suitable for this task because of the long simulation run times. There are several programming languages, which can be used to create more abstract models to increase simulation performance (see figure 2.1). The challenge here is the trade-off between accuracy and simulation performance.

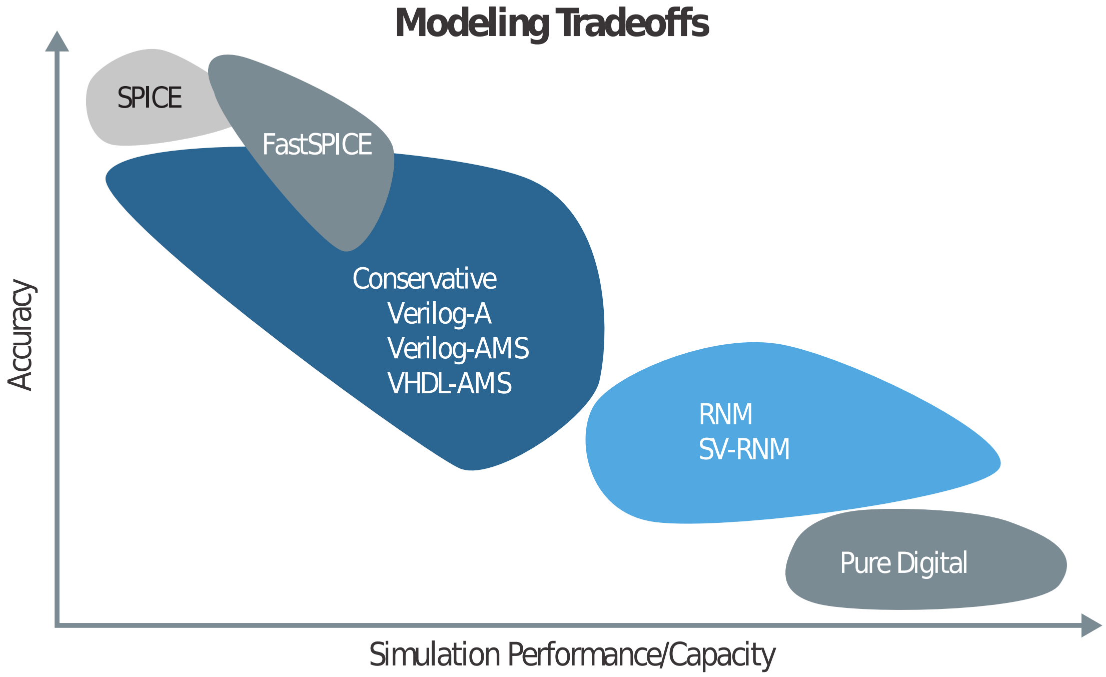


Figure 2.1: Performance to accuracy trade-off for different modeling styles [13]

The main languages for the creation of analog/mixed-signal behavioral models are

- Verilog A
- Verilog AMS
- SystemVerilog (with real number modeling extension since IEEE1800-2009)

VerilogA targets continuous time analog behavioral modeling. It allows to model behavior in the voltage/current domain and the simulator still needs to solve Kirchhoffs Current/Voltage Law (KCL/KVL). Although an abstract model written for an analog block in VerilogA executes faster than the schematic implementation, the need for a SPICE simulator is a drawback for fast system simulations.

There is no support for event driven digital logic, which makes it unsuitable for mixed signal modeling.

This restriction is resolved in VerilogAMS, which combines the capabilities of VerilogA with those of Verilog-2005 [14]. The parts of the behavioral model, which represent the analog part, can be described in the continuous time voltage/current domain. The digital part is described in standard Verilog. Events can be created from certain conditions happening in the analog domain to facilitate interaction between continuous time and event based simulation.

Another interesting feature, that is added in VerilogAMS is the introduction of the *wreal* signal type. Whereas in pure Verilog a module port can only have the values 0,1,X,Z this adds the capability to have real number valued ports, which makes it possible to transport floating point values from one module to another and therefore model analog behavior purely in an event driven simulator. When only *wreal* nets are used instead of electrical voltage/current nets, the simulation speed is increased a lot. Though, a huge drawback is, that only one real value can be assigned per port, which e.g. implicates that either voltage or current information can be transfered to the next module.

With IEEE1800-2009 the *wreal* capabilities were also added to SystemVerilog, which opens up the whole testbench code and verification capabilities to mixed signal modeling and more importantly makes it possible to simulate real number models in a pure Verilog/SystemVerilog simulator. After *wreal* seems a very promising approach, the newest revision of the standard (IEEE1800-2012 [4]) adds some more interesting features for analog/mixed signal behavioral modeling.

Specifically user defined types (UDT) and user defined resolutions (UDR) were added. UDTs allow the creation of custom types, which can be constructed of multiple values. This overcomes the problem that only one real value can be passed to other modules per port. Now a user defined type can be constructed that e.g. consists of one real value for current and one for voltage. As ports can be of a user defined type, one port can now transport multiple values. With UDRs, custom resolution functions can be defined, which handle the problem of multiple drivers when UDTs are involved. Every time a UDT net is driven from multiple drivers the defined UDR is called to compute the actual value of the net (or each value of the net, if it is a multi value net) and resolve the conflict.

To emphasize the possibilities of user defined types and resolution functions, the model of a simple ring oscillator as depicted in figure 2.2 is described for illustration below: To reflect the modular implementation structure of the ring oscillator, the inverter and the controllable capacitance should are modeled in

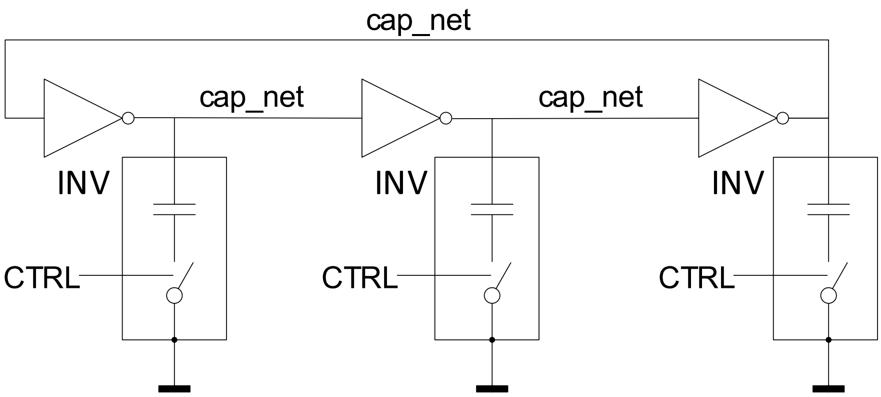


Figure 2.2: Simple ring oscillator model example

separate modules in this example. The capacitance bit model should be able to interact with the inverter model to modify its propagation delay and hereby eventually control the oscillator frequency. Still, the inverter model and capacitance bit should have the same portlist like the actual schematic implementation will later on have. Therefore, load capacitance and time domain waveform both have to be present at the inverter output. To accomplish this, a user defined net type *cap_net* together with a resolution function will be defined in the following.

A struct *cap_struct* is defined, which is constructed of a real value C for capacitance and a wire *net* for a digital time domain waveform.

```
typedef struct {
   logic net;
   real C;
} cap_struct;
```

On the basis of this struct, a resolution function *c_sum* is defined, which iterates over all drivers of the net, constructs the time domain waveform and accumulates all capacitance present at the node. This resolution function is used to create the nettype.

```
function automatic cap_struct c_sum (input cap_struct driver[]);
   foreach(driver[i]) begin
      c_sum.C += driver[i].C;
      if (driver[i].net === 1'bZ) begin
         c_sum.net |= 1'b0;
      end else begin
         c_sum.net = driver[i].net;
      end
   end
endfunction

nettype cap_struct cap_net with c_sum;
```

The simplified models for inverter and capacitance bank can use the nettype, to model the ring oscillator as follows:

```
module INV ( input cap_net IN, output cap_net OUT );
   reg out_net = 0;
   assign IN = cap_struct'{1'bZ, 0.1 }; //add load cap
   always @ (*) begin
      #(10.0+OUT.C); //compute delay
      out_net = ~IN.net;
   end
   assign OUT = cap_struct'{out_net,0.0 };
endmodule
```

The inverter model *INV* assigns a capacitance to its input *IN*, which represents the load for the preceding stage. The output net *OUT* is driven with the complement of the input after a delay, which is calculated from a fixed delay and the load capacitance *OUT.C* seen at the output.

```
module CAP ( input wire CTRL, output cap_net OUT );
   assign OUT = CTRL ? cap_struct'{1'bZ, 0.1 } : cap_struct'{1'bZ, 0.2 };
endmodule
```

The switchable capacitor model *CAP*, adds a capacitance to the output node *OUT*, depending on the control input net. It is not driving any waveform on the node.

From these modules the ring oscillator, which is depicted in figure 2.2 can be constructed:

```
module toplevel;
   interconnect wire12, wire23, wire31;
   ...
   INV INV1 (.IN(wire31), .OUT(wire12));
   INV INV2 (.IN(wire12), .OUT(wire23));
   INV INV3 (.IN(wire23), .OUT(wire31));

   CAP CAP1 (.CTRL(ctrl), .OUT(wire12));
   CAP CAP1 (.CTRL(ctrl), .OUT(wire23));
   CAP CAP1 (.CTRL(ctrl), .OUT(wire31));
   ...
endmodule
```

At the toplevel, the inverter and switchable capacitor instances can be connected using so-called *interconnect* nets. These nets are basically typeless and are coerced to a certain type by the sinks or drivers connected to it through the module hierarchy. This makes it possible to use the same structural Verilog description and swap and mix different model types, such as RNM or SPICE in and out depending on the current simulation scope.

These concepts allow to model the actual implementation very closely using real number models, while keeping all the module interface definitions of the actual implementation. This is a very important property, which is used in the following top-down design methodology.

2.4 Top-Down Design Methodology

The main goals of the mixed-signal top-down design methodology developed in the course of this work were to keep the model of the system in sync with the actual implementation and vice versa, to predict the performance early in the design cycle, identify and reuse as many common building blocks as possible and to be able to handle a complex design with a small team.

To accomplish this, the idea is to first model the complete system that is built from both synthesizeable logic and full custom modules in a top-down approach using SystemVerilog HDL. The design hierarchy is fully differentiated down to the individual modules, which later will have a transistor implementation, like depicted in figure 2.3. Hereby, it is important to distinguish between the different cell types which are present in the design and adhere to the following naming convention:

- "verilog": modules written in synthesizable verilog
- "structural": full custom blocks which only contain leaf cells, but no primitives such as transistors etc.
- "functional": digital only model for a full custom leaf cell
- "rnm": real number model for a full custom leaf cell
- "leaf": generalized leaf cell model usable in both real number and functional context

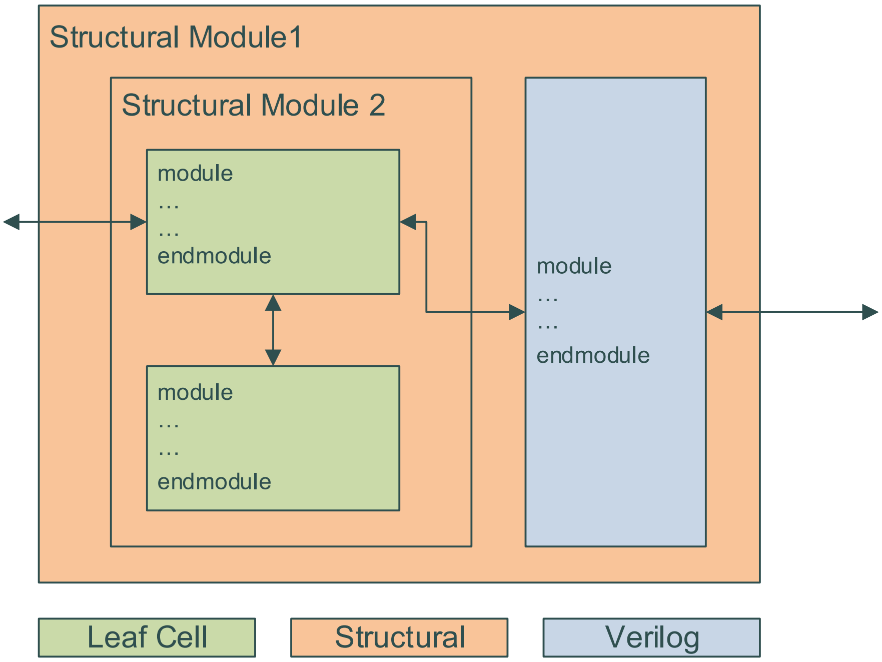


Figure 2.3: Design hierarchy example

Structural cells therefore only describe the connectivity of full-custom cells. This means the structural modules must only contain instantiations and connections between modules without any behavioral code statements which would require synthesis. Structural modules can either instantiate other structural modules or so called leaf cells.

A leaf cell needs at least two model views. A functional model which only exhibits basic behavior of the cell, like signal flow, resets, power downs but does not model cell performance metrics. These models are very fast and can be used in large system level simulations. The other type of simulation view is the real number model (RNM). These models exhibit analog metrics of importance to system scope and cell performance and can be used to explore the design space and system behavior. The metrics derived and defined for these models later serve as a specification for the full custom cell designer. For better code reuse there is also the *leaf* cell type which contains an instance of a more generalized model. This is for example useful, when there are multiple amplifiers present in the design, which all have different characteristics. First, a parameterized amplifier model is created for each functional and rnm context. Then, multiple different leaf type cells can instantiate this amplifier model, each with a different set of parameters. Because the actual full custom implementation will differ, it is necessary to have different cells, but for the simulation, they can all share the same parameterized model.

Over time a library of generalized models for all commonly used leaf cell times is developed, which accelerates design space analysis.

To keep the developed models in sync with the actual implementation as much as possible, the structural verilog and the leafs cells are used to automatically create the schematic hierarchy which is used in the full custom design flow. For this, a special tool was developed and integrated into the full custom design environment of Cadence Virtuoso, as will be described in section 2.5.

The text based Verilog views are handled well in version control tools like *svn* or *git*, which eases collaboration between multiple designers. The binary coded schematic view files used in EDA tools like Cadence Virtuoso can always be regenerated to keep consistency.

Changes like adjusting bus widths or introducing new pins in cells deep down in the hierarchy are faster to incorporate in Verilog than in schematics, where pins need to be added/edited and symbols regenerated for each level of hierarchy. Additionally integrated development environments (IDEs) can be used on text views, which have sophisticated refactoring capabilities to ease such time consuming tasks.

As the structural modules only contain instantiations and connectivity they can be directly synthesized to schematics. For the leaf cells, schematic templates for the designer are generated. These templates exhibit correct port names, widths and serve as specification for transistor level implementation. Additionally the parameters, which are passed to the models to specify circuit performance, such as gain, poles, zeros etc. are annotated as comments in the schematic.

Because the RNM models exhibit the critical performance parameters, the actual cell implementations as well as the complete system can be checked against them as a golden sample.

For this methodology to pay off, it is essential to push as much connectivity into structural modules and keep the leaf cells as simple as possible. The average leaf cell usually only contains a handful transistors or other devices. All connectivity and complexity, which is already present in the structural module hierarchy can be verified early and will later be automatically generated, hereby eliminating possible mismatches between model and implementation.

Once the complete mixed signal macro is described, the model can already be used at the SoC level for further integration and verification while the mixed signal macro development is still ongoing - guided by the leaf cell models.

Through the use of generated templates for the leaf cell implementation, the module boundary such as port widths, port names and instance names stay consistent throughout the design cycle. It is also easier to work with a team on the design as block level implementations derived from the templates are guaranteed to fit together in the end. The complete design flow is depicted in figure 2.4.

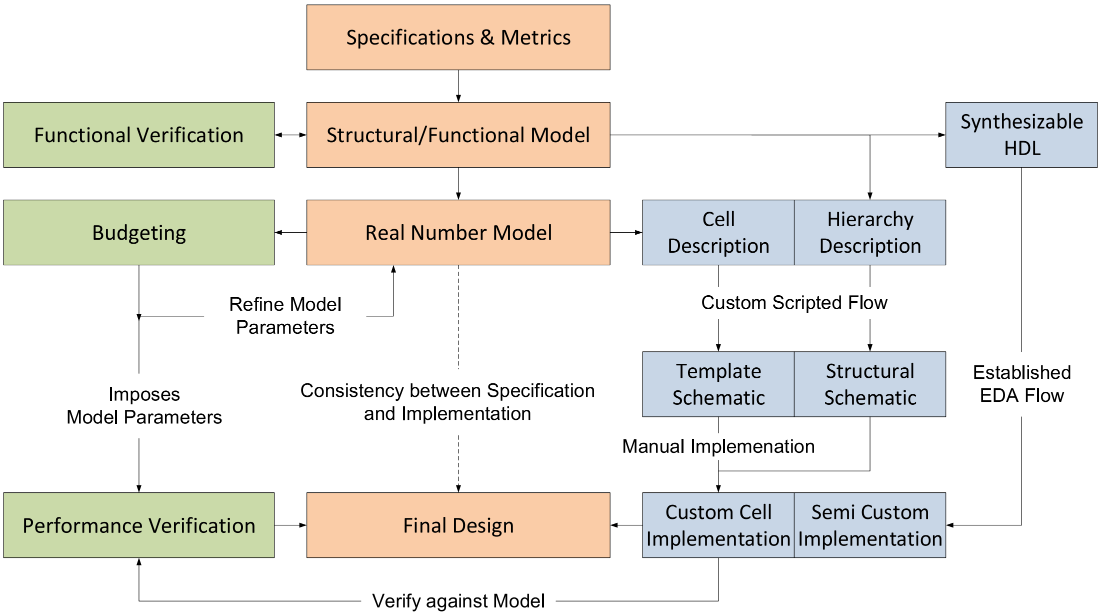


Figure 2.4: Overall mixed-signal design flow

In a top-down approach the structural/functional model is created from the specification. This model is already fully differentiated down to the leaf cell level and exhibits the same module hierarchy that the actual implementation is going to have. It is used for functional verification purposes. For implementation the digital portions of the design, which have been modeled in synthesizable Verilog are handled by the established semi-custom EDA flows. For the custom, analog parts first real number models are created, which are used as specification for the schematic implementations. As the real number models are used to accurately model the performance of the complete system, they are used in the Budgeting step to determine the required leaf cell parameters in the overall system context. Using the developed custom scripted flow, the full custom schematic hierarchy is generated, which ensures consistency between model an implementation.

With the help of schematic driven layout tools like Cadence Layout XL, the layout of structural cells can be directly generated from the schematics. This ensures, that the connectivity is the same like in the structural model and correct by design from model to full custom layout.

All early design space exploration is faster and more efficient in this top-down design flow. Cells, which can be reused at different places in the design are identified easier and problems arising from interactions on the system level are found early in the design phase. This increases the chance of avoiding fundamental problems during the implementation phase which are costly and can ruin the complete tapeout schedule. Also, the verification environment can be developed in parallel. The real number models can be used during the development phase of the testbenches and can later be replaced with SPICE netlists, where required. By this, the methodology accelerates the design process significantly, avoids errors and provides higher flexibility for design changes. Implementation and verification of the complete mixed signal system are effectively coupled into the same design flow.

2.5 Schematic Generation Tool

In this section the actual schematic and leaf cell template generation, depicted in figure 2.5, is now described in more depth. The tool is build around *Genus*, the HDL synthesis solution from Cadence and Virtuoso, which is the full custom implementation tool. It is mostly written in TCL with some parts being SKILL code, which is necessary to control Virtuoso.

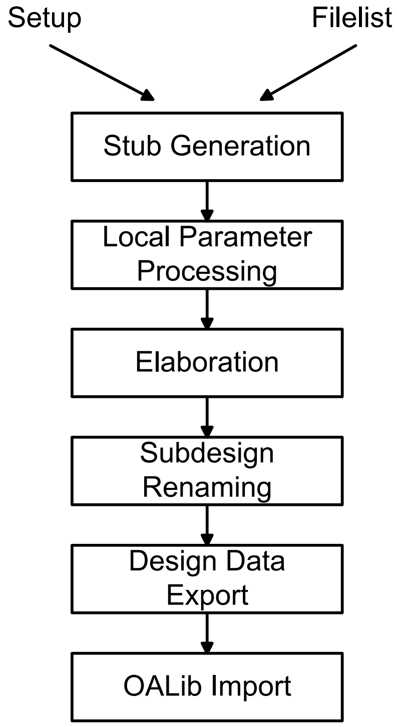


Figure 2.5: Schematic generation flow overview

The structural modules, as well as the leaf cells are assumed to be stored in the OpenAccess database used by Virtuoso. This makes sense, because the actual implementation based on the generated schematics and templates will take place in Virtuoso. Synthesizable HDL code can be stored anywhere on the filesystem and does not necessarily need be in an OpenAccess library (oalib).

Two input files are needed for the schematic generation tool to work properly. One is a filelist, which contains a list of all verilog sources necessary for the design (structurals, leafs, synthesizable HDL). The filelist can actually be the same file, which is normally used as input for the event driven simulator used for mixed signal verification - in this case Cadence Incisive. This helps to keep verification and schematic generation consistent. The other input is a setup file, which contains information about the path to the oalib, the toplevel instance name and the open access library names used for the generation process.

Next, the design hierarchy has to be elaborated. Module parameters need to be evaluated and values propagated from the toplevel instance down to all leaf cells. Verilog *generate* statements need to be evaluated, which can create or remove instances and modify the design hierarchy. For this, like mentioned earlier, the capabilities of *Genus* are used, as this is normally the first step for HDL code synthesis. A custom or open source solution might be used for HDL elaboration as well in the future, as the current approach also has some limitations, which need to be worked around. As *Genus* can only work on synthesizable constructs and the leaf cell modules contain all sorts of non synthesizable SystemVerilog code like User Defined Types, the first step before elaboration is to generate Verilog stub modules. These stubs only contain the module declaration, local parameter definitions and comments. UDTs are replaced with simple wires. Next, *Genus* is not able to correctly evaluate and elaborate real valued parameters, which makes it necessary to extract all local parameters of the leaf cells before elaboration, in order to be able to evaluate them separately afterwards. Regular expressions are used to extract module header, parameters etc. from the original source file.

During elaboration, beginning from the toplevel instance, which was specified in the setup file, the design hierarchy is build up and parameters are passed through the structure down to the leaf cells. Further more, problems in the design like unconnected ports, wrong port widths etc. show up now, allowing the designer to fix them early in the design cycle.

After elaboration the real valued local parameters, which might be dependent on other integer valued module parameters are evaluated. A SKILL script is created, which is later processed in Cadence Virtuoso to annotate these parameters to the schematic template cells as text comments and oalib properties.

Next, the design hierarchy is traversed to see if modules, which are going to have a full custom implementation have been instantiated with different parameters. If so, the names of these so called subdesigns need to be changed accordingly because the actual implementation needs to differ and multiple cells need to be generated. All parameters of the instances are compared and only differing ones are used to rename and identify the subdesigns. The parameters are appended to the original module name.

An example is a buffer leaf cell 'BUF', which has a parameter D for 'drive strength' and is instantiated with e.g. $D = 4$ and $D = 3$. Though there was only one leaf cell as input for the schematic generation, there will be two leaf cells created in the Virtuoso database: BUF_D4 and BUF_D3, because they need a different schematic implementation.

At this stage an overview of how often which cells are used in the design is generated as well. This helps to identify possible opportunities for cell reuse or can show issues with excessive parameterization, which later on will create manual implementation work.

At last, structural and leaf cells are exported as verilog files and afterwards imported as schematics to the open access library where they were originally read from. For this, the verilog import function of Virtuoso is used, which is able to create schematics and symbols from verilog source code. In order to be able to construct the structural schematics correctly, the import order has to be controlled in advance. This makes sure all necessary symbols are present for each schematic once it is generated. The SKILL scripts, which were generated earlier to annotate the parameters are afterwards processed in a Virtuoso session.

The result of the schematic generation tool are generated schematics for all modules, which had a structural SystemVerilog description and schematic templates for all leaf cells. The schematic templates contain correct port names and bus widths, derived through the hierarchy. Parameters, which were used in the RNM models are annotated to the schematic templates as comments to guide the designer in the implementation process.

To ease the usage of the schematic generation flow, it has been integrated in the context menu of the Virtuoso Library Manager. The designer can right click on a cell and chose to generate a schematic or template either for all hierarchy starting from the selected cell or only for the selected cell. The schematic generation tool is then invoked in the background. After completion the current libraries are refreshed. This integrates the methodology seamlessly into the known working environment, makes it easy to use and increases the acceptance by designers, which is an important factor.

The actual TCL implementation of the tool is not further discussed here, as there is not much scientific value in the code itself, but in the process that has been described earlier.

2.6 Liberty File Generation Tool

During the back-end implementation process of the mixed signal macro, full custom modules are used together with synthesized logic. Timing information, so called timing arcs, for the full custom blocks can be passed to the synthesis tool using timing abstract models. The industry standard to describe timing information during synthesis is the Liberty File Format (.lib). So, for proper integration into a mixed signal design flow, the generation of .lib files for the implemented full custom blocks is necessary.

There are commercial tools such as 'Cadence Liberate' available, which analyze and extract timing arcs from a design. Though in the course of this thesis it turned out, that these tools are mostly focusing on CMOS based standard cell logic characterization. Additionally the setup is quite complicated, which might be exaggerative if only a very small number of cells with fairly easy timing arcs are to be characterized. Therefore a .lib file generation tool was implemented in TCL, which complements the described methodology and helps enforcing consistency throughout the whole design flow.

The tool takes a setup file and the structural Verilog description of the full custom toplevel as inputs. All input/output ports are extracted from the Verilog description, which makes sure that the .lib file and the implementation can easily be kept in sync with respect to port names, bus width etc. The setup file defines all timing arcs, which are present in the design and additional information which should be added to the .lib file later on.

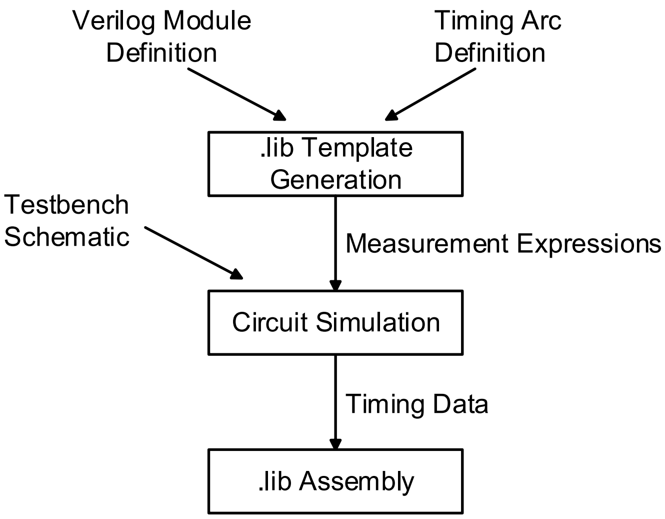


Figure 2.6: Timing library generation flow overview

In a first pass, the Verilog source code is read and a .lib file template is generated. This template file already contains all non corner related information, such as pin and timing arc definitions, but lacks actual values for propagation delay or setup/hold times. This corner dependent timing information needs to be derived from circuit simulations first and is then annotated to the template file in a second pass later on. To ease the testbench creation and timing annotation to the template, measurement expressions for Cadence' simulation environment *ADE XL* are created. These expression can be imported in the ADE XL GUI and make sure that simulation results, which are exported from the testbench match the ones expected by the .lib file generation.

The testbench to derive the timing arcs needs to be created manually. The creation of a suitable simulation environment as well as stimulus generation is very hard to automate, because every circuit that has to be characterized is slightly different and has to be driven differently in order to stimulate the desired timing arcs. A textual description of the necessary stimulus which would be needed for automatic testbench generation, that can support a wide range of different circuits would probably be more complicated for the user than the manual testbench schematic creation.

As the actual simulations then take place in ADE XL, all features like corner setup and distributed processing can be used to generate the desired timing arc data. After the simulations complete, the results of the measurement expressions are exported to comma separated values (CSV) files. CSV files exported from multiple testbenches can then be processed by the .lib generation tool in a second pass to build the timing tables and fill them into the template file. For every simulation corner defined in the ADE XL corner setup a copy of the template file is created and the respective timing data is inserted. The .lib file names are automatically generated from the operating conditions and include process, voltage, temperature (PVT) for a consistent naming scheme.

2.7 Link Budgeting

As one of the goals of this thesis is the implementation of a high-speed serial link, a major concern is to be able to verify and predict the performance of a SerDes design in advance.

Link budgeting tries to gives an answer to the question how every component in a serial high speed link contributes to the overall performance - or rather how much each component e.g. the transmitter, channel and receiver degrades the quality of a link. The main figure of merit is the bit error ratio (BER).

Besides all functional verification done on a SerDes design, it is a very important question which BER is going to be achieved for a given SerDes and transmission channel setup. The bit error ratios, which need to be achieved for protocols like PCIe or Ethernet lie in the range of 10^{-12} to 10^{-15} [15]. This means, that for a 10G Ethernet link with 10.000.000.000 bits being transmitted per second, one error is allowed to happen every 100 seconds to achieve exactly a BER of 10^{-12}. It is obvious then, that with practical circuit simulations where maybe only 1µs of normal operation is simulated, the existence of multiple bit errors during this time would indicate that the design is completely broken.

Additionally, because of the unbounded statistical processes which play a role in BER calculations, even the simulation of error free operation of 100 seconds would not be enough to indicate a BER of 10^{-12}, as at least $3 \cdot 10^{12}$ bits are required to measure such a BER with 95% confidence [16]. In the following, first some more detail is given on what factors play a role for serial link performance and afterwards the link budgeting procedure for the SerDes developed in the course of this work is described in more detail.

There are two types of graphs often used to asses the quality of a high speed serial link. One is the so called eye diagram and the other one is the bathtub curve.

An eye diagram (see figure 2.7) is usually constructed from a recorded time domain waveform. The time information of each sample that has been captured is taken modulo by the bit time, when plotting the diagram. Hereby the waveforms of all bits that are transmitted are overlayed multiple (thousand or even million) times. By this, a graph is constructed that essentially depicts what the probability of the sampled signal is to take on a certain value. From this plot it is then also clear that the best sampling position to decide if the received bit is a zero or a one, is the middle of the eye. If this point is never crossed by the received signal, every bit can be correctly identified. There will be no bit errors, and the eye is called open. The open space left in the middle of the eye is then the eye-margin.

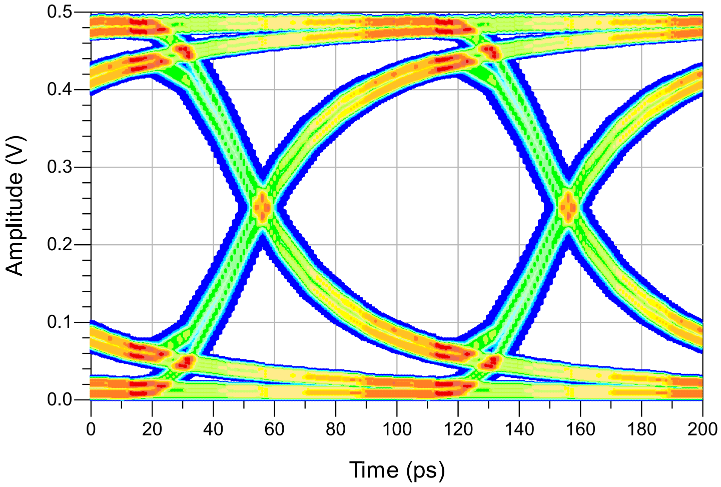
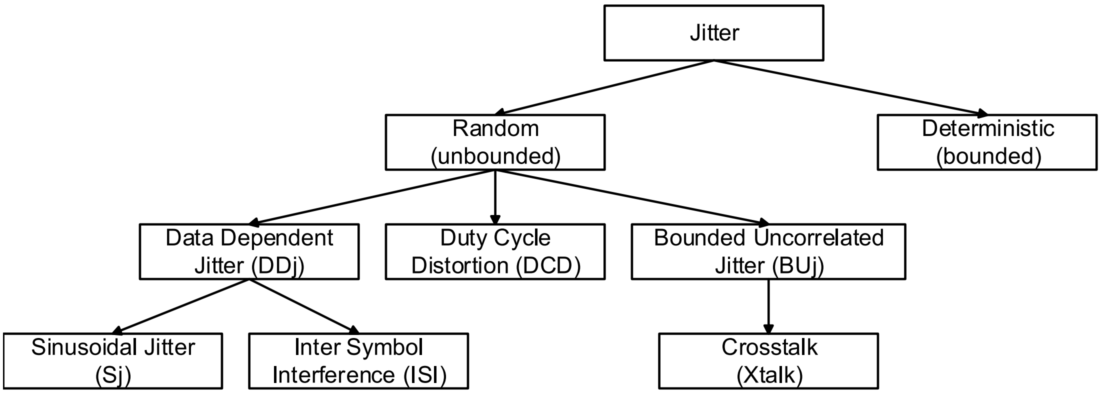


Figure 2.7: Eye diagram of a 10 Gbps signal

Variations in the time (horizontal) or the voltage (vertical) domain, which cause the eye to close by shifting the signal with respect to its ideal position are called jitter and noise respectively. There are different types of jitter, which are caused by different underlying phenomena in a real serial link. Jitter can be divided into two different main classes: deterministic and random.



Jitter
Random (unbounded)
Deterministic (bounded)
Data Dependent Jitter (DDj)
Duty Cycle Distortion (DCD)
Bounded Uncorrelated Jitter (BUj)
Sinusoidal Jitter (Sj)
Inter Symbol Interference (ISI)
Crosstalk (Xtalk)

Figure 2.8: Jitter type taxonomy

Whereas for deterministic effects, it can be said with certainty how they affect the signal, for random effects there is only a certain probability, which can be used to predict the effect with a certain confidence [16]. The effect of deterministic impairments is generally bounded, whereas random jitter is unbounded. Figure 2.8 gives an overview on different jitter classes which are briefly discussed in the following.

The most obvious deterministic effect in the context of serial links is so called inter symbol interference (ISI), a type of data dependent jitter (DDj).

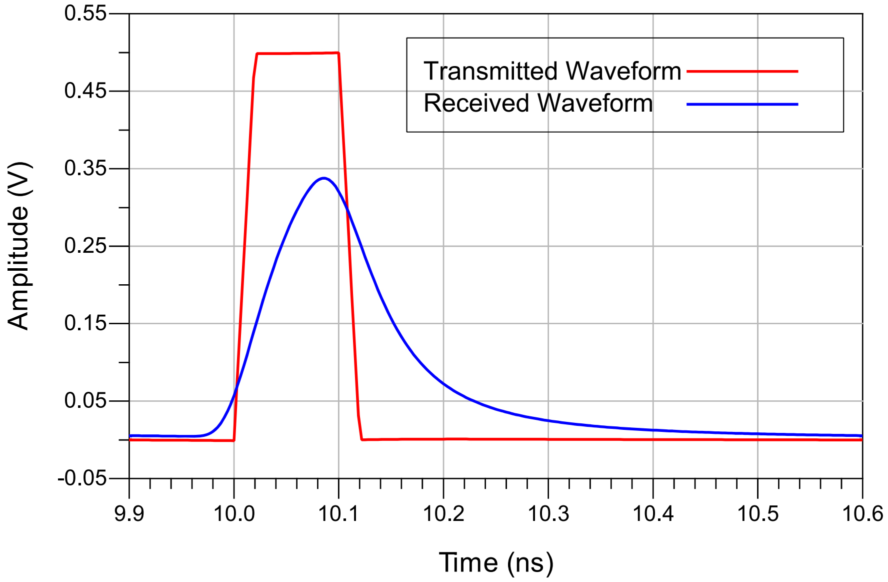


Figure 2.9: Single bit response, the waveform of a lone bit transmitted over the channel, spreading into multiple unit intervals

Because of the low pass characteristic of common transmission channels, the transmitted symbol (zero or one) will smear into the next symbol time(s) and affect the waveform. The single bit response (SBR) (figure 2.9) can be used to analyze how subsequent bit times are affected by a previously transmitted symbol. If the transfer function of the transmission channel is assumed to be fixed over the time of operation, the SBR and therefore the ISI will always be the same and hence deterministic for a given bit sequence. Coding of the transmitted data can be used to only allow certain patterns and limit the number of consecutive equal symbols (maximum run length), thereby limiting the ISI. Figure 2.10 shows the resulting ISI for a PRBS31 pattern which contains up to 31 consecutive ones and zeros and 8B/10B coded data, which has a maximum run length of 5 and a limited amount of allowed symbols. In modern protocols, such as PCIe 3.0 even longer run length of 130 bits are theoretically possible because of the use of scrambling polynomials to code the data [15]. The actual difference observed, though depends on the SBR and therefore on the channel characteristics.

Another deterministic jitter type is sinusoidal Jitter (Sj), which can be caused by deterministic noise sources on clock generation or power supply. If there is for example a sinusoidal perturbation on the sampler clock, this will have an impact on the eye diagram, because the sampling clock is now no longer in the center of

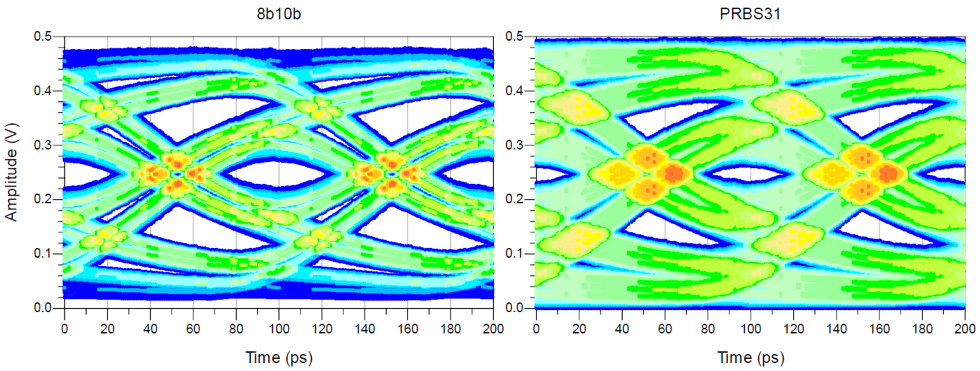


Figure 2.10: Eye diagrams of 8b/10b coded and PRBS31 data

the eye, but has a certain probability for its position. This is best described by a probability density function (PDF). In short, a PDF $f(x)$ gives the likelihood that a sample of a certain random variable X will take on the value x. It is important to say, that the PDFs of these sources are bounded and therefore can be characterized by a peak to peak value.

Additionally, real samplers in a receiver are not like ideal dirac samplers, that are used in signal processing theory, which are able to obtain an instantaneous signal value in an ideal way [17]. In reality a sampler has a sampling aperture. This aperture spans over a certain time interval and the input waveform over the whole interval is taken into account to form the sampled value. This can be characterized, extracted as a PDF and used in the link budgeting process [18].

Also, there might be static voltage or timing offsets in the receiver, duty cycle distortion in the transmitter or crosstalk from nearby channels. It should now be obvious that if the eye is closed solely by deterministic effects, the link will definitely not work. Though, usually various equalization techniques are applied on the transmitting and receiving side in modern serial links to counter ISI. Calibration methods can further be used to minimize static offsets. These techniques are discussed in chapter 3.1.

Once the eye was opened up through the means of equalization, random jitter becomes the limiting factor in terms of BER. Random jitter is caused by thermal or device noise. For the underlying physical process in general a Gaussian distribution of the jitter amplitude is assumed. The well known PDF of the normal or Gaussian distribution (figure 2.11) is given below:

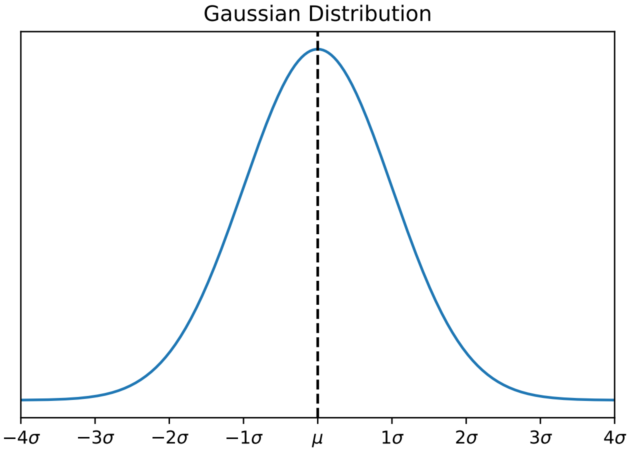


Figure 2.11: Gaussian distribution used to model random jitter

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.1}$$

For the Gaussian distribution μ is the mean, σ the standard deviation and σ^2 the variance. The integral over a Gaussian bell is 1, but the tails spread out to infinity. This means that the random jitter amplitude is unbounded, the eye will be closed eventually and a bit error will theoretically happen if the link runs long enough (maybe for an infinite amount of time). Of course, like stated earlier it is sufficient to guarantee e.g. that for PCIe the transmitted bit error ratio is better than 10^{-12}.

The integration of a PDF yields the cumulative distribution function (CDF) of a random variable. A CDF $f(x)$ gives the probability that a certain random variable will take a value less than or equal to x. For the special case of a normal distribution having a mean $\mu = 0$ and a variance $\sigma^2 = 1/2$ the CDF of the Gaussian distribution is called the error function $erf(x)$.

If a noise source can be approximated by a Gaussian distribution, it is fully characterized by its standard deviation. The complementary error function can then be used to convert the standard deviation of the noise into a peak-to-peak value at a given BER [19]. By this, a first order estimation of a link timing margin can be calculated in a quick way like shown below:

For a 10Gbps link, let the UI be 100ps ($T_{ui} = 100ps$). Let's also assume the following jitter contributions

- 30ps peak-to-peak of data-dependent jitter through ISI ($DD_{j,pp} = 30ps$)
- 5ps peak-to-peak of sinusoidal jitter through power supply noise ($S_{j,pp} = 5ps$)
- 2ps random jitter on the transmitter side from device noise in the driver and clock generation ($R_{j,tx,\sigma} = 2ps$)
- 3ps random jitter on the receiving side from device noise in the sampler and clock generation ($R_{j,rx,\sigma} = 3ps$)

From statistics textbooks we know, that two Gaussian distributions with σ_a and σ_b can be added to a new Gaussian distribution with variance σ_c by calculating the root sum square (RSS) of the contributing variances: $\sigma_c = \sqrt{\sigma_a^2 + \sigma_b^2}$. The Tx and Rx random jitter values can therefore be combined to a single R_j. With the complementary error function, we can calculate that the variance has to be multiplied by approximately 14 to yield the peak-to-peak value at a BER of 10^{-12} (as derived in [19]).

Therefore the timing margin T_{margin} at 10^{-12} is

$$\begin{aligned} T_{margin} &= T_{ui} - DD_{j,pp} - S_{j,pp} - 14 \cdot \sqrt{R_{j,rx,\sigma}^2 + R_{j,tx,\sigma}^2} \\ &= 100ps - 30ps - 5ps - 14 \cdot 3,6ps \\ &= 14,6ps \end{aligned} \tag{2.2}$$

Because T_{margin} is still positive, the link will work with the desired BER. Of course this is a very rough estimation, but it illustrates the interaction of deterministic and random sources. For a more accurate answer the eye diagram is much better suited, also because it gives the margin in both timing and voltage domain. The PDFs of different noise sources are added together by convolution (as illustrated in figure 2.12), yielding a more accurate picture than just peak-to-peak values.

Eventually by convolving all sources together an eye diagram containing both deterministic and random effects is constructed (see figure 2.13). While the eye diagram now basically displays the probability density function, with the BER coded as colors, a bathtub curve can be constructed from it through integration of the PDFs (figure 2.14) from both sides of the eye. The bathtub curve has the BER

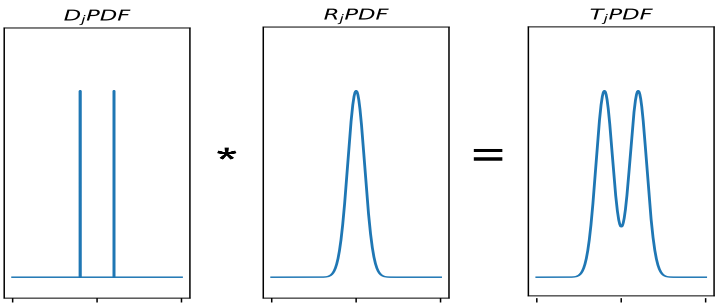


Figure 2.12: Convolution of deterministic (D_j) and random (R_j) jitter PDFs to derive a total jitter PDF

on the Y-axis and the timing margin on the X-axis. Because the deterministic parts get integrated first, the timing margin drops rapidly, whereas the tails of the bathtub are entirely dictated by the amount of random jitter. The bathtub curve can be used to easily get the timing (or voltage) margin for a given BER.

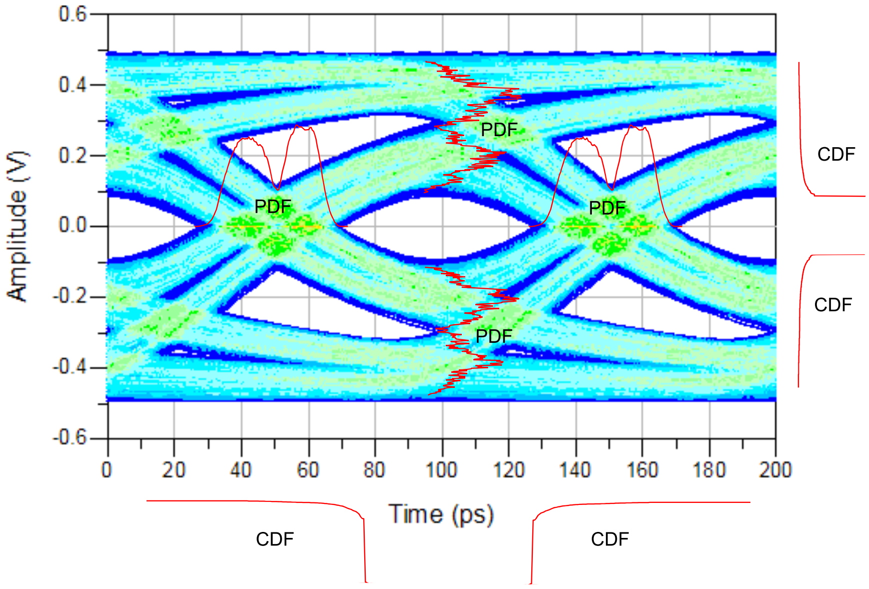


Figure 2.13: Eye diagram with random and deterministic jitter added. PDFs and CDFs for the transitions which are used to derive link budgets

There are statistical simulators like Seasim [20] available, which can calculate an eye diagram from a set of input jitter and channel parameters.

Though, for the SerDes developed in the course of this thesis an improved approach is used. It leverages real number modeling, integrates well with the overall design methodology and makes it possible to visualize and verify the impact of architectural decisions on bit error ratio.

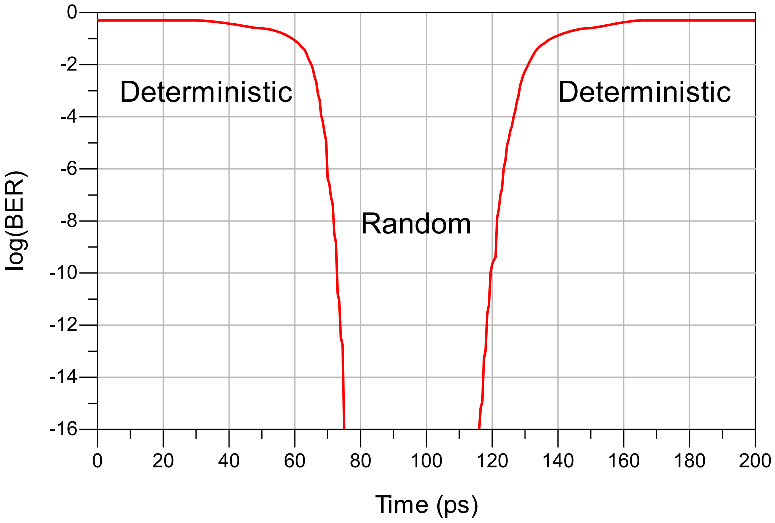


Figure 2.14: Bathtub curve for the horizontal centerline of an eye diagram.

In [18] a framework was developed, which allows S-parameter models of a transmission channel to be used in an event based simulator in order to be able to carry out the budgeting simulations. With the help of real number modeled transmitter, channel and receiver simulation speeds are high enough to use the same equalization adaptation and calibration mechanisms that are later used in the manufactured device. With SPICE simulations this would not be possible in reasonable time frames. Because the whole system is accurately modeled down to the leaf cell level like stated earlier, the real number model exhibits the same behavior and performance that are expected from the transistor implementation. This makes it possible to take additional effects like deterministic jitter induced by the clock data recovery (CDR) in the receiver into account, that are otherwise hardly covered by the statistical simulators mentioned above. Although the simulations are order of magnitudes faster, they would still take a lot of time if random effects would be simulated for extremely low BER like 10^{-12} or 10^{-15} as required by some standards. Therefore, in the budgeting simulations only deterministic effects are derived and random contributions are added through post processing the data. To verify the framework, the simulations can still be executed with random effects included, with the penalty of much longer runtime, depending on required BER.

In the following, the link budgeting procedure for the SerDes is briefly described (depicted in figure 2.15). For details on channel modeling, ISI extraction and post-processing be kindly referred to [18].

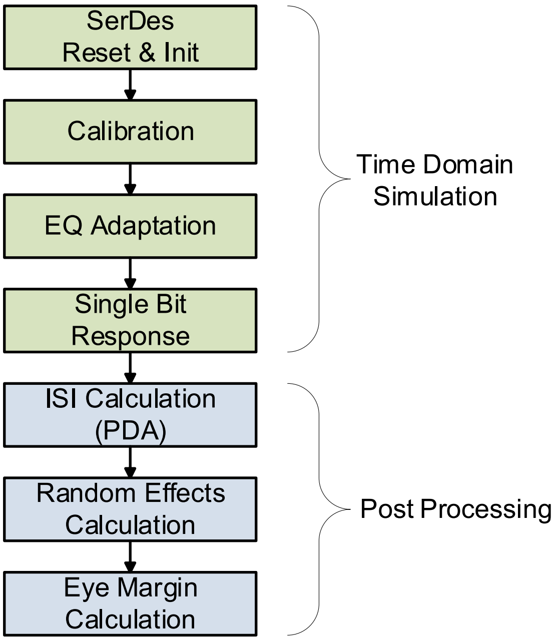


Figure 2.15: Budgeting process overview

The first step of the actual budgeting simulation is the initialization of the SerDes, which includes power-up and reset sequence of the different blocks like receiver front-end, clock dividers, CDR, etc.

Next, offset calibration takes place. This can either be executed using the actual built-in hardware control loops (see 3.5.7) or by determining the residual offsets in the specific blocks directly according to achievable granularity. Both should yield the same results and the hardware calibration can be bypassed once verified, to save simulation time. Next, random patterns are sent by the Tx side and the Rx side uses its built-in equalization adaptation to set the control vectors of all equalizers for the given implementation and channel. This, again can be verified against the theoretical optimum settings which are achievable. Simultaneously, the CDR is used to find and adjust the optimal sampling point. Once the adaptation is converged, the transmitter is used to send a single bit response. The resulting waveform is recorded and used for post-processing.

During post-processing an eye diagram, which includes ISI is constructed from the single bit response. Theoretically this could also be done through transient simulation and waveform overlay, but depending on the channel characteristics and pattern lengths quite long simulation times are needed to determine all possible ISI. A faster way is to use the single bit response and so called peak distortion analysis (PDA). The basic idea is to convolve the single bit responses of a lone zero and a lone one with shifted versions of itself to form an overlay of all possible patterns (see [21] for details). This eventually yields an eye diagram. A modified version of the PDA algorithm also takes the probability for a given pattern into account, yielding a more accurate result if true random traffic is assumed [22]. Though, depending on the line coding (like 8b/10b) used on the link, true randomness might not always be the case.

The effects of equalizers on both the Tx and Rx side are already fully taken into account, so that only residual channel ISI is used for BER calculations.

CDR induced bounded jitter - which is dependent on the residual ISI - is also accounted for in the final timing budget as it is extracted from time domain simulations.

Because the RNM models of the leaf cells exhibit parameters for impairments such as offsets or non-linearities, the impact of these effects on the system performance can be explored in a fast way and specifications for the leaf cell implementation can be derived. This can also be done the other way around, to find out if the performance achieved by a transistor implementation is appropriate in the system context. The advantage over other system modeling approaches such as Matlab/Simulink is, that the gap between model and implementation is very small and less abstract, while still providing a good simulation performance.

3 High Speed SerDes Architecture

3.1 State of the Art Serial Links

Although this work focuses mainly on the physical layer (SerDes PHY) of a serial link, a quick look on the broad topic of serial links in general should be taken. By understanding what applications for serial links exist, it will get clearer that a multitude of different constraints are imposed onto the actual SerDes.

The most prominent high level distinctions are data rates and transmission channel characteristics. Table 3.1 and figure 3.1 give a brief overview of line rates, channel lengths and channel loss for a small selection of standards.

Name	Datarates (Gbps)	Comment
PCIe	2.5, 5, 8, 16	Tens of centimeters PCB
10G Ethernet	10.3125	Several meters cable
HMC	10.0, 15.0, 30.0	Few centimeters PCB
OIF CEI[1]	10.3125, 25.x, 28.x	Few centimeters PCB

Table 3.1: Selection of serial link standards datarates and approximate transmission channel characteristics

While some standards, like the Hybrid Memory Cube (HMC) serial link are meant to only be used between chips on the same PCB spaced a few centimeters apart, others like 10G Ethernet can consist of multiple connectors and meters of cable. PCIe transmission channels normally also have at least one connector and are tens of centimeters of fairly low cost PCB long, which increases attenuation. Additionally a PCIe 3.0 link for example has to support multiple datarates, which are not even integer multiples of one another. When taking the channels defined for Common Electrical I/O (CEI) of optical transceiver modules into account, these range from ultra short range (USR) over medium range (MR) to long range (LR) with data rates up to around 28Gbps. Of course it would be possible to build an optimized custom transceiver for each of these use cases, but from the standpoint of a SoC designer it is highly desirable to have a single SerDes block which fits the needs of multiple protocols. Also, complex SoCs are implemented

[1] The Optical Internetworking Forum (OIF) defines a Common Electrical I/O (CEI), which is primarily used to electrically interface optical transceiver modules.

in semiconductor processes which focus on digital standard cell logic performance. Most traditional analog macros in contrast, do not benefit from advanced process nodes in the same way in terms of performance and scaling. For example, a 100 Ohm polysilicon resistor, which sustains a certain current will have approximately the same size in a 28nm technology as it had in a 180nm process and the size of an inductor used in an LC oscillator is also not going to scale down. Further, operating voltages keep decreasing to save power in the digital logic, but transistor threshold voltages are not decreasing in the same way. This means some proven analog circuit topologies are getting harder to realize. Noise margins are going down, which is a bigger problem for analog than digital circuits. Additionally local variations and leakage are increasing, which makes the design of structures that depend on well matched components more challenging. Power consumption is also becoming an even more important concern, as the serial I/O takes up a decent chunk of the overall power budget. The metrics of pJ/bit or mW/Gbit are therefore commonly used to asses the power efficiency of a serial link PHY. All the constraints mentioned above are driving the design process of state of the art multi-protocol SerDes designs.

An extensive study has been done in order to get an overview of current developments, implementations and trends in the design of high-speed SerDes PHYs. On this basis a short overview of state of the art serial links is given to motivate the architecture created in the course of this thesis.

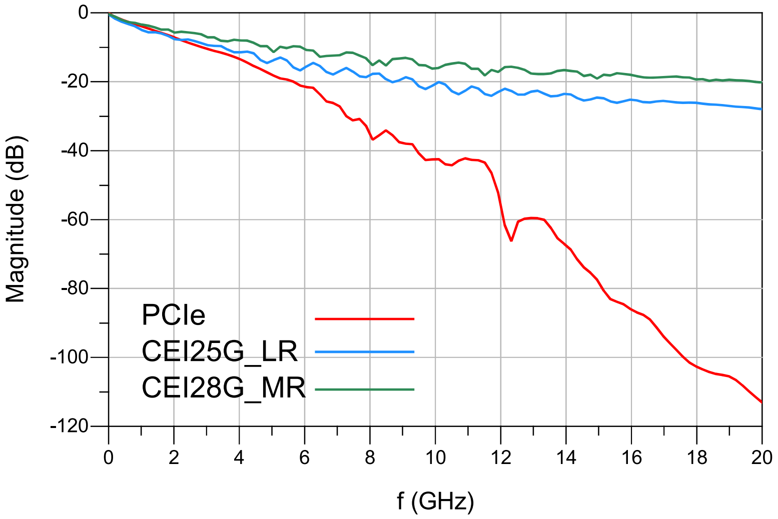


Figure 3.1: Channel loss for different transmission channel standards.

3.1.1 Line Coding

An important feature in the design space that does sometimes get overlooked is the line coding or modulation. The vast majority of high speed wireline SerDes nowadays used in the context of computing or high speed networking are built for non-return-to-zero (NRZ) coding. The advantages of this very simple scheme are that it does not impose additional complexity to modulate or demodulate the data, which would add latency or require additional power. This is only possible because of fairly good channel characteristics and exclusive use of the transmission medium. Wireline transmissions, which use higher order modulation schemes, such as quadrature amplitude modulation (QAM), that are for example used in digital cable TV networks, are out of scope of this thesis. Though, since data rates are increasing, it is getting harder to achieve further bandwidth increase with NRZ coding over the presently used transmission channels. This currently leads to the emergence of another modulation technique, namely 4 level pulse amplitude modulation (PAM4).

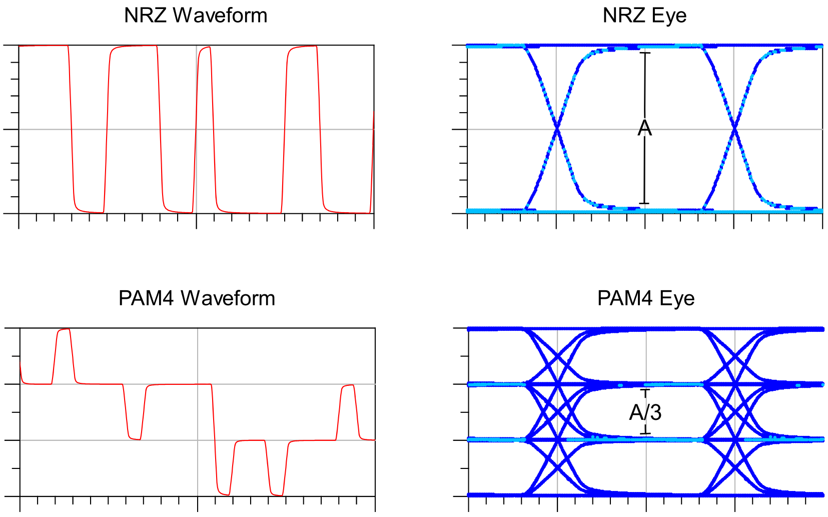


Figure 3.2: NRZ and PAM4 transient signal as well as resulting eye diagrams.

Compared to NRZ (which is equivalent to PAM2), PAM4 uses 4 levels compared to 2 levels of plain NRZ (see figure 3.2). This doubles the achievable data rate but cuts the signal to noise ratio (SNR) by 1/3 compared to NRZ. Bandwidth is therefore traded with SNR. This can be favorable, depending on the channel characteristics and data rate.

The additional complexity imposed on Tx and Rx are the need for multilevel signaling on the transmitting side and multilevel sampling on the receiving side.

Nevertheless a lot of research work is currently being done in the area of PAM4 wireline transceivers.

3.1.2 Synchronization

Another factor, that defines the complexity, mainly of the receiver, is the relationship between Tx and Rx clocking. Earlier source synchronous links like HyperTransport used to forward the Tx clock to the Rx via a dedicated clock lane so that it could be used in the Rx block to sample the received data synchronously. This obviously required additional space on die, package and board to transmit the high-speed clock. Moreover it induced very tight skew constraints on the data lanes if no additional phase shifting was used in the receiver. With shrinking UIs this practice is now only rarely used.

Instead of using an additional signal for clock transmission, the clock is nowadays embedded into the data stream. This means a coding is applied which guarantees a specific minimum amount of data transitions within a defined number of bits, and a clock data recovery (CDR) system is used on the receiving side to extract the optimum sampling phase. Depending on whether the reference clock of Tx and are Rx synchronous (like PCIe) or completely asynchronous (like Ethernet) this CDR may have to be able to compensate for a continuous frequency drift instead of only a static phase offset. Depending on the magnitude of the allowable frequency difference this requires additional effort from the CDR.

3.1.3 Line Drivers

At the transmitter side current mode logic (CML) drivers were dominant for a long time. This has now shifted towards voltage mode stub series terminated logic (SSTL) drivers. In figure 3.3 the two different circuit topologies are shown from a high level perspective.

Whereas the SSTL driver has the termination resistance in series, the current mode driver has its termination resistors in parallel. This leads to an inherent lower power consumption of 1/4 for an SSTL driver as a first order estimation, if the same swing is going to be achieved [23]. Additionally, the power consumption of the SSTL driver is really proportional to the number of data transitions, compared to static power burned by the CML driver, regardless of the datarate. The downside is, that SSTL is usually implemented as pseudo differential, having two rather independent single ended drivers which are just sending data with

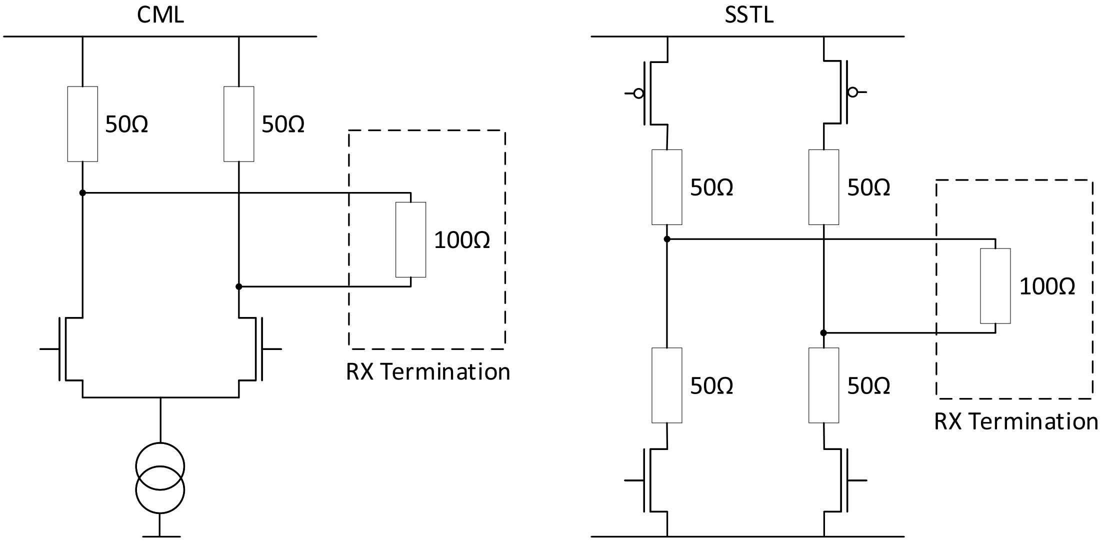


Figure 3.3: CML driver (left) and SSTL driver (right) topologies

opposite polarity. This makes them more susceptible to skew and power supply noise. CML on the other hand is truly differential and has a better power supply rejection because of the tail current source. Still, with the focus on efficiency and power consumption, most recent implementations are using SSTL drivers.

3.1.4 Equalization

To overcome the impact of channel loss and impedance discontinuities at very high data rates, equalization has also become a very important part of serial link design. Whereas legacy standards, like PCIe Gen1 relied only on a very simplistic static equalization scheme on the transmitter side, modern links usually split the equalization between Rx and Tx. Different types of equalizers, which are often complemented by adaptation algorithms to find the best settings for a given data rate and channel, are used. This makes equalization expensive in terms of die area and power consumption. In the following three basic concepts, which are implemented in most modern SerDes PHYs are discussed.

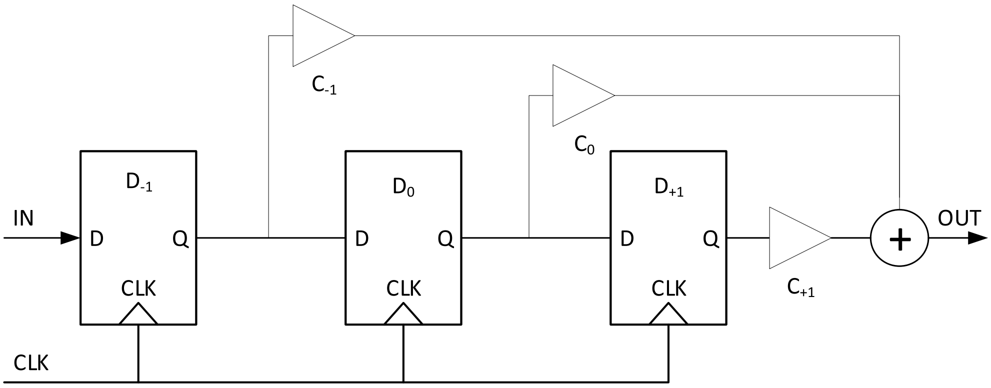


Figure 3.4: Feed forward equalizer

The transmitter side usually implements a so called feed forward equalizer (FFE), which basically is a finite impulse response (FIR) filter. In figure 3.4 a 3 tap FEE is depicted. The output signal is constructed by building a weighted sum from delayed versions of the input signal:

$$D_{out} = c_{-1} * D_{-1} + c_0 * D_0 + c_1 * D_1 \tag{3.1}$$

Where D_{-1} is called pre cursor, D_0 main cursor and D_1 post cursor. c_x are the respective pre, main, post cursor coefficients. Usually the sum over all absolute coefficient values $|c_x|$ is equal to 1. This means that the maximum signal swing is kept constant regardless of the chosen coefficients. A sample waveform is given in figure 3.5.

With an FFE the signal gets pre-distorted on the transmitting side. Because the overall signal swing is kept constant, the emphasis of high frequency content is effectively achieved by attenuating the DC content. The resulting eye at the receiver in more equalized, but at the cost of smaller signal swings.

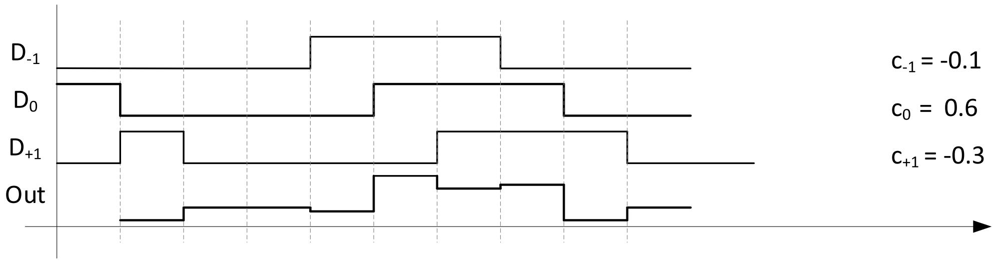


Figure 3.5: Feed forward equalizer resulting waveform with pre and post emphasis

The finite signal swing also limits the magnitude of distortion that can be applied to counter the channel characteristics. The optimum coefficients are specific for a given channel, and need to be derived either through simulation or in system adaptation. Most SerDes PHYs implement a 3 tap FFE, which is also a requirement for PCIe Gen3. It is interesting to note, that a 4 tap FFE transmit equalizer can also be used to encode data into PAM4 [24].

On the receiving side in state of the art designs usually two different types of equalizers, which complement each other, are implemented. At the input there normally is a first stage implementing a continuous time linear equalizer (CTLE), followed by a digital time discrete decision feedback equalizer (DFE).

The CTLE is a fully analog amplifier with a transfer function that counteracts the channel loss as seen in figure 3.6. This is usually partly achieved through actual high frequency gain and partly again by DC attenuation.

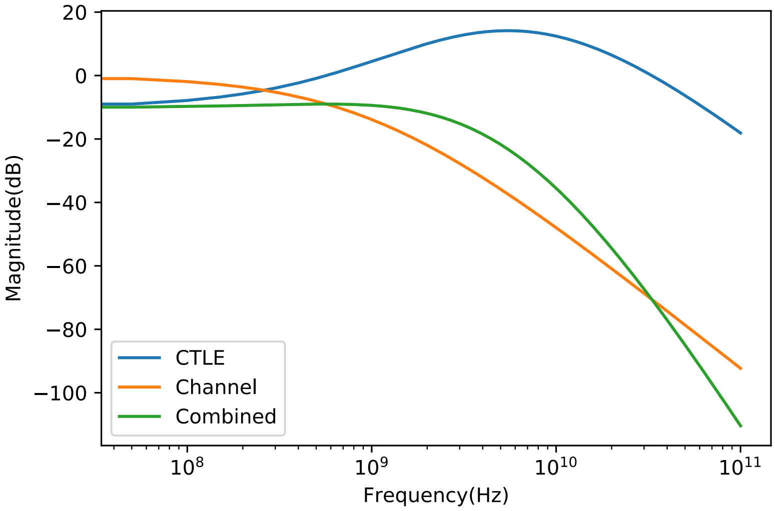


Figure 3.6: Low pass transmission channel, CTLE and equalized combined channel+CTLE transfer function

The basic circuit topology of a CTLE is given in figure 3.7. Basically it is a CML amplifier with a degenerated tail current source. The single tail current source is split into two sources which are again shorted with a combination of resistor and capacitor. The resistor defines the DC gain of the amplifier, whereas at high frequencies the capacitor shorts the tail current sources branches together leading to increased high frequency gain again [25]. The zero and 1st pole of the transfer function are dependent on both R and C. Usually both components are adjustable, so they can be used to adapt the CTLE frequency response.

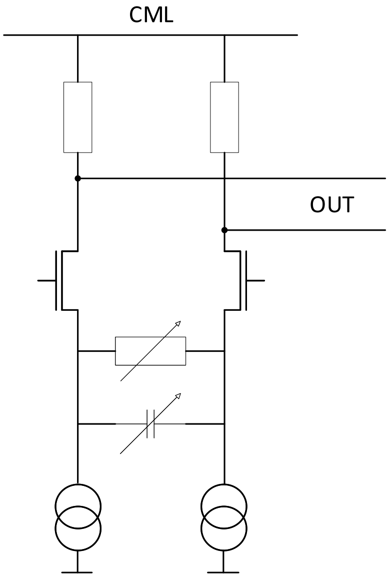


Figure 3.7: Basic CTLE circuit topology

The downside of a CTLE is, that every input is obviously treated the same way. This implicates that all noise and crosstalk is also amplified along with the desired signal. Therefore, in modern serial links, where a large amount of channel loss is present a CTLE is often used together with a decision feedback equalizer, which does not suffer from noise amplification in the same way.

Figure 3.8 depicts the working principle of a DFE, which is a digital, time discrete filter. At the input, a comparator is used to decide if the sampled signal is a logic zero or one. In front of the comparator a weighted summation of previously received bits and the input signal is performed. This basically implements an infinite impulse response (IIR) filter. If for example several consecutive logical ones are received, the summation leads to an offset at the comparator input, which then favors a weak zero. Because of the feedback path it is theoretically possible for a DFE to lead to error propagation once wrongly sampled bits entered the feedback path. In practice the bit error rates are normally low enough to get the DFE back to error free operation, if this happened.

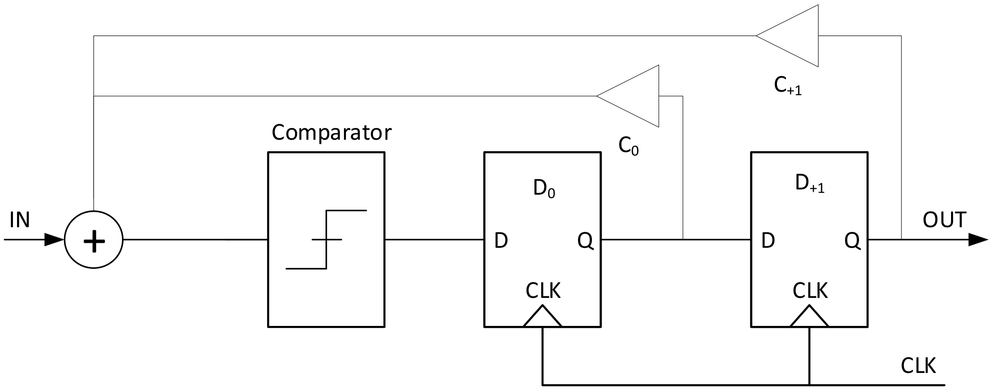


Figure 3.8: 2-Tap DFE working principle

The number of previously received bits, which are summed at the input determines the number of taps of the DFE. PCIe Gen3 demands at least a 1-tap DFE, to cancel the effect of the first post cursor tap. The first tap usually has the largest impact but in practice often 3 or 5 taps are implemented. There are also designs, especially for long backplane scenarios were DFEs with tens of taps are used [26]. The necessary number of taps of course depends on the channel characteristics and should be kept at a minimum because the actual implementations are often very power hungry. For the actual summation in front of the comparator different mechanisms are used. State of the art implementations have been analyzed in [27].

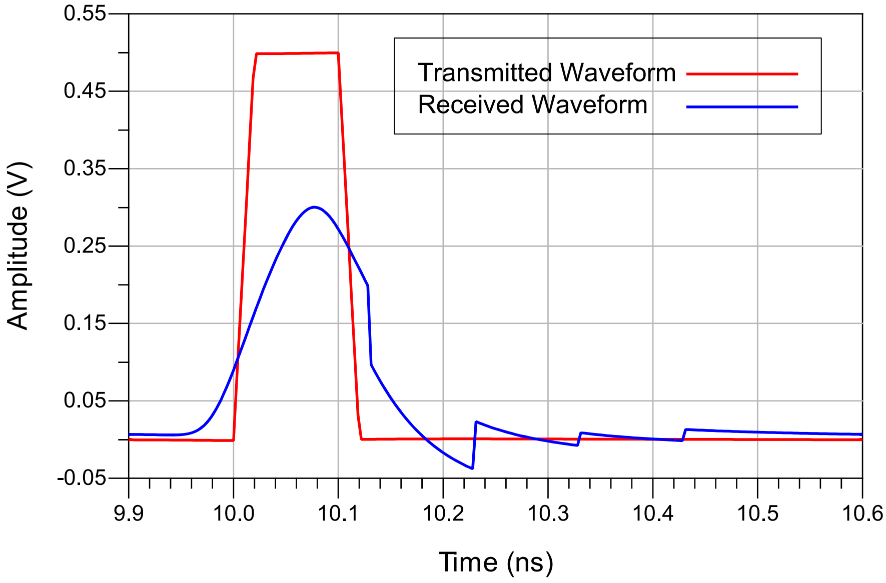


Figure 3.9: DFE corrected SBR

Figure 3.9 shows a single bit response (SBR) after DFE correction. Due to the time discrete nature, discontinuities are visible at the time the offset is applied. From this, it is also visible that a DFE equalizes the signal only at the data sampling points. If e.g. an oversampling clock data recovery circuit is used, which takes additional samples at UI boundaries, a DFE in the datapath does not remove ISI from these additional samples.

In contrast to a CTLE, a DFE does not amplify noise or crosstalk. Another helpful property is the ability to counter reflections, caused by impedance discontinuities at connectors or vias. Depending on channel length a reflection might arrive at the receiver several bit times after the bit which caused it. This can not be equalized using a CTLE, but if a DFE has a tap which corresponds to the necessary delay, the reflection can be completely canceled as it is a deterministic effect. For this, recent DFE implementations added so called sliding taps. These can be adjusted according to their delay in respect to the main cursor and positioned at times the reflections are received.

3.2 Architecture Overview

In this chapter the SerDes architecture developed in the course of this thesis is described. First a general overview is given and the main ideas driving the overall architecture are stated. Afterwards the different main components such as clock distribution and generation, the receiver and transmitter are described in more detail.

A serial link is normally build from multiple lanes. To support different protocols with different link widths, the individual lanes were built to be independent from each other and can be cascaded by abutment. The common clock is generated by a central phase locked loop (PLL), which is then distributed from lane to lane. Every lane contains a buffer to distribute the common clock to the next lane. The number of lanes, which share a common PLL is only limited by the jitter which is accumulated by the clock distribution buffers. The more lanes share a common PLL, the better the power efficiency of the clock generation. Every lane contains a receiver and a transmitter, which share some common clocking infrastructure.

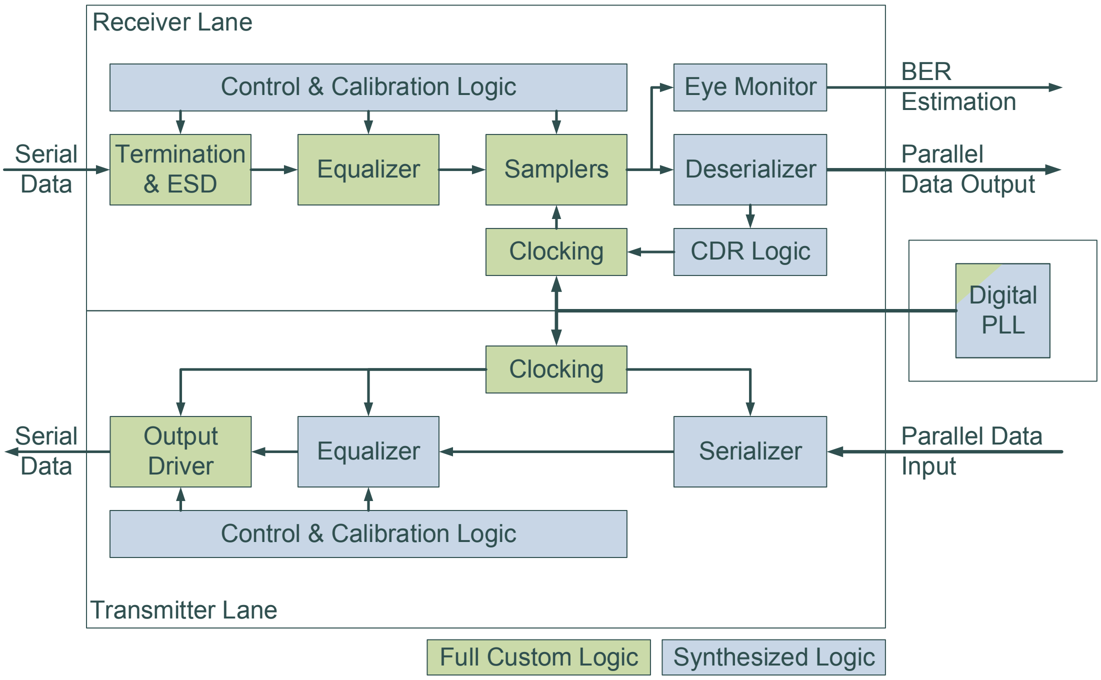


Figure 3.10: Coarse SerDes architecture overview

Figure 3.10 gives a coarse overview of the general architecture of a single SerDes lane. The main goal was to create a flexible, robust and portable design. This immediately lead to the conclusion to use as much synthesizable, digital logic as possible. Digital logic is much more immune against process variation than analog circuits and synthesizable logic written in an HDL is much more portable than full custom logic. This is especially true in advanced nodes such as 28nm and below where variations are increasing and supply voltages are decreasing. Full-custom and semi-custom parts are distinguished in figure 3.10.

Frontend parts in the high-speed datapath such as the output driver in the transmitter and the equalizers and samplers in the receiver are implemented as full-custom blocks. Same is true for most of the clocking resources, which are partly shared between Rx and Tx. All control and calibration logic, which includes most parts of the actual serializer/deserializer as well as the CDR are implemented in semi-custom fashion.

The main focus of the work carried out in the course of this thesis was on clock data recovery and clock synthesis using a digital PLL, which is subsequently presented.

3.3 All Digital PLL

Both transmitter and receiver require a high speed clock for their operation. This clock has to be generated and distributed to the individual lanes. Usually high speed clock generation is done using a phase locked loop (PLL). A PLL uses an external reference oscillator to generate a synchronous output clock, which is a multiple of the reference clock. For a high-speed SerDes very high clock frequencies of up to 14GHz and more are necessary. In order to achieve the desired bit error ratios at data rates of 28Gbps, a generated clock random jitter of below 1ps is necessary. This is very challenging to achieve. Moreover, the clock generation needs to be flexible enough to support different reference clocks and output frequencies, in order to support different SerDes rates. For increased power efficiency it is also desirable to share one clock generation unit between as many lanes as possible. As the main clocking component, the PLL which was developed in the course of this thesis is now described in detail.

3.3.1 Introduction

In figure 3.11 the general architecture of a PLL is depicted. In a traditional analog PLL, there is a phase detector (PD), that generates a voltage, which is proportional to the phase difference of the reference clock f_{ref} and the feedback clock f_{fb}.

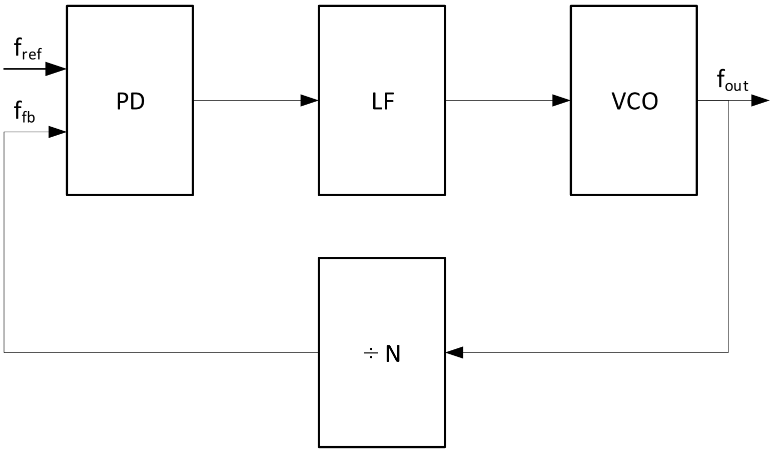


Figure 3.11: PLL working principle

The output signal of the phase detector is filtered by the loop filter (LF). As a PLL is basically a control loop which minimizes the phase error of reference and feedback clock, the loop filter is responsible for the control loop characteristics such as stability and bandwidth. The loop filter design is usually a major concern when building a PLL.

The filtered output of the PD is fed to an oscillator, which is traditionally a voltage controlled oscillator (VCO). The control voltage determines the instantaneous frequency of the VCO. To close the loop, the VCO output f_{out} is used to clock the feedback divider which has a division factor of N. The divided clock is fed back into the phase detector.

By this, the output frequency f_{out} is set as

$$f_{out} = f_{ref} * N \tag{3.2}$$

Besides the fact that one is usually interested in f_{out}, the PLL control loop does not work in the frequency but in the phase domain. This stems from the fact, that the VCO over time integrates an offset in frequency into a phase error. The analog PLL characteristics can be analyzed by using traditional control theory transfer functions.

The open loop transfer function of the PLL depicted in figure 3.11 can be written as

$$G(s) = K_{pd} * Z_{lf}(s) * \frac{K_{vco}}{s} \tag{3.3}$$

where K_{pd} is the phase detector gain coefficient, $Z_{lf}(s)$ the loop filter transfer function and K_{vco} the VCO gain coefficient [28].

The transfer function of the feedback path H only consists of the division factor N and can be written as

$$H = \frac{1}{N} \tag{3.4}$$

Finally, the transfer function for the feedback system, consisting of $G(s)$ and $H(s)$ can be put together according to control theory textbooks, to form the closed loop transfer function

$$CL(s) = \frac{G(s)}{1 + G(s) * H} \tag{3.5}$$

By analysis of poles and zeros of this transfer function the characteristics of the PLL can now be assessed. This can be complex, especially if higher order loop filters are used [28]. However, this is not the scope of this thesis, especially because the PLL, which was built is not an analog but a so called all digital PLL (ADPLL). Still the introduction above will be very useful for the general understanding of the following.

As stated earlier, in a traditional PLL the control information is represented as an analog voltage. As all subcomponents are susceptible to different sources of voltage noise, the performance can easily be degraded. This is especially a problem in advanced semiconductor process nodes where supply voltages are being scaled down, decreasing the voltage headroom and signal to noise ratio. Additionally in these types of processes analog properties such as linearity or device matching are getting worse and analog loop filters, built from resistors and capacitors, are not scaling down with the technology. This lead to the emergence of digital PLL architectures, which tend to benefit from process scaling. The control information is no longer stored as an analog voltage but encoded in digital codes stored in registers.

Despite the benefits that digital PLLs bring, there are also new problems which arise, such as quantization errors and non linearities in the control loop. These can degrade the performance and complicate the analysis compared to traditional PLLs.

In figure 3.12 the general architecture of a second order ADPLL is depicted and will be described in the following. As it can be viewed as a time discrete control system, it is possible to employ the z-transformation as a time discrete version of the Laplace transformation for system analysis [29].

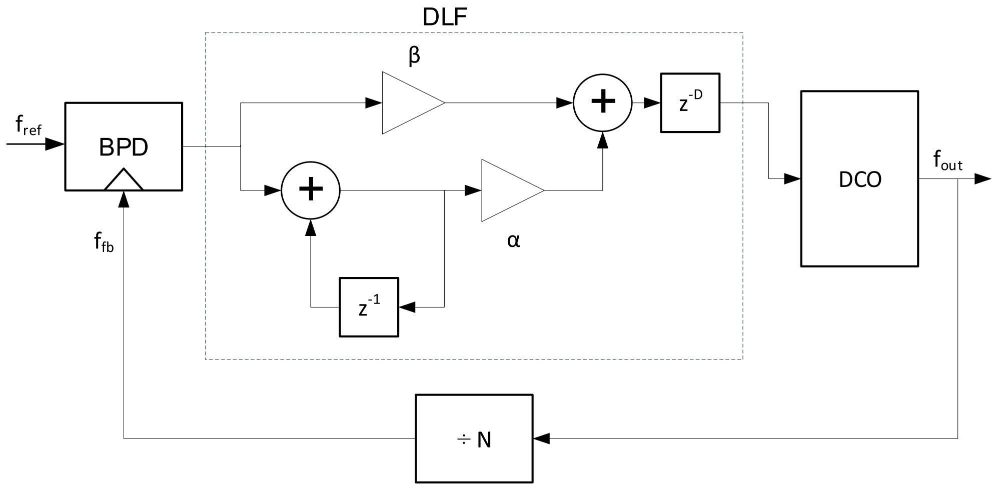


Figure 3.12: ADPLL working principle

Because the phase error information is processed by a digital loop filter, it has to be digitized first. This is done by means of a so called time to digital converter (TDC). The simplest implementation of a TDC is a binary or bang-bang phase detector (BPD), which is basically a 1-bit TDC. The output bit therefore only indicates if the reference phase is leading or lagging the feedback clock phase. As this is a highly non-linear behavior, it complicates analysis using traditional control theory for linear time invariant (LTI) systems. Though, there are several different approaches to linearize the BPD and alleviate this issue [30], [31].

The loopfilter itself is pure digital logic. The digital implementation makes it much easier to implement mechanisms like configurable loopfilter coefficients compared to an analog implementation, while not suffering from problems like leakage or voltage noise effects. The filter depicted in figure 3.12 includes a proportional path with coefficient β and a integral path with coefficient α. The number of delay cycles D necessary for processing the phase information obtained by the TDC can be modeled using a delay element.

An alternate approach to describing the ADPLL behavior using z-domain analysis are time domain approaches developed in [32] and [33]. In contrast to the traditional LTI-systems approach the time domain technique can be used to predict effects of random noise in the system on the behavior of the finite resolution control loop. The time domain approach can be used to accurately predict lower bounds for achievable jitter due to quantization and select proper loop filter coefficients to guarantee stability. In the following, equations predicting the ADPLL behavior which were derived in [33] are used to show how the systems parameters depend on each other. This is then used to explain design decisions that need to be considered when building an ADPLL.

Key parameters that have to be considered are:

- N: PLL multiplication factor
- D: PLL loop latency
- $\sigma_{T_{dco}}$: DCO random jitter
- K_T: DCO period gain
- β: PLL loop filter proportional path coefficient
- α: PLL loop filter integral path coefficient

The notation adheres to the one used in [33] to keep consistency. The period of the PLL output clock is defined as $T_\nu = T_{\nu 0} + K_T \cdot \omega$, where $T_{\nu 0}$ is the free-running period and ω is the tuning word. This tuning word is the output of the PLL loop filter.

It can be derived that for stability, the proportional path coefficient β needs to be sufficiently higher than the integral path coefficient α. The actual ratio is dependent on the loop delay D. The higher the loop delay is (the more cycles are used to compute the new tuning word from the phase detector output), the higher the β/α ratio needs to be. For $D = 1$ a ratio of 16 is sufficient for a phase margin of 60° [33].

According to [33], the PLL output jitter can be approximated as

$$\sigma_{t_v} \approx \frac{1+D}{\sqrt{3}} \cdot N\beta K_T + \sqrt{\frac{\pi}{8}} \cdot \frac{\sigma^2_{T_{dco}}}{\beta K_T} \tag{3.6}$$

The first term in equation 3.6 refers to the quantization noise of the loop (limit cycle), the second term describes the jitter of the DCO. To achieve minimum jitter, an optimum β has to be chosen. If β is too high, quantization of the loop filter is not scrambled enough by the random jitter of the DCO. If β is too small, the DCO random jitter is not filtered by the PLL properly. Both scenarios result in increased jitter. Because the first term is proportional to βK_T and the second one is proportional to $1/\beta K_T$ a minimum can be found.

The optimum β can be approximated by

$$\beta_{opt} \approx 2 \cdot \frac{1}{\sqrt{((1+D)N}} \cdot \sigma_{T_{dco}} \cdot \frac{1}{K_T} \tag{3.7}$$

as derived in [33].

From these equations the following can be deduced:

Equation 3.6 leads to the conclusion, that in order to minimize the PLL output jitter, all contributors need to be minimized. For best performance, the loop latency D needs to be minimal, which means the number of pipeline stages in the digital loop filter needs to be kept small. Also, the division factor N should be as small as possible. This is often something, that can not be directly influenced because both PLL output frequency as well as reference clock frequency are often given by a certain standard, which tries to minimize cost and therefore aims to use a low reference clock frequency.

Further, if intrinsic DCO jitter is not dominant, K_T needs to be minimized, which means the LSB resolution in the DCO needs to be minimized. This is mostly technology dependent but scales nicely with the semiconductor manufacturing technology feature sizes. Yet, there are additional constraints such as tuning range and parasitic capacitance, which prevent the usage of large minimum sized capacitor arrays in the DCO.

At last, β needs to be minimized. For stability reasons, and in order not to introduce some further quantization at the DCO input, α is chosen first, so that an LSB change in the loop filter results in a minimum period change of K_T in the DCO. Afterwards β can be chosen after equation 3.7, according to the DCO random jitter and period gain. Though, a lower bound for β exists because of the loop stability considerations.

This illustrates again that there are two cases. In case one, the quantization effects from the loop dominate. The DCO random jitter is so small that a lower β would result in lower jitter, but it can not be selected because it would break stability. In case two, DCO random jitter dominates. Therefore β needs to be increased according to equation 3.7, in order to still get the best result out of the given DCO random jitter.

Therefore, though not directly visible in equation 3.6, it is also very important to minimize the DCO random jitter, in order to be able to chose a small β, which is then only limited by DCO tuning word quantization. DCO quantization and random jitter both have to be maintained in a reasonable relation in order not to spoil the overall performance mutually.

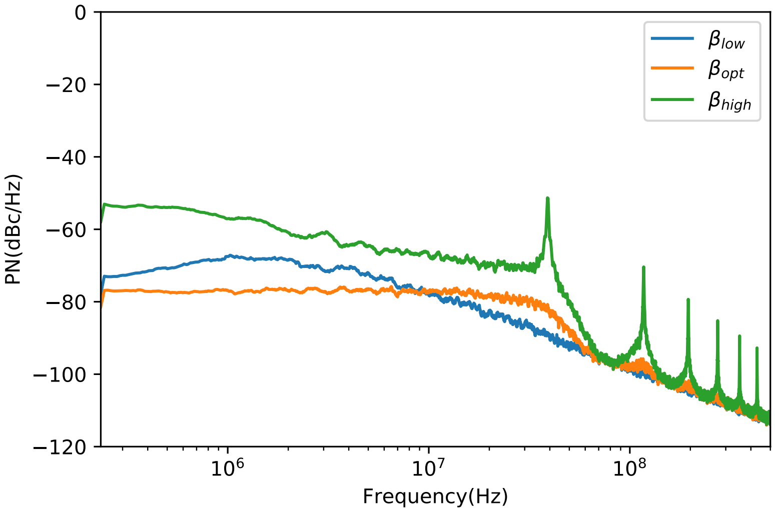


Figure 3.13: Phase noise plots for random noise regime (β too small), optimum β and limit-cycle regime (β too high)

To reassess, if the predictions made by time domain analysis fit simulations, a simple reference model was implemented in SystemVerilog. The results appeared to be in good agreement with the equations above. To illustrate the different cases of random jitter and quantization (limit cycle) regime, figure 3.13 shows phase noise plots for three different values of β. While keeping all other parameters constant, only β was varied to produce the different scenarios. The reference model was used later on to validate the actual implementation in terms of performance.

3.3.2 Implementation

The actual implementation of the designed PLL is depicted in figure 3.14. Semi custom parts are highlighted to emphasize that most complexity is pushed into synthesizable HDL code. This improves portability to other processes.

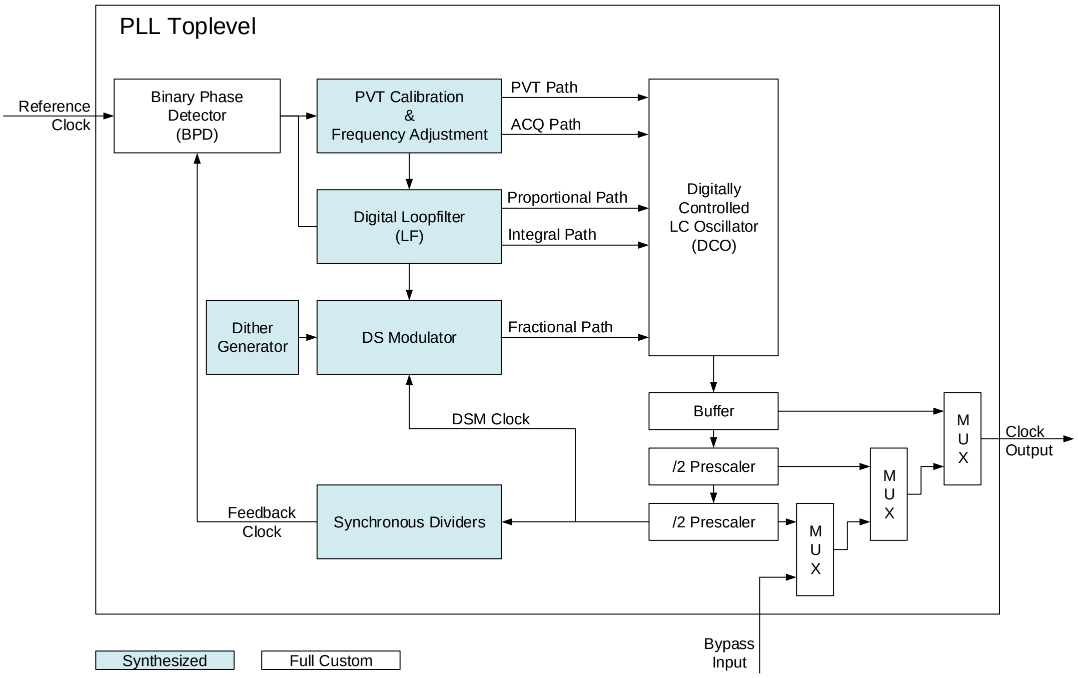


Figure 3.14: Block diagram ADPLL implementation

There are some differences compared to the general ADPLL architecture from figure 3.12.

A PVT calibration block is added, which is necessary to tune the oscillator free running frequency to the nominal range, where the loop filter with its proportional and integral path can be used.

Moreover, the addition of the loop filter proportional path and integral path accumulator is done intrinsically in the DCO itself by separately controlled parallel capacitor arrays. This saves an additional adder in the loop filter and thereby helps reducing the overall loop latency [34], which in turn helps reducing the output jitter.

The PLL output clock can be selected from the actual DCO frequency, half and quarter rate. A bypass input can also be selected to use an external clock for test and debug purposes.

A very important addition is the delta sigma (DS) modulator, which can be used to increase the effective resolution at the DCO as described in [35].

The different modules of the PLL are now described in more detail.

Digitally Controlled Oscillator (DCO)

To meet low jitter specifications required by high speed serial links, an LC tank has to be used in the DCO [36]. Because the general design of a DCO and LC oscillator has already been covered numerous times, e.g. in [37],[38] and [39], the architecture will only be discussed briefly. The general idea of the DCO is depicted in figure 3.15.

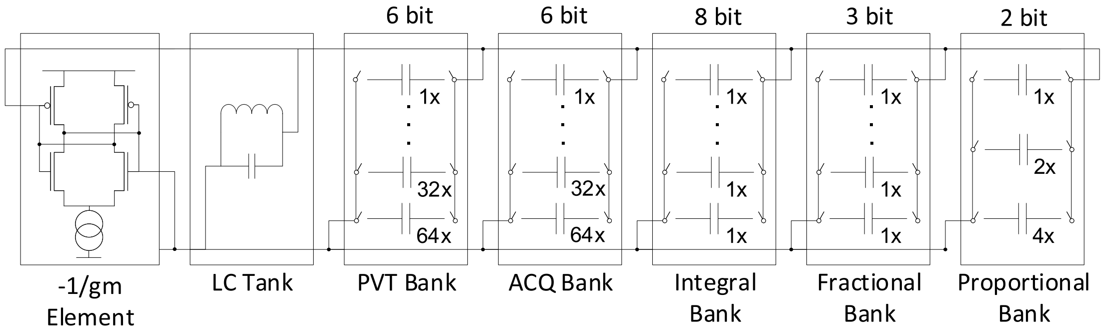


Figure 3.15: General oscillator architecture showing all elements, which are connected in parallel in the DCO

In parallel to the actual LC tank oscillator, there are several additional capacitor arrays, which can be controlled separately. The different tuning banks have different granularities to fulfill different purposes.

There are binary weighted PVT and acquisition (ACQ) banks, which are used to set the free running frequency of the DCO to the required range.

The two banks have 6 bits each, for easier implementation compared to a single 12 bit capacitor bank. Within 6 bits linearity and layout matching is much better than it would be over a complete 12 bit array.

For reasonable tuning, the ACQ bank, which has finer steps compared to the PVT bank, has to overlap the PVT LSB over all process corners.

The 8 bit integral path bank is constructed from equally sized capacitors, which are organized in a 8x8 matrix structure. Each capacitor element has its own decoder logic, which allows the 8x8 matrix to be controlled with 8 bit thermometer row and column codes (see figure 3.16). Through this only 2*N instead of 2^N wires are necessary to control the capacitor array, which greatly reduces the routing effort and parasitic capacitances. Since the integral path bank control code is constantly changed during PLL operation, equal weighting was chosen over binary weighting for better linearity and absence of glitches. To further prevent glitches when selecting/deselecting a row, the control inputs to the DCO are retimed again by sampling latches. For best achievable DCO LSB resolution (lowest K_T), the integral path capacitors are implemented using the smallest switchable capacitors available in the given technology.

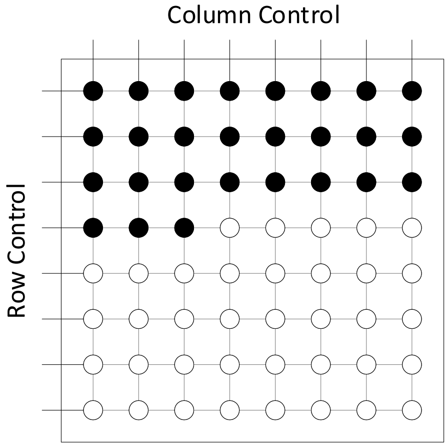


Figure 3.16: DCO integral path capacity matrix

The integral path frequency range spans at least double the range of an ACQ bank LSB to be able to compensate for temperature drifts in system, without having to switch ACQ bits during normal operation.

The fractional bank, which is controlled by the DS modulator is made from the same capacitors as the integral path, for proper matching.

Because all banks are placed in parallel in the LC tank, the DCO intrinsically adds all separate tuning PVT, ACQ, integral, fractional and proportional tuning words into the single DCO tuning word ω, rendering additional accumulators redundant. As otherwise a digital adder would be necessary, this helps to save latency in the loop filter.

With the tank inductance fixed, and the oscillation frequency F_{DCO} determined as

$$F_{DCO} = \frac{1}{\sqrt{LC}} \tag{3.8}$$

the relative amount of capacity is chosen to be larger compared to the inductance, in order to achieve a wide tuning range. This is beneficial in order to be able to support different line rates for multiple protocols using the overall SerDes architecture. Though, this also degrades noise performance and increases non-linearity of the oscillator tuning word. An alternative would be to have a switched inductor to increase tuning range with variable L and improved performance [40]. Though, the actual implementation of a switched inductor with predictable inductance is quite hard for practical reasons and was therefore not implemented in this DCO. Because such an inductor is not part of the usual device libraries, which are provided by semiconductor manufacturers it has to be implemented and characterized by the designer himself. This requires the use of an electromagnetic field solver and in-depth knowledge of the materials and stack up used in the specific manufacturing process, which is not always available.

Feedback Divider

To determine, which feedback dividers are necessary, it has to be analyzed first, which standards or line rates the SerDes should support using which reference frequencies. The following table gives an overview of the different frequencies taken into account.

Standard Name	Line Rate	Reference Clock	Division Factor
PCIe 1.0	2.5 Gbps	100 MHz	25
PCIe 2.0	5 Gbps	100 MHz	50
PCIe 3.0	8 Gbps	100 MHz	80
HMC 10G	10 Gbps	125 MHz	80
HMC 12.5G	12.5 Gbps	125 MHz	100
10G Ethernet	10.3125 Gbps	161.1328125 MHz	64

Table 3.2: Line rates and reference clock frequencies considered for PLL architecture

The division factors for the different rates can be constructed using the feedback divider and output multiplexers. The output multiplexers can be used to select a full, half or quarter rate clock. Therefore the lower PCIe rates can be constructed with a feedback divider of 100 and an output selection of 2 or 4 respectively. This leaves the required feedback divider values to 64, 80 and 100. They are factorized to find the common factors:

$$\begin{aligned} 64 &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \\ 80 &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot 5 \\ 100 &= 2 \cdot 2 \cdot 5 \cdot 5 \end{aligned}$$

As 2^2 is the least common multiple, this factor can be implemented in the full custom part of the PLL, as it brings down the working frequencies low enough so that a semi-custom implementation can be used for the subsequent configurable dividers. This improves the flexibility and lowers the implementation effort that would otherwise be required to implement a factor other than 2 as a full custom design.

A common approach to realize configurable feedback dividers in a PLL are so called dual modulus dividers [41]. Luckily, the residual factors of 16, 20 and 25 can be perfectly covered by two cascaded 4/5 modulus dividers. In figure 3.17 a block diagram of the 4/5 modulus divider is given.

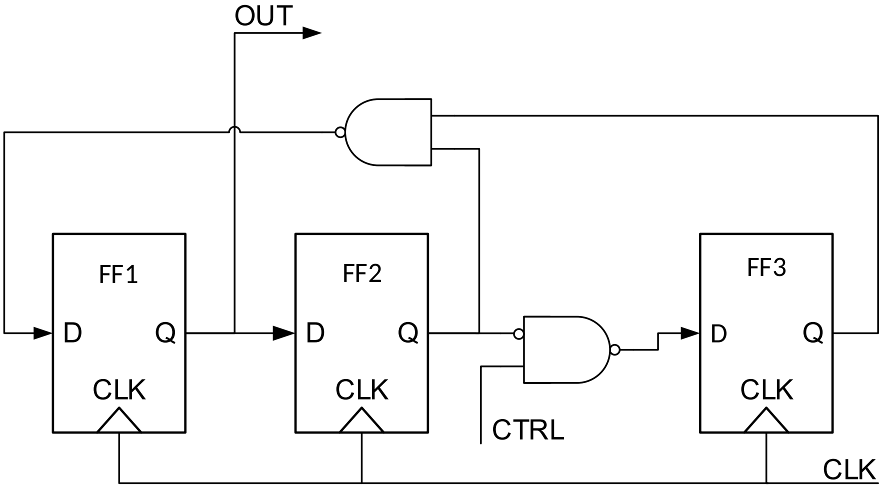


Figure 3.17: 4/5 modulus divider block diagram

If *CTRL* is set to 0, the output of *FF3* is kept at a constant 1 and the circuit works like a normal synchronous divide by 4 module. If *CTRL* is set to 1, FF3 injects the delayed version of FF2 into the feedback path, masking the high signal of FF2 for one additional clock cycle. Hereby a divide by 5 is realized. It should be noted, that the output clock signal does have a duty cycle of 40:60 instead of 50:50 in divide by 5 mode, but this is not a problem in case of the feedback divider.

Because the feedback dividers are currently implemented as a series of cascaded dividers, their jitter is accumulated, hereby degrading the overall jitter performance. In future implementations, it should be considered to realize especially the lower speed dividers as synchronous implementations by re-timing the output stage with the fast input clock for better jitter performance.

PVT Calibration

Before the actual PLL operation can take place, the DCO free-running frequency needs to be adjusted after power-up to bring it as close to the nominal frequency as possible. The algorithm to find the optimum free-running frequency is described in the following.

To be able to tune to the desired free-running frequency, the average DCO frequency after power-up has to be determined. The binary phase detector itself can not be used for this purpose, as it does deliver only phase, not frequency information - in contrast to the popular phase frequency detector (PFD) used in analog PLLs. Therefore, Bang-Bang PLLs often have an additional frequency detector.

One way to detect a frequency offset detection is by counting edges in both the reference and feedback clock. If the number of edges detected for the feedback clock is higher than the number of reference clock edges, the DCO is running to fast and vice versa. The accuracy of this method only depends on the measurement time.

As the feedback clock period T_{fb} will have a small difference to the reference clock period T_{ref}, it will accumulate over the number of reference clock cycles N_{ref}. Therefore it takes

$$N_{ref} = \frac{T_{ref}}{|T_{ref} - T_{fb}|} \tag{3.9}$$

reference clock periods until the difference accumulated to a whole period.

If a simple counter is used for each of the two clocks, a frequency difference can only be determined once it has been accumulated to a phase difference of a complete period and one counter actually "overtakes" the other by one. Because the reference clock frequencies are usually relatively slow this can take quite long for small offsets. Also, the two counters can also only be compared after a fixed time interval, because they obviously run in different clock domains and can not be compared easily while incrementing.

Therefore, the implementation in this work uses a modified approach for faster frequency acquisition. Instead of having a counter for each clock domain. The faster DSM clock (see figure 3.14) is used to oversample both feedback and reference clock and count the edges. This has the benefit that the resolution is improved by at least 16 (which is the smallest low speed divider value), which in turn results in a measurement time reduction. Also, as the two counters now reside in the same clock domain, a frequency difference can be detected as soon as one counter overtakes the other because no synchronization is necessary. This makes fixed measurement intervals redundant and further speeds up the PVT tuning.

The main finite state machine (FSM), which controls the PVT tuning procedure is depicted in figure 3.18.

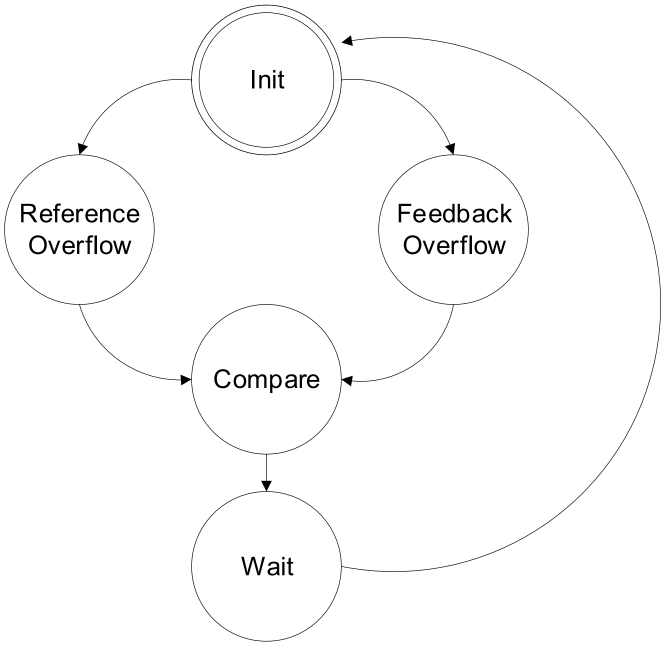


Figure 3.18: PVT tuning main FSM

To guarantee a stable operation, the FSM is clocked by the PLL reference clock. In the *init* step, all counters are reset and afterwards enabled. As soon as one counter overtakes the other, the results are compared and the capacitor array is adjusted accordingly in the *compare* state. After each tuning operation the feedback dividers are reset after a guard interval in the *wait* state, because it can not be ruled out that there are glitches on the feedback clock when switching the large tuning capacitors into the LC tank.

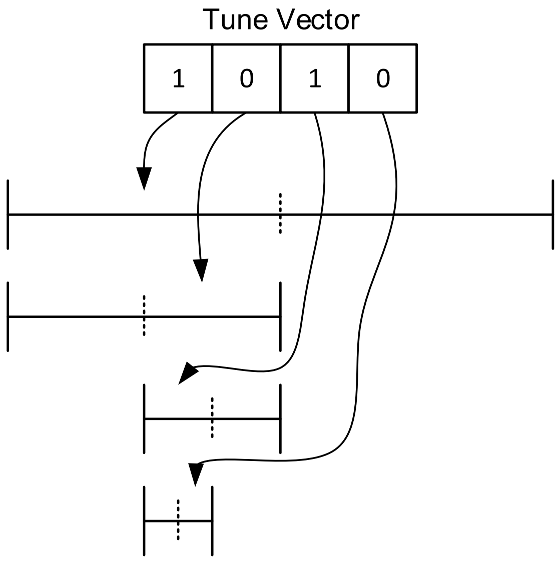


Figure 3.19: Binary search on PVT register

To speed up the tuning procedure, a binary search is implemented on top of the capacitor bank control registers. Instead of linear incrementing or decrementing the control vector, which would take $2^N - 1$ iterations for an N bit wide control register, the number of iterations is reduced to only N. The binary search is implemented in a very efficient way by walking linearly over the control vector, starting with the MSB. According to the determined frequency information the current bit is either flipped or kept "as-is" in order to get to the upper or lower part of the interval, like depicted in figure 3.19. This is done for each bit from MSB to LSB.

In figure 3.20 a complete PVT and ACQ bank tuning is depicted. It can be observed, that the time for a control vector step depends how far the current average frequency is from the nominal frequency. The bigger the offset, the faster it can be obtained.

After the PVT and ACQ tuning is finished, the loop filter is activated and normal PLL operation can begin.

Digital Loop Filter

For the loop filter, it is very important to minimize the number of pipeline stages and thus, latency D. This is especially true for the proportional path, whereas the integral path latency can be higher without performance impact [32].

Therefore the proportional path does not have a register at all, but uses combina-

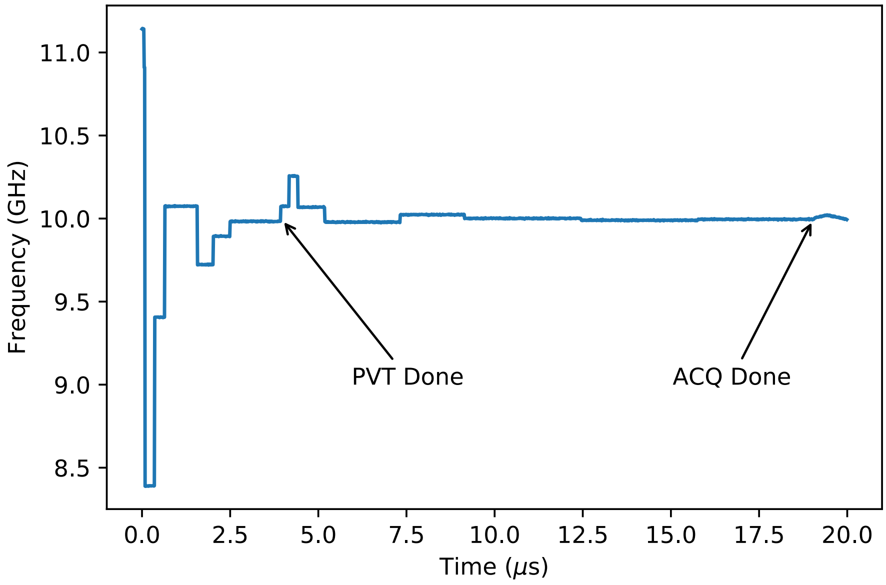


Figure 3.20: DCO free running frequency during PVT and ACQ tuning

tional logic in order to scale the proportional paths coefficient value. The latency is hereby kept at a small number of gate delays. The integral path accumulator is actually wider than the DCOs integral path. This means, that depending on the configured coefficients the accumulators LSBs get truncated, because a change in the accumulator does not change the DCO control vector. This truncation is actually a quantization, which does introduce additional noise in the PLL [33]. Still, it can be beneficial to use loop filter settings that apply this kind of quantization, if this allows a reduction of the proportional path coefficient β. This is the case as long as the jitter reduction by decreased β is higher, than the additional quantization noise caused by truncation in the integral path. To prevent this quantization to occur in this way and still use a higher resolution integral path accumulator, delta sigma modulation (DSM) can be used. This is described in the next sub section.

When the loop filter gets activated after PVT tuning, there might still be a small frequency offset, which has to be corrected by the integral path. If the loop filter is configured for very small proportional gain, it can still take quite long for the PLL to actually lock, or in extreme cases it might not lock at all. Therefore a variable gain mechanism has been implemented.

To activate it, the phase detector output is monitored. If it is constant for a configurable amount of feedback clock cycles, the proportional path gain is continuously increased in order to prevent a reference clock cycle to slip at the phase detector. As soon as a change at the phase detector output is observed, the gain is reduced to nominal. To be able to apply enough proportional path gain, the ACQ bank capacitance bits can be controlled by the loop filter, when variable gain is necessary. With this mechanism the last bit of frequency offset is tuned out by the loopfilter integral path.

Delta Sigma Modulator

Delta sigma modulators are used extensively in modern analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) to increase resolution and signal-to-noise ratio. Though they are also popular in the context of all-digital PLLs. As mentioned earlier, delta sigma modulation (DSM) can be used to increase the actual resolution of the DCO control word and hereby improve the frequency resolution and DCO period gain. This is achieved by a combination of oversampling and noise shaping, to provide smaller capacitance steps in the DCO through time-averaging. Figure 3.21 depicts the working principle of a 1-bit DSM.

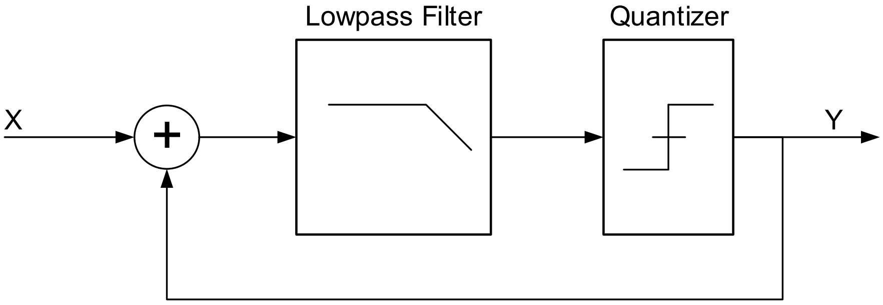


Figure 3.21: DSM working principle

At the input X, first the actual input data, which is m bits wide is added to the 1-bit feedback path value. The summation result is afterwards low pass filtered. The low pass filter, in turn is followed by a quantizer, which is the 1-bit output Y of the DSM and the feedback path value.

The general idea is, that similar to normal oversampling the delta sigma modulator is running at a higher frequency than the input data is changing. During one period of input data change, the output switches much faster between 0 and 1 in such a manner that the time averaged ratio of the two values matches the multi bit input value. The faster the output value switches compared to the input (the higher the oversampling ratio), the finer the resolution obtained because more output samples are used to create the average in the same time interval. Therefore, to reconstruct the input value an additional low pass filter is needed at the DSM output to create the average of the output values. In context of the PLL, the high frequency noise of the DSM gets filtered by the DCO and control loop.

The second aspect of the DSM is its noise shaping nature, which can be deduced when analyzing its transfer characteristics. Figure 3.22 depicts the linearized z-domain model of the 1-bit delta sigma modulator.

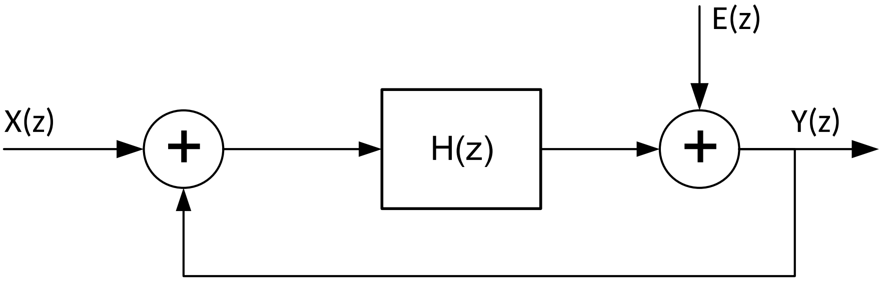


Figure 3.22: DSM z-domain model

The low pass filter has been exchanged with a generalized block of transfer function $H(z)$ and the quantizer is modeled a random noise source $E(z)$. This is a valid assumption as long as the quantization error is uncorrelated to the input value. The overall transfer function can be derived as a combination of the so called noise transfer function (NTF) and the signal transfer function (STF). Setting X(z) to 0 in figure 3.22 results in the NTF defined as

$$NTF(z) = \frac{Y(z)}{E(z)} = \frac{1}{1+H(z)}. \tag{3.10}$$

The same is done for $E(z) = 0$ to obtain the signal transfer function:

$$STF(z) = \frac{Y(z)}{X(z)} = \frac{H(z)}{1+H(z)} \tag{3.11}$$

Through superposition the overall transfer function of the DSM is formed as

$$Y(z) = X(z) \cdot STF(z) + E(z) \cdot NTF(z) \tag{3.12}$$

For a simple digital accumulator, the filter transfer function $H(z)$ would be

$$H(z) = \frac{1}{z-1} \tag{3.13}$$

which yields

$$NTF(z) = \frac{Y(z)}{E(z)} = \frac{1}{1+H(z)} = \frac{z-1}{z}$$

$$STF(z) = \frac{Y(z)}{X(z)} = \frac{H(z)}{1+H(z)} = \frac{1}{z} = z^{-1} \tag{3.14}$$

Substituting these into equation 3.12 gives the overall transfer function

$$Y(z) = X(z) \cdot z^{-1} + E(z) \cdot \frac{z-1}{z} \tag{3.15}$$

From this, it is now visible, that the quantizer noise $E(z)$ is shaped by the NTF which has a transfer function of a first order high pass, while the input signal $X(z)$ is only delayed by one cycle. The noise gets shaped and the original signal is passed to the output unchanged. Quantizer noise is not reduced but "pushed" to higher frequencies, where it can later be removed when low pass filtering is applied for reconstruction of the input value. Higher order filters can be used in the DSM to further increase the noise filtering performance. The number of integrators defines the order of the delta sigma modulator. The DSM in the former example therefore is of order 1.

The noise shaping aspect results in an increase in dynamic range compared to simple oversampling, which lowers the required oversampling ratio and therefore the frequency the DSM needs to be operated at for a desired resolution increase. Simple oversampling also increases the dynamic range, but only spreads noise over a wider frequency range. The increase in bits can be derived, according to [42] as

$$DR = 0.5 \cdot \log_2 OSR \tag{3.16}$$

where OSR is the oversampling ratio. With delta sigma modulation the gain in dynamic range also depends on the modulator order. In [43] the dynamic range of a N-order DSM is derived as

$$DR = \log_2 \frac{3 \cdot (2N+1)}{2\pi^{2N}} \cdot OSR^{2N+1} \tag{3.17}$$

With a DSM clocked at 16x oversampling ratio, which is the minimum in the PLL developed in this thesis, this yields a DR of around 10 bit for 1st order DSM and around 21 bit for a 3rd order DSM, which is sufficient for the fractional part bits of the loop filter.

The order of the DSM can either be increased by a higher order filter or by cascading multiple 1-bit modulators. While the former can introduce stability issues, cascading multiple modulators increases the delay. Because the DSM is used in the integral path of the loop filter, which is less sensitive to additional delay, a 3rd order multi stage noise shaping (MASH) DSM architecture was implemented in this work.

A key assumption for the DSM to work as intended and the analysis above to be valid, is that the input value is a uniform distribution in frequency. In the case of an ADC, this is generally satisfied, but not in case of the DCO fractional control word. Therefore additional dithering needs to be applied in order to prevent periods in the DSM output [44]. To accomplish this, a configurable PRBS generator is implemented and added as LSB into the fractional control word and DSM input. The configurable dithering polynomial provides additional pseudo random noise, which satisfies the assumptions.

3.4 Lane Clocking

In order to increase power efficiency, a central PLL is used to generate a link clock, which is then distributed to the individual lanes. In contrast to each lane having a dedicated PLL, the more lanes that share the same PLL, the higher the power efficiency. The actual number of lanes which can be fed by a single PLL depends on the clock distribution. When more lanes are added, the clock distribution has to cover a longer distance, which necessitates more clock buffers. Each clock buffer adds a specific amount of jitter, which step by step degrades the lane performance when getting farther away from the link PLL. It should be noted, that in contrast to clock distribution used in digital logic designs, no clock tree is built because balancing the clock delay to the individual lanes is not necessary. This approach is depicted in figure 3.23.

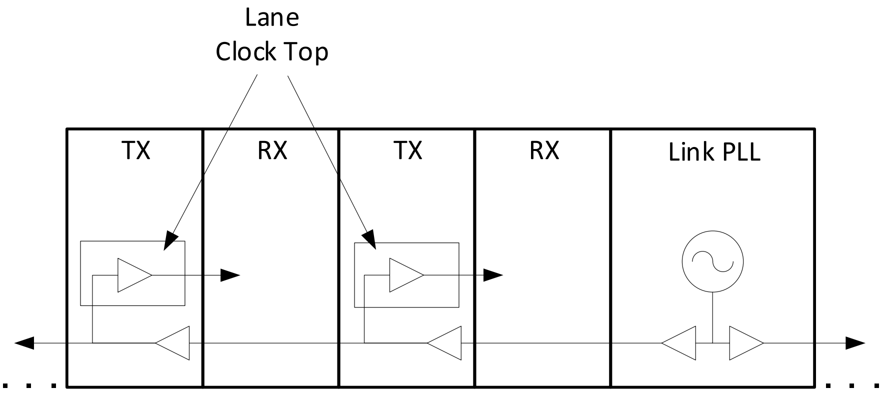


Figure 3.23: Lane clock distribution

Another potential drawback which should be taken into account is the obvious fact that when sharing a common PLL, all lanes of a link need to run at the same line rate. This is not an issue for the protocols or applications targeted with this architecture, but might be for others.

The lane clock distribution is implemented on-chip using the top most thick metal layers, which have the lowest resistance. Further, the geometry should be laid out to form an on-chip transmission line for lowest jitter [45].

Because of the Tx and Rx architecture, which is described in full detail in the following sections, four clock phases are required at each lane. Therefore, the question arises, whether to generate these four quadrature phases in the central PLL and distribute them or to generate them locally in each lane. This was analyzed in [46], taking into account the design space depicted in figure 3.24.

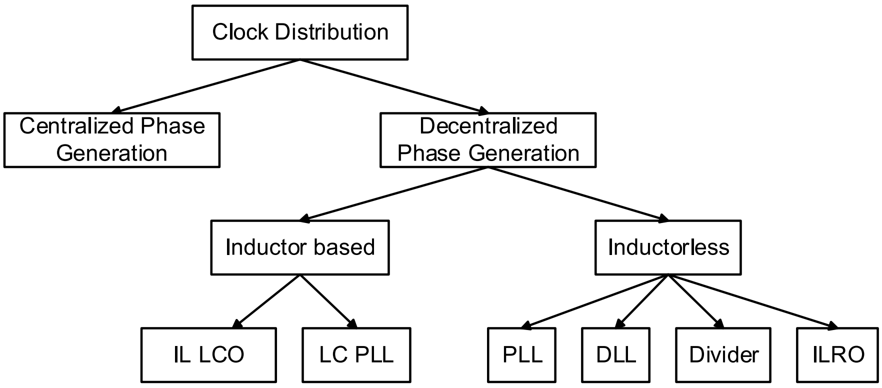


Figure 3.24: Different possibilities for quadrature clock distribution, after [46]

Centralized phase generation has the downside, that the amount of space and buffers necessary for distribution doubles. Also, there will be a skew introduced between the phases by the independent buffering, which in turn will degrade the lane performance. Therefore, it is desirable to generate the quadrature phases locally at each lane. The easiest way to generate quadrature clocks from a differential clock is by the use of a clock divider. Obviously, then either the central clock has to have twice the clock frequency or the achievable line rate is halved. If a divider should not be used because of the reason stated above, PLLs or delay locked loops (DLLs) could be used, but this would run contrary to the idea of a central link PLL resulting in usual power and area requirements. Further, there is the choice between inductor based oscillators (LCO) or ring oscillators. LCOs tend to have lower jitter, but also have higher area requirements and lower tuning range. After the analysis carried out in [46], the use of an injection locked ring oscillator (ILRO) eventually proved to be suited best.

An ILRO is a ring oscillator with additional ports that permit the injection of another clock signal into the feedback loop. When the free-running frequency of the ring oscillator is adjusted to be close to the injected clock, the oscillator will align itself to the injection frequency. As a multi-stage ring oscillator can be used to generate multi-phase clocks, this is a method to create local quadrature spaced clocks. When the ILRO is in injection locked mode, the noise characteristic is greatly improved if a low jitter clock is used as injection signal, as illustrated in figure 3.25.

Because the tuning range of the ILRO is limited, for lower speed line rates a divider based quadrature (I/Q) clock generation is still useful. Therefore, each lane contains a clock generation/selection module, which is depicted in figure 3.26. Depending on the line rate the clock is locally generated by division, using

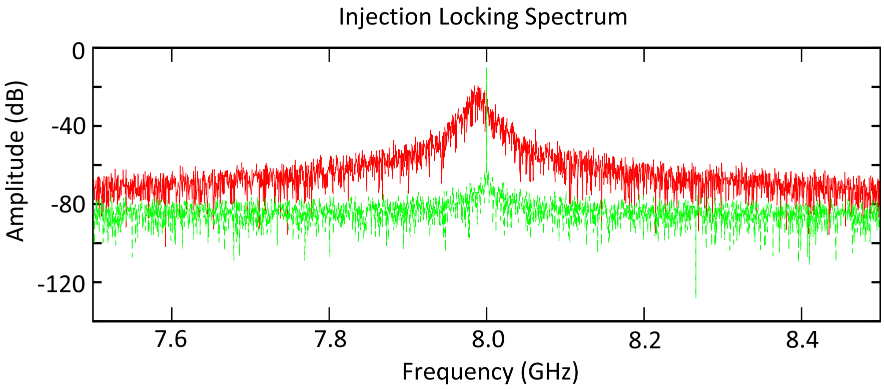


Figure 3.25: Oscillator spectrum for free running (red) and injection locked (green) ILRO at 8 GHz [46]

a quadrature divider or by using the ILRO, which generates the four phases without division. For lower line rates, the link clocks are prescaled by the PLL output dividers. Table 3.3 lists the combinations, which are used to achieve the necessary clock frequencies for some common line rates.

Line Rate	Link PLL Clock	Link PLL Output Divider	I/Q Gen
2.5 Gbps	10 GHz	4	Divider
5 Gbps	10 GHz	2	Divider
8 Gbps	8 GHz	1	Divider
10 Gbps	10 GHz	1	Divider
16 Gbps	8 GHz	1	ILRO
20 Gbps	10 GHz	1	ILRO
25 Gbps	12.5 GHz	1	ILRO

Table 3.3: Line rate and clocking mode combinations

The lane top clocking module also contains two additional multiplexers (DFT_MUX and TX_MUX), which allow the receiver recovered clock to be used to clock the transmitter. This is useful for debugging and characterization purposes. Further the ILRO can be used to clock the transmitter, while the receiver uses the link PLL clocks directly. This mode is necessary for calibration purposes and is described in section 3.5.7.

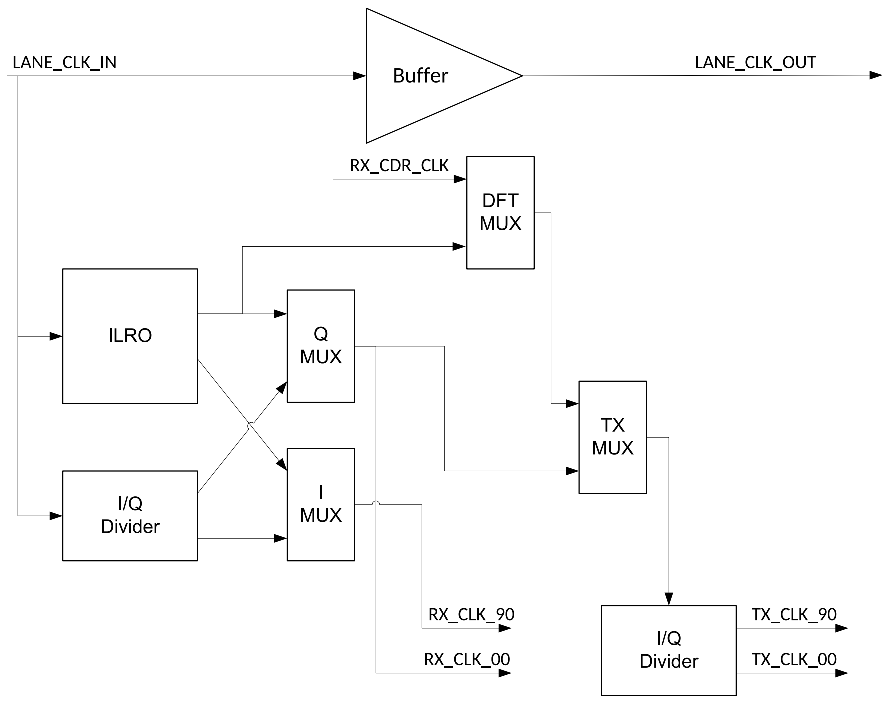


Figure 3.26: Common Rx/Tx lane clock top module

3.5 Receiver

3.5.1 Overview

The developed receiver architecture includes an analog frontend with a CTLE, a 5-tap decision feedback equalizer and digital clock data recovery. There are several digital calibration mechanisms to trim out both voltage and timing offsets in the datapath. The equalizers can be adapted automatically to a given transmission channel with an adaptation algorithm implemented in hardware.

The receiver toplevel is partitioned into a digital semi-custom (Rx Digital) and a full custom (Rx Core) part, as shown in figure 3.27.

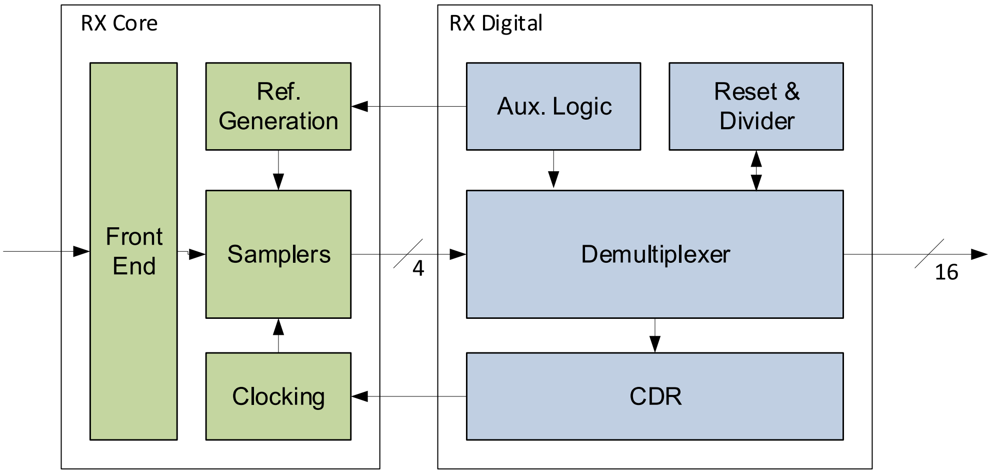


Figure 3.27: Rx overview block diagram

The full-custom part is divided into several main blocks. There is the datapath, which consists of the analog frontend, followed by all the samplers. Additionally there is a clocking module, which generates all the sampler clock phases from the lane clock received from the lane PLL, with the ability to shift phases according to the CDR control vectors. Further the Rx core contains the central bias generation for all full custom blocks as well as the reference level generation for the DFE. All control and status signals are brought to the Rx core module boundary and are processed by the digital toplevel.

As mentioned earlier the goal of the design was to move as much complexity as possible into the digital part. Therefore it includes the actual demultiplexers for the various samplers, several clock dividers and corresponding divider initialization circuits as well as the digital CDR.

The auxiliary digital logic contains the equalization adaptation logic, a concurrent eye monitor, the calibration loops as well as pattern checkers for testability.

The different blocks are now described in more detail.

3.5.2 Datapath

The datapath can be divided into the front end and the sampling stage followed by demultiplexers (see figure 3.28). The sampling stage is actually split into three different paths. There is the actual data sampling stage, the edge sampling stage used for the CDR and the eye sampling stage used for the concurrent eye monitor as well as the equalization adaptation. All stages have the same signal as input but are sampling at different phases.

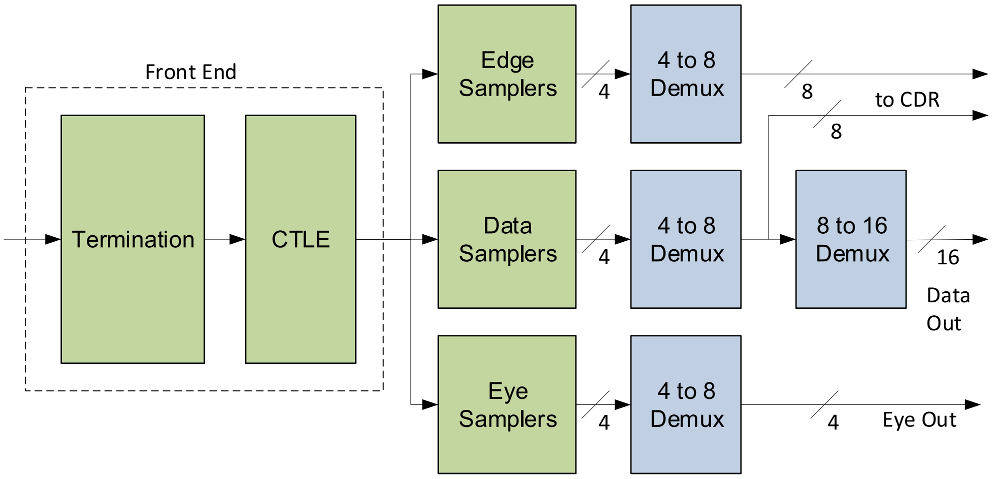


Figure 3.28: Receiver datapath overview

At the receiver input a termination is necessary to match the characteristic impedance of the transmission line and eliminate reflections, which show up as ISI and degrade the received signal quality. Figure 3.29 depicts the termination used in the developed architecture.

To prevent damage of the input transistors by electrostatic discharge (ESD) additional ESD prevention structures are needed. Unfortunately these structures add a very high input capacitance, which has a negative impact on the high frequency characteristic of the termination by degrading bandwidth and impedance matching. Therefore, inductive compensation techniques have to be applied to counter the capacitive load. This has been analyzed in depth in [47].

Further, because the manufacturing process variations of the internal termination resistors are quite high, they need to be adjustable in order to be able to tune

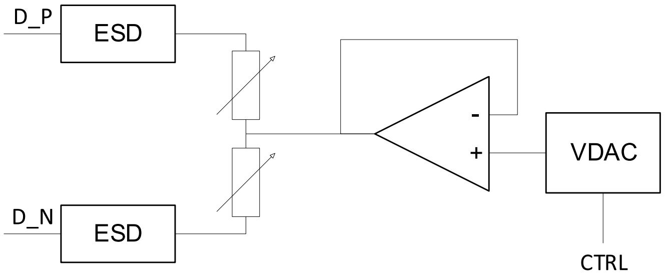


Figure 3.29: Receiver termination overview

them to the required line impedance. As the receiver is meant to be AC-coupled to a transmitter, a receiver input common mode has to be generated. This is done by an operational amplifier used as voltage follower and a DAC to generate the common mode reference voltage.

After passing the ESD structures, the differential signal is fed into the continuous time linear equalizer stage. The CTLE is built as a multi stage amplifier with source degeneration in the different stages, like described in section 3.1. This gives the flexibility to adapt to a wide range of different channels. For implementation details of the CTLE please refer to [18].

After passing the CTLE, the partly equalized signal is processed in the sampler stages.

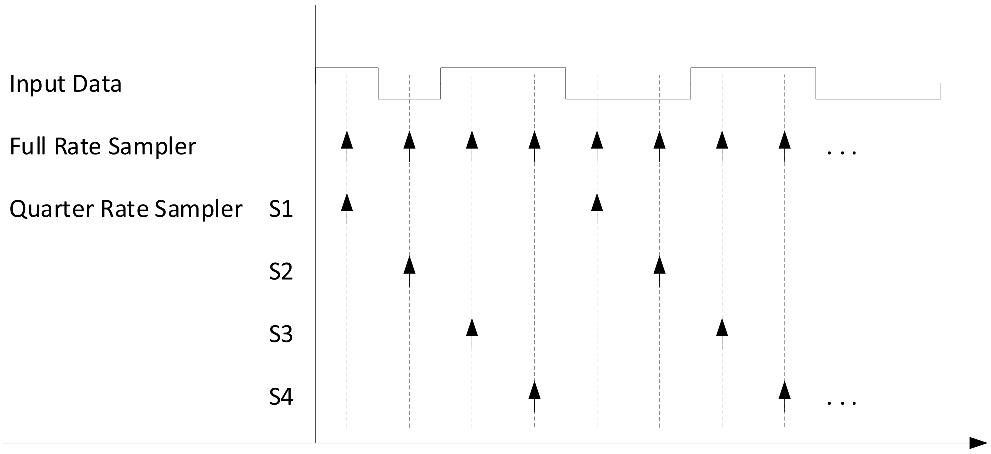


Figure 3.30: Quarterrate receiver principle

Generally the receiver is built as a quarter rate design. This means that instead of a single sampler which samples at the full data rate there are four samplers in parallel, which are interleaved using different sampling phases running at a quarter of the line rate (see figure 3.30). The main benefit is, that the highest clock speed in the design is reduced by a factor of 4. Another advantage is, that the input signal is already deserialized to 4, which makes high speed deserialisation stages from 1 to 2 and 2 to 4 obsolete, as they would be necessary in a full rate design. The downside is that the capacitive loading of the CTLE stage is higher because of the increased number of samplers. Still, quarter rate designs are more power efficient than full rate designs and are therefore generally preferred for high data rates. For better power efficiency at lower data rates, it is also possible to e.g. power down two samplers completely when only half the maximum lane rate is used.

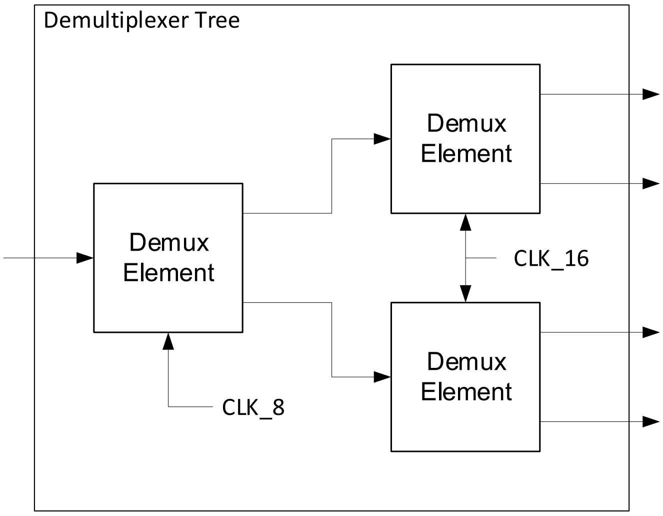


Figure 3.31: 1 to 4 demultiplexer tree after each sampler

The samplers themselves have been designed and characterized by [18].

Each sampler is followed by a two stage demultiplexer tree, as depicted in figure 3.31. Hereby the quarter rate data bit of each sampler is demultiplexed to 4 bit, now at 1/16th of the line rate. This clock is called the word clock.

Each demultiplexer tree in turn is constructed from demultiplexer (demux) elements.

Each demux itself (figure 3.32) has a data input D_{in}, which is sampled by two flip-flops (FF). One FF is sampling at the rising edge (D_{even}), the other one at the falling edge (D_{odd}). To be able to cascade the demux elements, the D_{odd} is followed by a latch, which retimes the output data transition to match the time of the even path. The timing diagram is given in figure 3.33 for better understanding.

With this scheme all samples taken by the different samplers are deserialized to the same word clock, in order to be used by subsequent logic.

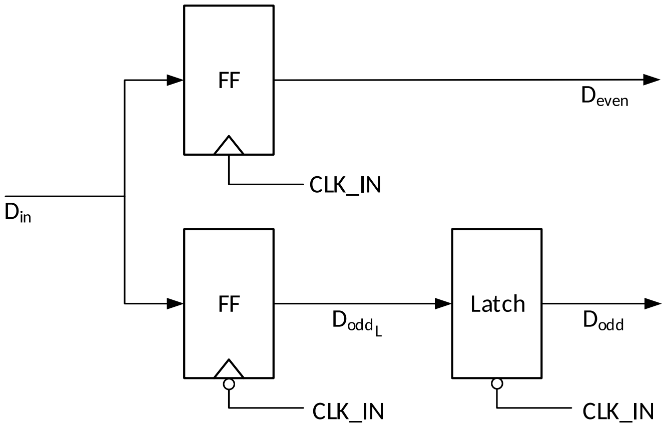


Figure 3.32: Demultiplexing element

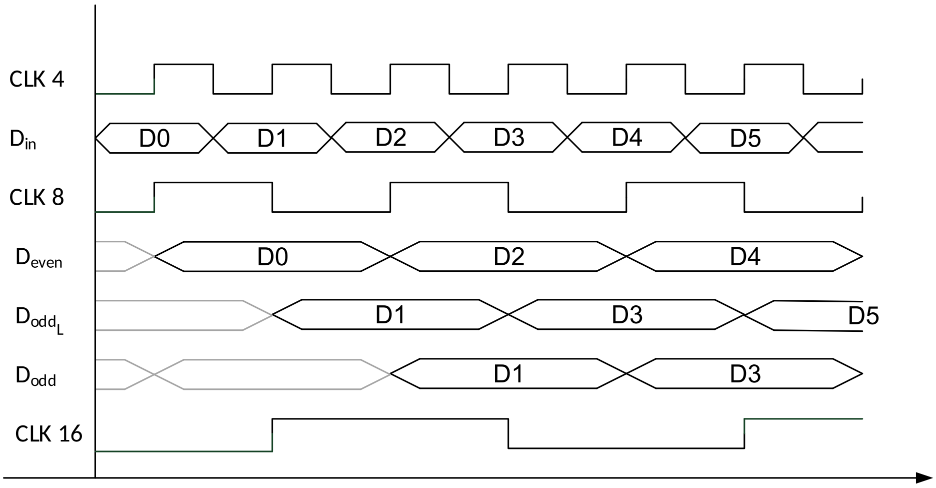


Figure 3.33: Demultiplexing element timing diagram

In contrast to the data and edge samples, the eye sample path does contain only one sampler. This has been done in order to reduce power and area requirements for the eye sampling path, which is used to analyze the incoming data levels and adapt the equalization control vectors. In contrast to the normal data path, the number of samples can be traded against increased adaptation time of the algorithms. Since measurement time is not important in this case, the area for additional samplers was saved.

An additional benefit of the quarter rate architecture is the possibility to realize a decision feedback equalizer without a lot of overhead. The implementation of the speculative 5-tap DFE which fits the overall architecture was done in [27].

The datapath is complemented by the clock generation, which is now described in more detail.

3.5.3 Clocking

The purpose of the Rx clocking module is to generate all the clock phases needed by the different samplers. The main challenge here is, that these phases need to be adjustable according to the control vectors of the digital clock data recovery logic for the optimum sampling instant. This gets especially difficult because of the different clock rates, which are necessary for a true multi-rate SerDes design. Figure 3.34 gives a high level overview of the overall Rx clocking architecture.

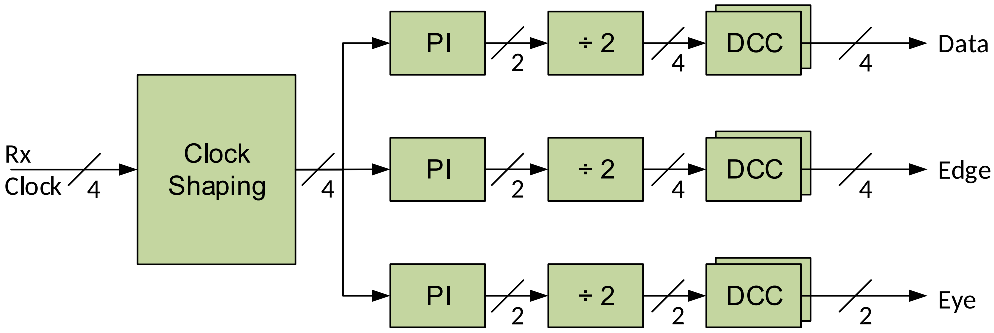


Figure 3.34: Rx clocking overview

The Rx receives four quadrature clock phases from the transmitter. These are necessary to realize the phase adjustment method via phase interpolation. For proper use in the phase interpolators (PI), the input clocks have to be shaped in advance. Because the three sampling paths (data, edge, eye) need to be adjustable independently, there is an independent PI for each of them. A PI interpolates four input phases into two output phases. To generate the quadrature clocks, which are needed in every sampling path, each PI is then followed by a divider. By this division four quadrature phases are generated again. Each phase is fed to a phase and duty cycle correction (DCC) buffer, before it is used by the sampler. Because every sampler has its individual DCC buffer, all timing offsets can be calibrated (see section 3.5.7). In the following, the main blocks of the receiver clocking architecture are described in more detail.

Clock Shaping

As mentioned earlier, the Rx clocking module uses phase interpolation to shift the sampler clocks to the desired time instant. The phase interpolation is achieved through analog weighted summation of quadrature spaced input clocks. For best interpolation results, sinusoidal shaped clocks lead to the highest accuracy. This unfortunately comes into conflict with the requirements of the clock distribution, where fast rise times are necessary to minimize jitter accumulation. This is illustrated in figure 3.35, which shows the simulated RMS random jitter at the output of a CMOS inverter that is driven by clock signals with different rise times. It can be observed that the jitter increases steadily with the rise time. Voltage noise, which overlays the original input signal can lead to higher timing offsets, if the signal slope decreases. Therefore, fast rise times inevitably need to be used throughout the the clock distribution when possible and some means of shaping need to be employed in front of the phase interpolator for proper interpolation.

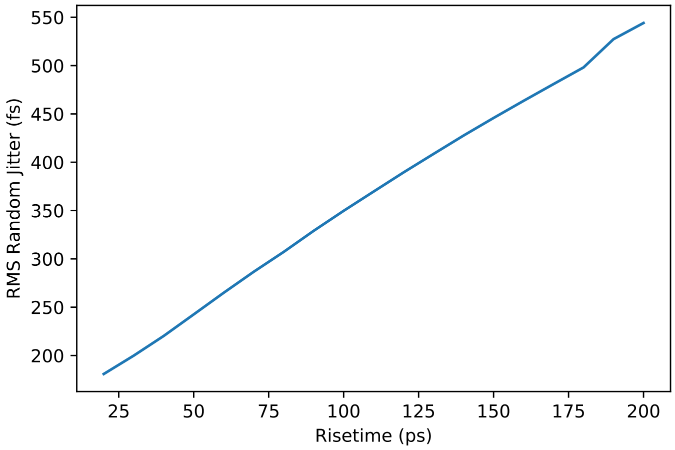


Figure 3.35: Simulated RMS random jitter at the output of a CMOS inverter driven with clock signals exhibiting different rise/fall times

To construct a sine or at least triangular waveform from a nearly rectangular clock several approaches are possible. One main constraint, which limits the solution space is the requirement for the shaping to produce reasonable clocks ranging from 1.25 to 12.5 GHz, in order to fit the multi-rate targets of the SerDes architecture.

To implement the shaping, different mechanisms like analog low pass filtering, digital FIR filtering and a clocked charge-pump approach were analyzed and are now briefly discussed.

The straight forward approach of analog low pass filtering to bring down the edge rate has the drawback that either a higher order filter would be necessary or the fundamental frequency would be attenuated significantly. Also a very wide tuning range to adjust the cutoff frequency would be necessary to cover all the different clock speeds. As passive on-chip components are very expensive in terms of area and their variation is quite high in modern CMOS processes, this was deemed not to be an acceptable solution.

As the different input clock phases could also be seen as delayed versions of each other, there could be the idea to construct a digital FIR filter by building weighted sums of the different phases and hereby construct a low pass filter. This approach is documented by [48]. The benefit is, that it can be adapted to different rates by adjusting the summation weights (filter coefficients), but the downside is, that many stages are necessary in order to get a decent waveform and very low swings are used in the FIR stage depending on the coefficients, which makes them susceptible to jitter.

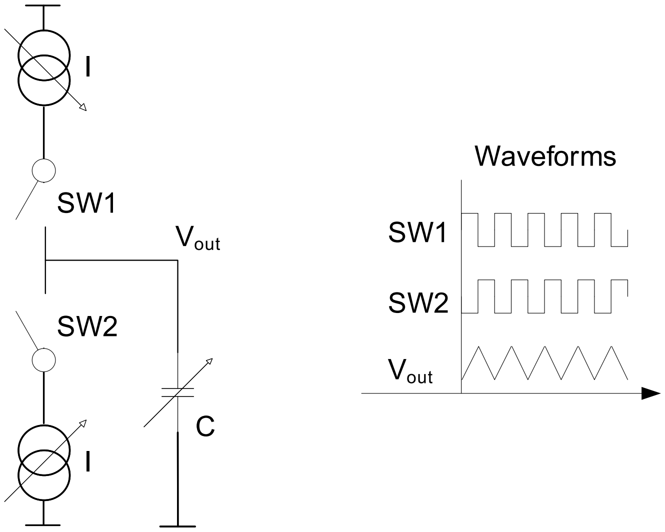


Figure 3.36: Charge pump working principle

The charge pump approach, which turned out to be the most flexible and robust one, was implemented in [49] and later on extended further to work with multiple clock rates. The general idea is depicted in figure 3.36. A capacitor C is periodically charged and discharged, via current sources $I1$ and $I2$, which are connected and disconnected by switches $SW1$ and $SW2$. As the charge on a capacitor is defined by

$$Q(t) = Q(0) + \int_0^t I(t)\,\mathrm{d}t. \tag{3.18}$$

for constant $I(t)$ and $Q(t) = 0$ the equation simplifies to

$$Q(t) = I \cdot t \tag{3.19}$$

By taking

$$Q = C \cdot U \tag{3.20}$$

into account, the voltage U_c after a time t on the capacitor is defined as

$$\begin{aligned} U_c &= \frac{Q}{C} \\ U_c &= \frac{I \cdot t}{C} \end{aligned} \tag{3.21}$$

and therefore linear in C, I and t. Therefore, ideally rectangular clocks at the switches yield a triangular clock at the capacitor. Additionally, the current and capacitance can be adjusted to get the same voltage swing for different clock periods. The actual implementation is built as a differential charge pump for better immunity against power supply variations. Because of current sources in both pull-up and pull-down branches the output common mode voltage is poorly defined [50]. This makes a common mode feedback circuit necessary, which regulates the upper branch so that the output common mode matches a reference voltage. The triangular waveforms are then used at the input of the phase interpolator.

Phase Interpolation

The idea of phase interpolation is to construct the weighted sum of two input phases to obtain an output phase in between as a result. This is illustrated in figure 3.37.

The positive y-axis denotes 0° and the positive x-axis 90° . If e.g. these two phases are summed together weighted equally by 0.5, the new output phase is located exactly in the middle at 45° with an amplitude of $1/\sqrt{2}$. In figure 3.37 different weighting functions were used to construct 64 points to approximate the 360° phase space. The ideal constellation would be to have the points equally spaced on the unit circle. This results in a constant amplitude and equal phase spacing for all points. Obviously, this can be achieved by a sinusoidal weighted summation of the respective input phases. The problem is, that such weighting is rather impractical for an actual implementation.

Usually, phase summation is performed like depicted in figure 3.38. There is an differential input stage for each of the four phases, all sharing common load resistors. The input phases are then summed, weighted by the tail current sources.

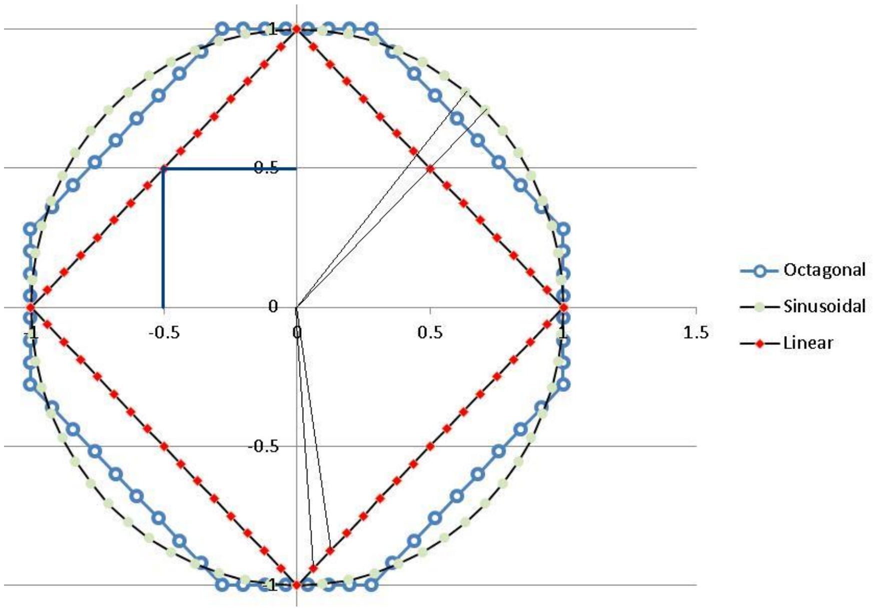


Figure 3.37: Comparison of different summation weighting functions to construct interpolated phases

The design of the tail current source DACs defines the phase positions, approximating the unit circle.

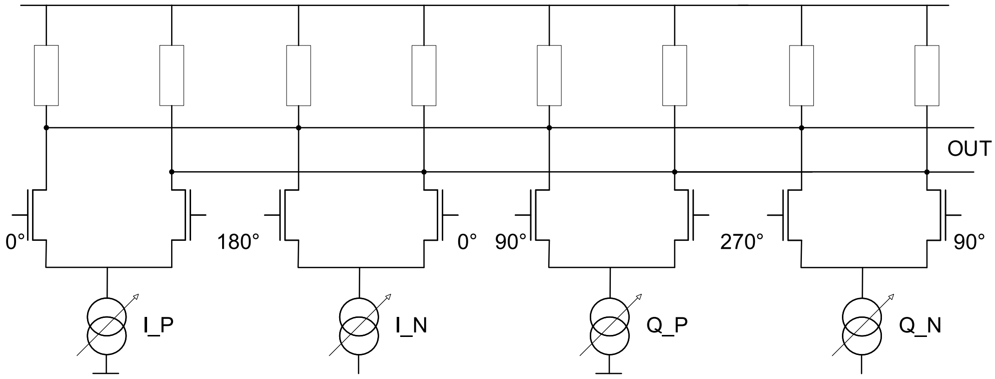


Figure 3.38: CML phase interpolator

The simplest implementation would be to use linear weighted DACs. Current taken from one branch is steered into the other branch, while keeping the overall current constant. At quadrant boundary current is steered almost completely through one branch, whereas the other branches receive almost no current. This also modifies the bias conditions of the mixing transistors, therefore reducing their bandwidth and modifying their delay. This possibly introduces non-linearities in the phase interpolation.

An ideal linear DAC implementation constructs the points on the diamond shape depicted in figure 3.37. The points are equally spaced on the straights, yet the phase angles are not equal. Also the amplitude varies by up to 30%. A much better approximation to the ideal unit circle is achieved by an octagonal phase diagram. The main improvement is a much smaller amplitude variation and a decreased maximum phase error. A very efficient implementation of an octagonal weighted interpolator is described in [51].

For a 6 bit quantized weighting, key figures are summarized in table 3.4.

	Linear Weighting	**Octagonal Weighting**
Max. Amplitude Error	29.2%	3.8%
Max. Phase Step DNL	1.81°	1.46°
Max. Phase Step INL	4.06°	4.56°

Table 3.4: Maximum errors for linear and octagonal weighting

The error of the interpolated phase can be also described as integrated non-linearity (INL) and differential non-linearity (DNL) error. The DNL error describes the difference between ideal step size and the actual step, whereas the INL error describes the accumulated difference which stems from the DNL error of the individual steps. The INL error, though is not important in the context of a CDR, as it will be automatically compensated by the control loop.

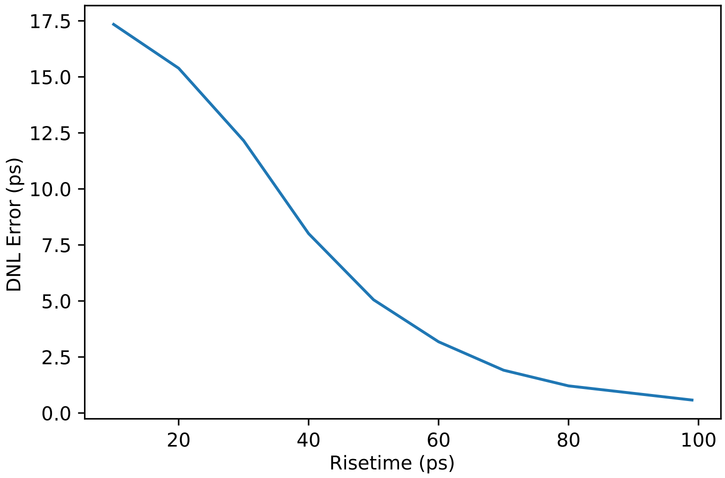


Figure 3.39: DNL error of the phase interpolator for clock signal with different rise times

In figure 3.39 the simulated DNL error of the interpolator output is shown for different input waveform rise times at 5GHz. The values are obtained from SPICE transistor level simulations under typical conditions. It can be observed that the lowest DNL is achieved for 100ps rise time, which is a triangular waveform like produced by the clock shaping circuit.

In figure 3.40 the simulated DNL error of the interpolator output versus the digital control code at 5GHz is plotted. The reference is the simulated DNL error for ideal octagonal weighting, which is obtained from the real number model. The data for sine wave and triangular wave inputs is obtained from schematic level simulations. The plot of sine wave and triangular waveform look almost identical. Further, it can be observed that the maximum DNL error for both sine and triangular waveform is smaller than the reference DNL error, which is due to non idealities of the schematic implementation, which work in favor of the overall DNL error under the simulation conditions.

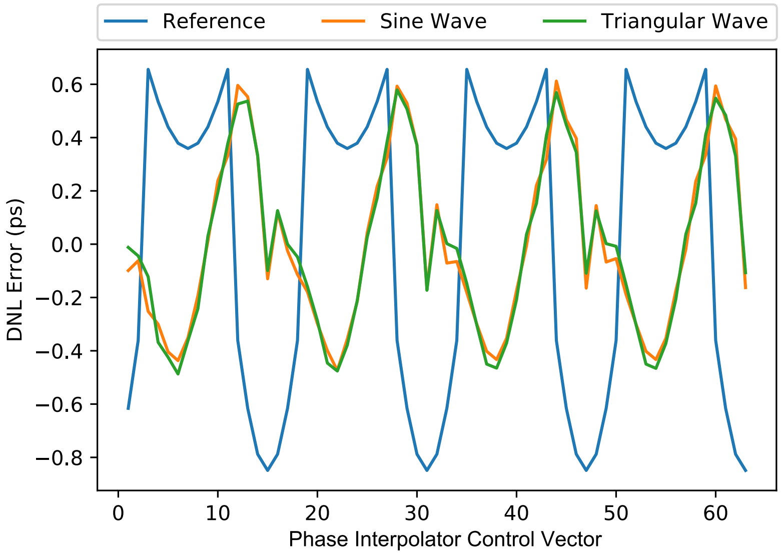


Figure 3.40: DNL error of the phase interpolator for different input waveforms at 5GHz

3.5.4 Digital Clock Data Recovery

The clock data recovery is a very important part of the receiver and greatly determines the overall performance. Its purpose is to extract the phase information from the incoming data stream and adjust the receiver sampling phase accordingly.

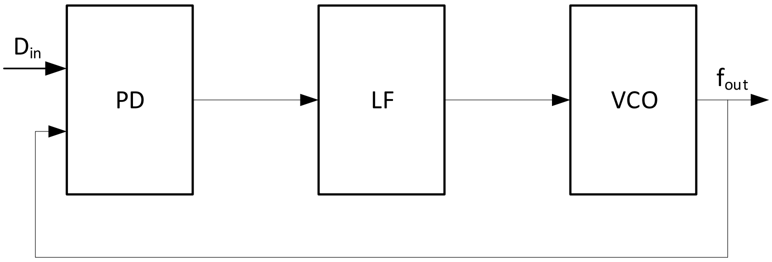


Figure 3.41: Analog CDR working principle

Figure 3.41 depicts the basic working principle of an analog CDR. It is in fact very similar to that of a PLL. From a high level perspective, there is a phase detector (PD), which compares the incoming data phase with the current sampling phase. The phase detector generates a control voltage proportional to the phase difference, which is filtered by the loopfilter (LF) to determine the CDR control loop characteristics. The filtered control voltage is used to set the instantaneous frequency of a voltage controlled oscillator (VCO), which integrates into the sampling clock phase. Unlike in a PLL, there is no feedback divider required, because no frequency multiplication takes place.

Yet, the traditional analog implementation has some drawbacks when it comes to scaling, performance and area efficiency. Because the LF has to be implemented as a time continuous linear filter, it usually consists of resistors and capacitors, which have high tolerances when implemented on-chip and require a lot of area. Further, they do not scale well with shrinking manufacturing technologies and are inherently noisy.

Just like in the field of PLLs, this lead to the emergence of digital clock data recovery circuits. The digital implementation results in higher noise immunity and robustness against power supply variations. It also provides better scaling characteristics concerning process shrinks because of the lack of passive components in the loop filter. Additionally it provides higher flexibility because of digitally configurable filter coefficients.

However the digital approach contains its own set of challenges, such as the introduction of quantization noise in the oscillator and phase detector as well as control loop latencies.

Figure 3.42 depicts the general high level architecture of the so called dual-loop CDR, which was implemented as part of this thesis.

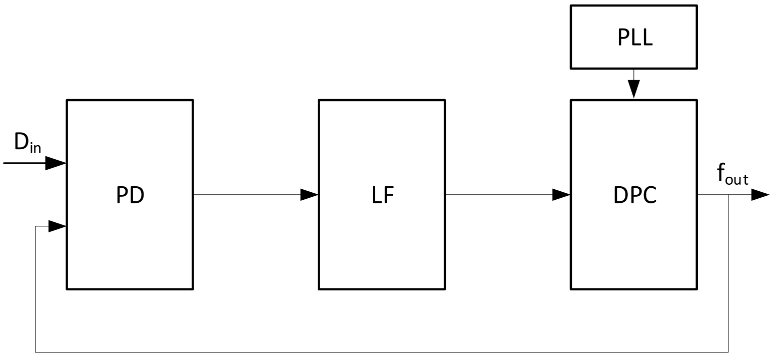


Figure 3.42: Digital dual loop CDR working principle

Just like the analog counter part, this CDR has a phase detector and loop filter. The PD now has to provide the phase difference in digital representation, hence it is often also designated as time to digital converter (TDC). Different approaches to accomplish phase detection exist and will be discussed later in this section. As the phase difference is present in a digital representation, the loop filter can be implemented as a completely digital filter, which allows to leverage the benefits of digital semi-custom design.

The dual-loop CDR architecture does not include a dedicated oscillator on its own, but utilizes a clock which is generated by an additional PLL. The sampling phase is adjusted by a so-called digital-to-phase converter (DPC), which uses the supplied central PLL clock and shifts it by the amount determined by a digital code supplied by the loop filter. In the actual implementation this is accomplished by the phase interpolator already described in section 3.5.3.

The dual-loop approach solves a problem, which arises from two orthogonal requirements on the CDR and (traditionally) its oscillator. On the one hand, it should be able to track changes in the incoming data phase, which suggests a high control loop bandwidth. On the other hand, the sampling clock should have as little jitter as possible, which would suggest a rather low control loop bandwidth. This is especially true, when the reference - which is the datastream itself in this case - is rather jittery. With a dual-loop CDR a low bandwidth PLL can be used to produce a low jitter high speed clock, which is then only phase shifted by a high bandwidth CDR loop.

To obtain the CDR characteristics, the system can be viewed as linear time invariant (LTI) system and analyzed using standard control theory approaches. Because of the digital time discrete nature of the system, the natural choice is to use the z-transformation to describe the transfer function [52]. Though most of the components in the digital CDR are truly linear and can be accurately modeled, the phase detector often is a non-linear circuit which needs to be linearized first.

It should be noted, that although a CDR is very similar to a PLL, not all analysis can be carried out the same way. In a digital PLL the DCO jitter is the dominant noise source and the reference is often assumed to be very clean. The focus lies on the DCO and its quantization effects. In a CDR, the 'reference clock jitter', which is the phase variation of the received data, is dominant. Therefore, the analysis and research focuses much more on the phase detector and its linearized model than in the case of a PLL [53], [54], [55].

Figure 3.43 depicts the z-domain model of the CDR architecture seen in figure 3.42.

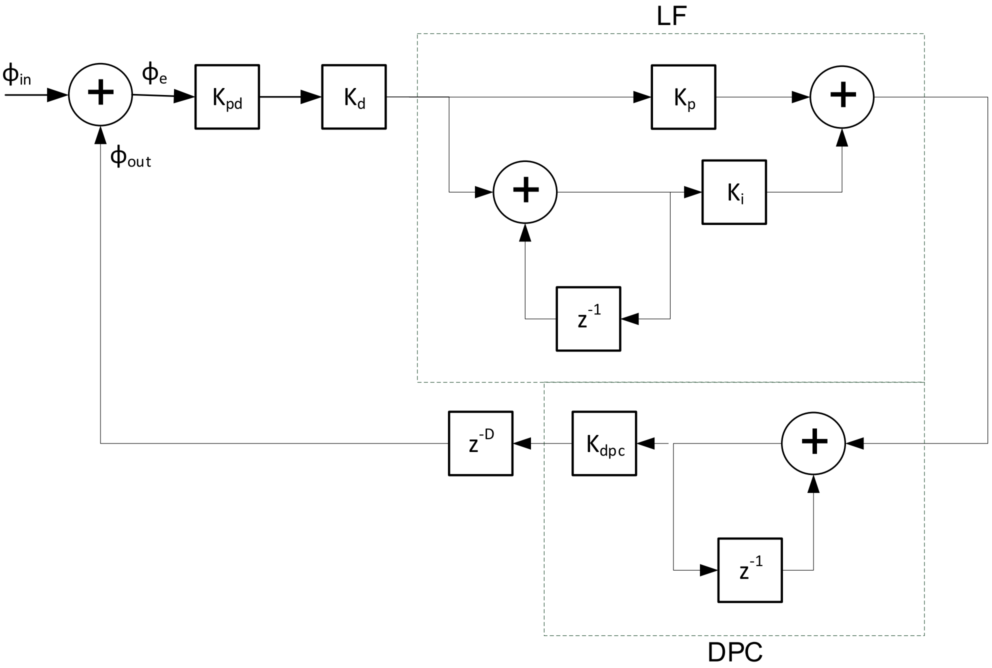


Figure 3.43: Digital CDR Z-domain model

The phase detector is linearized and represented by a constant gain coefficient K_{pd}. As the actual phase information is supplied at symbol rate, but the digital loop filter has to work at a lower clock frequency for practical reasons, another factor K_d is inserted to represent the decimation gain introduced by the demultiplexing. The loop filter itself consists of a proportional and integral path, with adjustable filter coefficients K_p and K_i. As no oscillator is present in the CDR, an additional adder is necessary in front of the digital-to-phase converter to accumulate the phase information. The DPC itself is represented by a gain coefficient K_{dpc}. To model the loop latency which is introduced by the pipeline stages in the digital logic, finally a delay element of latency D is added.

The open loop transfer function is given by:

$$OLTF(z) = \frac{K_{pd} K_d K_{dpc}}{1 - z^{-1}} \cdot (K_p + \frac{K_i}{1 - z^{-1}}) \cdot z^{-D} \tag{3.22}$$

The z domain transfer function can be transformed to s-domain space by means of the backward Euler method, which approximates the Laplace integration by setting $z = 1/(1 - s \cdot T)$. This happens at the expense of non-linear distortion being present in the resulting equation. The transformed transfer function is therefore only meaningful for frequencies much lower than the sampling frequency $1/T$, which is not a major drawback because the CDR loop bandwidth is usually orders of magnitude lower.

The open loop transfer function is used to determine the phase margin of the control loop, in order to ensure stable operation.

The closed loop transfer function

$$CLTF(z) = \frac{\phi_{out}}{\phi_{in}} = \frac{OLTF(z)}{(1 + OLTF(z))} \tag{3.23}$$

of the feedback system can be obtained from the open loop transfer function using standard control theory. The CLTF will be used later on to asses important performance metrics of the CDR.

Phase Detection

As mentioned earlier, the phase detection mechanism in the CDR is a critical point. There are generally different methods to obtain the optimal sampling instant from the incoming data. In the following three methods, which are used in modern digital CDRs are briefly discussed. Hereby the focus is on so called bang-bang phase detection (BBPD) techniques which only supply 1-bit phase information, in contrast to multi-bit TDCs. Bang-bang phase detection is widely used in both PLLs and CDRs because of its robustness and simplicity.

The most common method by far, which is used in the majority of reported designs uses the threshold crossing of the data to extract the phase information. The general working principle of this scheme is depicted in figure 3.44.

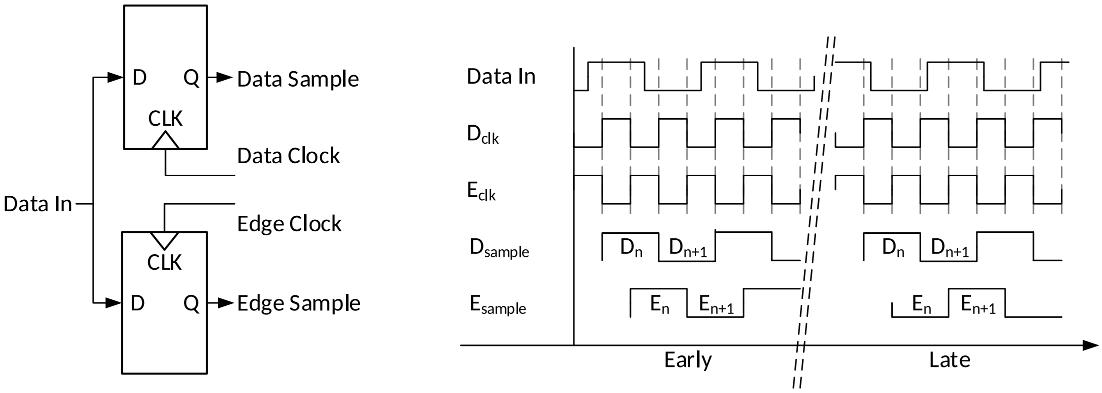


Figure 3.44: Bang bang phase detection

The general assumption is, that the optimum sampling instant is in the center of the eye and therefore half a bit time away from the data crossing the reference threshold. Therefore one sampler is used to sample the data directly at the crossing point (edge sampler), whereas another sampler obtains the actual data (data sampler). The sampling instants of the two samplers are spaced by half a bit time, which means that once the edge sampler is aligned with the data crossings, the data sampler will be in the middle of the eye.

From the comparison of data and edge samples the phase information can be extracted according to table 3.5.

		D_n	
		0	**1**
E_n	**0**	early	late
	1	late	early

Table 3.5: Binary phase detector phase information

The advantages of this type of implementation is its simplicity, while still achieving good accuracy. Though, the disadvantage is the inherent non-linearity, which can lead to noise generation and complicates the overall analysis. This disadvantage is common to all the discussed techniques, because it is inherent to bang-bang phase detection.

Another approach is the Mueller-Mueller phase detection (MMPD) scheme [56], which does not work on data transitions, but uses samples at different reference levels to equalize the impulse response between the sampling instants like depicted in figure 3.45.

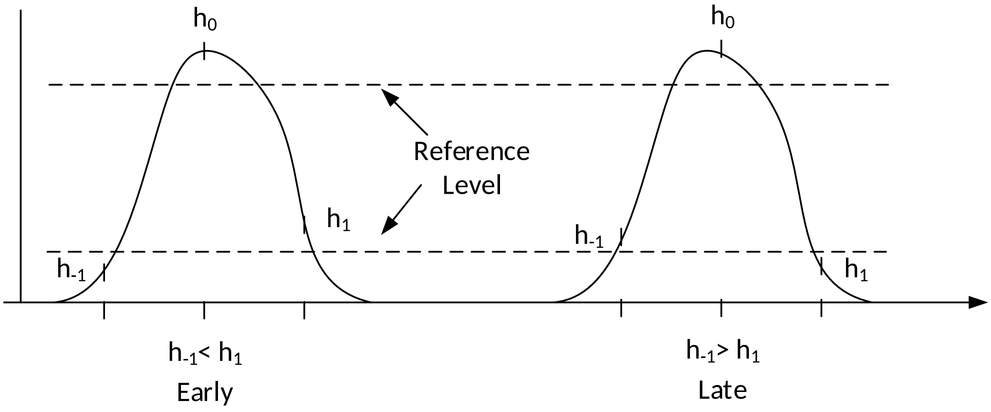


Figure 3.45: Mueller-Mueller based phase detection scheme

The assumption is, that when the amplitude of the impulse response $h(t)$ is equal for samples h_{-1} and h_1, then the sample h_0 is taken at the maximum of $h(t)$, hence at the optimum position. The properties of the MMPD were further analyzed through simulations in the work carried out in [57]. The advantage is that the phase detection happens at baud rate, in contrast to the oversampling, which is required by the first technique. The disadvantage though is, that the obtained sampling position is quite sensitive to the reference levels and the shape of the impulse response, which itself depends on the transmission channel and equalization.

At last there is a phase detector approach, which works on the spectral content of the received data to obtain phase information. These are so called spectral line or mixer based phase detection schemes. The general working principle is depicted in figure 3.46.

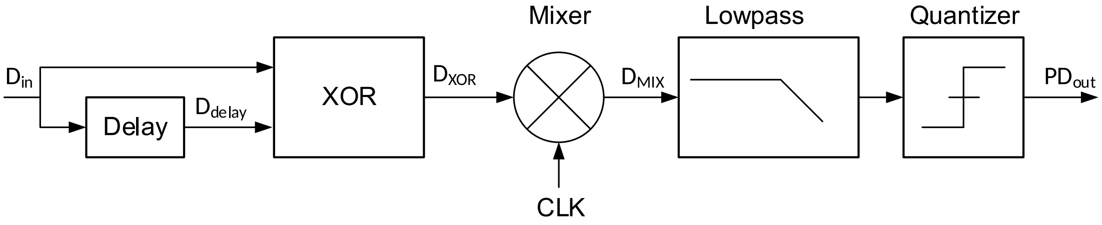


Figure 3.46: Spectral line based phase detection scheme

Non-return-to-zero (NRZ) coded random data generally exhibits a spectral harmonic at $T_{UI}/2$ [58]. This can be extracted by feeding the received data, as well as a $T_{UI}/2$ delayed version of it into an XOR gate. The XORed data is then mixed with the sampling clock. The average mixer output is proportional to the phase difference, like depicted in figure 3.47. For use in a digital CDR the output is low pass filtered and sampled with a low speed comparator to obtain an early/late indication. Because no sampling is involved in the high speed domain, this phase detection is suitable for extremely high data rates. The requirement for a precise delay element in a full-rate mixer implementation like depicted in figure 3.46 is removed in a half-rate quadrature mixer scheme like described in [59].

One disadvantage of this scheme is, that because of the rather analog nature of the phase detector, for use in a digital CDR an analog low pass filter is required at the mixer output, which needs passive components and does not scale very well with technology. The requirements on the filter transfer function also depend on the actual line rate, which is a problem in a multi-rate SerDes design, where the line rate is not fixed.

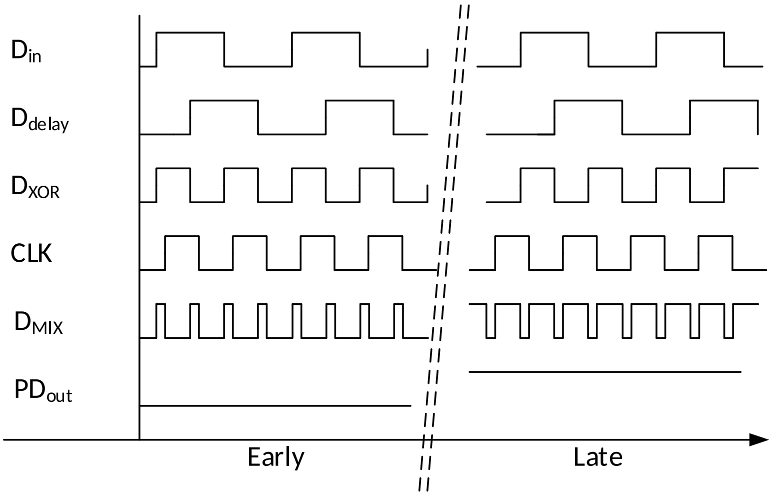


Figure 3.47: Timing diagram of spectral line PD

Though the different phase detection schemes all have their strengths and weaknesses, the classic BBPD was chosen for the architecture of this thesis, because it shares all circuit elements with the normal data sampling path and so no dedicated modules are necessary, which is important for a modular and portable design.

As stated earlier, because the BBPD outputs only early/late phase information, which is highly non-linear behavior, it needs to be linearized in order to be able to apply standard control theory to the CDR.

Although the transfer function of an ideal BBPD is in theory a step function, in practical implementations the probability density function of phase detector output versus input phase does have a finite slope. This stems from the fact, that the non linear behavior of the BB phase detector is averaged by the sampling clock and the metastability of the comparator flip-flop [60].

Because the input signal can be generally viewed as uncorrelated to the PD, the output PDF for a given input signal can be obtained from convolution of the ideal step-like PDF and the PDF of the input signal as depicted in figure 3.48. The resulting PDF exhibits a finite slope which can be linearized to the gain coefficient K_{pd}.

In practical implementations the input jitter is the dominant factor, while intrinsic metastability is comparatively small. All different jitter components of the input signal can be combined to a single PDF and convolved with the PDs PDF to obtain the phase detector gain under these conditions [61].

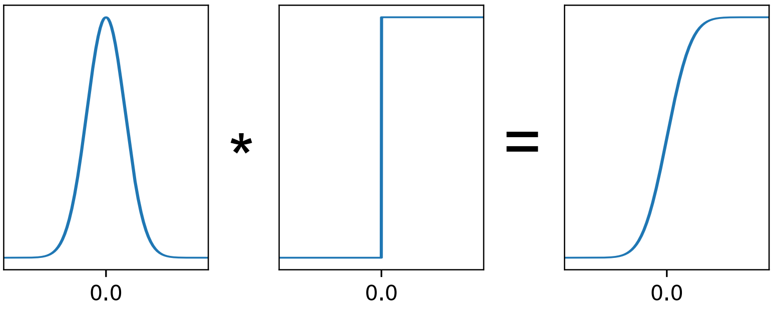


Figure 3.48: Bang-Bang phase detector linearization

If only random jitter with a Gaussian distribution is assumed, K_{pd} can be calculated (as derived in [52]) as

$$K_{pd} = \frac{T}{2\pi \cdot \sigma_j} \tag{3.24}$$

where σ_j is the random jitter standard deviation and T the unit interval. This is the value of the derivative at the origin of the convolution of the step PDF and a Gaussian shaped PDF with standard deviation σ_j, normalized to the unit interval.

Because the slope is not really linear, this means this approximation is only valid for very small phase differences in the locked state. It is also apparent that the whole CDR system behavior heavily relies on the input jitter distribution, which might not be known exactly during the design phase. This has to be taken into account in the implementation.

Implementation

Figure 3.49 depicts the actual implementation of the CDR, which matches the previously discussed theoretical Z-domain model.

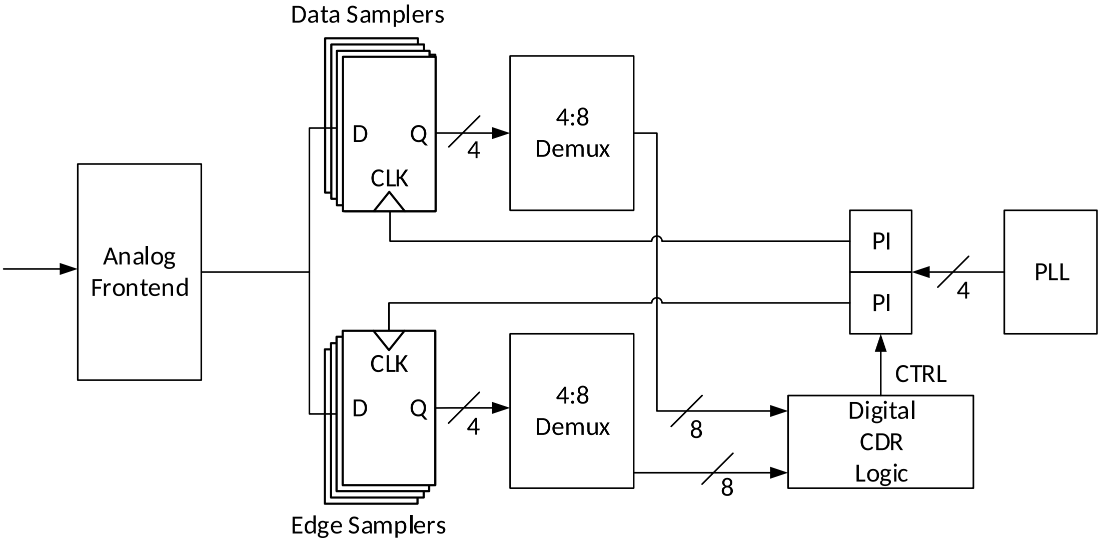


Figure 3.49: CDR architecture

The data is supplied from the analog frontend to the 4 edge and data samplers. The samples are then demultiplexed to 8 bit to lower the clock frequency. Afterwards the 8 edge and data samples are processed in the semi-custom digital CDR block. This block, in turn controls the two phase interpolators, which shift the edge and data sampler clocks. This closes the CDR control loop.

As loop latency is a major concern, the number of pipeline stages in the digital CDR block needs to be minimized. Figure 3.50 depicts the processing steps undertaken to compute the phase interpolator control vector from the edge and data samples.

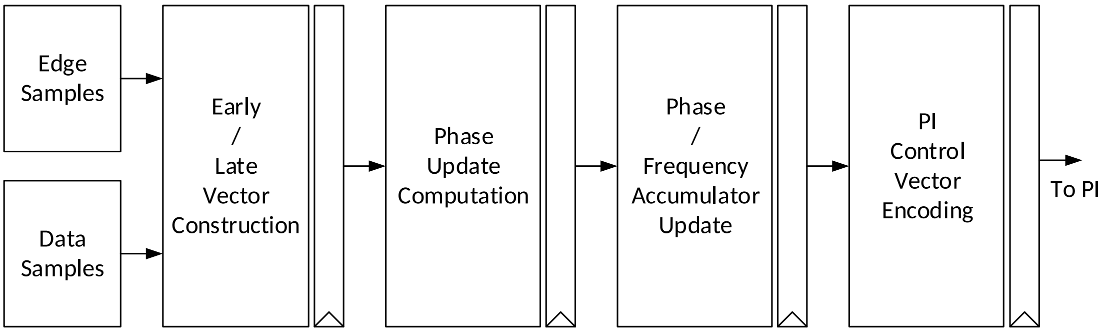


Figure 3.50: Digital CDR logic pipeline

The first pipeline stage extracts early/late information from the edge/data samples according to table 3.5. The second stage uses these vectors to compute the resulting phase increment or decrement. In the third stage, the phase and frequency accumulators are located, which compute the new phase interpolator control vector depending on the loop filter coefficients. The last stage decodes the binary interpolator vector to the actual control vector required by the DACs of the octagonal phase interpolator. Because the CDR clock is divided by 8 compared to the line rate, the 4 pipeline stages add a delay of 32 bit times to the loop latency.

As the input jitter is not known during the design phase, the loop filter coefficients are implemented to be highly adjustable, in order to be able to tune the control loop characteristics to fit a wide range of usage scenarios.

Because there are separate PIs for edge and data sampling, possible skews between edge and data path can be calibrated with the help of the eye monitor.

Metrics

There are two metrics, which are most important to characterize the performance of the CDR in this context, namely jitter transfer (JTRAN) and jitter tolerance (JTOL).

The jitter transfer function describes, how a phase error at the input is translated to a phase error at the output (the sampling clock) of the system in respect to the frequency of the phase change. This is actually the same, as the closed loop transfer function.

The jitter tolerance function describes, what the maximum tolerable jitter amplitude at a certain BER is, in respect to the frequency of a sinusoidal modulation of the data. This is probably the most important property of a CDR and often also part of serial link specifications. Sinusoidal input phase modulation is of course a simplification, but there are several reasons which make sinusoidal jitter a good candidate for this metric.

Random components are not predictable and can therefore not be tracked by the CDR. Deterministic jitter due to channel ISI is bounded, but its frequency content is usually beyond the CDR bandwidth and can not be be tracked either for the most part.

Sinusoidal jitter is predictable and well defined. A frequency offset between Tx and Rx will result in deterministic jitter, as well as for example the (undesired) modulation of the sampling point by a switching converter through the power delivery network. Another view on sinusoidal jitter is, that it represents a worst case scenario, because most of the probability mass of the jitter is located at $\pm a$ for a modulation of the form $a \cdot \sin(\omega_j \cdot t)$ [62]

During the jitter tolerance test, the amplitude of the modulation is increased until a particular BER limit is exceeded [63]. This is repeated for increasing modulation frequencies. The available sampling time T_{slack} for the CDR can be approximated as

$$T_{slack} = 0.5 \cdot UI - D_j - \sigma_j \cdot \rho \tag{3.25}$$

where D_j is the deterministic jitter cause by channel ISI, σ_j is the standard deviation of the random jitter and ρ is the number of standard deviations to calculate the random jitter peak-to-peak value at the desired BER. It can be obtained from the Q-function and is approximately 7 for a BER of 10^{-12} (see section 3.8.5). From the linearized phase domain model of the CDR, additionally to the phase transfer function ϕ_{out}/ϕ_{in}, a phase error function ϕ_e/ϕ_i can be derived from the phase domain model (figure 3.43) as

$$\phi_e = \frac{1}{OLTF(f) + 1} \cdot \phi_i \tag{3.26}$$

where ϕ_i is the input phase difference and $OLTF$ the open loop transfer function.

Therefore, the jitter tolerance at a given frequency can be calculated as

$$JTOL(f) = T_{slack} \cdot (OLTF(f) + 1) \tag{3.27}$$

Below, jitter transfer and jitter tolerance for the implementation described above are plotted. Jitter tolerance, has also been simulated using the real number model implementation in order to validate the design.

As long as the non-linear behavior of the BBPD is sufficiently linearized by random noise, the analytical expressions are in fairly good agreement with the simulations, as seen in figure 3.51. Tough, because the phase detector gain K_{pd} highly depends on the input jitter, it is very difficult to choose loop filter coefficients in advance. Because of this, the automatic adaptation of loop filter coefficients in system is an active research topic.

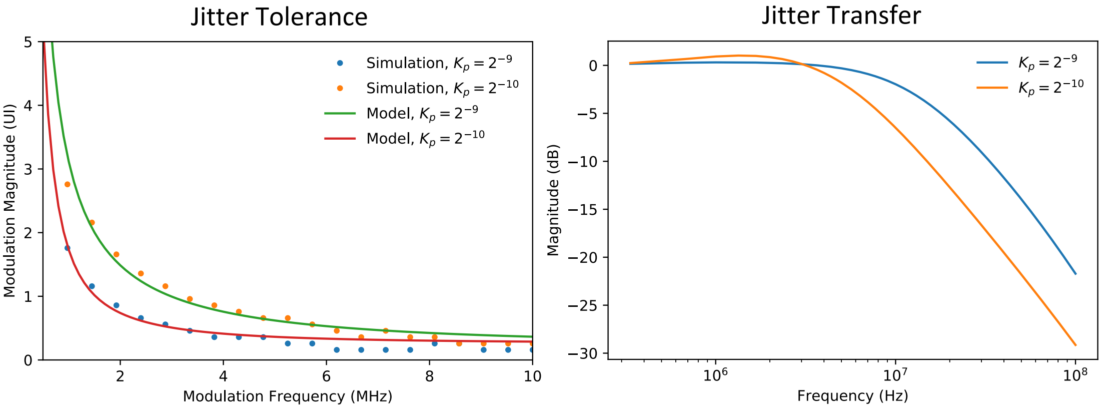


Figure 3.51: Jitter transfer and jitter tolerance functions for different loop filter coefficient and 3ps RMS random jitter in JTOL simulations

3.5.5 Divider Initialization

All the different clocks in the receiver need to have a specific phase relationship to each other for proper operation. For the lower speed clock dividers in the deserialization stages this is solved by initializing the divider flip-flops to known values after power-up. This results in a known phase relationship after reset. A different approach is chosen for the high speed divider path.

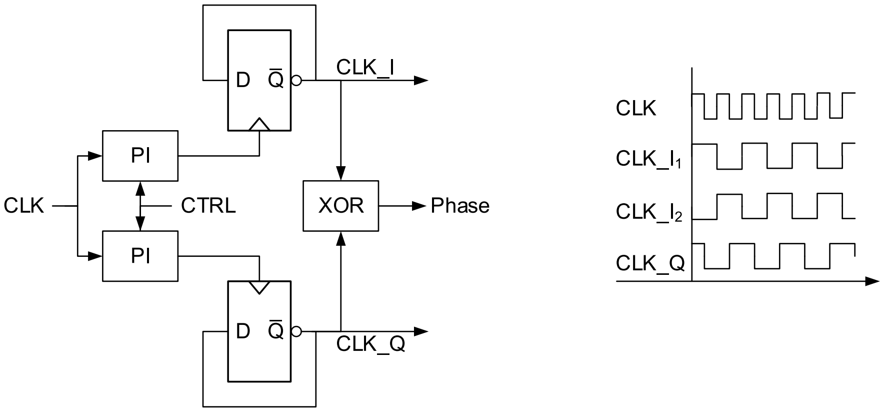


Figure 3.52: High speed divider and phase sense logic (left) and waveforms for two different reset cases CLK_I_1 and CLK_I_2 (right)

As depicted in figure 3.52, there is a clock divider to generate the quadrature phases (IQ divider) for data and edge path respectively. For proper operation of the CDR, the edge clock rising edge has to follow the data clock rising edge. Because the divider has to operate at clock speeds up to 12.5GHz, it is not desirable to add an initialization or reset signal like in the low speed divider case. Additionally the challenge is to pass the initialization signal to both dividers simultaneously. It would be necessary to use a synchronizer to sample the init signal into the clock domain of one divider, and from there pass it synchronously into the domain of the other divider.

The synchronizer itself, which consists of flip-flops, would need to work at the highest clock speed of 12.5GHz. This would lead to additional clock load, area and power consumption. Also the synchronizer would be clocked all the time, while it is only used during initialization, which is a fraction of the overall operation time of the receiver. Of course the synchronizer could be clock gated, but this would only lead to additional complexity.

Therefore, a special initialization scheme was developed which makes the need of init signal synchronization at the dividers redundant.

The scheme uses the special fact that both dividers are driven by phase interpolators.

Because the IQ dividers are located behind the interpolators the effective phase between them can be controlled by the digital control vector of of the PIs. After the power up, when the dividers started operation, the actual phase only needs to be determined. This is achieved by an XOR gate, which is used as a phase detector.

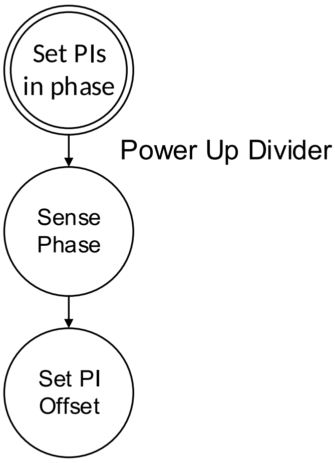


Figure 3.53: High speed divider initialization sequence

Figure 3.53 shows the overall initialization procedure. On startup, the PIs are set in phase using the digital control vectors. The dividers are powered up and the output of the XOR gate is sampled by a low speed flip-flop. As the output clocks of the IQ dividers can now only be in phase or 180 deg out of phase, the output of the XOR is always static. Small glitches due to very small difference in rise time or duty cycle of the two outputs are filtered out by the load capacitance and finite bandwidth of the XOR gate. After the initial phase has been determined, static offsets are added to the control vectors of the interpolators in the CDR in order to get the desired phase relationship of the data and edge PIs. The hardware overhead is very small and this approach is very area and power efficient. The prolonged initialization phase in contrast to a synchronizer approach can be neglected, because the procedure is only necessary once after power up.

3.5.6 Bit Slip Mechanism

Protocols, which are built on top of the SerDes interface, usually work on the granularity of symbols/words which have a width of several bits. In contrast to this, the receiver operates on raw bits of a serial data stream. The CDR finds the optimum sampling point in the middle of the eye, and the data gets deserialized. From this perspective, the alignment of a word boundary at the parallel side of the SerDes, like depicted in figure 3.54, is entirely random.

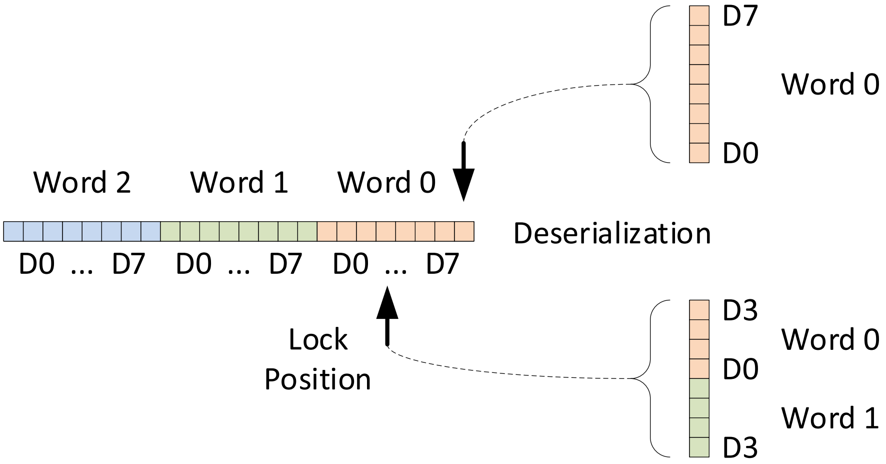


Figure 3.54: Possible word alignments at SerDes parallel side

Because subsequent logic usually needs a specific word alignment at the parallel side, additional logic, which aligns the parallel data to desired word boundaries is therefore required. This additional logic can be built from buffers and barrel shifters, in order to select the desired bits from a number of buffered received bits. This adds additional latency and often also makes the delay through the SerDes block nondeterministic.

Latency variations for different word alignments in Xilinx Virtex 4 and 5 FPGAs have been investigated in [64]. Further, the latency through the so-called 'Comma Alignment' module is reported to be between 32 to 55 UI according to [65] for the latest available Xilinx FPGA devices.

Still, in SerDes applications such as readout networks for physics projects, deterministic latency is often a requirement, and achieving low latencies is very desirable in high-performance networking applications.

While [66] states the existence of implementations based on the mentioned FPGA devices which allow almost fixed latencies irrespective of the alignment, these implementations still increase the overall latency.

In the following a bit slip mechanism which adds no additional latency and provides deterministic and identical delay for every possible alignment is introduced.

The mechanism is directly integrated into the digital CDR as depicted in figure 3.55. As the current sampling position is determined by the digital control input of the phase interpolator, the sampling position can be rotated into the next UI by adding an offset to the phase accumulator. This keeps the latency from sampler clock to word clock edge constant and the symbol alignment is shifted by one bit on the 16 bit parallel side of the SerDes.

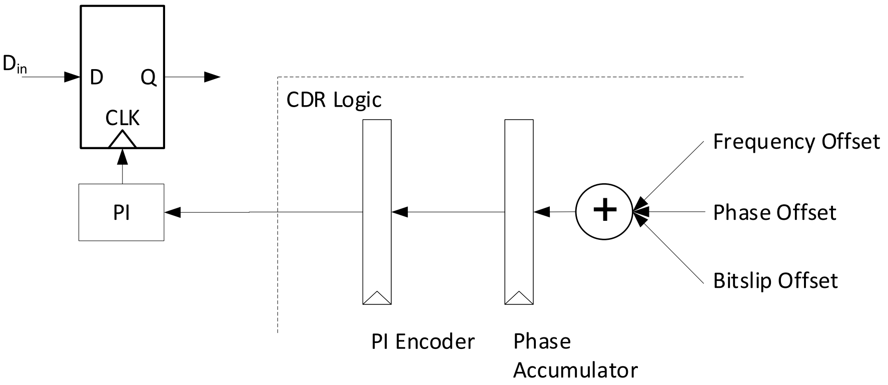


Figure 3.55: Bitslip functionality in the Rx CDR

The penalty is an additional input at the phase accumulator, to add the fixed offset for one UI on top of the updates from the CDR loop filter, when a bit slip is requested. The offset accumulation does not happen in one cycle, but is spread into multiple clock cycles to prevent glitches from sudden code changes.

When the sampling point is rotated to the adjacent UI, the phase information will be corrupted for a number of bit times. Still, because the rotation happens in a very short time frame compared to the time constant of the CDR control loop, the operation of the CDR is not disturbed. The CDR stays locked after the rotation is finished.

This shows the benefit of a digital CDR architecture, which allows precise control of the sampling phase compared to an analog counter part, where such a scheme could not be implemented.

The mechanism has been verified in simulations to prove its reliability.

3.5.7 Calibration

While a static 10ps sampler offset from the ideal position in a receiver at 5Gbps results in only 5% loss of eye margin, at 25Gbps this increases to already 25%. Therefore, the removal of static offsets in both time and voltage domains is getting more and more important with increasing data rates. The single bit times are shrinking, but static offsets are not necessarily decreasing the same way. In fact, higher data rate SerDes PHYs are usually implemented in advanced manufacturing nodes, which tend to increase local variations [67].

A basic problem of calibration usually is that either some kind of reference or additional sensing is necessary. Since the calibration should happen completely on-chip, without the use of external components, a method has to be used which does not involve the use of external references. Also the goal was to add as little additional hardware as possible to measure both timing and voltage offsets, since additional sense hardware is most likely also afflicted by the same offset issues, in turn requiring a calibration. Additionally, dedicated hardware would consume additional area and power which is undesirable. The optimum solution to all of this is, to use the actual data samplers of the receiver themselves to sense the voltage and timing offsets at their input. This approach is called *in-situ* calibration. In the following the calibration mechanisms, which are implemented by using unique features of the developed SerDes architecture are described in detail.

Vertical Calibration

Because voltage offsets in the sampling paths would eventually translate into timing offsets during horizontal calibration, the overall offset removal in the receiver has to start with vertical/voltage offset calibration.

The actual voltage magnitude which needs to be calibrated is technology specific and needs to be obtained from transistor level simulations using mismatch models from the respective foundry. Acceptable residual offsets, which define the finite calibration resolution can be obtained from link budgeting.

The datapath in the receiver frontend up to the samplers is basically an amplifier chain, as depicted in figure 3.56.

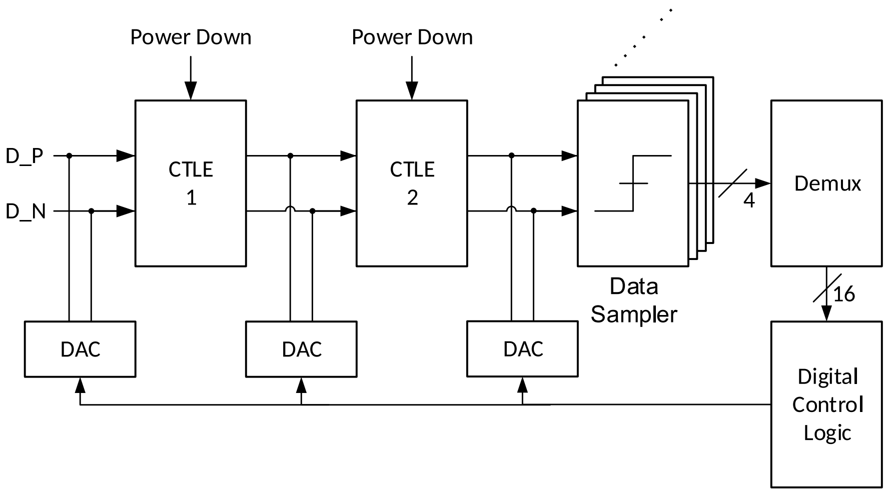


Figure 3.56: Vertical calibration setup

Each of the CTLEs as well as the samplers can have their own intrinsic offset. Therefore they have to be calibrated one by one.

First, everything in front of the samplers is powered-down and a common mode signal is applied to the differential inputs of the sampler comparator. The datapath demultiplexer is used to obtain the sampler results. With its differential inputs basically tied together, due to intrinsic noise, the comparator will on the long term resolve the input signal an equal number of times to both possible logic levels. If an offset exists, one level is favored and either more logic ones or logic zeros are visible at the demultiplexed output. The individual bits in the parallel data output can be associated to a specific sampler. Usually there will be an offset in an differential amplifier stage because of a shift in the differential transistor pair threshold voltage V_t or a mobility mismatch due to local doping variations [68].

A DAC at the sampler input (the same one, which is used to adjust the offsets required by the DFE) is used to introduce an additional offset. With this, a digital control loop can be build.

The DAC control code is monotonically incremented and an up-down counter is used to record the number of logic zeros and ones resolved by the comparator. Hereby a histogram like depicted in figure 3.57 can be obtained. For extreme offset codes always the same logic level is resolved, while there is also an optimum DAC control code, which yields an almost equal number of logic zeros and ones. The histogram can also give a qualitative hint to the amount of noise in the sampling process by the spread of non-saturated counter results, This can be helpful to asses the impact of other noise sources in the entire system.

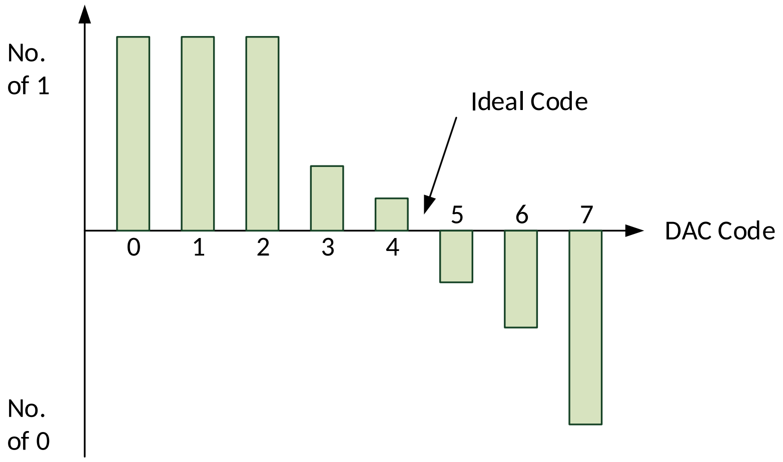


Figure 3.57: Vertical calibration histogram

To calibrate the entire chain, first the optimum code for the sampler offset DAC is obtained. Afterwards, the next amplifier stage in front of the sampler is powered on, its differential inputs tied to a common mode and an offset DAC is used to introduce an additional offset. This procedure is repeated until the complete receiver input chain has been calibrated.

Horizontal Calibration

After vertical calibration has been performed, timing offsets can be addressed. As mentioned in section 3.5.3 each sampler has its own clock buffer which allows to adjust the delay. The sampler phase offset is not calibrated against some external reference. Instead one sampler is chosen as reference and the timing offset against this reference phase sampler is measured.

The actual measurement is then executed as a modified code density test (CDT) in the following way as depicted in figure 3.58.

The internal near end serial loopback is activated, and the transmitter is used to send a low speed clock pattern at the receiver. Hereby, the lane clocking is configured in such a way that the transmitter is clocked by the free running injection locked ring oscillator. In contrast, the receiver is still clocked by the locked lane PLL. Also, the clock data recovery in the receiver is deactivated during calibration.

The received signal and the sampling instant of the receiver are now (ideally) completely asynchronous and uncorrelated. Therefore, from the Rx perspective a rising edge in the received signal can occur at any time with the same probability,

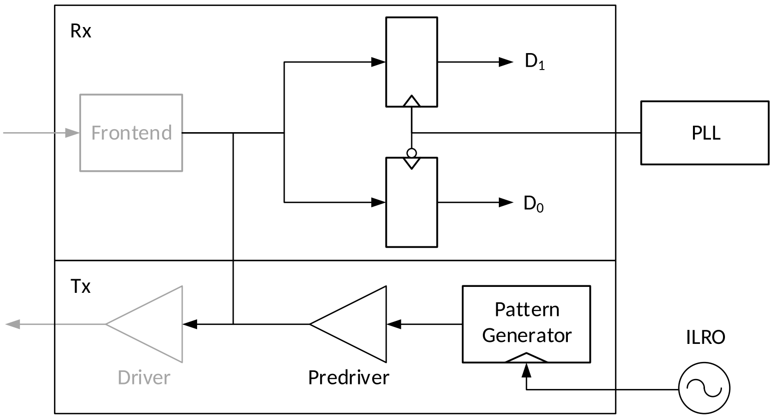


Figure 3.58: Horizontal calibration setup

as depicted in figure 3.59. The phase difference to be measured between the reference sampler D_0 and sampler D_1 is designated ΔT. If the transmitted clock signal period is guaranteed to be greater than T, the probability for a rising edge to fall between two sampling points of D_0 is equal to 1. Therefore, the probability for a rising edge to fall in the interval ΔT is $\Delta T/T$. By counting the transmitted rising edges and the rising edges seen between D_0 and D_1, the phase shift in degrees between the two samplers can be calculated as

$$\Phi = \frac{edges\ detected}{edges\ transmitted} * 360 \tag{3.28}$$

The accuracy of the measurement is improved with the number of samples taken. Using this technique a repeatability of 10fs rms and absolute accuracy 250fs has been achieved in [69].

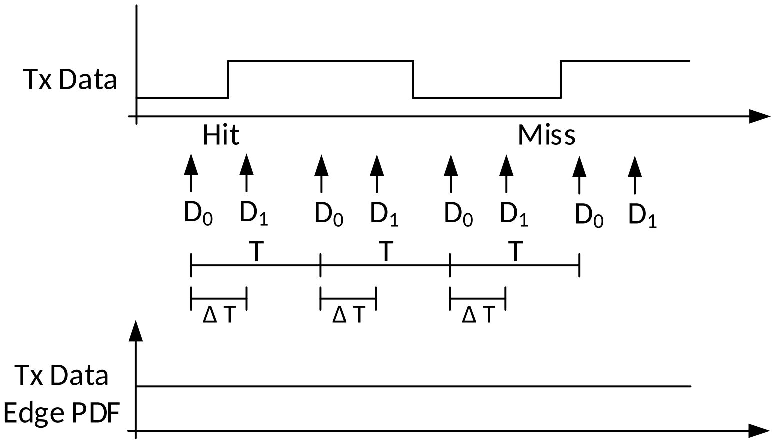


Figure 3.59: Horizontal calibration waveform

After the offset has been determined, the delay of the respective sampler can be adjusted using the DCC buffer. Afterwards the measurement is repeated. This has to be iterated until the offset is minimized. Theoretically all samplers can be calibrated in parallel. However, to keep the number of counters in the digital part small, the actual implementation is limited to one sampler pair at a time. Because the calibration is only performed once on power-up, the additional time needed for calibration can be neglected.

The advantage of this technique is that no additional hardware is necessary in the full custom part of the SerDes. The actual samplers are used to determine the offsets, with no difference to the actual receiver operation. Moreover a very high accuracy is achievable, which is superior to most direct measurement techniques, that would all require some additional sensing hardware.

3.6 Transmitter

3.6.1 Overview

The transmitter is, just like the receiver divided into a full custom partition and a semi custom implementation part as depicted in figure 3.60 The functionality implemented in the full custom partition is kept at a minimum in terms of complexity, in order to improve portability and reduce manual implementation work. Therefore it mainly consists of a segmented SSTL driver and clocking resources which are necessary to distribute high-speed clocks to the driver segments and implement the interface synchronization described in section 3.6.3. The synthesizable logic implements multiplexers from 16 bit parallel data up to 4bit at quarter rate. Additionally the data selection logic for the 4-tap FIR filter is implemented in front of the multiplexer trees.

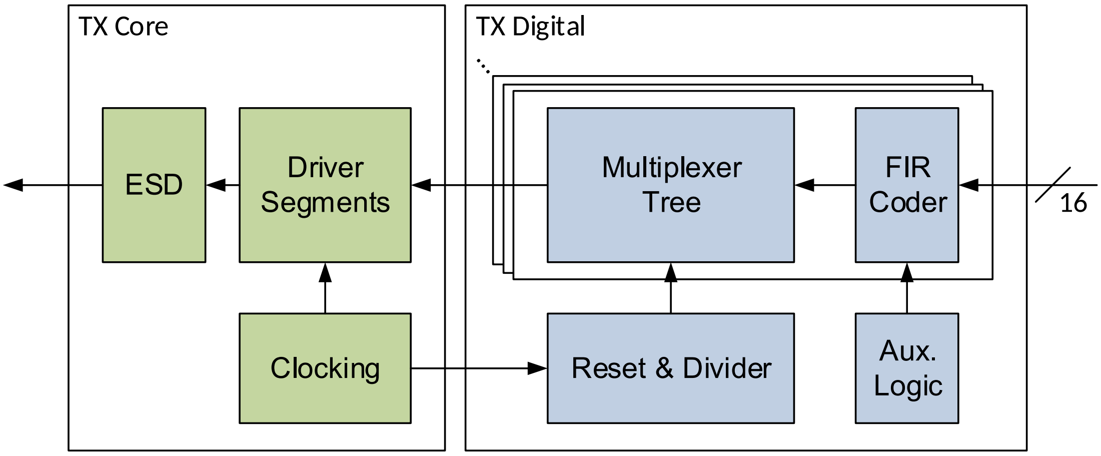


Figure 3.60: Overall Tx overview

Further, all low speed dividers to generate clock phases, which are necessary in the multiplexer stages along with their respective reset logic are located in the semi custom partition. Same is true for auxiliary debug logic used for test pattern generation and loopback.

In the following, the datapath of the transmitter is described in more detail.

3.6.2 Datapath

As stated earlier, the transmitter implements a 4-tap FIR filter in order to apply pre-distortion to the output signal to partly counter the channel characteristics. The number of taps was chosen to be 4, because it also allows to generate PAM4 signal levels.

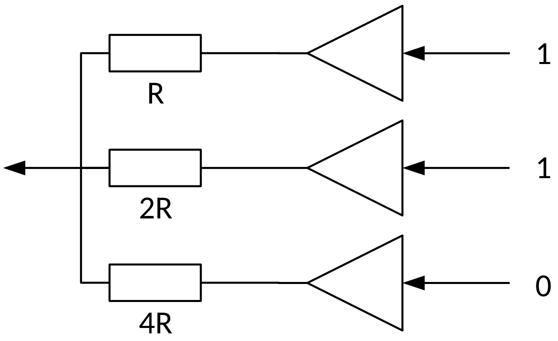


Figure 3.61: SSTL driver segments

Figure 3.61 depicts the working principle of the segmented SSTL output driver. There are multiple CMOS buffers, connected in parallel. Each buffer has a weighted resistance attached in series. The overall segmented driver is designed in a way that all parallel resistors yield an effective 50 Ω resistance. To create a differential output driver, two single ended drivers are used with one driver sending the logical complement.

To create the pre-distorted FIR output waveform, not all segments are configured to send the same data, but some are sending the previous or subsequent bit. Due to the weighted series resistors, the output driver then works like a voltage divider. The number and weighting of the segments dictates the resolution of the FIR coefficients.

For maximum flexibility the cursor time each segment is assigned to, is fully configurable in the synthesized part. Because the driver is built as a quarter rate design, each segment is preceded by a 4 to 1 multiplexer. Therefore the semi custom part needs to supply $4 \cdot N_{seg}$ data bits at quarter rate to the full custom segmented differential output driver, where N_{seg} is the number of output driver segments.

Figure 3.62 depicts the Tx datapath in more detail. Like previously stated, the transmitter is implemented as a quarter rate design to improve energy efficiency [70].

Therefore, the 16 bit parallel input data is split into 4 x 4 bit, and every 4 bit slice is then multiplexed to 1. Because every segment can be configured in respect to the cursor value it drives, each segment requires its own dedicated multiplexer tree. Therefore the 4 quarter rate multiplexer are replicated N_{seg} times.

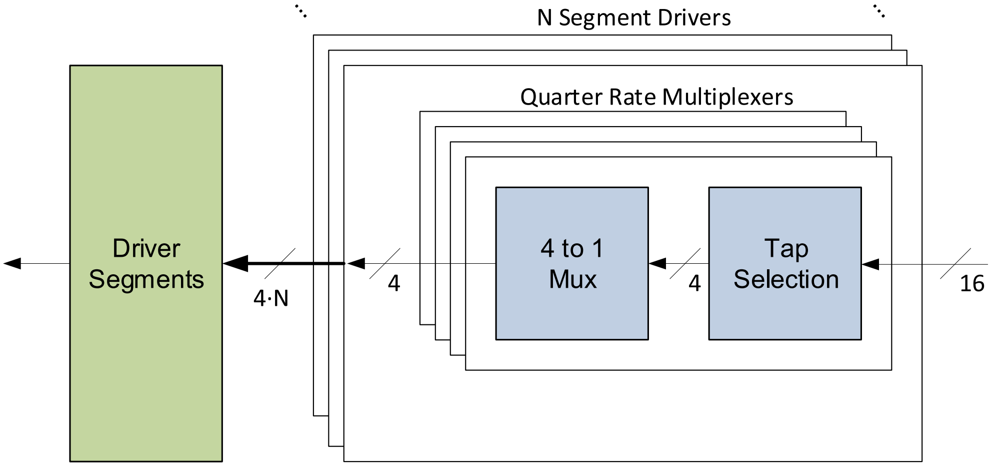


Figure 3.62: Tx datapath

Finally, in front of each multiplexer is a tap selection logic, which allows to select the cursor that the driver segment is going to send. By this tap selection logic, the FIR coefficients are determined.

3.6.3 Interface Synchronization

A special interface synchronization scheme was implemented in the transmitter to facilitate two things: First, the input signals for the 4 to 1 full custom multiplexers need to arrive with proper timing to avoid glitches at the transmitter output. Secondly, it is desirable to use the same transmit side parallel clock for multiple lanes. Both issues are addressed by the mechanism described in the following.

As stated earlier, the transmitter is split into a semi- and a full-custom part. Figure 3.63 depicts the interface boundary without synchronization mechanism to illustrate the problem. High speed clocks are generated within the transmitter core partition and used to clock latches, which are in front of the actual full custom driver segments to retime the incoming data. Because of the relatively small number of latches, they are all driven by the same clock buffer. The clocks are also used in the digital partition to clock the last multiplexer stages. During semi-custom implementation, a clock tree is usually inferred into the design in order to distribute the clock to the individual clock tree leaf elements. This buffer tree adds an additional latency t_{skew} from the clock root to the leaf. The problem is, that this latency degrades the setup timing margin of the latch ($t_{slack,setup}$), which can be calculated as

$$t_{slack,setup} = t_{cycle} - t_{co} - t_{setup} - t_{skew} - t_{pd} \tag{3.29}$$

where t_{cycle} is the clock cycle time, t_{co} the flip-flop clock-to-output time, t_{setup} the sampling latch setup time, t_{skew} the clock skew and t_{pd} the propagation delay.

Because of the timing variations introduced by process, voltage, temperature (PVT) and very small cycle times in the range of 200ps or less, positive setup slack is very hard to achieve.

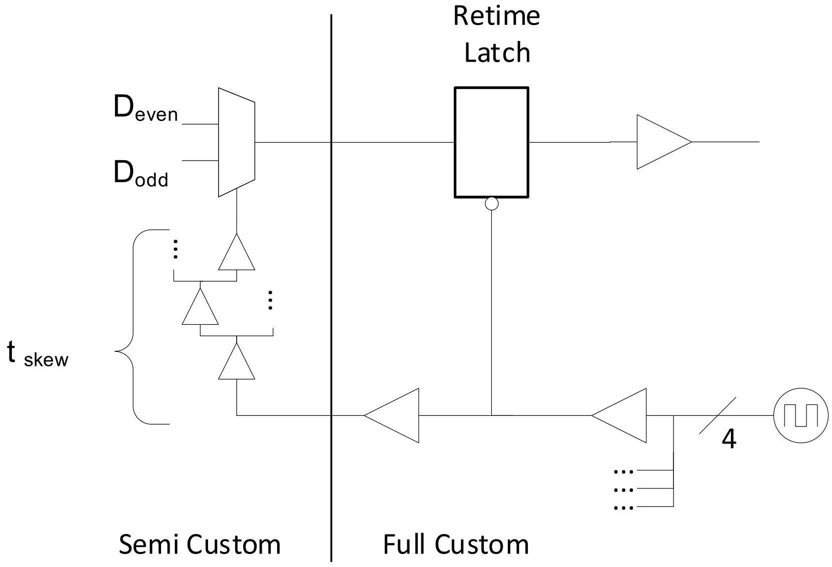


Figure 3.63: Tx semi to full custom interface

To overcome this issue, a synchronization mechanism similar to a delay locked loop (DLL) was implemented at the interface boundary, as depicted in figure 3.64.

A phase interpolator is introduced in front of the clock output to the digital partition. By this, the high speed clock (and the derived word clock) can be shifted, to tune out t_{skew} and align the clock phase at the flip flips and latches on both sides of the interface. It should be noted, that the transmitting clock itself is not touched and no additional jitter at the output driver is introduced.

In order to adjust the interface clock properly, the ideal phase adjustment has to be determined first. This could be facilitated by adding a phase detector, which compares the clock phase at the clock tree leaf and the retime latch, like in a

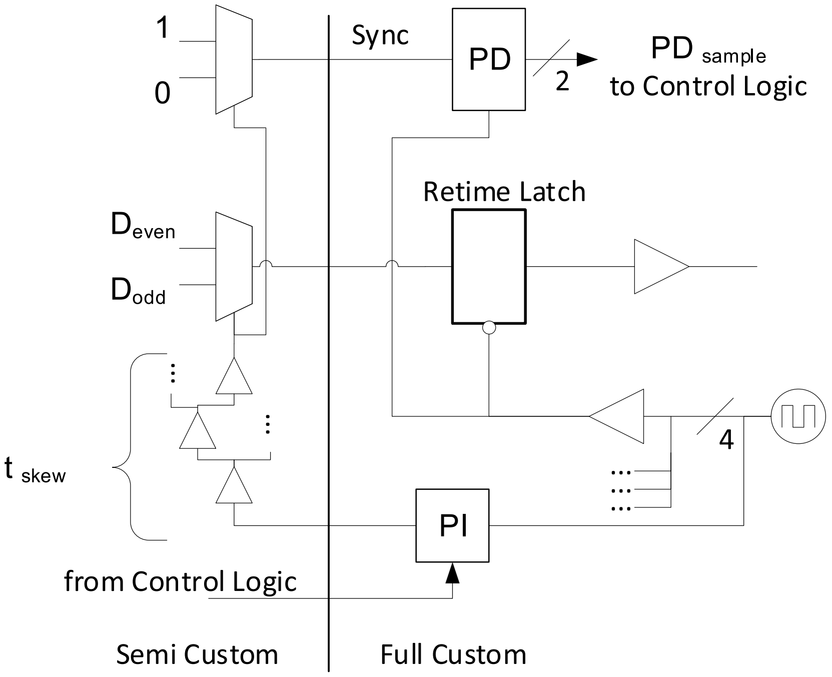


Figure 3.64: PI used for interface synchronization

traditional DLL. Still, a very good timing characterization of the retime latch setup time would be necessary, to allow proper balancing in the semi custom design flow. To overcome this issue, another scheme is used, which adjusts the clock to the optimum retiming instant, regardless of the actual t_{co}, t_{setup} or t_{pd}.

In parallel to the actual data path multiplexers an additional multiplexer, which constantly switches between 1 and 0 is added. This clock signal, which is hereby generated is sampled by two retime latches at the full custom side, like depicted in figure 3.65. The outputs of the two latches work like a bang-bang phase detector and can be used to determine the optimum sampling phase. Because the synchronization signal as well as the sampling is done using the same circuits like the actual data path, the optimum phase adjustment can be determined, without knowing the actual delays of the individual components.

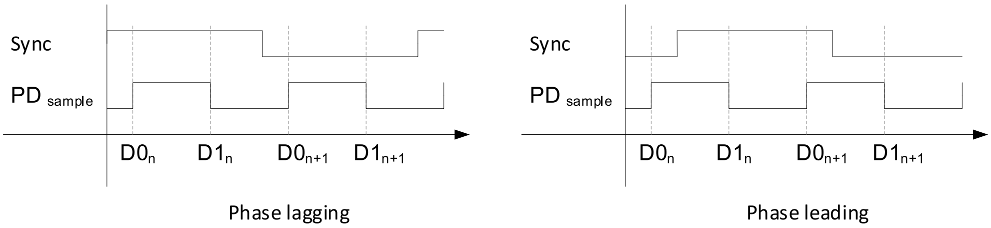


Figure 3.65: Interface synchronization phase detection scheme

		$D0_n$	
		0	**1**
$D1_n$	**0**	lag	no decision
	1	lead	lag

Table 3.6: Two bit phase detector truth table

The phase information can be extracted from the two bit value like shown in table 3.6

No extra components need to be implemented in the full-custom design, because all can be reused from the receiver CDR.

As hinted earlier, once this mechanism is implemented, it can also be used to synchronize multiple transmitter lanes. While all Tx clocks in a link are derived from the same PLL and therefore have fixed phase relationship, this relation is not known a priori when multiple lanes are used together. Therefore, often one Tx word clock is chosen to drive the main logic of a chip and all other word clocks are treated as asynchronous to the chosen word clock. This means that data, which is processed by the main logic and is to be sent over the multi-lane link first has to be synchronized into each local transmitter clock domain. This adds additional latency, which is often undesired.

A two step approach can now be used to synchronize the transmitter word clocks and get rid of any additional synchronization in the datapath. First, phase detectors are necessary at the word clock outputs of two adjacent transmitter lanes. The phase interpolators at each transmitter clock root can now be used to shift the word clock of one transmitter until the phases are aligned. Afterwards the high speed interface synchronization has to take place like described above. This will introduce a small skew between the two transmitter word clocks. Still, the offset is smaller than half a high speed clock period, which is around 100ps at the highest rate. This is the maximum uncertainty that has to be taken into account between two word clock domains. For lower data rates, this uncertainty increases, because the high speed clock frequency decreases - yet the word clock period decreases as well, which compensates for the higher uncertainty.

3.7 Testability Concept

Though sometimes overlooked, it is very important to think about testability and debug capabilities during the design phase.

To be able to characterize and debug the design in silicon or on the system level, it has to be ensured that necessary hardware structures which are needed to facilitate the measurement tasks are actually present in the design. Signals or data samples which are easily accessible in simulation need to be made observable in the final chip. Therefore a number of additional functions, which are dedicated to test and characterization need to be implemented.

The test and debug functions are integrated in each individual lane. As the SerDes is developed as a multi-protocol transceiver this makes sure that in each scenario all debug functionality is present, regardless of the capabilities which might be provided by the respective protocol.

In the following, different functions which all contribute to the overall testability of the SerDes are introduced.

Test pattern Generation and Checking

Test pattern generation using pseudo random bit sequences (PRBS) is a standard approach in the industry. They are normally employed during bit error rate testing (BERT). The test patterns are usually generated by the means of linear feedback shift registers (LFSR), like depicted in figure 3.66.

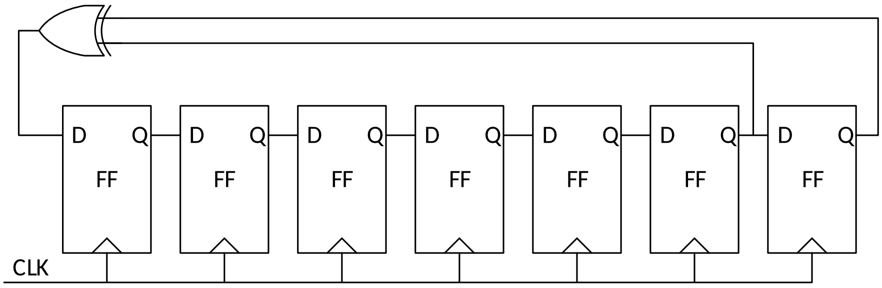


Figure 3.66: LFSR for PRBS7

The input of the shift register is constructed from feedback taps of the shift registers and XOR gates.

The position of the taps determines the generator polynomial of the LFSR. Different generator polynomials can be used to mimic data encodings used by different protocols. For this multi-protocol SerDes, PRBS generators and checkers for different standard polynomials are included, specifically:

- PRBS7 ($X^7 + X^6 + 1$)
- PRBS15 ($X^{15} + X^{14} + 1$)
- PRBS23 ($X^{23} + X^{18} + 1$)
- PRBS31 ($X^{31} + X^{28} + 1$)

The number identifying the PRBS sequence hints the generator polynomial, which is used and also states the number of possible pattern combinations, the length of the shift register and how many consecutive ones or zeros can occur. Because of the nature of LFSRs, the checkers used on the receive side are self-aligning, which means there is no extra hardware needed to obtain a symbol alignment. It should be noted, that some extra logic needs to be added to identify if the received data is all zero, because all zero data fed into the checker LFSR will always return zero. This is because mathematically the checker divides the incoming data stream by the generator polynomial and zero divided by anything will always return zero and can be mistaken for no erroneous bits being received. This could lead to a lane being falsely identified as working perfectly while actually being stuck at zero.

Additionally, a 128bit long custom pattern can be send, which can be used to characterize specific details in the transmitter, such as multiplexer setup/hold issues, duty cycle distortion, PLL and clock generation jitter as well as channel issues like worst case ISI.

On the receive side, there is no checker for the custom pattern generator because this would require symbol alignment logic, which requires additional logic.

The bit sequence, which is generated by the LFSR also depends on the initial values of the shift registers (seed value). It is beneficial to be able to control the seed value in order to load different seeds for neighboring lanes, when crosstalk is going to be analyzed. If all lanes are using the same seed, the bit pattern is synchronized and cross talk effects might be underestimated during PRBS testing, when all lanes are sending the same pattern.

Loopback Paths

Loopbacks are essential for debugging to determine in which portion of the design or link errors are introduced. Figure 3.67 shows the different types of loopbacks, which are usually present in a SerDes design. They can be divided into near-end and far-end as well as serial or parallel.

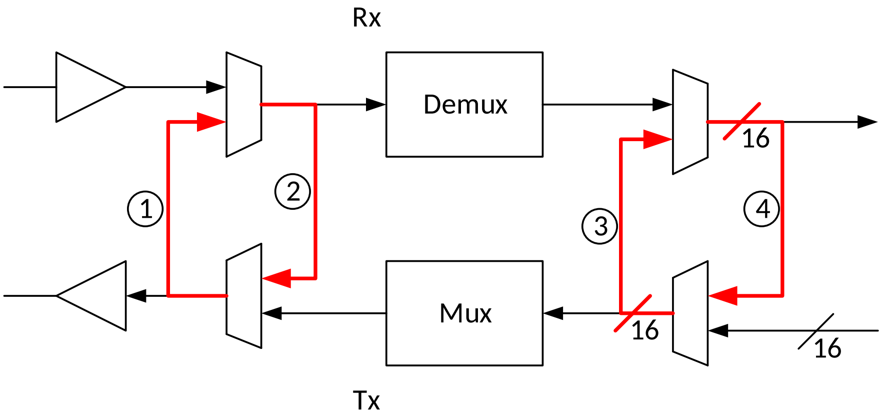


Figure 3.67: Different types of loopback locations: 1) near end serial 2) far end serial 3) near end parallel 4) far end parallel

At the near end parallel loopback, data which is sent on the transmitter is looped back to the parallel side of the own receiver before even being multiplexed to the highest data rates. Using this loopback, timing errors in the digital part and on the interface of the SerDes to the surrounding logic can be identified. The actual serial part, nor the transmission channel nor the receiver on the other side (far end receiver) are involved. Some designs introduce additional near end parallel loopbacks at different stages of the multiplexers for internal testing of the high speed multiplexer/demultiplexer structures.

The near end serial loopback takes serialized data of the transmitter and loops back to the own receiver serial input. This is useful to test all serial and high speed logic of a complete lane, without the degradation of the actual transmission channel. It can be used to make sure that the actual transceiver is operating correctly, taking all external structures, like PCB, package, soldering etc. out of the equation. To stress the receiver, some designs have the ability to add a degrading buffer in the loopback path to mimic channel loss.

A far end parallel loopback takes received and fully deserialized data at the parallel digital side and sends it on the own transmitter. This can be used to test a lane, with a transmission channel and a remote transceiver involved.

At last a far end serial loopback takes the serially received data and sends it back out on the local transmitter right away without using its own CDR to sample the data, while the transmitter often only works in a simple buffer mode.

In the current design, all these possibilities except the far end serial loopback have been implemented. If bit errors turn up, the loopback modes greatly help to narrow down the problem and identify the location of the root cause.

Digital Observation and Override

Each lane has a dedicated control and status register file as well as a direct configuration and control interface. The register file is built using a generator, which allows to automatically generate HDL and verification code from a special register file description language [71]. The register file can be accessed over a memory interface, which can for example be hooked up to the internal register file structure of an SoC or is directly attached to an off-chip interface like I2C. A single lane has about 250 different configuration and status registers which sum up to around 4kb in combination.

The direct interface is intended to be controlled by hard-wired FSMs, which e.g. control power-up, rate change, calibration, bitslip, equalization adaptation. Despite all verification efforts, there is always the probability that such FSMs contain bugs which prohibit proper system functionality and are very hard to debug in system. To address such issues, every functionality in the SerDes can be controlled and observed via the register file, using the scheme depicted in figure 3.68.

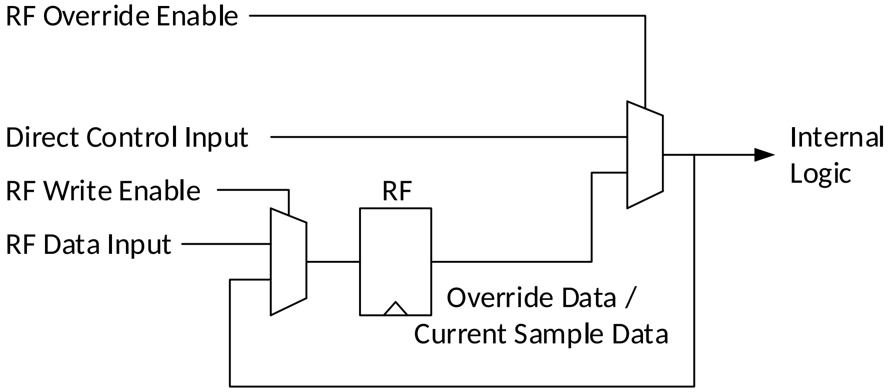


Figure 3.68: Observation and override through the register file

For each direct control signal, there is a multiplexer which is controlled by an override enable from the register file. The multiplexer determines, if the internal logic is controlled by the direct interface or the register file itself. When the override is not enabled, the current value, which is driven to the internal logic can be observed from the register file. This can be used to debug external FSMs and override their behavior if necessary. Via this mechanism the whole PHY can be initialized and controlled entirely by external software as a fall back solution.

On-Die Eye Monitor

The on-die eye monitor is implemented using the eye/error sampler, which can also used for equalization adaptation. The eye sampler is an additional sampler in parallel to the data samplers which can be configured to different sampling instants and signal levels independently. Sampled eye data can be compared with the values obtained from the data samplers. By sweeping the eye sampler horizontal and vertical offsets compared to the data samplers, a 2D map of the received data eye can be constructed. If a sufficient number of samples is taken, this information can be used identify the effect of transmitter and receiver equalization as well as random noise levels or construct bathtub curves to analyze the quality of the link in terms of BER. In figure 3.69 on-die eye diagrams obtained using the actual hardware implementation and RNM models for an equalized and unequalized channel are plotted for illustration.

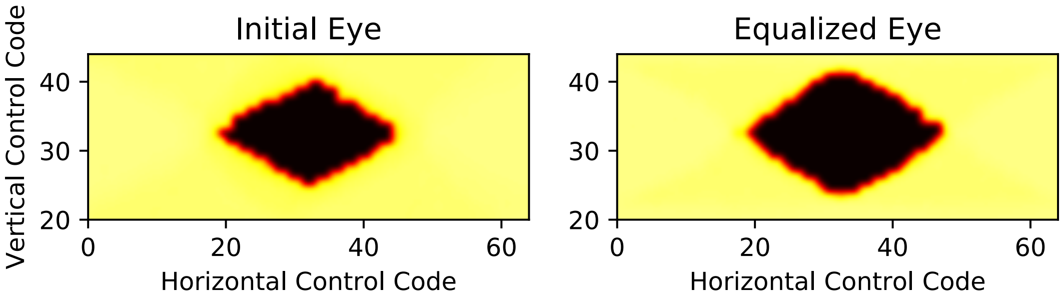


Figure 3.69: Plots for different equalizer settings using the on-die eye monitor

It can be observed how the eye for the equalized channel on the right side opens up because of the reduction of ISI. Because the eye diagram is obtained with an additional sampler, it can be built concurrently, while real traffic is sent over a link. This can give better insight on the nature of additional noise effects, which might arise when more digital logic on a larger chip is activated and produces switching noise on the power supply rails. The on-die eye monitor is especially useful because the actual sampling hardware of the SerDes is used, which obviously takes into account all the degradations on signal and power supply the samplers suffer. Additionally it is nearly impossible to probe the signal at the receiver using external equipment without introducing further changes to the transmission channel. It would not be possible to use an external probe in a real system, because there is simply no probing location where a meaningful signal could be obtained at such high frequencies.

Analog Test Bus

Besides the digital testability functions, sometimes it is useful to be able to probe actual voltages in the design. As it is obviously impossible because of pin limitations to add dedicated pins for every node that should be observable, some kind of multiplexing has to be employed.

A single analog test bus (ATB) pin is used to access probes, which are added to the design for specific nodes. The probes and multiplexer structure can then be controlled via a control register file to connect the node of interest to the shared ATB pin.

Whereas in [72] an actual passgate based multiplexer structure is proposed, which also allows nodes to be controlled, in [73] source followers are used to sense the test point voltages over a common test pin as depicted in figure 3.70. This approach has been adopted for the current design.

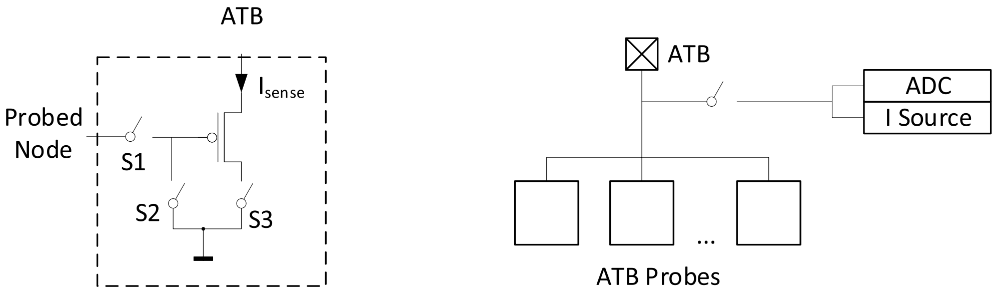


Figure 3.70: ATB probe circuit (lef) and on-chip probing architecture (right)

As described in [73], to measure an internal node, a current I_{sense} is applied at the ATB pin and the specific test point is activated by closing switch S3. First, switch S2 is closed, while S1 is open and the voltage at the ATB is measured. Afterwards S1 is closed, while S2 is open and the voltage is measured again. The voltage at the probed node can then by calculated as the difference of the two measurements.

This approach does only permit sensing voltages, but also results in higher isolation between probed nodes. Also, the bandwidth is very limited because the ATB bus is loaded by the capacitance of all probes and interconnect.

The ATB can be extended in the future, by adding an on-chip ADC and current source as depicted in figure 3.70. The ATB could then not only be used for test an debug, but also as shared means for PVT calibrations after power-up. Because all probed voltage values could be accessible from a central register file, this could be used for software controlled calibration loops to adjust bias currents or resistors to optimum values. As the ADC can be shared over all probes connected to the bus, the area penalty is small, while more sophisticated implementations can be used.

JTAG Boundary Scan

The Joint Test Action Group (JTAG) developed the IEEE standard 1149.1 [74] in the mid 1980s, which defines so called boundary scan testing. This mechanism is intended to test interconnect structures between different ICs after assembly on a printed circuit board (PCB), which was formerly done by connecting test probes in a "bed of nails" - like approach.

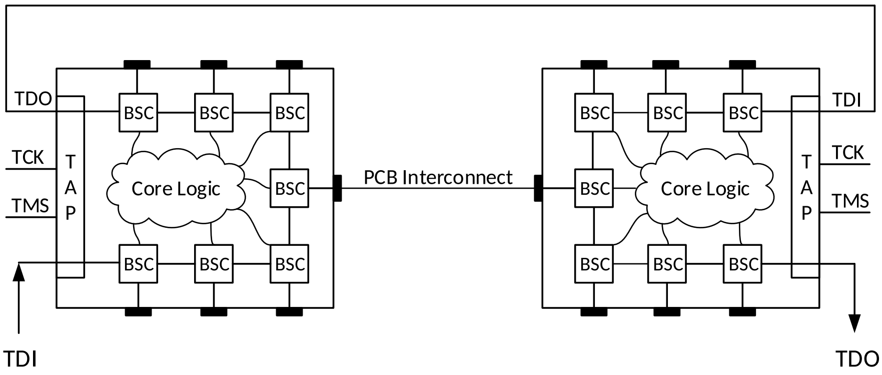


Figure 3.71: JTAG boundary scan

As depicted in figure 3.71, to accomplish this, every I/O has to contain a boundary scan cell (BSC). The BSC does provide means to drive or capture the data on that specific pin, independently from its original functionality. It also contains multiplexer structures in order to be transparent during normal operation. All BSCs are connected like a shift register chain, in order to be able to serially write or read test patterns using the standardized test access port (TAP). Multiple ICs can be connected in series, in order to have a low pin count interface for testing that well suits the automatic test equipment (ATE). While the original standard only addressed the use of static test pattern, with the emergence of high-speed links, the IEEE standard was extended in [75] to cover testing of differential AC-coupled signals.

Obviously for a SerDes it is also desirable to have boundary scan capability for the serial input/output in order to be able to integrate it in standard manufacturing tests. Yet, because of the very high data rates it is not straight forward to add a simple boundary scan cell to the I/O port, like it is e.g. on a digital CMOS configuration pin. Every additional logic adds capacitive loading at the high speed nodes, which potentially degrades the performance. Though, because boundary scan is usually only used in high volume testing, it was not considered in the architecture developed in this thesis.

Digital Internal Scan

To test synthesized, digital logic for manufacturing faults, internal scan is the standard approach. During implementation, all flip-flops in the design have to be exchanged with scan flip-flops like depicted in figure 3.72. A scan FF has a multiplexer in front of the actual storage element, which selects between the data- and the scan input. Via the scan input all flip flops in the design can be connected as a shift register (or multiple parallel shift registers). When the scan enable (SE) is activated a specific test pattern can be serially shifted into all FFs.

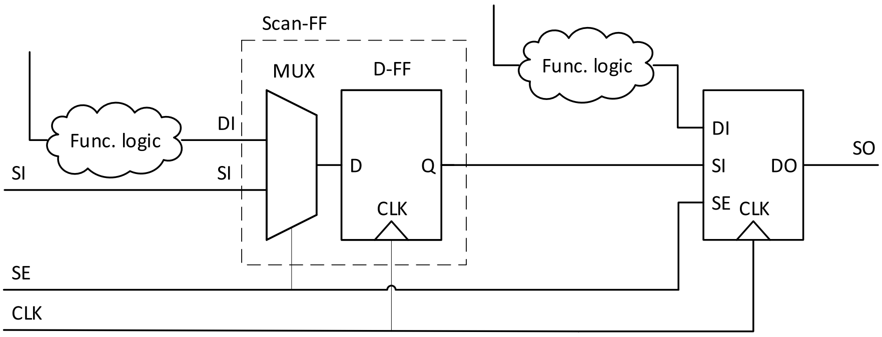


Figure 3.72: Internal scan

Afterwards, the SE is deactivated and the clock signal is toggled. Now the SE can be activated again and the results are shifted out for evaluation. Special automatic test pattern generation (ATPG) software is used to generate and evaluate the scan flip-flip data based on fault models. With this technique consequently applied, a digital flip-flop based design can be exhaustively tested for manufacturing faults. The combinational logic in between of the FFs is directly tested this way, in contrast to normal operation where it would be very difficult, if not impossible to identify manufacturing faults down to the root cause.

3.8 Physical Coding Sublayer

The Open Systems Interconnection (OSI) model, which partitions a communication system into 7 abstract layers, designates the lowest layer as the physical layer. On this layer the actual raw data transmission takes place, independent from higher level protocol related issues.

The physical layer itself is often also split into two sublayers, the physical media attachment (PMA) layer and the physcial coding sublayer (PCS). The PMA is essentially the raw SerDes, which is discussed in chapter 3, that electrically (or optically) interfaces the transmission medium and provides a parallel interface to a serial channel.

The PCS builds on top of the PMA and fulfills a number of slightly higher level tasks, such as applying line coding/decoding and data scrambling, establishing symbol alignment and word synchronization or providing elastic buffering to overcome frequency offsets between far end and local reference. These functionalities will be covered in context of a 'PHY Interface for the PCI Express Architecture' (PIPE) implementation, which is described in the following.

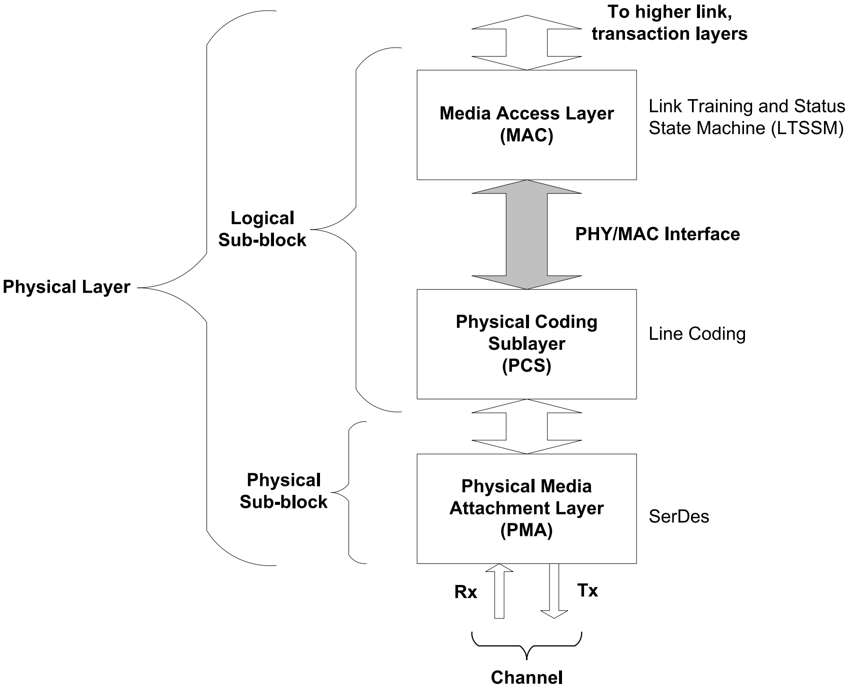

To higher link, transaction layers
Media Access Layer (MAC)
Link Training and Status State Machine (LTSSM)
Logical Sub-block
PHY/MAC Interface
Physical Layer
Physical Coding Sublayer (PCS)
Line Coding
Physical Sub-block
Physical Media Attachment Layer (PMA)
SerDes
Rx
Tx
Channel

Figure 3.73: Physical layer partitioning, after [76]

The PIPE is a standard interface between the SerDes PHY composed of PMA/PCS and the media access controller (MAC), which handles the overlying protocol layers. Figure 3.73 depicts the physical layer as it is defined according to the PCIe specification. Because PHY and MAC are often not developed by the same company, the PIPE interface is an attempt to ensure interoperability and ease concurrent developments.

The PIPE specification only defines the interface signals, but makes no assertions on the actual implementation. To be able to use the SerDes developed in the course of this thesis with a standard PCIe MAC a PIPE compatible PCS was implemented. In the following, the PCS and the challenges that had to be overcome will be discussed in more detail. Hereby the focus is on the general ideas, which are universal to PCS layers and not only specific to PCIe. Therefore the actual PIPE interface, which is defined in [76] is not described in detail.

The PIPE PCS is written in Verilog HDL, in order to be used in an FPGA or implemented as a semi-custom design in an ASIC.

3.8.1 Datapath Overview

Figure 3.74 depicts the complete PIPE PCS of a single lane. Each lane top module contains the receiver and transmitter datapath top modules. Because the line coding is different for PCIe Gen 1/2 and Gen 3, each datapath is split and the appropriate coding can be selected depending on the mode of operation. Whereas PCIe Gen 1/2 uses 8b/10b coding, Gen 3 uses 128b/130b coding together with scrambling.

The top module of the Tx data path contains encoders for 8b/10b and 128b/130b as well as a 20 to 16 bit gearbox and a rate converter respectively.

The 8b/10b encoder takes 2 bytes and maps them to 2 10bit symbols according to the coding defined in [77]. The coding ensures DC balance, which means that the number of 1 and 0 bits over a certain interval is equal on average, which is important for AC-coupled serial links. It also ensures a maximum runlength of 5 bits, which guarantees a minimum amount of data transitions, which is important for the CDR. The downside is that there is a coding overhead of 25%, which reduces the net data rate.

Gen3 operation uses a 128b/130b encoder, which directly works on a 16bit data path instead of buffering to 128bit to save latency. 128b/130b coding only inserts 2 additional bits into the data stream and appends 128bit payload. The payload is scrambled data [15], which means the raw data is XORed with a scrambling polynom like PRBS23 in order to increase the number of transitions. The two

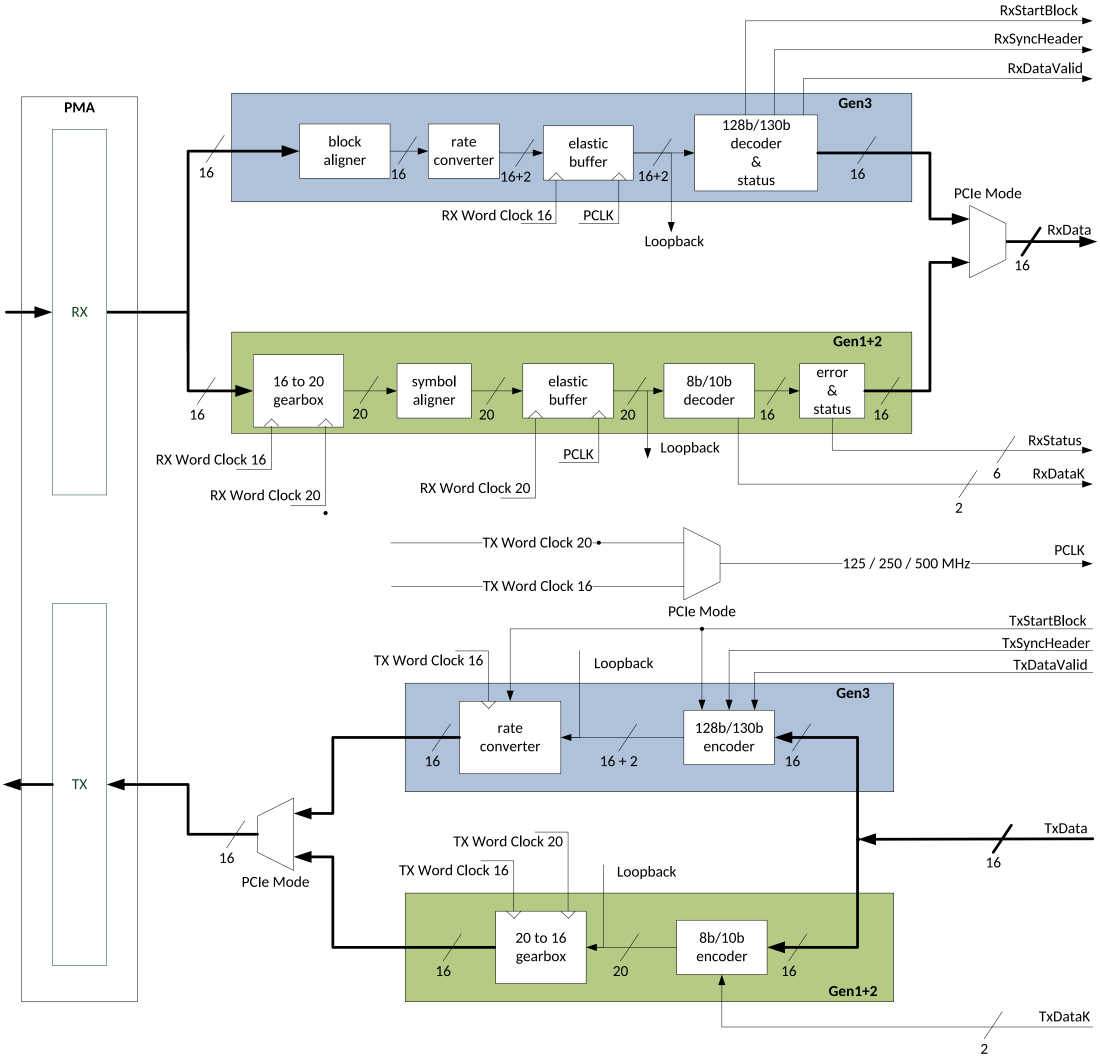


Figure 3.74: PIPE implementation datapath overview

additional bits are used for block alignment and identification of control characters. Though the net data rate is increased compared to 8b/10b coding, DC balance and maximum runlength are not tightly controlled anymore.

The 128b/130b encoder is followed by a rate converter to accommodate the two extra control bits which must be added to the data stream every 8 cycles as part of the 128b/130b encoding. It relies on the previous unit – which is in fact the PCIe MAC - to insert a gap of one clock cycle every eight blocks (after 64 clock cycles) to account for the extra bits being added.

Whereas in PCIe Gen3 modes gaps are inserted by the MAC to account for the coding overhead, this is not the case in Gen1/2 mode. The 20 bit output of the 8b10b encoder has to be supplied to the 16 bit SerDes input without interruptions. This is of course only possible by using two different clocks and passing the data from one clock domain to another in an efficient way. This is facilitated in the 20 to 16 gearbox, which is described separately in a subsequent section.

The top module of the Rx data path contains a 16 to 20 gearbox, block and symbol alignment logic as well as so called elastic buffers. There are also the 8b/10b and 128b/130b decoder modules for Gen1/2 and Gen3.

The 16 to 20 gearbox uses the same scheme as the 20 to 16 gearbox to transform the 16 bit stream from the SerDes Rx data output domain to a 20 bit stream in a slower clk_20 clock domain.

The symbol aligner module is used to find the correct starting point of the 8b/10b symbols in Gen1/2 mode.

The implementation uses a barrel shifter to align the data output to encoded 8b10b symbol boundaries. Instead of using a barrel shifter to change the alignment of the input data to the desired bit position, the bit slip mechanism of the SerDes as described in section 3.5.6 can be used to save latency. Though to keep the implementation generic, a barrel shifter is used. The barrel shifter position is obtained by matching the input data against the 8b/10b COM character (K28.5) in the bit stream which is used for symbol alignment during link initialization. The matching is done in parallel at all possible positions in the 20 bit input word. When the pattern is found, it has to be found on the same position again for the alignment to get locked. If the 8b/10b decoder module which sits at the end of the data path detects too many bit errors at some later point in time during operation, it can request a relock from the symbol aligner.

The block alignment module is needed when operating in Gen3 mode. 128b/130b block boundaries are initially detected by using a special repeating pattern (the so called EIEOS) during the initialization phase of the PCIe link.

The block aligner buffers two symbols to accommodate the 128b/130b data rate difference. Every 8 blocks a gap is inserted into the data stream to the MAC, to buffer a new data word to be able to allow an uninterrupted data stream for the next 8 blocks.

If block lock is lost, this is signaled to the subsequent data path, which is then drained. The loss of block lock is recognized by either the detection of an EIEOS at a different position in the data stream than the current alignment position or a corrupted 128b/130b sync header. Therefore the block alignment logic needs to keep track of the current block start position, which is constantly moving because of the two extra sync header bits every 8 cycles.

3.8.2 Clocking Architecture

As serial links are usually formed from multiple lanes, a complete link contains multiple parallel PCS layers, e.g. up to 16x for PCIe. At the PCS level, all lanes are completely unrelated to each other in terms of data transfer and coding. Synchronization mechanisms are required because a single interface clock to the MAC is used, even though every lane's Tx an Rx has its own word clocks. The clocking architecture to facilitate this is depicted in figure 3.75.

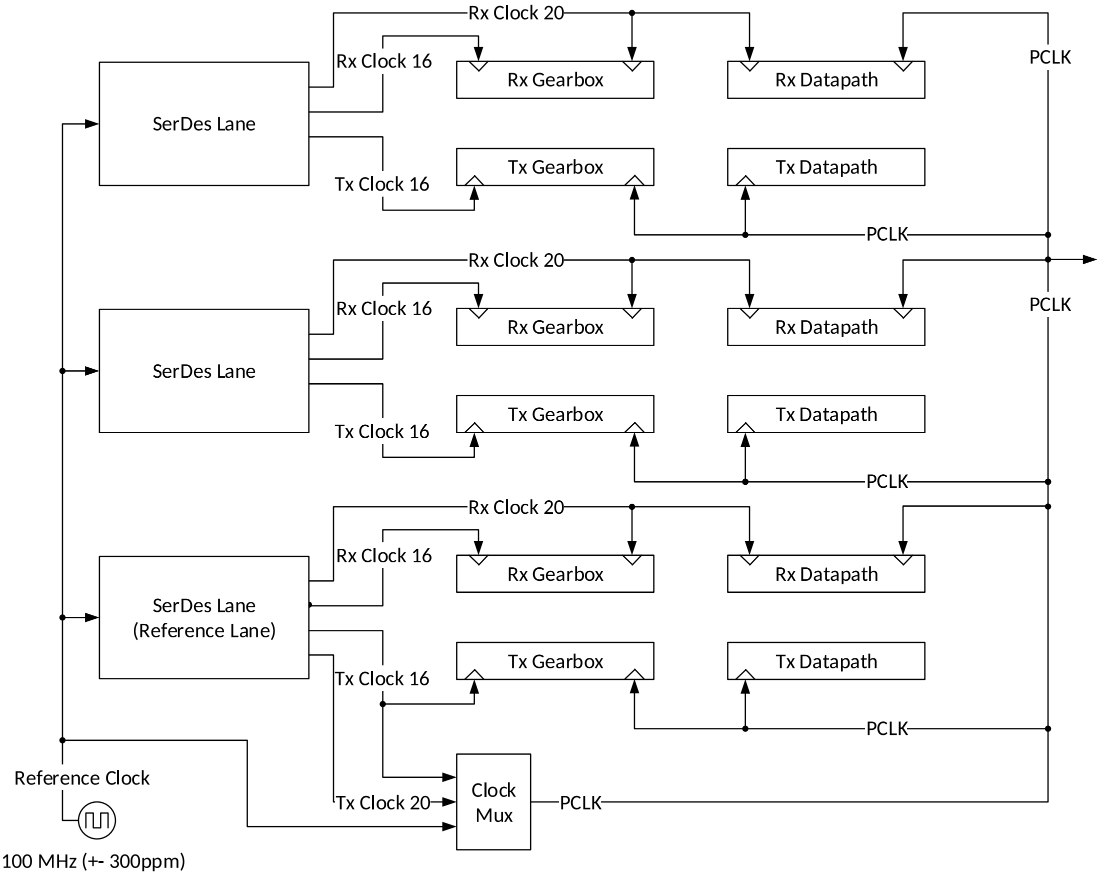


Figure 3.75: PCS clocking overview (3 lane example)

There are a number of clock domain crossings in the design because the PCS logic itself needs to run on a single clock. This single clock, which is used as PCLK is selected to be the Tx word clock of lane 0.

On the Tx side, each SerDes lane has /16 and /20 word clocks, which are all assumed to run at the same frequency, but do not necessarily have the same phase. The /20 word clock is necessary to be able to process the data from the 8b/10b encoder without gaps in Gen1/2 mode. All clock domain crossings on the Tx side are handled in the 20to16 gearbox.

On the Rx side things are more complicated due to the additional clock domain crossing from the Rx clock domain to the PCLK domain. There is the 16to20 gearbox which is equivalent to its 20to16 counterpart in the Tx data path and additionally the elastic buffer, which implements the clock domain crossing from the Rx to the Tx domain to maintain a constant flow of data without gaps by compensating up to ± 300ppm frequency offset between the far end and local reference clock.

3.8.3 Gearbox

There are two different gearboxes present in the PCS. One to transforms the 20bit wide stream from the 8b/10b encoder into a 16bit wide stream which is expected by the SerDes parallel side and the other to transform the 16bit wide receive data to a 20bit wide stream for use in the symbol aligner. Both work on the sample principle, which is described by example of the 20to16 gearbox below.

To work continuously without gaps or overflows, the 20bit stream runs at a slower clock and is transformed to a 16bit stream in a faster clock domain. The clocks are assumed to have a ratio of 4 to 5 and fixed but not priorly known phase relationship. The goal is to facilitate this rate conversion and clock domain crossing with minimum latency.

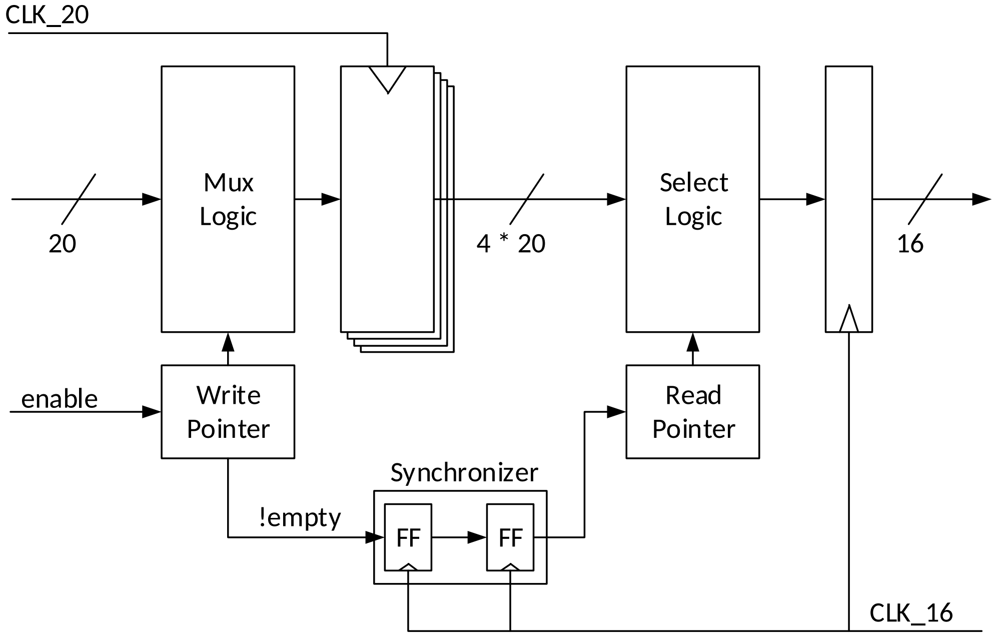


Figure 3.76: Gearbox block diagram

Figure 3.76 shows the block diagram of the 20 to 16 gearbox. The operation is similar to an asynchronous FIFO, but because the phase relation of the two clocks is fixed and the ratio is known, no extra buffer space needs to be allocated and latency can be minimized.

The input data is sampled into a ring buffer on the clk_20 side depending on the write pointer state and is read out from the clk_16 side. The clk_16 starts reading from the ring buffer, after the *!empty* signal was received from the clk_20 side. A selection logic multiplexes the 20bit data stream into 16bit data chunks depending on the read pointer state. This ensures that data, which is read has already settled, thereby preventing metastability and setup violations. The *!empty* is generated from the write pointer, once the gearbox is enabled.

In the following the clock relationship is analyzed closer, in order to determine the minimum achievable latency.

Because of the 4 to 5 ratio of the clocks, one out of four clk_20 cycles is always sampled twice by the clk_16 domain. This is always the last edge right before clk_20 and clk_16 are aligned (or the closest to being in phase). Figure 3.77 shows one possible phase relationship for illustration.

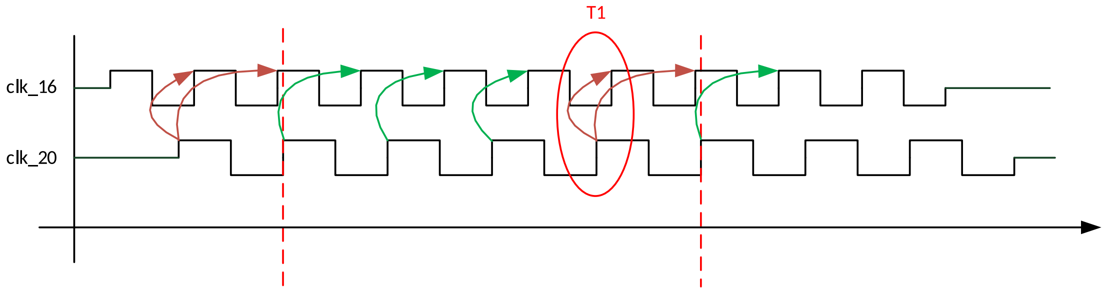


Figure 3.77: Gearbox clocks timing diagram

To get rid of the tough timing requirements, which could lead to setup/hold violations for the first clk_16 edge at T1, a ring buffer is used to transfer data between the two clock domains. In clk_16 only data, which has already settled for one cycle is read. Only one signal which indicates the ring buffer fill grade must then be synchronized from clk_20 to clk_16. This signal is sampled in the clk_16 domain using a synchronizer circuit to prevent possible meta stability.

When sampling only settled data values, timing requirements are greatly relaxed (see figure 3.78). Nevertheless care must be taken that the ring buffer has proper depth. No data should be overwritten and there must always be enough data available to facilitate the gearbox mechanism.

The synchronizer circuit, which is used to transfer the *!empty* signal, introduces

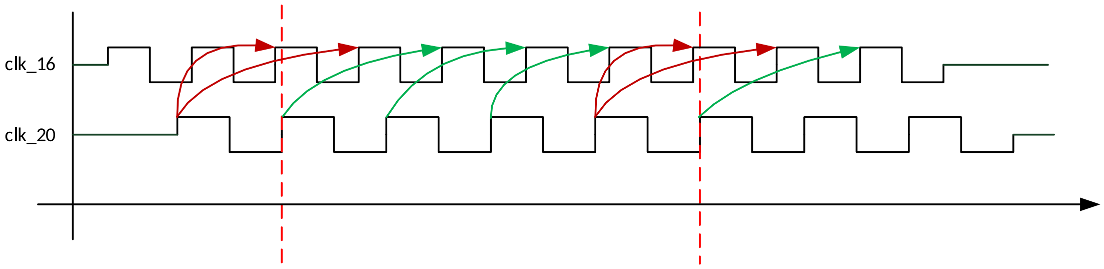


Figure 3.78: Gearbox clocks relaxed timing diagram

a best and worst case scenario. In the worst case scenario the ring buffer fill grade is missed by the first clk_16 edge and not sampled correctly (or the first synchronizer FF goes metastable), so one additional buffer space must be reserved for this case, as seen in figure 3.79.

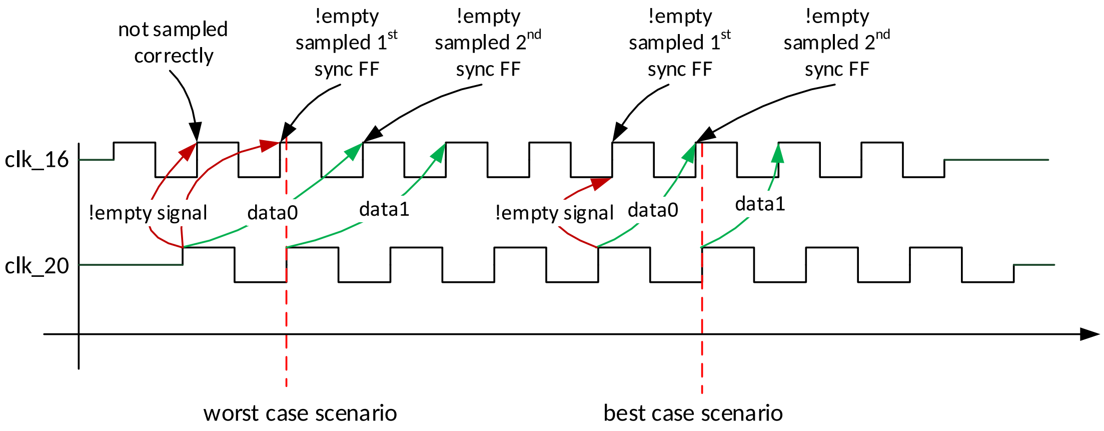


Figure 3.79: PIPE gearbox timing diagram

Because the start of the data transfer has no relation to the current clock phase, it is not know when a clk_20 edge is sampled twice from clk_16. Therefore at least two data samples must be available in the buffer before starting in the clk_16 domain, to allow the gearbox mechanism to work properly. This leads to a best case latency of one clk_20 cycle and a worst case latency of two clk_20 cycles to cross the clock domain.

3.8.4 Elastic Buffer

The elastic buffer has the purpose to compensate a possible frequency difference between remote transmitter and the local PCS clock, which originates from an offset between the reference clocks. The PCIe specification for example allows a reference clock difference of ± 300 ppm between host and device.

If the remote transmitter runs faster than the local PCS clock the buffers in the receiver would eventually overflow and the data would be corrupted. In the opposite case, if the remote transmitter reference is slower than the local reference, the receive buffers in the MAC would eventually underflow. To keep latencies small it is further desirable to buffer as little data as possible.

To compensate such frequency differences special symbols or so called 'ordered sets' are inserted into the data stream by the transmitter in defined intervals. In the case of PCIe these sets consist of multiple symbols and need to be scheduled between 1180 to 1538 symbols [15], to be able to compensate worst case frequency differences. The receiver has to detect these sets and can either add or remove symbols to them in order to prevent a buffer over or under run. Using this mechanism the actual payload data can be received undisturbed. It is up to the transmitter to insert as many ordered sets as necessary to be able to compensate the possible frequency offset. If there is no offset, the number of symbols is not modified by the PCS. In the following the actual implementation of an elastic buffer depicted in figure 3.80, which can be used for PCIe is described.

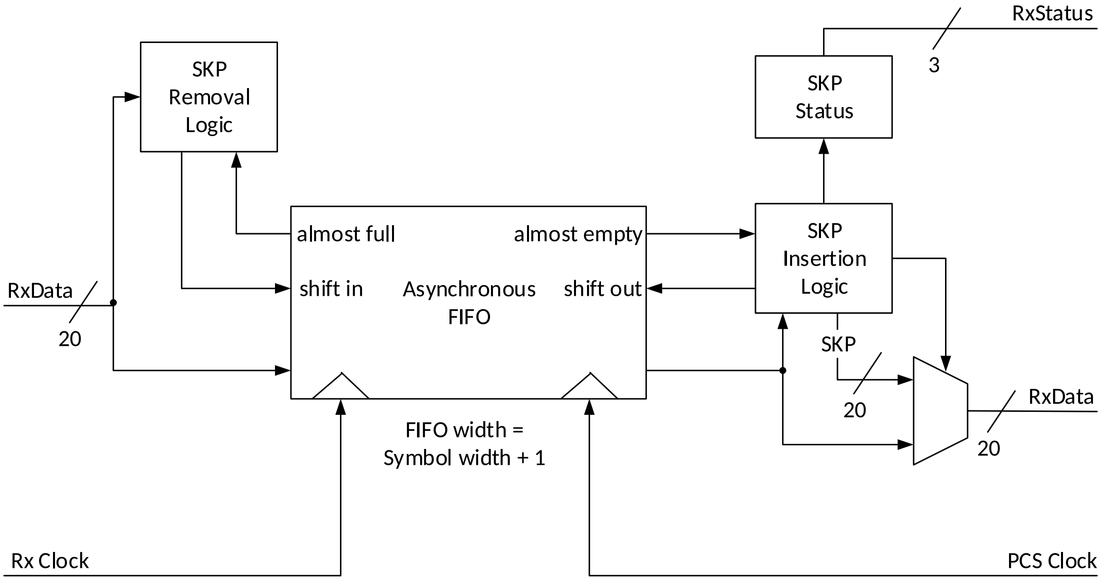


Figure 3.80: Elastic buffer block diagram

The main building block in the elastic buffer is an asynchronous FIFO, which is used for clock domain crossing. Additional control logic is added, to keep the fill grade of the FIFO at a steady level and prevent it from running empty or full. In the case of PCIe the symbols which are used for clock compensation are called skip (SKP) symbols.

To detect the need for SKP insertion/removal, almost full/empty signals of the asynchronous FIFO are used in contrast to the actual FIFO full/empty. The almost full/empty signals can be configured to signal when the FIFO holds only a specific number of entries or has only a specific number of entries left. Hereby it is ensured that there is always either still some data or respectively some buffer space left. This is necessary because the SKP ordered sets are only added in fixed intervals into the data stream by the transmitter independent from the actual frequency offset. The decision to add/remove a SKP symbol must take the maximum number of symbols that may increment/decrement the FIFO fill grade due to the frequency offset in between two SKP sets into account. SKP symbols can be removed on the input side to prevent a FIFO overrun. For this, there is a logic which detects the beginning of a SKP ordered set at the FIFO input side. If the FIFO is almost full, the shift_in is not asserted and the SKP symbol is dropped.

Respectively, SKPs can be added at the output side to prevent a buffer underrun. A logic detects, that there is a SKP ordered set to be shifted out of the FIFO. If the FIFO is almost empty, the shift_out is not asserted and the current SKP symbol in the FIFO is replicated at the elastic buffer output.

Because SKP removal has to be signaled to the MAC, this information also needs to cross the clock domain, synchronous to the SKP ordered set. Therefore the removal of SKPs is signaled through an extra bit in the asynchronous FIFO going from the Rx to the Tx clocked side. This allows keeping track of the exact data point where the SKPs have actually been removed in the data stream over the clock domain crossing.

To save latency, SKP insertion/deletion thresholds (FIFO almost full/empty levels) and the initial fill level are configurable. If the frequency offset between remote and local reference is smaller, the fill level can be reduced, which in turn reduces the latency through the elastic buffer and vice versa.

3.8.5 Figure of Merit Calculation

Modern serial link standards such as PCIe Gen3 require, that the MAC gets information on the current bit error ratio from the PCS. This 'figure of merit' (FOM) is used to indicate if a link operates within its specification, which means if it has a BER better than e.g. 10^{-12}. Only if this is the case, the specific lane or complete link is seen as functional and can be used for data transmission.

During the initialization phase a training sequence is used to determine the optimum equalizer settings. While there are algorithms to adapt DFE and FFE equalizer coefficient values in system, as presented in [78], these optimum equalizer settings usually do not relate to a specific BER.

Though on-die eye monitors, like the one presented in section 3.7, can be used to create eye diagrams and bathtub curves concurrently to the training sequences to assess link quality, there is usually not enough time to complete such exhaustive measurements.

For example the PCIe specification defines that the evaluation of a requested transmitter equalization setting must not take longer than 2 ms (see [15], section 4.2.6.4.2.). In contrast to this, to be able to assume with a confidence of 95% that the BER is better than 10^{-12} at least $3 \cdot 10^{12}$ bits need to be received without an error happening during that time (as derived in [16]). At 8 Gbps it would therefore take at least 375 seconds to measure a BER of 10^{-12}. Therefore, the required FOM can not be measured directly. Under some assumptions, estimations on the current BER can be made using the following technique.

The assumption is, that the jitter at the middle of the eye is totally in the random jitter regime. This is a valid assumption, because if there would be bit errors, which are caused by deterministic effects such as ISI or crosstalk, the link would not be able to work at all and the link initialization would already fail completely.

To simplify the following analysis, a normalized type of the standard bathtub curve, introduced in section 2.7, will prove to be very useful. The so-called Q-Scale [79] uses the inverse error function to normalize a bathtub plot from BER to the standard deviation of a normal distribution. The benefit is, that a random jitter source of Gaussian shape translates to a constant slope of $1/\sigma_{rj}$ on a Q-scale bathtub curve [79], where σ_{rj} is the standard deviation of the random jitter. As the jitter near the center of the eye is defined by random jitter, the bathtub curve tails in the Q-Scale are therefore linear in σ_{rj}, which simplifies the following considerations.

For reference, the Q-function

$$Q(BER) = \sqrt{2} \cdot erf^{-1}(1 - \frac{1}{0.5} \cdot BER) \tag{3.30}$$

is tabulated for common BER figures below:

BER	10^{-4}	10^{-5}	$5 \cdot 10^{-6}$	10^{-6}	10^{-7}	10^{-8}	1^{-9}	10^{-10}	10^{-11}	10^{-12}
Q	3.7	4.3	4.4	4.8	5.2	5.6	6.0	6.4	6.7	7.0

Table 3.7: Q-Scale vs BER values

As the actual jitter present in the system is not known, but the FOM also only needs to reflect if the BER is better than a predefined BER, the worst case jitter, which is allowed as per specification can be assumed. Therefore, for a given worst case σ_{rj} and a defined target bit error rate, the BER can be measured for a lower BER, at a distance of $(q_t - q_o) \cdot \sigma_{rj}$ from the middle of the eye, where q_t is the target BER and q_o is the observed BER, as seen in figure 3.81. If the actual jitter present in the system is better than the maximum value allowed in the specification, then the measurement will be too pessimistic, but the FOM is still valid, as the actual BER is better than the targeted BER.

For the PCIe example above, the BER, which can be measured in 2 ms with a confidence of 95% is about $5.3 \cdot 10^{-6}$, which translates to a Q of 4.4 as per equation 3.30. Further, the worst case random jitter allowed for PCIe Gen3 is 3 ps RMS at the Rx for a stressed eye, according to the specification [15]. Using the Q-scale, the observation point offset t_{FOM} for the FOM can be determined as

$$\begin{aligned} t_{FOM} &= (q_t - q_o) \cdot \sigma_{rj} \\ &= 7 - 4.4 \cdot 3ps \\ &= 7.8ps \end{aligned} \tag{3.31}$$

as seen in figure 3.81.

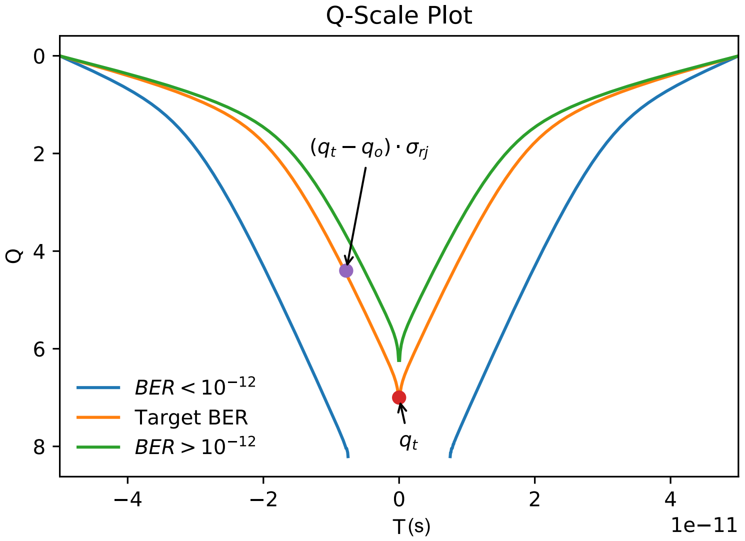


Figure 3.81: Q-Scale bathtub curves for same σ_{rj}, but different deterministic jitter due to ISI

A problem arises, if the eye is dominated by deterministic effects and the random jitter is much lower than the maximum allowed value and the deterministic jitter is in contrast very high. From the observation point it will be assumed, that the target BER is not achieved, while it actually is (see figure 3.82). The estimation would be overly pessimistic. A link might be labeled as not working by the FOM, while it actually achieves the desired BER.

This problem could be solved by complementing this scheme by a circuit, which is able to estimate the current random jitter in the system (like described in [80]) or provide pre-characterized random jitter figures for a known system. This might be possible depending on the application, since the random jitter is independent from the transmission channel.

Another option, which might be feasible in some cases is to increase the measurement times. The smaller the distance between the target BER and the observed BER, the lower the chance to overestimate the BER.

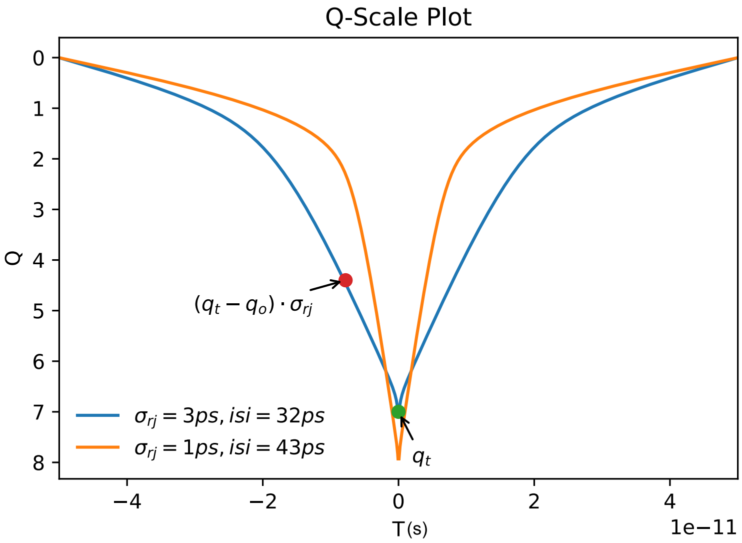


Figure 3.82: Overly pessimistic estimation due to domination of deterministic jitter

The short timeframe of about 2ms allows only a rough estimation of the BER. Assumptions on the deterministic and random jitter magnitudes have to be made in order to pick an observation point. Still, if the channel is not heavily dominated by deterministic jitter, this method can be used to provide a figure of merit for protocols like PCIe that indicates the BER at the center of the eye as an upper bound.

4 Implementation

4.1 Overview

While the previous chapters focused on the design methodology and the SerDes architecture itself, this chapter describes the actual implementation that was done on the basis of these considerations.

The complete SerDes architecture, presented earlier, was modeled top-down to every leaf cell using SystemVerilog real number models, afterwards schematics and layouts of the complete design were created in a 28nm manufacturing process. The SerDes was integrated into a testchip and was eventually taped out to a foundry for manufacturing. These actual implementation tasks were carried out in a team effort, which also proved the effectiveness of the proposed design methodology.

In the following sections, first the physical implementation is reviewed, afterwards simulation results as well as the toplevel simulation setup are described. At last the considerations done for the testchip are discussed.

4.2 Layout Implementation

The floorplan of the physical layout implementation is, from a high-level perspective, dictated by the location of the interfaces. The serial interface pins are located on a bump array and the internal digital interface for the parallel data and all the configuration signals has to be well accessible by other logic on the chip. Further, each lane has a dedicated power supply for the full custom part in order to be able to reduce power supply noise. An example bump configuration for four lanes and a PLL, is depicted in figure 4.1 below.

Because of the connectivity between Rx and Tx such as the loopbacks and common clocking resources, they have to be located next to each other. Further, to keep the aspect ratio reasonable when a larger number of lanes is placed side by side, the serial I/O pins have to be placed vertically to each other, instead of horizontally. The rest of the space, which is necessary for the implementation of

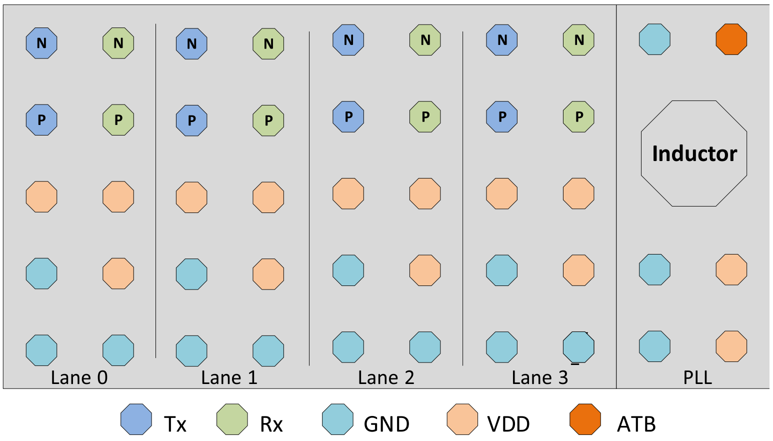


Figure 4.1: Bump layout of four lanes and a common PLL

the SerDes can be used for the dedicated power supply bumps. Overall, one lane has to have a minimum height of 4 bumps to accommodate the differential pair and the power supply. With the utilized technology, this space was not sufficient to implement the receiver, so one additional power bump was added. For the internal interface to be accessible, only the bottom side of the SerDes macro is left.

In the following, the considerations driving the receiver floorplan, which is depicted in figure 4.2 are given.

The Rx floorplan can generally be broken down into the analog frontend, sampling stage, clock generation and digital synthesized logic. The frontend is located directly at the serial input bumps and far away from the clocking resources. The largest individual components are the termination resistors and ESD protection structures, which are placed symmetrically to the input pins.

As the datapath spans from the serial pins to the parallel side, the CTLE stages are spread vertically to transfer the signal down to the sampling stage.

Above the sampling stage, where there is no clocked digital logic, there is room for the common bias generation and the reference level generation for the DFE, which is used by the sampler and DFE stage.

Because of the implemented quarter rate architecture, there are many samplers in the design, which all need to receive the same signal. Therefore the samplers are placed in parallel in order to be able to balance the input signal traces. The sampler layout itself has to have a very stretched portrait aspect ratio, in order to be able to fit all the samplers next to each other. The width of a lane is, as mentioned earlier, determined by the bump pitch so that multiple lanes next to each other align to the bump grid.

Below the sampling stage, the left side contains all the clocking resources, while the datapath continues down to the bottom side of the macro on the right side.

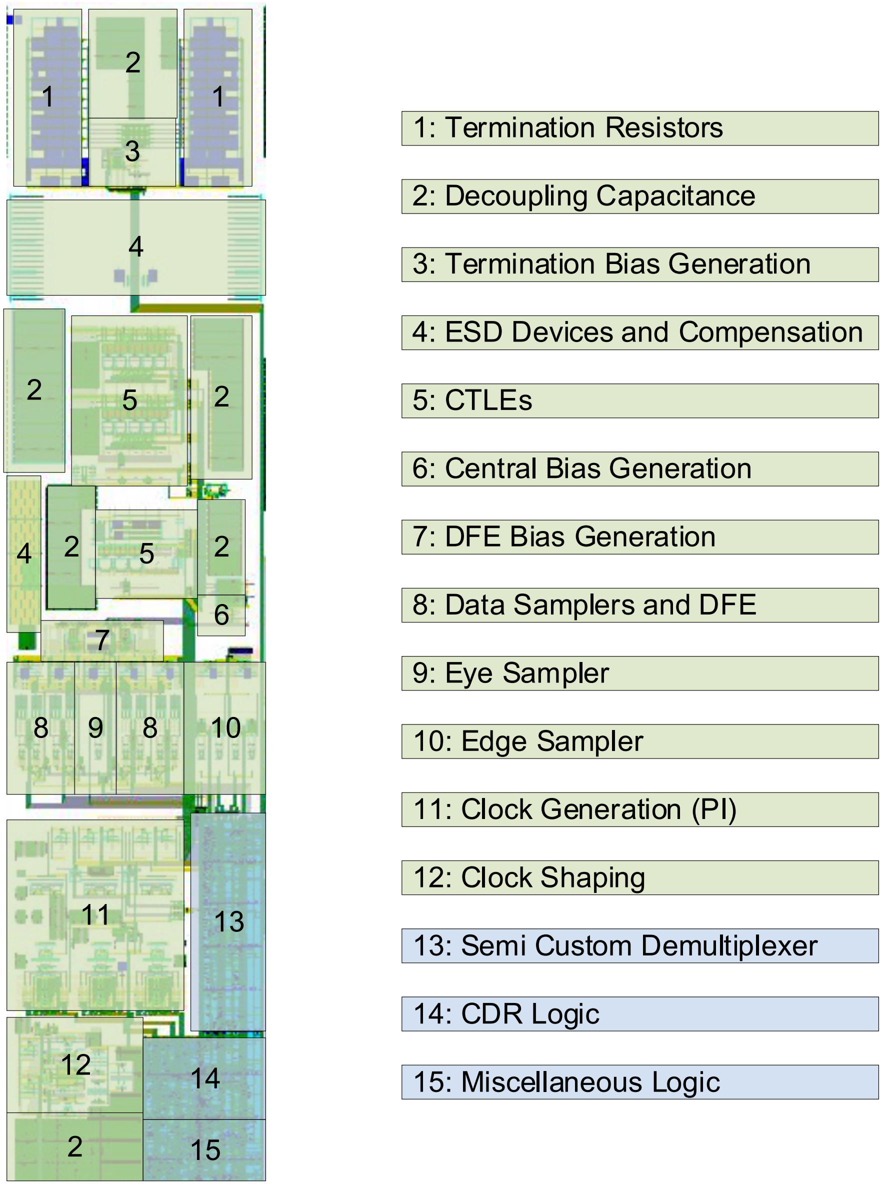


Figure 4.2: Receiver floorplan overview. (Full custom parts in green, Semi custom parts in blue)

The clock generation module below the samplers, includes the clock buffers which are used to supply the different clock phases to the respective sampling stages. The clock buffers themselves are driven by the three phase interpolators, which produce data, edge and eye path clock phases. Below the interpolators is the shaping circuitry, which receives its input clocks from the adjacent transmitter.

The full custom part, which is depicted by the green blocks in figure 4.2, is not a rectangular box, but the lower right edge is cut off to fit in the synthesized digital partition. Below the samplers there is the semi custom demultiplexer and the CDR logic, next to the phase interpolators.

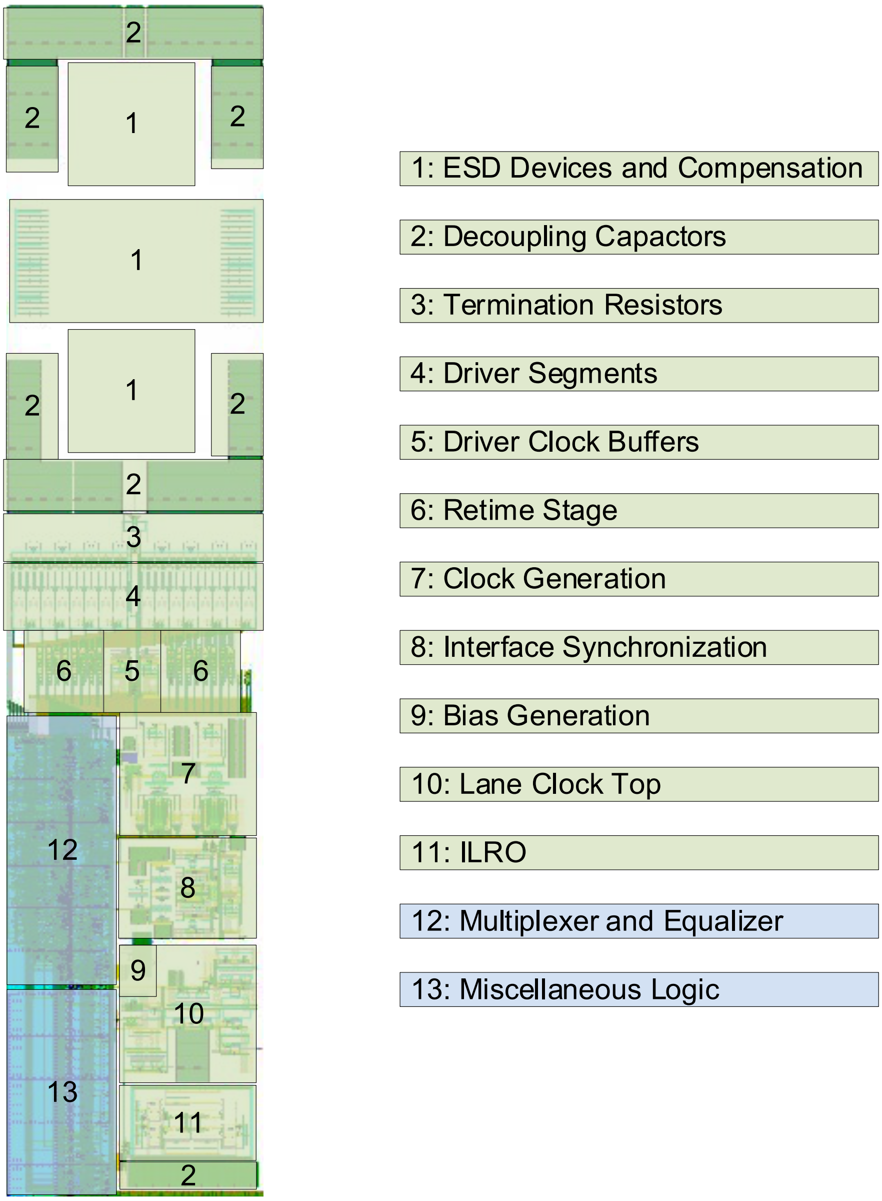


Figure 4.3: Transmitter floorplan overview (Full custom parts in green, Semi custom parts in blue)

The structure of the transmitter is simpler than the one of the receiver and requires less area. Therefore, common clocking resources of the lane clock top module, which contains the ILRO and all the clock multiplexers, are located in the transmitter floorplan.

The constraints for the datapath of the transmitter, shown in figure 4.3 are of course the same as for the receiver. The parallel input is located at the lower side and interfaces the FIR encoder and the semi custom multiplexer path. Next, there is the full custom retiming stage and the actual driver. All the segments of the output driver are put in parallel and therefore have to have a very stretched aspect ratio. Right at the bumps of the differential output, there is the ESD structure along with the compensation structures.

The common clocking resources are on the right side, because they interface with the receiver lane which is abutted there.

All gaps, which are left in the floorplan are used to place additional decoupling capacitance, in order to reduce power supply noise caused by high frequency transient currents.

In figure 4.4 below, a micrograph of a section of the manufactured SerDes testchip is given, which shows the common lane PLL and two adjacent lanes constructed from transmitter and receiver.

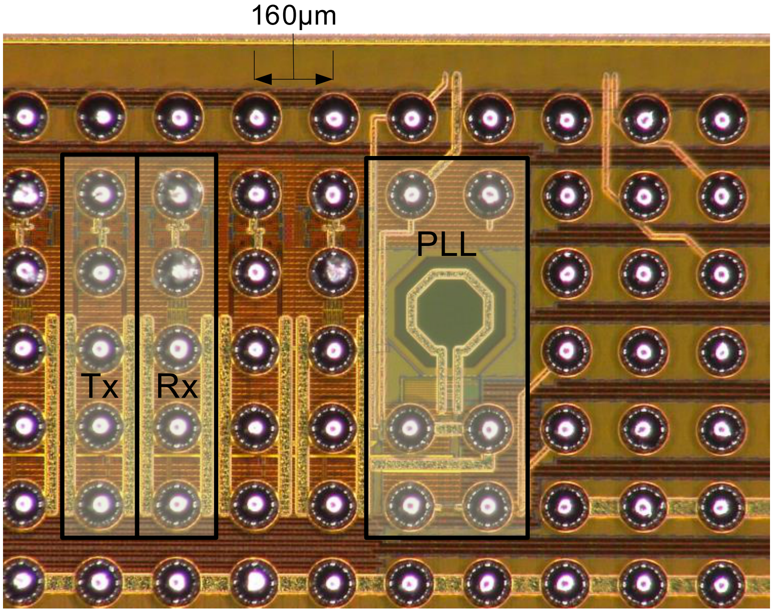


Figure 4.4: Micrograph of two SerDes lanes next to the common lane PLL, with leftmost lane Tx and Rx areas being marked

4.3 Simulations

As per the methodology developed, the whole SerDes system was modeled as a structural SystemVerilog description down to the leaf cells. For the leaf cells plain functional (digital) and real number models, which cover the different simulation scopes were implemented. Transistor level SPICE simulations were carried out for all the leaf cell schematics, which were afterwards developed to verify their performance against the real number models. Additionally, system-level simulations of a complete lane were performed to verify performance and functional correctness.

Because of the size and the different time constants in the design (bit time vs. number of bits required in a simulation) the real number modeling proved to be essential for the verification process. This is especially visible from table 4.1 below:

	Functional	**RNM**	**SPICE**	**Extracted SPICE**
Simulation Time	0.6s	6.3s	5.3h	>24h

Table 4.1: Simulation time (wallclock) for the same SerDes initialization sequence lasting 1µs

To illustrate the speedup of the different models, the same SerDes initialization sequence is simulated on a 16 core Intel Xeon E5-1660 CPU running at 3.20GHz with 64GB RAM, using different leaf cell views in the same structural Verilog hierarchy. While there is of course a degradation of accuracy, huge speed-ups for the different leaf cell models are visible. Depending on the simulation scope it has to be assessed, which kind of accuracy is necessary in order to chose the right model.

For example, when the SerDes is integrated into a larger design, which can contain hundreds of lane instances, and only digital higher level protocol related functions need to be verified, the functional models are sufficient and provide the highest simulations speeds. If e.g. the equalization adaptation mechanisms in a protocol such as PCIe should be verified, the real number models can be used in order to be able to actually verify SerDes performance. Because the real number models already provide a lot of performance related information, time consuming SPICE simulations could be reduced to a minimum. This greatly speeds up the overall design and verification process, especially in the context of larger designs.

Nevertheless some SPICE simulations have to be carried out, also at the scope of a complete lane. From table 4.1 it is visible that simulator runtimes increase dramatically. Though, runtimes of the SPICE simulations depend on their accuracy and vice versa. Every SPICE simulator has several parameters to tweak tolerances and minimum time steps in order to relax accuracy and improve simulation speeds.

To determine, which kind of relaxation is tolerable, the design has to be simulated at highest accuracy first to obtain the 'correct' result. Afterwards, accuracy can be relaxed step by step, as long as the difference of the simulation results is tolerable. The larger the design and the more complex the metrics are to assess the loss of accuracy, it can be very hard to determine which settings are still tolerable. Because the toplevel simulations need to be carried out over PVT corners, a possible speed-up gets multiplied by the number of corners.

Still, by using functional and real number model simulations the number of SPICE simulations can be reduced to a minimum. Because the design has most of its sequential complexity, like control loops and calibration algorithms, implemented in digital logic, pure functional simulations can be used to identify most bugs early in the design cycle.

Extracted SPICE simulations on the toplevel, which also include parasitics were performed to gain more confidence in the design.

Schematic level simulations are mainly necessary to ensure correct timings at the custom/ semi-custom boundary and identify setup/hold problems, to check correct biasing over corners, and to predict power consumption. Only relatively short simulation times are required for this.

Real performance estimations, such as BER predictions are not possible using SPICE simulations. These performance figures are obtained from post processing and analysis as described in [18].

To ensure consistency and keep the maintenance effort for different test scenarios reasonable, the same simulation setup, which is depicted in figure 4.5, is reused for all toplevel tests.

Rx and Tx of a single lane are connected in an external loopback through a channel model. PRBS generator and checker modules are used at the parallel side to create and check the traffic.

The actual test procedure is coded in the mgt_test module, which is specific to the individual tests. This concept to run different tests with same testbench is inspired by the unified verification methodology (UVM) [2].

This single testbench can be used to either run tests from command line, using functional and real number leaf cells, as well as to run tests from inside of Ca-

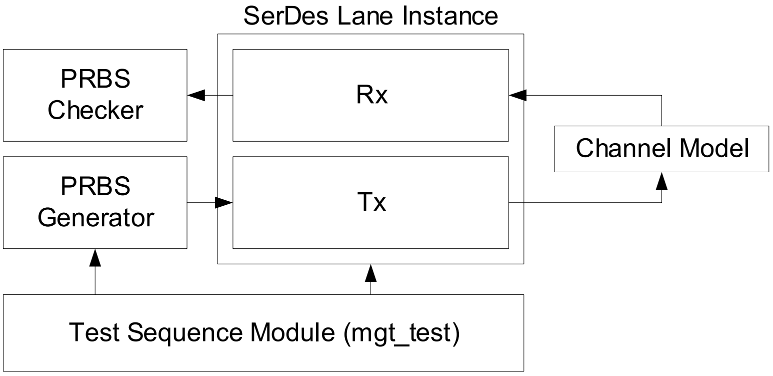


Figure 4.5: Reusable testbench setup

dence Virtuoso using schematic or extracted leaf cells over different PVT corners. Test scenarios can be developed and debugged with fast functional or RNM simulations before finally deploying them to schematic simulations, to improve test development times.

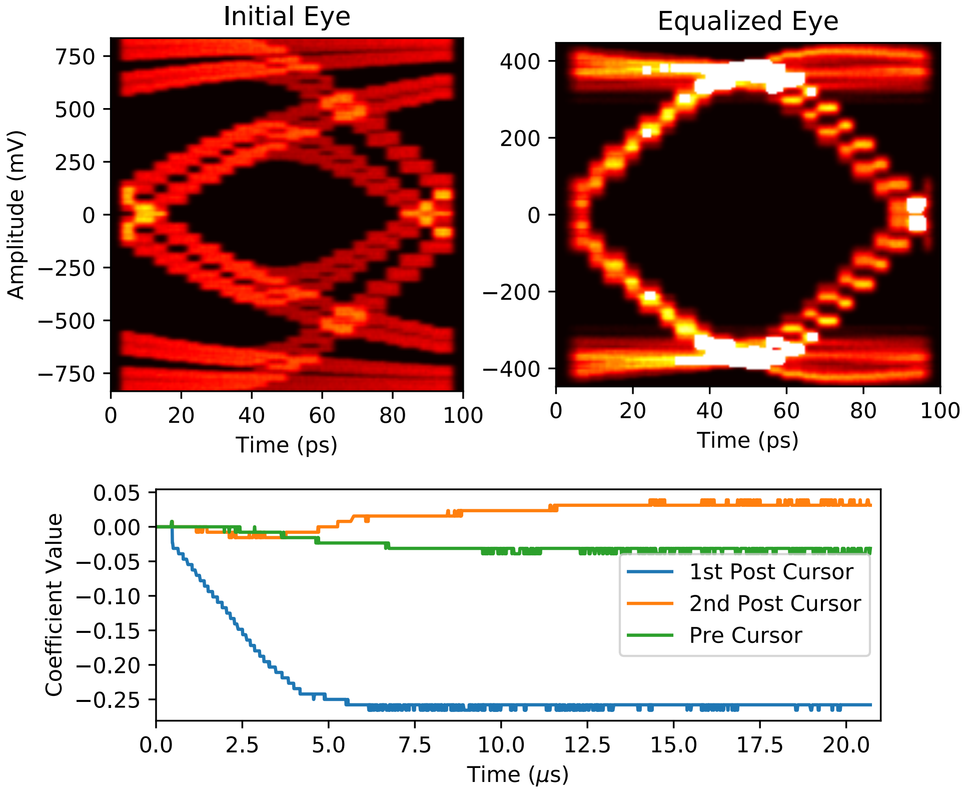


Figure 4.6: Eye diagrams obtained from RNM simulations before and after equalizer coefficient adaptation using the built-in hardware adaptation

With this setup complex scenarios can be simulated. As an example, figure 4.6 shows pre- and post equalized eye diagrams, which were obtained from RNM simulations. The equalizer coefficients of the transmitter were adapted in the simulation, using the built-in hardware logic. Because of the long runtime requirements, such simulations can not be carried out with SPICE models in reasonable time frames.

4.4 Testchip

A testchip, which included the SerDes design was planned, implemented and manufactured. The main goal was to verify the full custom mixed-signal part of the SerDes in silicon against the simulation results to assess their accuracy and correctness.

A block diagram of the testchip is given in figure 4.7.

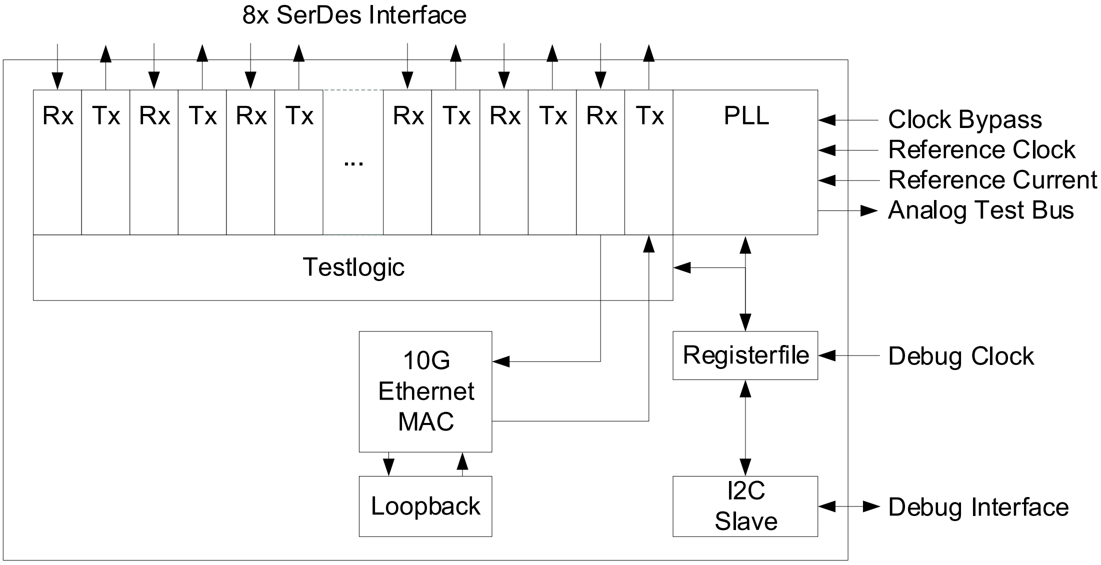


Figure 4.7: Testchip blockdiagram

Though the SerDes performance is simulated over different process corners and operating conditions with respective channel models, there still exists an uncertainty whether the projected electrical performance is really achieved in silicon. Therefore a characterization of performance parameters such as jitter, signal amplitude and bit error ratio (BER) on real silicon is necessary to validate the design. Additionally, the interaction of the SerDes together with a low latency 10G Ethernet MAC and commercial hardware, such as 10G Ethernet devices, should be tested.

Careful planning of the testchip is necessary, in order to be able to perform the measurements, which are necessary to characterize the design afterwards.

Measurements, which are simple to carry out in simulation environments might be difficult to accomplish in silicon, or require special hardware structures, which need to be implemented in advance. All the test structures, which were described in section 3.7, have been implemented in the SerDes PHY and can be used to validate simulation results against measurements.

10G Ethernet functionality is intended to be tested with the MAC acting as a far end protocol loopback. This means another external Ethernet device is used to create traffic, which is received and decoded by the testchip MAC. This decoded traffic is then sent back to the device through the testchip transmitter lane. By this, the interaction of MAC and PHY with each other as well as with other devices is tested in a straight forward way.

To accomplish these tasks, the testchip includes:

- 8 SerDes transceiver lanes capable of 2.5, 5, 8, 10.3125, 16 and 20 Gbps
- Common lane LC ADPLL
- 10G Ethernet low latency MAC

As well as the following hardware structures for control, test and characterization:

- Control and status registerfile
- I2C debug interface
- Microcode engine for PHY configuration and calibration tasks

The testchip was taped out and manufactured in a 28nm silicon process. A micrograph of a die is given in figure 4.8.

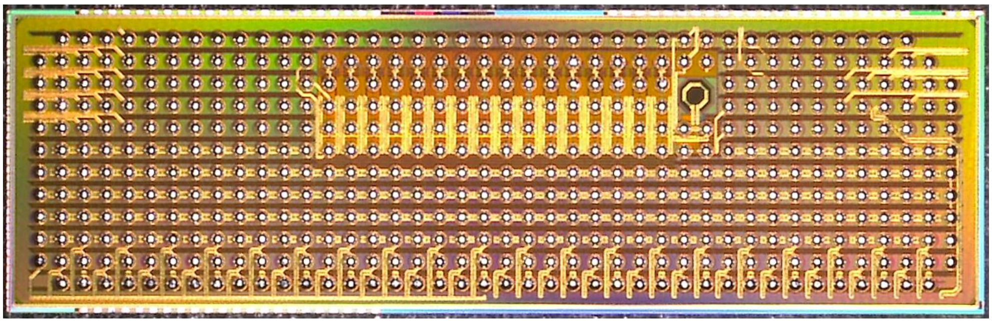


Figure 4.8: Micrograph of the complete testchip

In the middle of the die, the eight SerDes lanes to the left of the PLL are visible. The PLL can be identified by the inductor of the LC-oscillator.

4.5 Lessons Learned

This section summarizes lessons learned during the design and implementation process and states ideas on how the process can be improved in the future.

When looking at the durations of individual tasks throughout the overall design process in the entire project, it becomes clear that a lot of time was spent in the layout phase. As the whole methodology was originally targeting simulation, verification and schematic design, this suggests that more automation in the layout phase is necessary. Though the design is very modular and many leaf cells are reused, each leaf cell layout itself has many recurring structures, such as differential pairs or current sources, which had to be manually created over and over again with only minor modifications. As this is a repetitive task which can be automated, this needs to be addressed better in a future project.

A possibility would be to identify recurring primitives and create parameterized cells (PCells) for their layouts. PCells are coded in the SKILL programming language and allow geometry such as number of transistor fingers, length or width to be controlled by parameters on their instantiation. Leaf cell schematics and layouts can be constructed from these primitive PCells instead of single transistors. The PCell implementation will require additional time, but if the cells can be reused often throughout the design, this time will be well spent. It will also ease porting to other technologies, as a major part of the layout is already ported once the primitive PCells have been implemented in the new technology. The goal should be to break the leaf cells down to a small number of primitives, instead of trying to build complete layout generators on the granularity of whole DACs or operational amplifiers, which is a very complex task.

As the layout implementation happened in a team effort, common design constraints/standards were established initially to allow integration of the different layouts on the toplevel. These constraints for example defined the pin layers to be used on different cell levels, maximum number of metal layers on cell levels, power rail locations, minimum metal widths for different types of signals and so on. Still, as the layout allows many degrees of freedom, which can eventually complicate integration, the issue of common design standards also needs to be addressed in the future.

One attempt to reduce these degrees of freedom and harmonize different cell designs is to introduce additional grids for cell outlines and signal lines. These grids should be much more coarse than the manufacturing grid. This approach is very similar to the concept of routing tracks, which is commonly used in semi custom design. The grids can have different spacings for different layers, such as coarser grids on higher metal layers to account for different manufacturing rules. The devices, such as transistors, do not have to be on these grids, as the transistor geometries of different width/length gates used throughout the design normally do not align very well to a common grid. The toplevel block integration will be much easier, if every block has its pins, outline and power on common grids, rather than only on predefined layers.

Another important topic in layout that needs to be addressed better in the future, is the requirement for uniform density. Advanced node technologies of 28nm and beyond have extremely strict density requirements for all the different layers in order to enforce uniform pattern density which is required for manufacturing.

If density requirements are only considered after the layout is completely assembled at the top level, problems can be extremely hard to fix. For example, excessive chaining of many transistor fingers on the same diffusion to improve matching, might be undesirable, because it can create areas of high poly silicon density, which can not be fixed on the toplevel.

Therefore, density requirements need to be taken into account on leaf cell level. If every leaf cell already has uniform density, a design which is constructed from leaf cells, which align to a common grid, will also have a uniform density. This would require the designer to already include shapes, which only serve density requirements at design time and check density at leaf cell level. If primitive PCells are used to construct the leaf cells, these PCells can also include density related shapes and generate them automatically. This will also lead to a better parasitic estimation at leaf cell level, but might increase simulation times for extracted SPICE simulations.

In conclusion, it can be said, that the layout in advanced nodes is much more driven from a manufacturing requirements point of view than by the designer, which removes many degrees of freedom and may hereby eventually lead to easier integration.

5 Conclusion and Outlook

High-speed serial I/O is a key technology, necessary to build state of the art, highly integrated systems on chips. Without SerDes technology the degree of integration and processing power that is achieved in modern ASICs can not be fully exploited.

Though, the design of a SerDes for data rates up to 25 Gbps, that fits the multitude of requirements is a very challenging task. This is manifested by the fact that such designs are only commercially available from a handful of companies.

In this thesis a mixed signal design methodology was introduced, which addresses both implementation and verification of such complex systems. It leverages real number modeling as a key element to overcome the consistency gap between model and implementation. A tool flow, which supports the designer was developed to complement state of the art EDA software to efficiently apply this methodology in practice.

The analysis of todays high-speed serial link architectures lead to a design, which moves most of its complexity into digital semi custom logic. This paves the way to fully leverage the advantages of advanced node semiconductor manufacturing.

The developed high-speed SerDes architecture was analyzed and verified using the methodology introduced earlier. The SerDes design was implemented on a testchip, taped out and manufactured in a 28nm silicon process.

The contribution of this work is a new mixed signal design methodology, which tightly couples model and implementation and hereby enforces consistency throughout the design flow. The methodology was used in practice to implement a complex high-speed SerDes with a small design team and proved to be indispensable for future developments.

Further, architectural improvements to high-speed SerDes implementations, such as an innovative divider initialization and bit slip mechanism for word alignment were proposed.

For the future, improvements to both the SerDes architecture, as well as the design methodology are planned.

As a next step, the SerDes testchip, which was manufactured, will be characterized in order to determine the accuracy of the real number model based performance prediction, which was priorly confirmed through circuit simulations.

The methodology will be extended to cover the layout process, as discussed in section 4.5. With the usage of PCells for primitive building blocks such as current sources and differential pairs, the layout reuse will be improved to accelerate the implementation process in the future. This will also improve portability to other process nodes.

Further, a schematic generator approach for frequently used elements such as amplifiers or buffers will be integrated into the schematic generation flow to further assist the designer with already initially dimensioned schematic templates for leaf cell development. As the high level specifications for the leaf cells are already derived through the hierarchy in the current methodology, these can be used as inputs to generator scripts, which then in turn compute initial transistor geometries for a given technology based on predefined circuit topologies and technology specific g_m/I_d tables.

Concerning the SerDes architecture, two major issues, namely robustness and power efficiency need to be further investigated. Means to efficiently remove PVT mismatch of the Rx and Tx termination resistors in order to improve impedance matching need to be integrated. Further, the implementation of an in-situ calibration method for the Tx quarter rate clocking structure, similar to the mechanism in the receiver should be investigated. This could reduce duty cycle distortion and deterministic jitter effects, hereby improving the link timing margin. Finally, the automatic adaptation of CDR loop filter coefficients to the current jitter profile is an important topic that needs to be addressed.

In order to improve power efficiency, which is one of the major challenges for future high-speed serial links, the use of CMOS logic, primarily in the clock distribution, should be explored.

List of Figures

Bibliography

[1] Nick Heaton. White paper: Maximizing Verification Effectiveness Using MDV. Technical report, Cadence Design Systems, Inc.

[2] IEEE Standard for Universal Verification Methodology Language Reference Manual. *IEEE Std 1800.2-2017*, pages 1–472, May 2017.

[3] Walter Hartong. White paper: Plan-Based Analog Verification Methodology. Technical report, Cadence Design Systems, Inc.

[4] IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. *IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009)*, pages 1–1315, Feb 2013.

[5] B. Murmann. A/D converter trends: Power dissipation, scaling and digitally assisted architectures. In *2008 IEEE Custom Integrated Circuits Conference*, pages 105–112, Sept 2008.

[6] Yaron Kashai Neyaz Khan. From Spec to Verification Closure: a case study of applying UVM-MS for first pass success to a complex Mixed-Signal SoC design. Technical report, Maxim Integrated Products.

[7] Prabal Bhattacharya Donald O'Riordan. PSL/SVA Assertions in SPICE. Technical report, Cadence Design Systems, Inc.

[8] Ken Kundert. Top-Down Design and Verification of Mixed-Signal Circuits. *EE Times*, Jun 2005.

[9] Juergen Scheible and Jens Lienig. Automation of Analog IC Layout: Challenges and Solutions. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, pages 33–40, New York, NY, USA, 2015. ACM.

[10] J. Crossley, A. Puggelli, H. P. Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. J. An, A. L. Sangiovanni-Vincentelli, and E. Alon. BAG: A designer-oriented integrated framework for the development of AMS circuit generators. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 74–81, Nov 2013.

[11] B. Prautsch, U. Eichler, S. Rao, B. Zeugmann, A. Puppala, T. Reich, and J. Lienig. IIP framework: A tool for reuse-centric analog circuit design. In *2016 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 1–4, June 2016.

[12] D. Marolt, M. Greif, J. Scheible, and G. Jerke. PCDS: A new approach for the development of circuit generators in analog IC design. In *22nd Austrian Workshop on Microelectronics (Austrochip)*, pages 1–6, Oct 2014.

[13] Pete Hardee Sathishkumar Balasubramanian. Whitepaper: Solutions for Mixed-Signal SoC Verification Using Real Number Models. Technical report, Cadence Design Systems, Inc.

[14] IEEE Standard for Verilog Hardware Description Language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pages 1–560, 2006.

[15] PCI-SIG. *PCI Express Base Specification*, November 2010. Revision 3.0.

[16] McHugh Russ Müller Marcus, Stephens Ransom. Total Jitter Measurement at Low Probability Levels, Using Optimized BERT Scan Method. In *DesignCon 2005*, 2005.

[17] Norbert Fliege and Markus Gaida. *Signale und Systeme - Grundlagen und Anwendungen mit MATLAB ; mit 8 Tabellen und 38 MATLAB-Projekten.* Schlembach Fachverlag, Wilburgstetten, 1. aufl. edition, 2008.

[18] Maximilian Thürmer. *Modelling and performance analysis of multigigabit serial interconnects using real number based analog verification methods.* PhD thesis, Heidelberg University, 2017.

[19] Maxim Integrated. *Converting between RMS and Peak-to-Peak Jitter at a Specified BER*, 2008.

[20] Seasim. *Release 0.54*. PCI-SIG, 2015.

[21] B. K. Casper, M. Haycock, and R. Mooney. An accurate and efficient analysis method for multi-Gb/s chip-to-chip signaling schemes. In *2002 Symposium on VLSI Circuits. Digest of Technical Papers (Cat. No.02CH37302)*, pages 54–57, June 2002.

[22] Sam Palermo. ECEN689: Special Topics in High-Speed Links Circuits and Systems, Lecture 22: ISI and Random Noise, Spring 2010.

[23] Thomas Toifl. Low-power High-Speed CMOS I/Os: Design Challenges and Solutions. Sept 2012.

[24] J. Kim, A. Balankutty, A. Elshazly, Y. Y. Huang, H. Song, K. Yu, and F. O'Mahony. 3.5 A 16-to-40Gb/s quarter-rate NRZ/PAM4 dual-mode transmitter in 14nm CMOS. In *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, pages 1–3, Feb 2015.

[25] S. Gondi and B. Razavi. Equalization and Clock and Data Recovery Techniques for 10-Gb/s CMOS Serial-Link Receivers. *IEEE Journal of Solid-State Circuits*, 42(9):1999–2011, Sept 2007.

[26] T. Kawamoto, T. Norimatsu, K. Kogo, F. Yuki, N. Nakajima, M. Tsuge, T. Usugi, T. Hokari, H. Koba, T. Komori, J. Nasu, T. Kawamata, Y. Ito, S. Umai, J. Kumazawa, H. Kurahashi, T. Muto, T. Yamashita, M. Hasegawa, and K. Higeta. 3.2 multi-standard 185fsrms 0.3-to-28Gb/s 40dB backplane signal conditioner with adaptive pattern-match 36-Tap DFE and data-rate-adjustment PLL in 28nm CMOS. In *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, pages 1–3, Feb 2015.

[27] Stefan Kosnac. Design-Aspects of a Decision Feedback Equalizer in a 28 nm Technology. Master thesis, University of Heidelberg, 2016.

[28] Dean Banerjee. *PLL Performance, Simulation, and Design.* Fourth edition edition, 2006.

[29] W. Li and J. Meiners. Introduction to phase-locked loop system modeling, 2005.

[30] N. D. Dalt. Markov Chains-Based Derivation of the Phase Detector Gain in Bang-Bang PLLs. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53(11):1195–1199, Nov 2006.

[31] Jri Lee, K. S. Kundert, and B. Razavi. Analysis and modeling of bang-bang clock and data recovery circuits. *IEEE Journal of Solid-State Circuits*, 39(9):1571–1580, Sept 2004.

[32] Nicola Da Dalt. *Theory and Implementation of Digital Bang-Bang Frequency Synthesizers for High Speed Serial Data Communications.* PhD thesis, RWTH Aachen, Feb 2007.

[33] M. Zanuso, D. Tasca, S. Levantino, A. Donadel, C. Samori, and A. L. Lacaita. Noise Analysis and Minimization in Bang-Bang Digital PLLs. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 56(11):835–839, Nov 2009.

[34] Robert Bogdan Staszweski. *Digital Deep-Submicron CMOS Frequency Synthesis for RF Wireless Applications.* PhD thesis, University of Texas at Dallas, August 2002.

[35] G. Marucci, S. Levantino, P. Maffezzoni, and C. Samori. Analysis and Design of Low-Jitter Digital Bang-Bang Phase-Locked Loops. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(1):26–36, Jan 2014.

[36] Ming-Ta Hsieh and Gerald E Sobelman. Comparison of LC and Ring VCOs for PLLs in a 90 nm Digital CMOS Process. 01 2006.

[37] M. M. Mansour and M. M. Mansour. On the design of low phase-noise CMOS LC-tank oscillators. In *2008 International Conference on Microelectronics*, pages 407–412, Dec 2008.

[38] P. Kinget, B. Soltanian, S. Xu, S. a. Yu, and F. Zhang. Advanced Design Techniques for Integrated Voltage Controlled LC Oscillators. In *2007 IEEE Custom Integrated Circuits Conference*, pages 805–811, Sept 2007.

[39] R. B. Staszewski, Chih-Ming Hung, D. Leipold, and P. T. Balsara. A first multigigahertz digitally controlled oscillator for wireless applications. *IEEE Transactions on Microwave Theory and Techniques*, 51(11):2154–2164, Nov 2003.

[40] B. Sadhu and R. Harjani. Capacitor bank design for wide tuning range LC VCOs: 850MHz-7.1GHz (157In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1975–1978, May 2010.

[41] Ching-Yuan Yang. A High-frequency CMOS Multi-modulus Divider for PLL Frequency Synthesizers. *Analog Integr. Circuits Signal Process.*, 55(2):155–162, May 2008.

[42] Silicon Labs. *AN118:Improving ADC resolution by oversampling and averaging.*

[43] Juan Jesus Ocampo Hidalgo. *System and Circuit Approaches for the Design of Multi-mode Sigma-Delta Modulators with Application for Multi-standard Wireless Receivers.* Dissertation, TU Darmstadt, 2004.

[44] J. Zhuang, K. Waheed, and R. B. Staszewski. Design of Spur-Free Sigma Delta Frequency Tuning Interface for Digitally Controlled Oscillators. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(1):46–50, Jan 2015.

[45] Kangmin Hu, Tao Jiang, and P. Chiang. Comparison of on-die global clock

distribution methods for parallel serial links. In *2009 IEEE International Symposium on Circuits and Systems*, pages 1843–1846, May 2009.

[46] Tobias Markus. High-Speed Clock Generation Architecture for a Multi-Rate SerDes in 28 nm. Master thesis, University of Heidelberg, 2015.

[47] Stefan Kosnac. Design of a T-Coil Termination and an Electrostatic Discharge Structure in 28 nm. Project report, University of Heidelberg, 2015.

[48] H. Camara and S.V. Rylov. Conditioning input buffer for clock interpolation, 2010. US Patent 7,659,763.

[49] Stephan Walter. On-Chip Measurement Units for High-Speed Signals. Diploma thesis, University of Heidelberg, 2015.

[50] Diarmuid Collins. A Fully Differential Phase-Locked Loop With Reduced Loop Bandwidth Variation. Master thesis, National University of Ireland, Maynooth, 2011.

[51] S.V. Rylov. Phase shifting using asymmetric interpolator weights, Aug 2009. US Patent App. 12/024,043.

[52] J.L. Sonntag and J. Stonick. A Digital Clock and Data Recovery Architecture for Multi-Gigabit/s Binary Links. *Solid-State Circuits, IEEE Journal of*, 41(8):1867–1875, Aug 2006.

[53] Jri Lee, K. S. Kundert, and B. Razavi. Analysis and modeling of bang-bang clock and data recovery circuits. *IEEE Journal of Solid-State Circuits*, 39(9):1571–1580, Sept 2004.

[54] Richard C. Walker. *Designing BangBang PLLs for Clock and Data Recovery in Serial Data Transmission Systems*, pages 34–45. Wiley-IEEE Press, 2003.

[55] Myeong-Jae Park and Jaeha Kim. Pseudo-Linear Analysis of Bang-Bang Controlled Timing Circuits. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 60(6):1381–1394, June 2013.

[56] K. Mueller and M. Muller. Timing Recovery in Digital Synchronous Data Receivers. *IEEE Transactions on Communications*, 24(5):516–531, May 1976.

[57] Tobias Markus. Verilog-AMS Model of a Mueller Mueller CDR. Project report, University of Heidelberg, 2015.

[58] Maxim Integrated. *AN3455:Spectral content of NRZ test patterns*, 2004.

[59] S. Erba, M. Pozzoni, M. Pisati, R. Brama, D. Sanzogni, E. Depaoli, P. Viola, and F. Svelto. A 10Gb/s receiver with linear backplane equalization and

mixer-based self-aligned CDR. In *2008 IEEE Custom Integrated Circuits Conference*, pages 559–562, Sept 2008.

[60] Yehui Sun and Hui Wang. Analysis of digital bang-bang clock and data recovery for multi-gigabit/s serial transceivers. In *Custom Integrated Circuits Conference, 2009. CICC '09. IEEE*, pages 343–346, Sept 2009.

[61] A. Zargaran-Yazd and W.T. Beyene. Discrete-time modeling and simulation considerations for high-speed serial links. In *Electrical Performance of Electronic Packaging and Systems (EPEPS), 2014 IEEE 23rd Conference on*, pages 165–168, Oct 2014.

[62] Chao He and T. Kwasniewski. Bang-Bang CDR's acquisition, locking, and jitter tolerance. In *Electrical Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*, pages 1–4, April 2012.

[63] Kundert Ken. Verification of Bit-Error Rate in Bang-Bang Clock and Data Recovery Circuits. *The Designers Guide Community*, 2010.

[64] Sven Schenk. Architecture Analysis of Multi-Gigabit-Transceivers for Low Latency Communication. Diploma thesis, Mannheim University, 2008.

[65] Xilinx. UltraScale GTY Transceiver: TX and RX Latency Values.

[66] Raffaele Giordano, Vincenzo Izzo, and Alberto Aloisio. High-Speed Deterministic-Latency Serial IO. In George Dekoulis, editor, *Field - Programmable Gate Array*, chapter 11. InTech, Rijeka, 2017.

[67] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance CMOS variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4.5):433–449, July 2006.

[68] Sam Palermo. ECEN689: Special Topics in High-Speed Links Circuits and Systems Spring 2012, Lecture 15: RX Circuits, Spring 2012.

[69] M. Mansuri, B. Casper, and F. O'Mahony. An on-die all-digital delay measurement circuit with 250fs accuracy. In *2012 Symposium on VLSI Circuits (VLSIC)*, pages 98–99, June 2012.

[70] A. A. Hafez, M. S. Chen, and C. K. K. Yang. A 32-to-48Gb/s serializing transmitter using multiphase sampling in 65nm CMOS. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 38–39, Feb 2013.

[71] TCL based extensible Register File Generator. http://unihd-cag.github.io/odfi-rfg/. Accessed: 2017-09-29.

[72] V. A. Zivkovic, F. v. d. Heyden, G. Gronthoud, and F. d. Jong. Analog Test Bus Infrastructure for RF/AMS Modules in Core-Based Design. In *2008 13th European Test Symposium*, pages 27–32, May 2008.

[73] M. Franco, J. Güiza, E. Chiappetta, S. Rueda, H. Luis, J. Bertuzzo, J. Koeppe, T. Robins, J. Jenkins, and T. Hamilton. Electronically programmable test points for on-chip analog/digital measurements. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 2670–2673, May 2013.

[74] IEEE Standard Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2001*, 2001.

[75] IEEE Standard for Boundary-Scan Testing of Advanced Digital Networks. *IEEE Std 1149.6-2003*, pages 0–132, 2003.

[76] Intel Corporation. *PHY Interface for the PCI Express Architecture, PCI Express 3.0, Revision 0.5.*

[77] A. X. Widmer and P. A. Franaszek. A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code. *IBM Journal of Research and Development*, 27(5):440–451, Sept 1983.

[78] V. Stojanovic, A. Ho, B. W. Garlepp, F. Chen, J. Wei, G. Tsang, E. Alon, R. T. Kollipara, C. W. Werner, J. L. Zerbe, and M. A. Horowitz. Autonomous dual-mode (PAM2/4) serial link transceiver with adaptive equalization and data recovery. *IEEE Journal of Solid-State Circuits*, 40(4):1012–1026, April 2005.

[79] Inc. Agilent Technologies. Jitter analysis: The dual dirac model, RJ/DJ, and Q-scale. December 2004.

[80] J. Liang, M. S. Jalali, A. Sheikholeslami, M. Kibune, and H. Tamura. On-Chip Measurement of Clock and Data Jitter With Sub-Picosecond Accuracy for 10 Gb/s Multilane CDRs. *IEEE Journal of Solid-State Circuits*, 50(4):845–855, April 2015.