Clock and Data Recovery over Optical Links and Networks

A MEng Project Final Report

Ammar Bin Shaqeel Ahmed

ammar.ahmed.16@ucl.ac.uk

16080322

University College London

Supervisor:
Dr. Georgios Zervas

Secondary Assessor:
Dr. Domaniç Lavery

May, 2020

Acknowledgements

I would like to thank Dr. Georgios Zervas for his help and support during this project. He was always approachable and encouraging throughout.

I am also very grateful to Vaibhawa Mishra and Kari Clark for being generous with their time and for sharing their knowledge with me.

Abstract

In this project we aim to develop a system that can perform burst-mode source-synchronous optical transmission to demonstrate the feasibility of such communication in data centres. Modern data centres require rapid switch reconfiguration, and optical switches could theoretically meet those requirements. However they are limited by CDR locking times which would be bypassed in a source-synchronous system. We were not able to demonstrate a fully working system but were able to demonstrate bust-mode transmission and prepare hardware for optical transmission. We were not able to demonstrate source-synchronous reception.

Contents

Chapter 1

Introduction

Bandwidth demands in data centres have been doubling every 12-15 months. For data centre providers to keep pace with the increased demand (at the same price point) network switches have had to double their capacity while staying at roughly the same cost [1], and so far they have done so. However this seems to be coming to an end for two reasons: The first is a predicted increase in the rate of growth of demand, due to trends like hardware accelerated programming and dis-aggregated workloads. The second is because electrical switches are predicted to reach a limit due to the physical limits on pin density [2].

For these reasons optical switching is being explored, as it has the potential to overcome many of these problems. Optical switches do not require opto-electrical (OEO) conversion, and hence the number of expensive and power hungry transceivers required is reduced. Furthermore, as buffering is not needed, the latency of the optical switches is much lower. Lastly, they do not use electronics for switching, thus bypassing the aforementioned physical limit [2].

In data centres much of the traffic that is transmitted between servers is in the form of small data packets, with 97.8% of packets being 576 bytes or less [3]. With 100 Gb/s ports this means that switching should take place on the order of hundreds of nanoseconds.

When data is transmitted without a clock signal, the clock has to be regenerated at the receiver before the data can be decoded - this is known as clock and data recovery (CDR). The time taken for the local clock to "lock" to the data stream, adds latency. In optical switches physical links are created between each transceiver-receiver pair. Hence each time the switch is reconfigured, the CDR must re-lock to the new link. This means that the network throughput is limited by the sum of the optical switching time and the CDR locking time - which can be hundreds of nanoseconds in the worst case and tens of nanoseconds in the best case [4]. Assuming an optical switching time of 1 nanosecond, it is evident that the CDR locking time acts as bottleneck that can drastically reduce the throughput [5].

In a source synchronous system the clock is transmitted alongside the data, removing the CDR locking time. This would remove the bottleneck, theoretically increasing the throughput.

Chapter 2

Theoretical Basis

2.1 Background Theory

Here we go deeper into the theory of certain elements of the system.

Bang-Bang CDR

Commonly a serial data stream is sent over a channel without a clock signal. Clock and Data Recovery (CDR) is the process of extracting timing information from a serial data stream, then using it to decode the received data stream. A CDR circuit has two primary functions. The first is to extract a clock based on the input data, and the second is to resample the data.

To extract the clock from the data, a local clock is generated, then is adjusted as "early" or "late" when compared with the incoming data signal [6]. We can think of this as a control system, as shown in Figure 2.1.

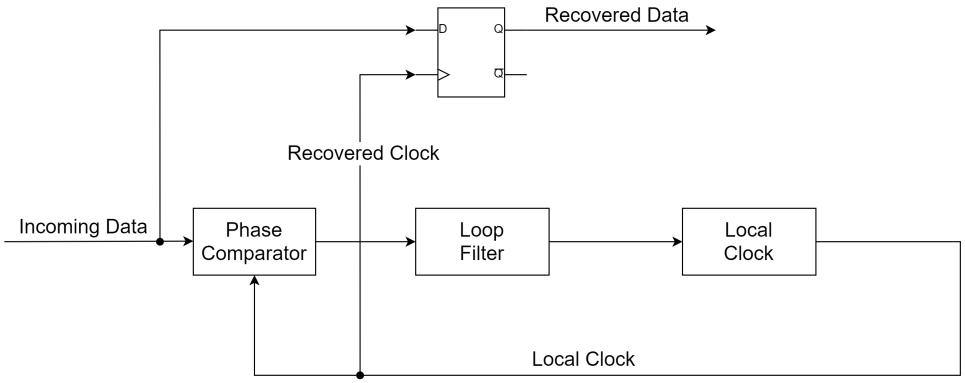


Figure 2.1: Basic CDR design

Phase detectors can be divided into two types, linear (where the output has a linear relationship to the input) and binary or bang-bang phase detectors (where the output is either positive or negative). Binary phase detectors are more commonly used in digital CDR circuits [7]. An example of one is the Alexander detector [8] which gives out a high D0+ and a low D0- if the clock lags and vice-versa if the clock leads, as shown in Figure 2.2.

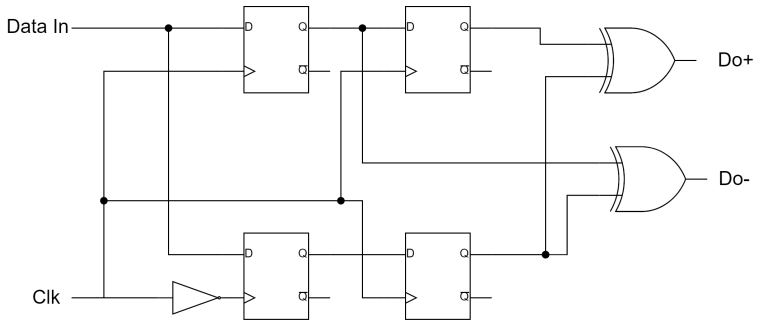


Figure 2.2: Alexander Phase Detector

Pseudorandom Binary Sequence

A pseudorandom binary sequence (PRBS) is a sequence of bits that appears to be random. However as it is generated using a deterministic algorithim, it can be replicated if the inital conditions are the same.

A common practical implementation of PRBS generation uses linear-feedback shift registers. As an example, a PRBS-4 sequence could be generated by using a 4 bit register. We seed the register with a non-zero number, then tap two bits of the register as an input. We then shift the contents of the register, taking the last bit as an output and the new bit as an input, as illustrated in Figure 2.3.

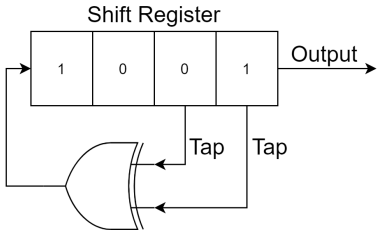


Figure 2.3: Shift Register Implementation

The full operation can be seen in Table 2.1. As 0000 cannot appear (the value of the register would never change) we see that for a register of size N, the bitsequence is $2^N - 1$ bits long.

Source Synchronous System

In a source synchronous system a clock signal is provided alongside the data signal, as shown in Figure 2.4. This has the advantage of not needing a CDR circuit. Furthermore as both the clock and the data come from the same device any jitter will be similar across both signals and can likely be ignored [9]. A downside is that there will be crossing of clock domains at the reciever as the transmitted clock will not be synchronous with the clock domain of the receiving device.

Cycle	Input	Shift Register				Output
0	1	1	0	0	1	1
1	0	1	1	0	0	0
2	1	0	1	1	0	0
3	0	1	0	1	1	1
4	1	0	1	0	1	1
5	1	1	0	1	0	0
6	1	1	1	0	1	1
7	1	1	1	1	0	0
8	0	1	1	1	1	1
9	0	0	1	1	1	1
10	0	0	0	1	1	1
11	1	0	0	0	1	1
12	0	1	0	0	0	0
13	0	0	1	0	0	0
14		0	0	1	0	0

Table 2.1: Shift Register Operation

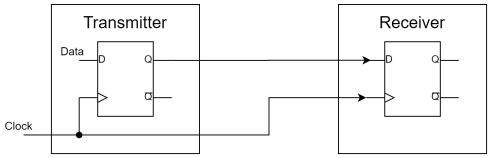


Figure 2.4: Source Synchronous System

Semiconductor Optical Amplifier

Optical amplifiers are devices that can amplify an optical signal without needing to convert it to an electrical one. A silicon optical amplifier (SOA) is one that uses a semiconductor as the gain medium. As light passes through this gain medium it is amplified. SOAs are electrically pumped (do not require the use of another laser) and are of small size.

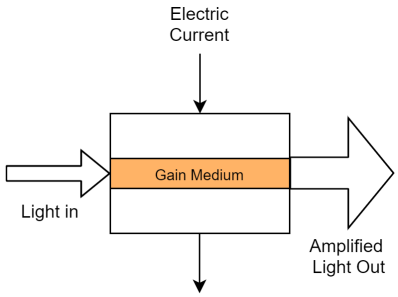


Figure 2.5: Basic SOA Structure

2.2 Literature Review

[5] outlines how CDR circuits are a limiting factor in optical switching and proposes a method of phase caching to overcome this. The data is transferred over the high-speed Xilinx transceivers. The phase measurements that are being compared are that of the received data to the local clock in a bang-bang CDR circuit. The PRBS data is pregenerated and is written to memory. It is sent in short bursts with a known sequence at the end. When the data arrives it is written to memory, is processed (searching for the known sequence), and is then checked for errors. The phase caching improved locking time on switching by 12 times.

[10] This paper describes a very low latency burst mode optical receiver. They were able to achieve 25 Gb/s burst mode operation with a 31 ns lock time. In this paper, the receiver receives a burst of data, must calibrate to the voltage of the data, then engage the CDR circuit to recover the incoming data. Again the CDR is a bang-bang CDR that is modified using a phase interpolator. While this paper is not directly applicable as the receiver architecture differs from that of the transceiver of the board we are using, the problem they are trying to solve is similar and provides an idea of the locking times of a well optimised CDR.

[11] this presentation describes a system where the phase of a transceiver on Xilinx board is kept stable over resets. While this was done on the transmitter side, it is applicable to this project as it uses the same transceivers (GTY Ultra-Scale+) and involves reading phase data and setting the phase of the transmitters. As the architecture of the transmitters and receivers is not dissimilar what is applicable at the transmitter may well be applicable at the receiver.

[12] This application note describes the design of a circuit that allows for the reception and transmission of 7:1 data using low-voltage differential signaling (LVDS) data transmission. Of note here is that the board's inbuilt circuitry allows makes designing serialiser/deserialiser circuits very simple. This may be preferable to using the inbuilt transceivers as it may be possible that we have more control over the inner workings of the system.

[13], [14] describe a Xilinx intellectual property that allows the high speed serial transceivers to be used at much lower data rates. This is done by oversampling the data. However it may not be applicable as it essentially operates as an IP that is placed after the transceiver interface and does not directly effect the behaviour of the transceiver. However it remains of interest as it may be easier to demonstrate a working system with lower data rates as a lower data rate would be forgiving in term of sampling/clock accuracy.

[15] described an optical source synchronous system. It describes how choosing the correct wavelength for the clock can minimise the modal cross-talk. Furthermore, in conjunction with [9] it describes how source synchronous systems are able to track correlated jitter between clock and data channels, and how system performance can be degraded by channel slew between clock and data channels. If the project is run at a lower data rate over multiple channels, this means that the jitter between the channels should be correlated if they are coming from the same device which would minimise the effects of such jitter.

[16] further explored reducing the modal crosstalk by proposing an architecture with re-configurable clock and data paths, thus allowing the user to chose the optimal lane for the sensitive clock for each photonic interconnect. This is unlikely to be needed in this project, as each transmitter should have a fixed data characteristic, and as the hardware for both transmitters is similar, they should have a similar data characteristic. In a situation where there are many different transmitters this may be useful.

In [17], [18], and [19], the white rabbit project is discussed. A white rabbit system provides sub-nanosecond synchronisation accuracy. To achieve this, accurate measurements of the link delay between the nodes of the network must be calculated. In a White Rabbit system, all the nodes are locked to the same frequency. Hence the link delay can be calculated by having a node receive a clock signal from another node, then return the same signal. The link delay can then be calculated by comparing the phase offset of the two signals. A similar system could potentially be used to determine the phase offset between a source and the receiver, although it is possible that the frequency of two sources may differ, and hence make this technique unusable.

[20] and [21] describe fixed latency links. In the event we were unable to bypass the CDR, it may be possible to organise the system to have a fixed latency, then force the CDR to the appropriate fixed phase. This is a workaround as it is not a true source-synchronous system. Nonetheless it may be useful as a point of comparison.

Chapter 3

Proposed System and Objectives

3.1 Proposed System Overview

To demonstrate the efficacy of a source synchronous system we propose a single pseudorandom binary sequence (PRBS) source that optically transmits over two channels to a single receiver. If transmission is alternated the effect is that the receiver would receiver bursts of data from two different channels. If the full PRBS sequence is receiver then the system would be working correctly.
An overview of the system is shown in Figure 3.1.

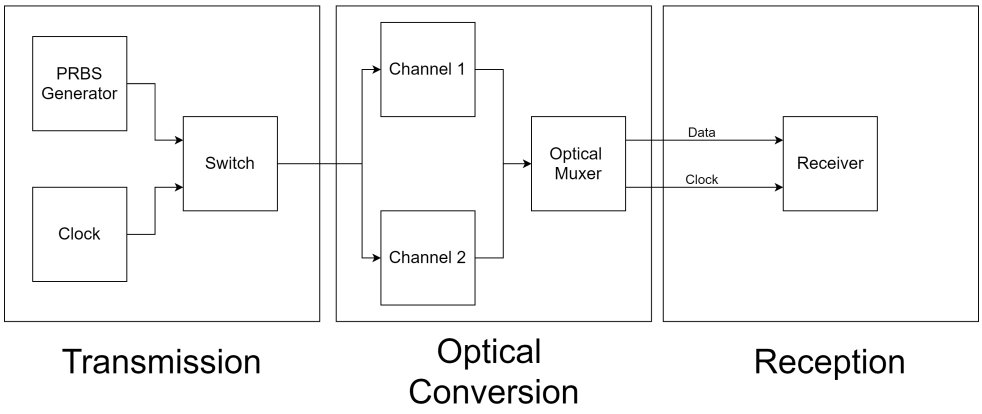


Figure 3.1: Overview of System

3.2 Objectives

The overall objective is to demonstrate successful burst source-synchronous communication for comparison with a system that uses a CDR. Overall we can break down the project to the following sub-objectives:

- Burst mode PRBS transmission over two channels alongside clock
- Optical transmission over two channels, then muxing the two channels together
- Source-synchronous reception of PRBS data

Chapter 4

Implementation and Results

As outlined in the Objectives we can divide the tasks into three main parts: transmission, optical conversion, and reception. In this project we looked at using an FPGA board for the generation and reception of the PRBS data. Hence the overall design is of a board in a loopback configuration as shown in Figure 4.1.

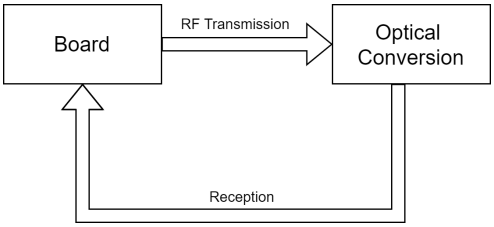


Figure 4.1: Loopback Configuration

4.1 Transmission and Reception

4.1.1 Hardware

To generate and receive PRBS data the VCU118 board was used. The transmission and reception of the data was handled by the onboard high-speed parallel to serial GTY transceivers in conjunction with a Si5345 external clock. To connect with the transceiver the HiTechGLobal FMC-MSMP module was used.

Figure 4.2: VCU118 Board

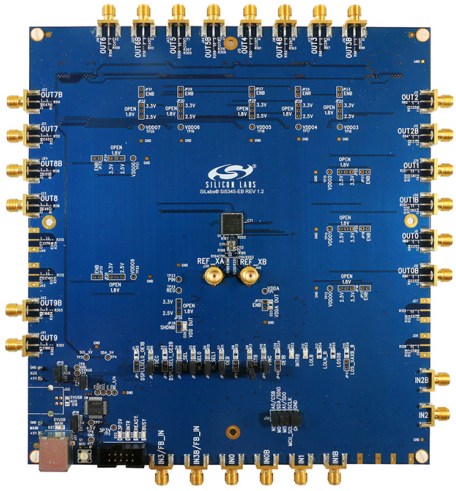


Figure 4.3: Si5345 Clock

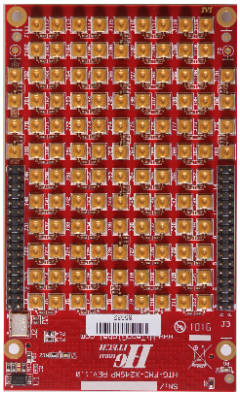


Figure 4.4: FMC-MSMP module

4.1.2 Transceiver Setup

Most of the project took place using the transceivers in a simple RF loopback configuration (no optical transmission).

The setup of the transceivers followed the example design outlined in the user guide to the transceivers [22]. Places where choices were made (as the example design is not specific to a particular board) or the design was deviated from are outlined below.

Selection of Quads

The GTY transceivers in the VCU118 are grouped into four channels or quads. Seven GTY quads on the left side of the device and six GTY quads on the right side of the device. There are 52 transceivers on VCU118 board, in total.

- Four of the GTY transceivers are wired to Samtec Firefly Module Connector
- Four of the GTY transceivers are wired to QSFP1 module connector
- Four of the GTY transceivers are wired to QSFP2 module connector
- Sixteen of the GTY transceivers are wired to the PCIe 16-lane edge connector
- Twenty-four of the GTY transceivers are wired to FMC+ HSPC connector

As we were using a FMC-HSMP module to connect to the transceivers, we were limited to the transceivers wired to the FMC+ HSPC connector. Furthermore only certain quads can be driven through an external clock. As such we chose Quad 120 and Quad 122 as the quads for testing.

There are also limitations to the number of quads that can be driven with an external clock while still meeting jitter requirements, but as we are only driving two transmitters, it was not an issue.

Pin Configuration

It was also necessary to map the pins on the board to those on the FMC-MSMP module. This was done as in Table 4.1. As two transceivers were used, we mapped out the pins for the clocks, transmitters, and receivers of both.

For a full pinout diagram refer to the user guide of the board [23].

Function	Channel	Net Name	FPGA PIN	Connected Pin	FMC+ Pin
clk_p	GTYE4_COMMON_X0Y1	FMCP_HSPC_GBTCLK5_M2C_P	AN40	Z20	J106
clk_n	GTYE4_COMMON_X0Y1	FMCP_HSPC_GBTCLK5_M2C_N	AN41	Z21	J115
tx_p	GTYE4_CHANNEL_X0Y4	FMCP_HSPC_DP20_C2M_P	BD42	Z8	J117
tx_n	GTYE4_CHANNEL_X0Y4	FMCP_HSPC_DP20_C2M_N	BD43	Z9	J116
rx_p	GTYE4_CHANNEL_X0Y4	FMCP_HSPC_DP20_M2C_P	BC45	M14	J55
rx_n	GTYE4_CHANNEL_X0Y4	FMCP_HSPC_DP20_M2C_N	BC46	M15	J54
clk_p	GTYE4_COMMON_X0Y3	GBTCLK2_M2C_P	AF38	L12	J21
clk_n	GTYE4_COMMON_X0Y3	GBTCLK2_M2C_N	AF39	L13	J20
tx_p	GTYE4_CHANNEL_X0Y8	DP0_C2M_P	AT42	C2	J77
tx_n	GTYE4_CHANNEL_X0Y8	DP0_C2M_N	AT43	C3	J78
rx_p	GTYE4_CHANNEL_X0Y8	DP0_M2C_P	AR45	C6	J100
rx_n	GTYE4_CHANNEL_X0Y8	DP0_M2C_N	AR46	C7	J99

Table 4.1: Transceiver Pin Out

Transceiver Configuration

The transceivers were programmed to run at a standard rate of 10GB/s, with a reference clock taken from the external Si5345 clock which was configured to run at 156.25 MHz with a format of 2.5V LVDS. The free-running clock (used to drive resets) was driven at 125 MHz, sourced from the onboard 300 MHz system clock. The reset functions were bound to the physical push buttons BE23 and BB24. Status indicators (Link Up and Link Down) were bound to LED1 and LED0 respectively.

At the current stage there was no encoding as we were trying to troubleshoot an issue with the PRBS checking module. However if running the system for a long period of time, encoding would be necessary to ensure the CDR remained locked. This can be easily done from the transceiver wizard. The full details of the setup can be found in Appendix A.

4.1.3 Transmitter

In Figure 4.5 we see a block diagram of the transmitter (TX) of the transceiver. Parallel data flows into the TX interface, is serialised, then finally flows out of the transmitter as high-speed serial data.

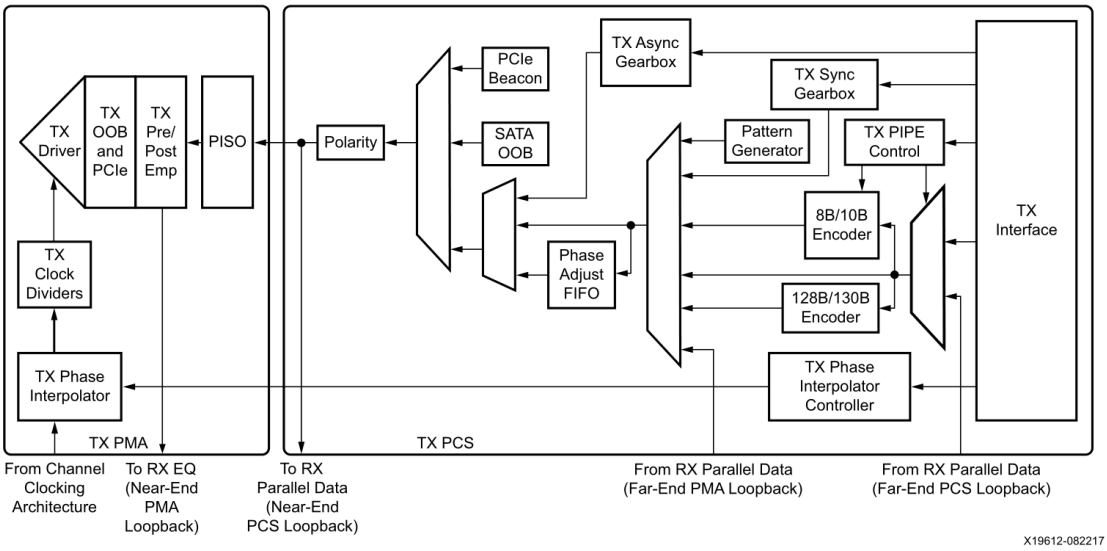


Figure 4.5: Transmitter Block Diagram [24]

We looked to modify the functionality of a basic implementation of the transmitter. In the basic implementation the output of a PRBS generator is fed to the transmitter interface through a wrapper as in Figure 4.6.

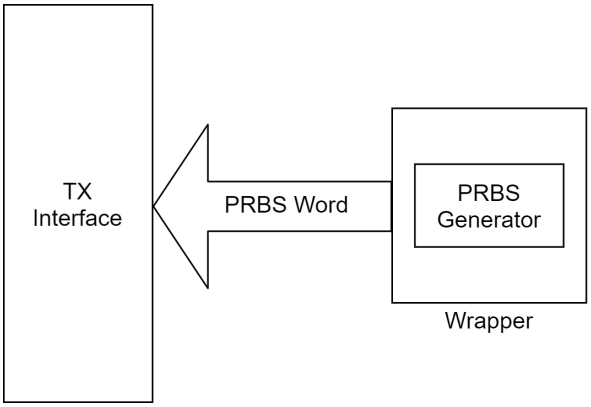


Figure 4.6: Generator to TX Interface

There were two variations of the PRBS generation module that were developed.

Burst Mode over Single Channel

Here we modified the PRBS generation module by setting it to output zeros if the enable command was not asserted. In combination we added a register to the wrapper, which on overflow toggles the enable flag. This has the effect of causing the PRBS module to output a sequence interspersed with zeros. The code can be found in Section B.2.

A testbench where the unmodified PRBS generator and burst PRBS generator are compared is shown in in Figure 4.7. We see the burst generator generating the expected result: a PRBS sequence interspersed with zeros. We note also that the full sequence is transmitted in burst mode (no data is lost).

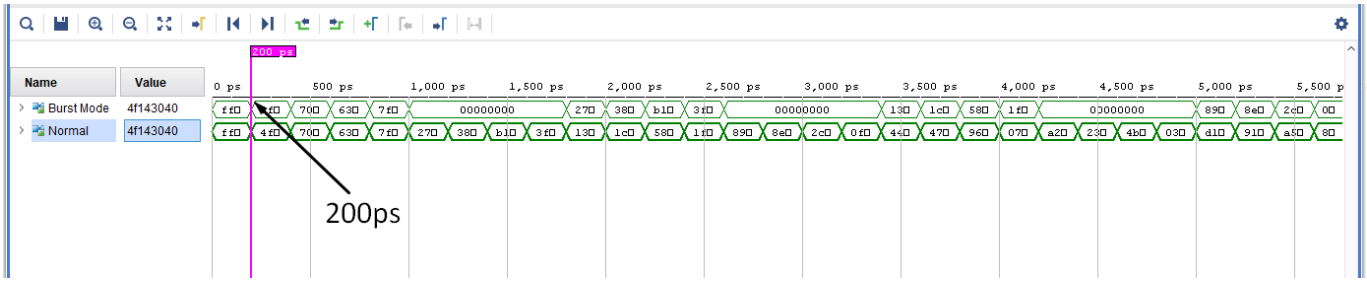


Figure 4.7: Single Burst Mode Behaviour

Comparing Figure 4.8 and Figure 4.9 we see that the recurrence time (in this testbench we use PRBS7, as it is a short sequence) for the normal sequence is about half that of the burst mode sequence. This is expected as the burst mode sequence is operating on a duty cycle of 50% and hence should take double the time to recur.

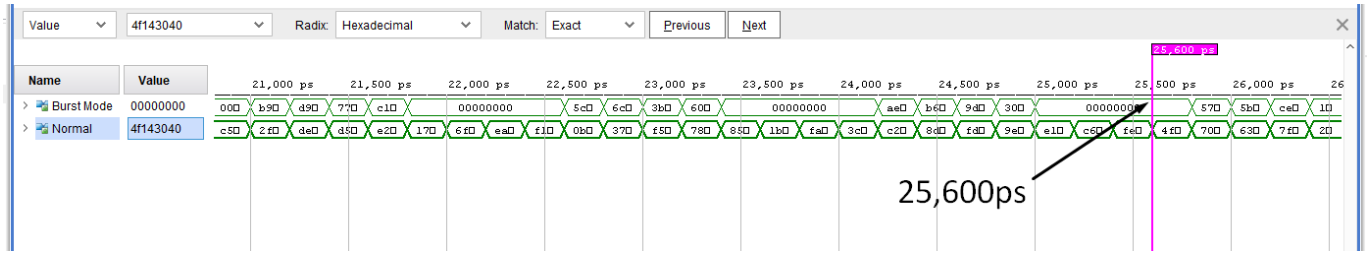


Figure 4.8: Normal Mode Recur Time

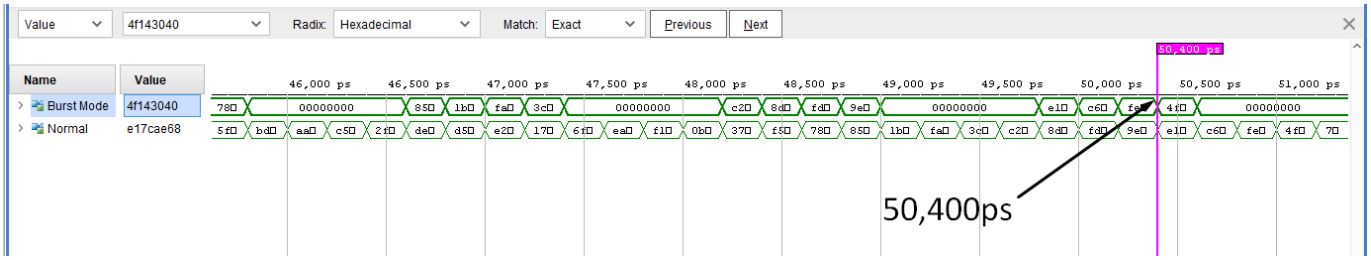


Figure 4.9: Burst Mode Recur Time

Switching Between Two Channels

Here the PRBS wrapper was changed to feed the two different outputs. The PRBS generator was unchanged. We added a register which on overflow alternated the output of the generator module. The code can be found in Section B.3 This had the overall effect of having the whole sequence be sent over two different channels as shown in Figure 4.10.

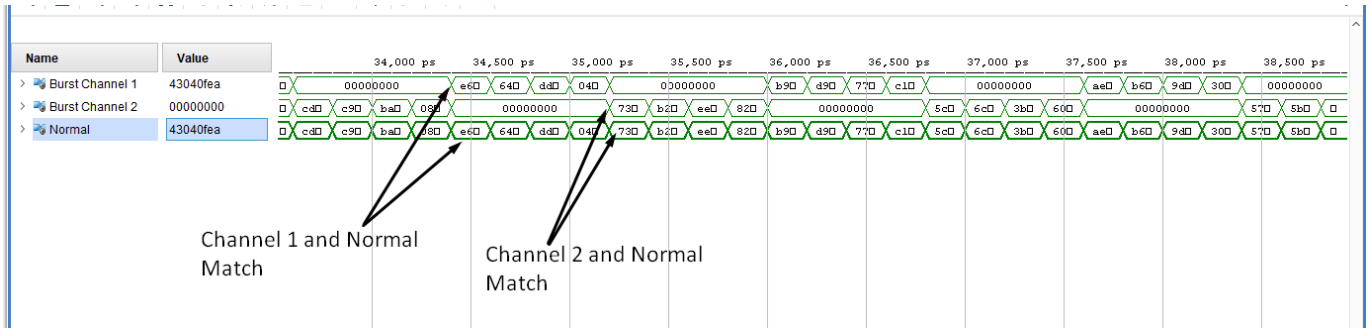


Figure 4.10: Two Channel Burst Mode

4.1.4 Receiver

In Figure 4.11 we see a block diagram of the receiver (RX) of the transceiver. Serial data flows into the reciever, is deserialised, and is finally passed to the RX Interface where it can then be passed to the rest of the board.

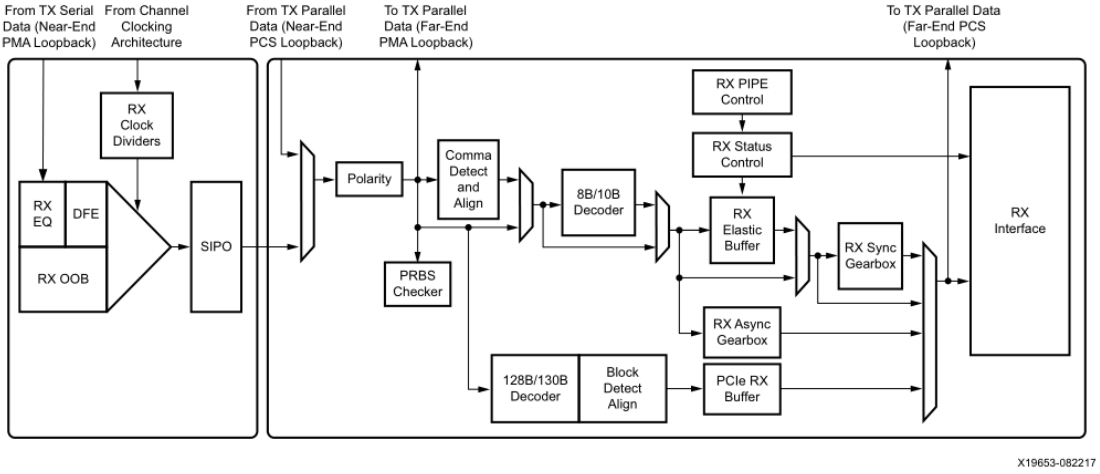


Figure 4.11: Receiver Block Diagram [24]

In the basic implementation the RX interface passes data to the PRBS checker through a wrapper. We looked to modify the wrapper so that burst mode checking would work correctly.

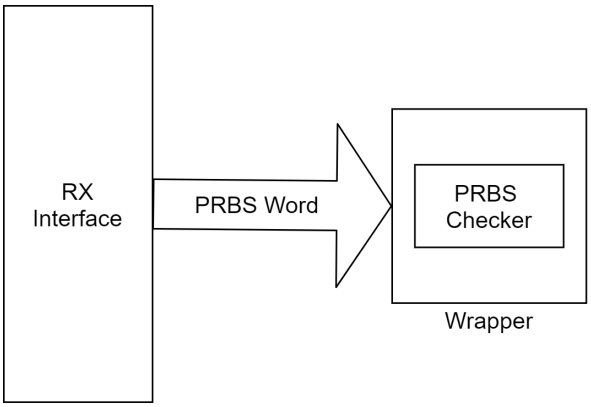


Figure 4.12: RX Interface to Checker

Burst Mode Checking

In burst mode checking there are periods when the incoming bitstream is all zeros. The PRBS checker module takes the incoming data as a seed to calculate the next expected sequence. If zeros are provided then this interferes with the module (as the next expected word will be calculated based on zeros). To compensate for this we loaded incoming data to a register and checked if it was zeros. If not, it was passed to the PRBS module. The code can be found in Section B.4.

In simulation when the PRBS generation module was connected directly to the PRBS checker this worked correctly as shown in Figure 4.13 with no errors (the sequence was repeated nearly 4,000 times).

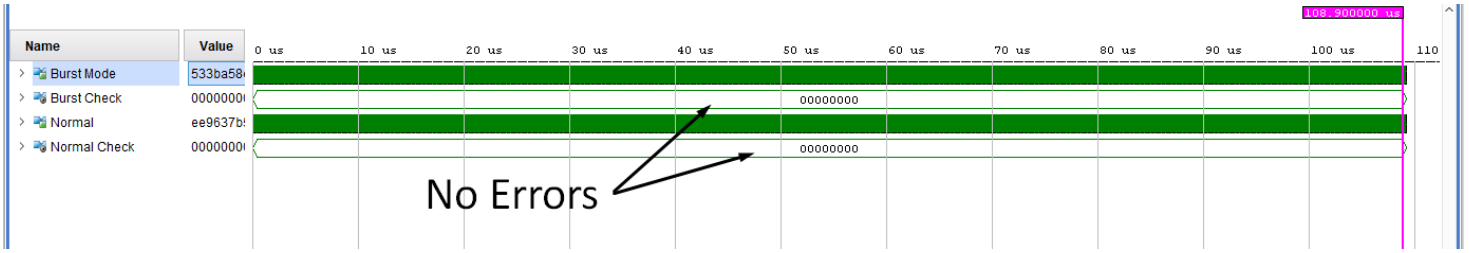


Figure 4.13: Direct Connection Checking

However, when passed through the transceiver, the PRBS checker found errors consistently, as shown in Figure 4.14. These errors appear to originate in situations where zeros are sent to the TX interface, but some bits are received at the RX interface. We suspect that this may be due to some encoding by the transceiver, but were unable to pinpoint the exact cause before the end of the project. There are other workarounds that may have been possible, such as loading the entire sequence into memory and processing it there to remove the zeros, or by sending shorter sequences, so that each burst would contain a full sequence, but we were not able to explore this fully.

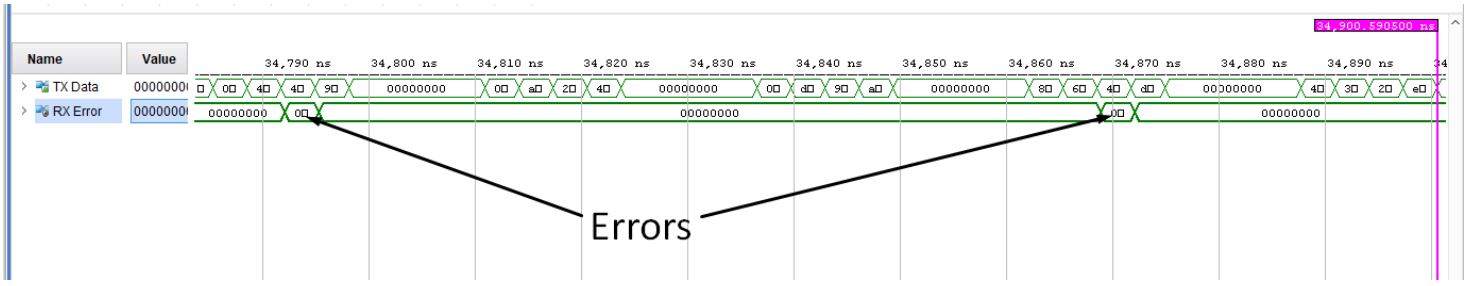


Figure 4.14: Transceiver Checking

Two Channel Checking

In the case where two transmitters were muxed together and were sent to a single receiver, it should not have been necessary to change the behaviour of the PRBS

checking module. We were unable to check this due to the closing of the lab.

Disabling The CDR

The receiver CDR block detailed block diagram is shown in Figure 4.15. We see it operates as a control system similar to what is described in Section 2.1: incoming data is sampled by edge and data samplers to get phase information, which is passed to a state machine which makes decisions on how to adjust the phase of the local clock (provided by the PLL) using phase interpolators.

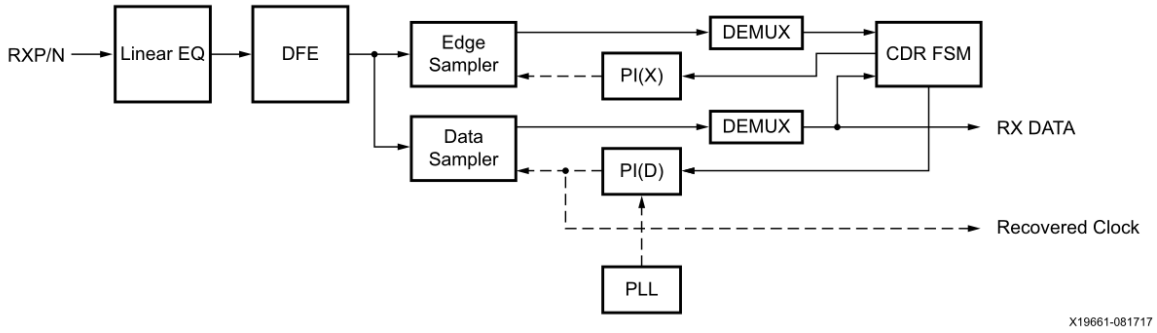


Figure 4.15: CDR Block Diagram

The final goal of the project would have been to run the receiver source-synchronously. The transceiver did not allow much flexibility here, and were limited to the configuration options that were offered by the Xilinx IP. We were not able to bypass the CDR entirely. We attemped to mimic this a much as possible by locking the CDR to reference clock and using the same clock to drive the receiver and transmitter, ideally then taking readings of phase, and making the necessary compensations.

We were able to disable the CDR by asserting the CDRHOLD ports. This was done by hardcoding the CDRHOLD value. We further extended this by adding CDRHOLD in conjunction with the QPLLLOCK and RXCDRLOCK ports to the Xilinx Virtual Input/Output (VIO) which allowed us to toggle the CDR. The link became unstable after the port was asserted.

We were then unable to read the phase. The phase caching paper by Kari, et al. [5] was done on Xilinx Ultra-Scale board and they were able to read and modify the phase by reading/writing values to the appropriate register. However the details of that implementation were specific to the Ultra-Scale boards. The board used in the project is an Ultra-Scale+ board and there is no publicly available information on how to modify/read the phase. As such we were unable to explore this further.

4.2 Optical Transmission

We were not able to test this hardware in conjuction with the burst mode transmission, as the labs closed due to external factors. However the boards were used in other projects that took place in the research group, including a recent paper where machine learning was used to optimise the switching times of the SOA [25].

4.2.1 SOA Board

The design consisted of a commercial Inphenix IPSAD1502 using input from a matched RF line to modulate an incoming laser. The SOA was also temperature controlled at 25 degrees Celsius. The driving current was 200mA. As the input resistors are 50Ω, the power dissipation is 2W, and the resistors must be rated as such.

The circuit diagram can be seen in Figure 4.16 (developed by Dennis Goh).

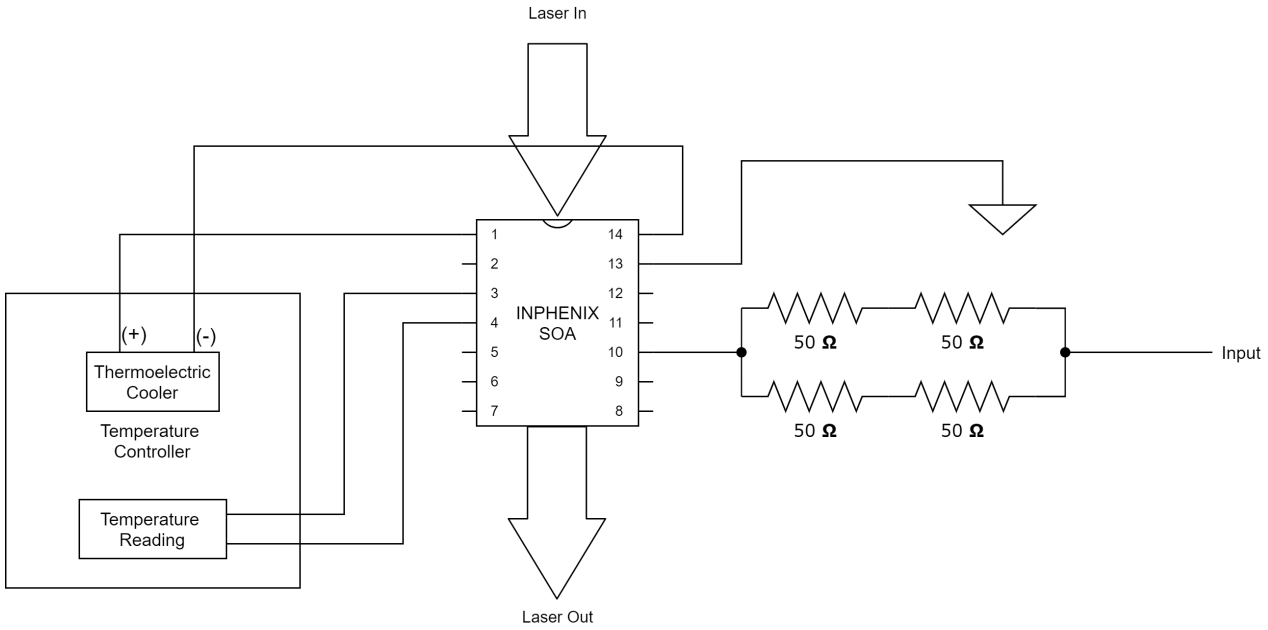


Figure 4.16: SOA circuit

4 PCBs were prepared and made. The PCB design can be found in Appendix C. The Temprature Controller was connected to the IC uing a 9-pin D-type connector.

4.2.2 Heatsink and Mount

It was also necessary to attach a aluminium substrate underneath the PCB board for heat dissipation. An Aluminium substrate was machined to match the PCB (as shown in Appendix D), then the SOA was placed in contact with the substrate. We then stacked the four boards above each other as in Figure 4.17.

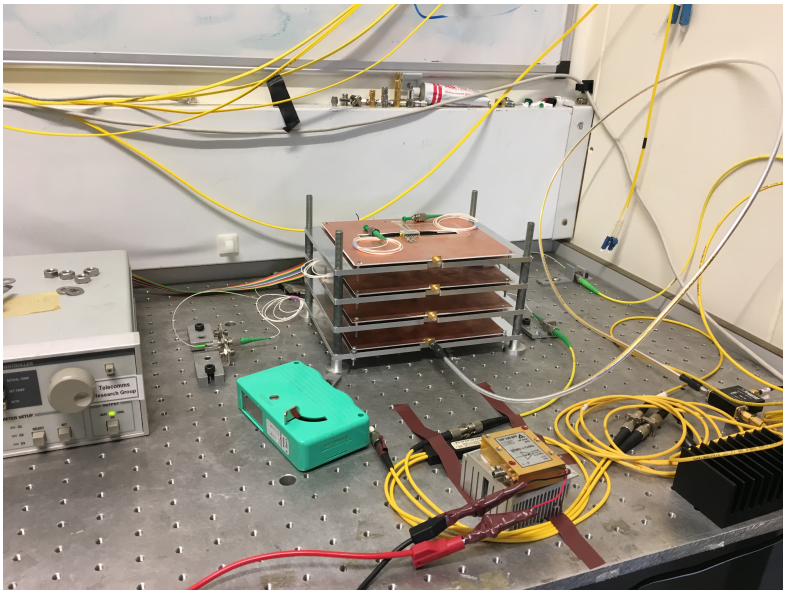


Figure 4.17: Stacked Boards

Chapter 5

Conclusion

In this project, we aimed to achieve a fully working burst-mode source-synchronous optical transmission.

We developed a wrapper for a PRBS generator that enabled burst mode transmission and were able to successfully demonstrate this in simulation. We also developed hardware mounts for the SOAs. We were not able to test optical transmission as the labs were closed.

On the receiver side we developed a burst-mode receiver which worked in simulation, but had issues when tested in the full system, that we did not have time to troubleshoot. Given more time it is possible that these issues could have been resolved.

With regards to running the receiver source-synchronously we were not able to bypass the CDR. We attempted a workaround, but were hindered due to a lack of publically available information about certain aspects of the transceiver. It is possible that if a different board was used, or a different type of transmitter, that this could have been accomplished.

Overall we were able to progress towards a workable prototype, even though we were not fully able to accomplish the stated goal.

Bibliography

[1] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: a decade of clos topologies and centralized control in google's datacenter network," *Communications of the ACM*, vol. 59, no. 9, pp. 88–97, 2016.

[2] H. Ballani, P. Costa, I. Haller, K. Jozwik, K. Shi, B. Thomsen, and H. Williams, "Bridging the last mile for optical switching in data centers," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*. IEEE, 2018, pp. 1–3.

[3] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proceedings of the 2017 Internet Measurement Conference*. ACM, 2017, pp. 78–85.

[4] X. Chen, S. Chandrasekhar, G. Raybon, S. Olsson, J. Cho, A. Adamiecki, and P. Winzer, "Generation and intradyne detection of single-wavelength 1.61-tb/s using an all-electronic digital band interleaved transmitter," in *Optical Fiber Communication Conference Postdeadline Papers*. Optical Society of America, 2018, p. Th4C.1. [Online]. Available: http://www.osapublishing.org/abstract.cfm?URI=OFC-2018-Th4C.1

[5] K. Clark, H. Ballani, P. Bayvel, D. Cletheroe, T. Gerard, I. Haller, K. Jozwik, K. Shi, B. Thomsen, P. Watts *et al.*, "Sub-nanosecond clock and data recovery in an optically-switched data centre network," in *2018 European Conference on Optical Communication (ECOC)*. IEEE, 2018, pp. 1–3.

[6] S. Y. Sun, "An analog pll-based clock and data recovery circuit with high input jitter tolerance," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 325–330, 1989.

[7] H. Zhang, S. Krooswyk, and J. Ou, "Chapter 4 - link circuits and architecture," in *High Speed Digital Design*, H. Zhang, S. Krooswyk, and J. Ou, Eds. Boston: Morgan Kaufmann, 2015, pp. 163 – 198. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124186637000046

[8] J. Alexander, "Clock recovery from random binary signals," *Electronics letters*, vol. 11, no. 22, pp. 541–542, 1975.

[9] A. Ragab, Y. Liu, K. Hu, P. Chiang, and S. Palermo, "Receiver jitter tracking characteristics in high-speed source synchronous links," *Journal of Electrical and Computer Engineering*, vol. 2011, p. 5, 2011.

[10] A. Rylyakov, J. E. Proesel, S. Rylov, B. G. Lee, J. F. Bulzacchelli, A. Ardey, B. Parker, M. Beakes, C. W. Baks, C. L. Schow *et al.*, "A 25 gb/s burst-mode receiver for low latency photonic switch networks," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 12, pp. 3120–3132, 2015.

[11] Eduardo Mendes, "Xilinx transceiver study," URL: https://indico.cern.ch/event/717613/contributions/2948664/attachments/1637690/2613637/hptd_fixed_phase_xcvr_04_18_eduardo_mendes.pdf, 2018.

[12] N. Sawyer, *LVDS Source Synchronous 7:1 Serialization and Deserialization Using Clock Multiplication*, Xilinx.

[13] P. Novellini and G. Guasti, *Dynamically Programmable DRU for High-Speed Serial I/O*, Xilinx.

[14] P. Novellini, G. Guasti, and A. Di Fresco, *Clock and Data Recovery Unit based on Deserialized Oversampled Data*, Xilinx.

[15] C. Williams, B. Banan, G. Cowan, and O. Liboiron-Ladouceur, "A source-synchronous architecture using mode-division multiplexing for on-chip silicon photonic interconnects," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 22, no. 6, pp. 473–481, 2016.

[16] C. Williams, D. Abdelrahman, X. Jia, A. I. Abbas, O. Liboiron-Ladouceur, and G. E. Cowan, "Reconfiguration in source-synchronous receivers for short-reach parallel optical links," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[17] J. Serrano, M. Lipinski, T. Wlostowski, E. Gousiou, E. van der Bij, M. Cattin, and G. Daniluk, "The white rabbit project," *N/A*, 2013.

[18] P. Moreira, P. Alvarez, J. Serrano, I. Darwezeh, and T. Wlostowski, "Digital dual mixer time difference for sub-nanosecond time synchronization in ethernet," in *2010 IEEE International Frequency Control Symposium*. IEEE, 2010, pp. 449–453.

[19] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer, "White rabbit: Sub-nanosecond timing distribution over ethernet," in *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2009, pp. 1–5.

[20] K. Chen, H. Chen, W. Wu, H. Xu, and L. Yao, "Optimization on fixed low latency implementation of the gbt core in fpga," *Journal of Instrumentation*, vol. 12, no. 07, p. P07011, 2017.

[21] X. Liu, Q.-x. Deng, B.-n. Hou, and Z.-k. Wang, "High-speed, fixed-latency serial links with xilinx fpgas," *Journal of Zhejiang University SCIENCE C*, vol. 15, no. 2, pp. 153–160, Feb 2014. [Online]. Available: https://doi.org/10.1631/jzus.C1300249

[22] *UltraScale FPGAs Transceivers Wizard v1.7*, Xilinx.

[23] *VCU118 Evaluation Board*, Xilinx.

[24] *UltraScale Architecture GTY Transceivers*, Xilinx.

[25] T. Gerard, C. Parsonson, Z. Shabka, P. Bayvel, D. Lavery, and G. Zervas, "Swift: Scalable ultra-wideband sub-nanosecond wavelength switching for data centre networks," 2020.

Appendix A

Transceiver Settings

The configuration of the Transceiver Wizard can be found here.

A.1 Transceiver Wizard Settings

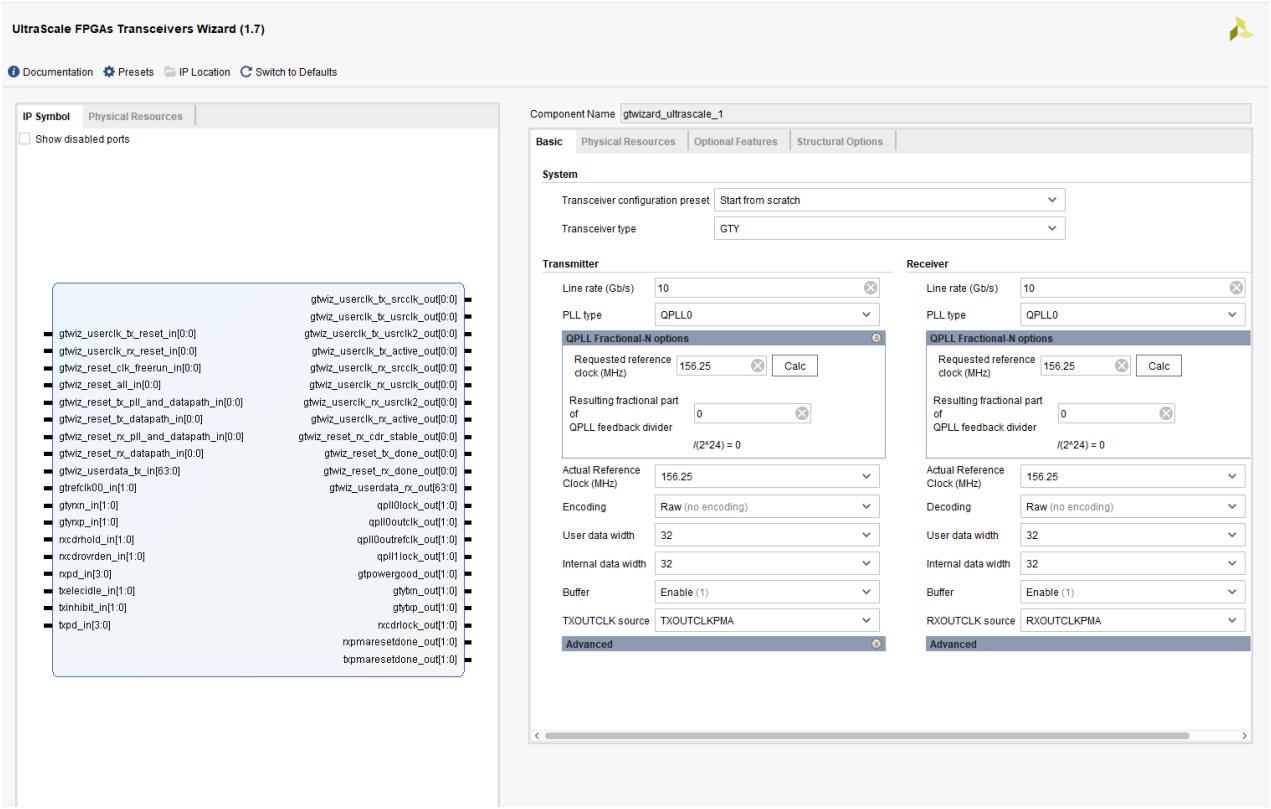


Figure A.1: Transceiver Wizard Settings

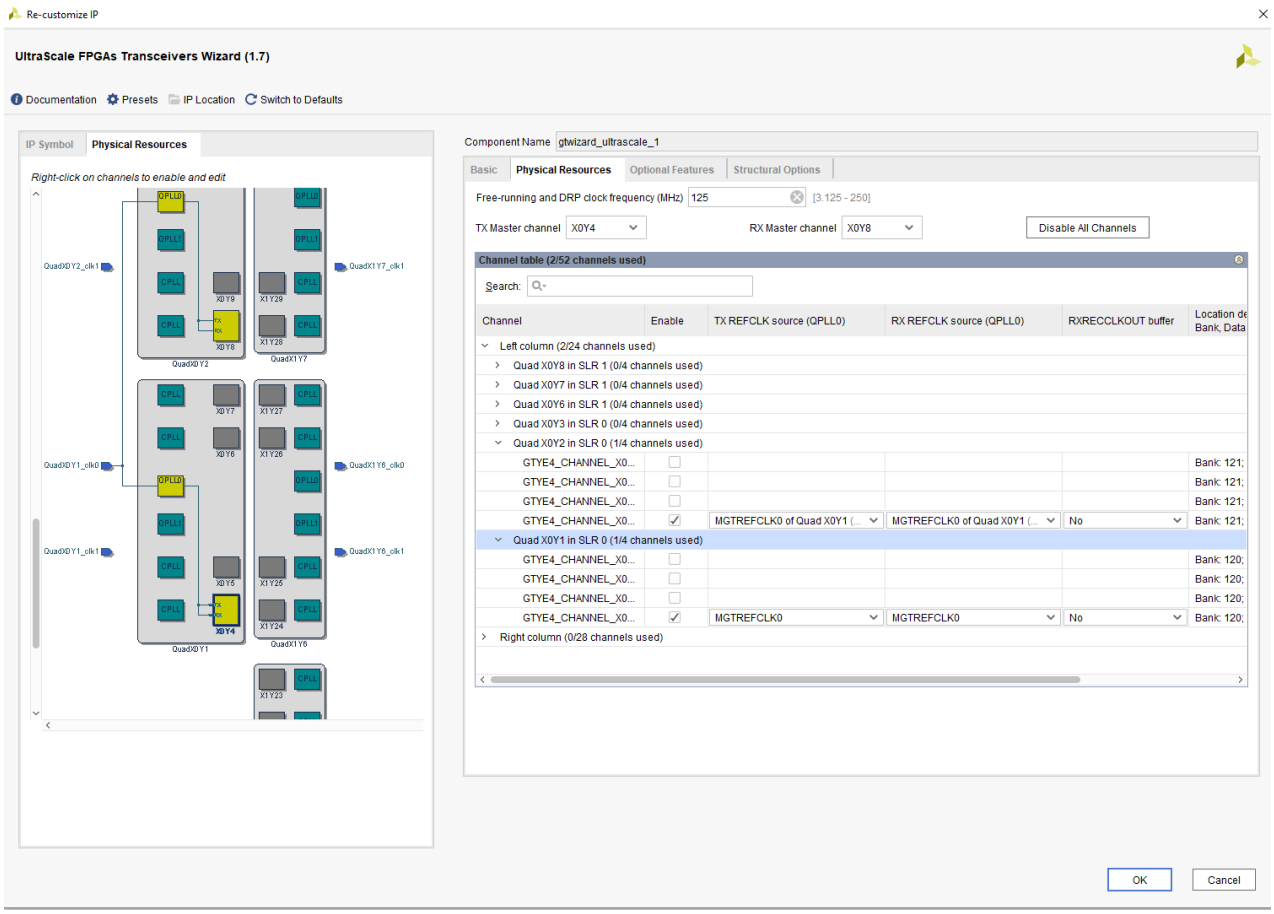


Figure A.2: Transceiver Wizard Settings 2

A.2 External Clock Settings

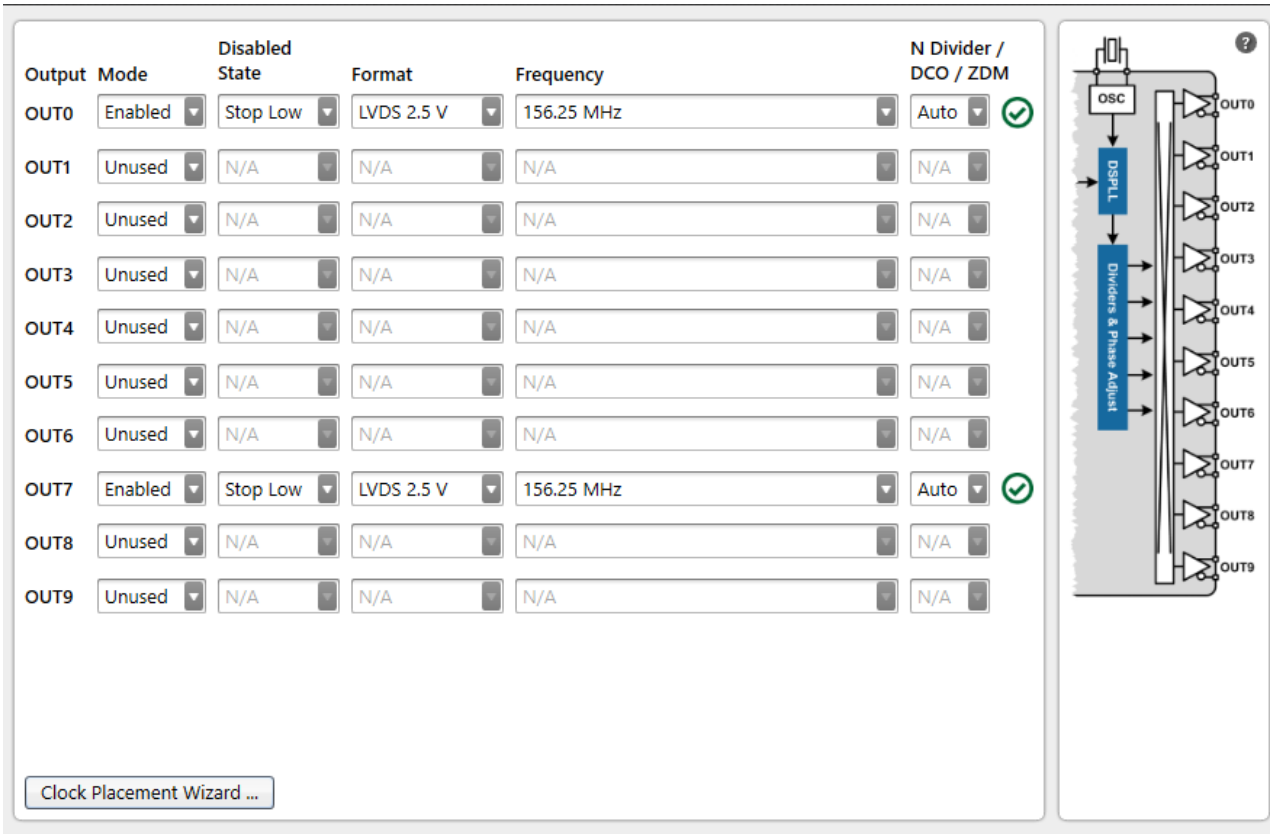


Figure A.3: External Clock Settings

A.3 Constraints

```
set_property PACKAGE_PIN AN41 [get_ports mgtrefclk0_x0y1_n]
set_property PACKAGE_PIN AN40 [get_ports mgtrefclk0_x0y1_p]

set_property IOSTANDARD DIFF_SSTL12 [get_ports hb_gtwiz_reset_clk_freerun_in_p]

set_property PACKAGE_PIN AY23 [get_ports hb_gtwiz_reset_clk_freerun_in_n]
set_property PACKAGE_PIN AY24 [get_ports hb_gtwiz_reset_clk_freerun_in_p]
set_property IOSTANDARD DIFF_SSTL12 [get_ports hb_gtwiz_reset_clk_freerun_in_n]

set_property package_pin BE23 [get_ports hb_gtwiz_reset_all_in]
set_property IOSTANDARD LVCMOS18 [get_ports hb_gtwiz_reset_all_in]

set_property PACKAGE_PIN BB24 [get_ports link_down_latched_reset_in]
set_property IOSTANDARD LVCMOS18 [get_ports link_down_latched_reset_in]


# LED1 (working correctly)
set_property PACKAGE_PIN AV34 [get_ports link_status_out]
set_property IOSTANDARD LVCMOS12 [get_ports link_status_out]

# LED0 (not working)
set_property PACKAGE_PIN AT32 [get_ports link_down_latched_out]
set_property IOSTANDARD LVCMOS12 [get_ports link_down_latched_out]

# Clock constraints for clocks provided as inputs to the core
--------------------------------------------------------------------------------
create_clock -period 8.000 -name clk_freerun [get_ports hb_gtwiz_reset_clk_freerun_in_p]
create_clock -period 6.400 -name clk_mgtrefclk0_x0y1_p [get_ports mgtrefclk0_x0y1_p]

# False path constraints #
--------------------------------------------------------------------------------
set_false_path -to [get_cells -hierarchical -filter {NAME =~
*bit_synchronizer*inst/i_in_meta_reg}]
set_false_path -to [get_cells -hierarchical -filter {NAME =~
*reset_synchronizer*inst/rst_in_*_reg}]
set_false_path -to [get_pins -filter REF_PIN_NAME=~*D -of_objects [get_cells
-hierarchical -filter {NAME =~ *reset_synchronizer*inst/rst_in_meta*}]]
set_false_path -to [get_pins -filter REF_PIN_NAME=~*PRE -of_objects [get_cells
-hierarchical -filter {NAME =~ *reset_synchronizer*inst/rst_in_meta*}]]
set_false_path -to [get_pins -filter REF_PIN_NAME=~*PRE -of_objects [get_cells
-hierarchical -filter {NAME =~ *reset_synchronizer*inst/rst_in_sync1*}]]
set_false_path -to [get_pins -filter REF_PIN_NAME=~*PRE -of_objects [get_cells
-hierarchical -filter {NAME =~ *reset_synchronizer*inst/rst_in_sync2*}]]
set_false_path -to [get_pins -filter REF_PIN_NAME=~*PRE -of_objects [get_cells
-hierarchical -filter {NAME =~ *reset_synchronizer*inst/rst_in_sync3*}]]
set_false_path -to [get_pins -filter REF_PIN_NAME=~*PRE -of_objects [get_cells
```

```
-hierarchical -filter {NAME =~ *reset_synchronizer*inst/rst_in_out*}]]


set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub] set_property
C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub] connect_debug_port dbg_hub/clk
[get_nets hb_gtwiz_reset_clk_freerun_buf_int]
```

Appendix B

Verilog Code

Here we include the different wrappers for the PRBS burst mode/checking. Most of the code is provided by the Xilinx wizard [22]. The modifications are the sections titled "Burst Mode Logic".

B.1 PRBS Generator/Checker Module

Mainly unmodified, with the exception of line 60 - changed to output zeros when the enable flag is 1'b0.

```
1  module gtwizard_ultrascale_1_prbs_any(RST, CLK, DATA_IN, EN, DATA_OUT);
2
3    //------------------------------------------
4    // Configuration parameters
5    //------------------------------------------
6     parameter CHK_MODE = 0;
7     parameter INV_PATTERN = 1;
8     parameter POLY_LENGHT = 31;
9     parameter POLY_TAP = 28;
10    parameter NBITS = 16;
11
12   //------------------------------------------
13   // Input/Outputs
14   //------------------------------------------
15
16    input  wire RST;
17    input  wire CLK;
18    input  wire [NBITS - 1:0] DATA_IN;
19    input  wire EN;
20    output reg  [NBITS - 1:0] DATA_OUT = {NBITS{1'b1}};
21
22   //------------------------------------------
23   // Internal variables
24   //------------------------------------------
25
26    wire [1:POLY_LENGHT] prbs[NBITS:0];
27    wire [NBITS - 1:0] data_in_i;
28    wire [NBITS - 1:0] prbs_xor_a;
29    wire [NBITS - 1:0] prbs_xor_b;
```

```
30    wire [NBITS:1] prbs_msb;
31    // reg [NBITS - 1:0] DATA_HOLD = {NBITS{1'b1}};
32    reg  [1:POLY_LENGHT]prbs_reg = {(POLY_LENGHT){1'b1}};
33
34   //-----------------------------------------
35   // Implementation
36   //-----------------------------------------
37
38    assign data_in_i = INV_PATTERN == 0 ? DATA_IN : ( ~DATA_IN);
39    assign prbs[0] = prbs_reg;
40
41    genvar I;
42    generate for (I=0; I<NBITS; I=I+1) begin : g1
43       assign prbs_xor_a[I] = prbs[I][POLY_TAP] ^ prbs[I][POLY_LENGHT];
44       assign prbs_xor_b[I] = prbs_xor_a[I] ^ data_in_i[I];
45       assign prbs_msb[I+1] = CHK_MODE == 0 ? prbs_xor_a[I]  :  data_in_i[I];
46       assign prbs[I+1] = {prbs_msb[I+1] , prbs[I][1:POLY_LENGHT-1]};
47    end
48    endgenerate
49
50    always @(posedge CLK) begin
51       if(RST == 1'b 1) begin
52          prbs_reg <= {POLY_LENGHT{1'b1}};
53          DATA_OUT <= {NBITS{1'b1}};
54       end
55       else if(EN == 1'b 1) begin
56          DATA_OUT <= prbs_xor_b;
57          prbs_reg <= prbs[NBITS];
58       end
59       else if(EN == 1'b 0) begin
60         DATA_OUT <= 32'b 0;
61       end
62   end
63
64 endmodule
```

B.2 Single Channel Burst Mode

```
module gtwizard_ultrascale_1_example_stimulus_raw (
  input  wire         gtwiz_reset_all_in,
  input  wire         gtwiz_userclk_tx_usrclk2_in,
  input  wire         gtwiz_userclk_tx_active_in,
  // input  wire         enable,
  output wire [31:0] txdata_out
);


  // ---------------------------------------------------------------------
  // Reset synchronizer
  // ---------------------------------------------------------------------

  // Synchronize the example stimulus reset condition into the txusrclk2 domain
  wire example_stimulus_reset_int = gtwiz_reset_all_in || ~gtwiz_userclk_tx_active_in;
  wire example_stimulus_reset_sync;

  (* DONT_TOUCH = "TRUE" *)
  gtwizard_ultrascale_1_example_reset_synchronizer
  example_stimulus_reset_synchronizer_inst (
    .clk_in  (gtwiz_userclk_tx_usrclk2_in),
    .rst_in  (example_stimulus_reset_int),
    .rst_out (example_stimulus_reset_sync)
  );


  // ---------------------------------------------------------------------
  // PRBS generator output enable and sideband control generation
  // ---------------------------------------------------------------------

  reg prbs_any_gen_en_int = 1'b1;

  // ---------------------------------------------------------------------
  // Burst Mode Logic
  // ---------------------------------------------------------------------

  reg [1:0] prbs_ctr = 2'd0;

  always @(posedge gtwiz_userclk_tx_usrclk2_in) begin
    if (example_stimulus_reset_sync) begin
      prbs_ctr <= 2'd0;
      prbs_any_gen_en_int  <= 1'b0;
    end

    else begin
         if(prbs_ctr >= 2'd3)begin
```

```
47                      prbs_ctr <= 0;
48                      prbs_any_gen_en_int  <= ~prbs_any_gen_en_int;
49              end
50              else begin
51                      prbs_ctr <= prbs_ctr + 2'd1;
52              end
53        end
54      end
55
56    // -------------------------------------------------------------------
57    // PRBS generator block
58    // -------------------------------------------------------------------
59    // The prbs_any block, described in Xilinx Application Note 884 (XAPP884), "An
60    // Attribute-Programmable PRBS Generator and Checker", generates or checks a
61    // parameterizable PRBS sequence. Instantiate and parameterize a prbs_any block
62    // to generate a PRBS31 sequence with parallel data sized to the transmitter
63    // user data width.
64
65    gtwizard_ultrascale_1_prbs_any # (
66      .CHK_MODE    (0),
67      .INV_PATTERN (0),
68      .POLY_LENGHT (7),
69      .POLY_TAP    (6),
70      .NBITS       (32)
71    ) prbs_any_gen_inst (
72      .RST      (example_stimulus_reset_sync),
73      .CLK      (gtwiz_userclk_tx_usrclk2_in),
74      .DATA_IN  (32'b0),
75      .EN       (prbs_any_gen_en_int),
76      .DATA_OUT (txdata_out)
77    );
78
79
80  endmodule
```

B.3 Two Channel Burst Mode

```
module example_stimulus_two_channel (
  input  wire         gtwiz_reset_all_in,
  input  wire         gtwiz_userclk_tx_usrclk2_in,
  input  wire         gtwiz_userclk_tx_active_in,
  // input  wire         enable,
  output wire [31:0] txdata1_out,
  output wire [31:0] txdata2_out
);


  // -------------------------------------------------------------------
  // Reset synchronizer
  // -------------------------------------------------------------------

  // Synchronize the example stimulus reset condition into the txusrclk2 domain
  wire example_stimulus_reset_int = gtwiz_reset_all_in || ~gtwiz_userclk_tx_active_in;
  wire example_stimulus_reset_sync;

  (* DONT_TOUCH = "TRUE" *)
  gtwizard_ultrascale_1_example_reset_synchronizer
  example_stimulus_reset_synchronizer_inst (
    .clk_in  (gtwiz_userclk_tx_usrclk2_in),
    .rst_in  (example_stimulus_reset_int),
    .rst_out (example_stimulus_reset_sync)
  );


  // -------------------------------------------------------------------
  // PRBS generator output enable and sideband control generation
  // -------------------------------------------------------------------

  // For raw mode data transmission, the PRBS generator is always enabled
  reg prbs_any_gen_en_int = 1'b1;

  // -------------------------------------------------------------------
  // Burst Mode Logic
  // -------------------------------------------------------------------

  reg [1:0] prbs_ctr = 2'd0;
  reg select = 1'b0;
  reg [31:0] txdata1;
  reg [31:0] txdata2;
  wire [31:0] txdata_out;


  always @(posedge gtwiz_userclk_tx_usrclk2_in) begin
```

```
47      if (example_stimulus_reset_sync) begin
48        prbs_ctr <= 2'd0;
49        txdata1  <= 31'b0;
50        txdata2  <= 31'b0;
51      end
52
53      else begin
54           if(prbs_ctr >= 2'd3)begin
55                      prbs_ctr <= 0;
56                      select  <= ~select;
57               end
58               else begin
59                      prbs_ctr <= prbs_ctr + 2'd1;
60               end
61         end
62       end
63
64       always@(*) begin
65          if (select == 1'b1) begin
66              txdata1 = txdata_out;
67              txdata2 = 31'b0;
68          end else begin
69              txdata1 = 31'b0;
70              txdata2 = txdata_out;
71          end
72      end
73
74      assign txdata1_out = txdata1;
75      assign txdata2_out = txdata2;
76
77
78
79    // -------------------------------------------------------------------------
80    // PRBS generator block
81    // -------------------------------------------------------------------------
82
83    // The prbs_any block, described in Xilinx Application Note 884 (XAPP884), "An
84    // Attribute-Programmable PRBS Generator and Checker", generates or checks a
85    // parameterizable PRBS sequence. Instantiate and parameterize a prbs_any block
86    // to generate a PRBS31 sequence with parallel data sized to the transmitter
87    // user data width.
88
89    gtwizard_ultrascale_1_prbs_any # (
90      .CHK_MODE    (0),
91      .INV_PATTERN (0),
92      .POLY_LENGHT (7),
93      .POLY_TAP    (6),
94      .NBITS       (32)
95    ) prbs_any_gen_inst (
96      .RST      (example_stimulus_reset_sync),
```

```
97      .CLK      (gtwiz_userclk_tx_usrclk2_in),
98      .DATA_IN  (32'b0),
99      .EN       (prbs_any_gen_en_int),
100     .DATA_OUT (txdata_out)
101   );
102 endmodule
```

B.4 Burst Mode Checking

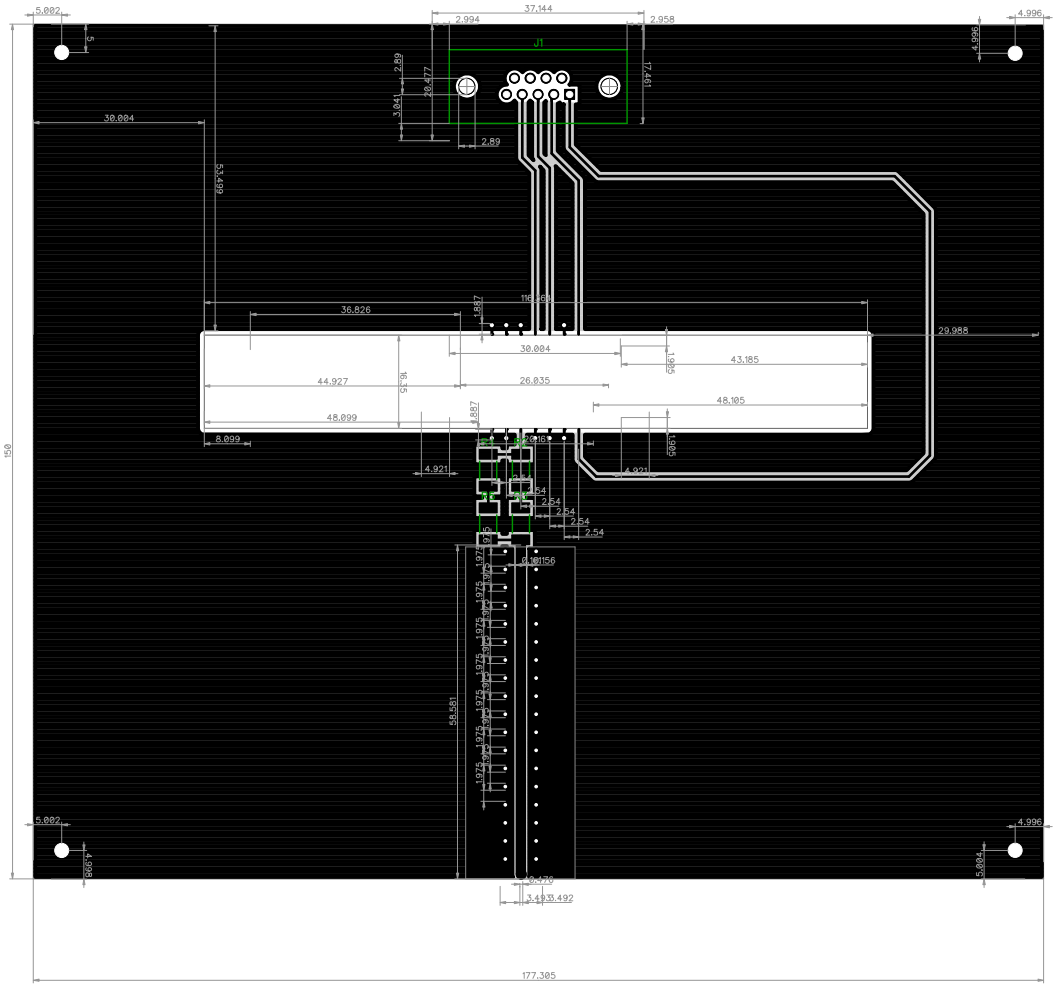
```
module gtwizard_ultrascale_1_example_checking_raw (
  input  wire         gtwiz_reset_all_in,
  input  wire         gtwiz_userclk_rx_usrclk2_in,
  input  wire         gtwiz_userclk_rx_active_in,
  input  wire [31:0] rxdata_in,
  output reg          prbs_match_out = 1'b0
);


  // -------------------------------------------------------------------------
  // Reset synchronizer
  // -------------------------------------------------------------------------

  // Synchronize the example stimulus reset condition into the rxusrclk2 domain
  wire example_checking_reset_int = gtwiz_reset_all_in || ~gtwiz_userclk_rx_active_in;
  wire example_checking_reset_sync;

  (* DONT_TOUCH = "TRUE" *)
  gtwizard_ultrascale_1_example_reset_synchronizer
  example_checking_reset_synchronizer_inst (
    .clk_in  (gtwiz_userclk_rx_usrclk2_in),
    .rst_in  (example_checking_reset_int),
    .rst_out (example_checking_reset_sync)
  );

  // -------------------------------------------------------------------------
  // PRBS checker enable and sideband control generation
  // -------------------------------------------------------------------------

  // For raw mode data reception, the PRBS checker is always enabled
  reg prbs_any_chk_en_int = 1'b1;

  // -------------------------------------------------------------------
  // Burst Mode Logic
  // -------------------------------------------------------------------

  reg [31:0] last_prbs = 32'b0;
  reg [31:0] checked_prbs = 32'b0;

  // AND gate to check if incoming data is all zeros
  always @(posedge gtwiz_userclk_rx_usrclk2_in) begin
      if (example_checking_reset_sync) begin
          last_prbs <= 1'b0;
          prbs_any_chk_en_int <= 1'b0;
      end
      // loads incoming data to register
```

```
47      else begin
48          last_prbs <= rxdata_in;
49      end
50
51      // checks if registered data is 0
52      if (~(|last_prbs) | (last_prbs == 32'b1)) begin
53          prbs_any_chk_en_int <= 1'b0;
54      end
55      else begin
56          prbs_any_chk_en_int <= 1'b1;
57          checked_prbs <= last_prbs;
58      end
59  end
60
61
62  // -------------------------------------------------------------------------
63  // PRBS checker block
64  // -------------------------------------------------------------------------
65
66  // The prbs_any block, described in xilinx application note 884 (xapp884), "an
67  // attribute-programmable prbs generator and checker", generates or checks a
68  // parameterizable prbs sequence. instantiate and parameterize a prbs_any block
69  // to check a prbs31 sequence with parallel data sized to the receiver user data
70  // width.
71
72  wire [31:0] prbs_any_chk_error_int;
73
74
75  gtwizard_ultrascale_1_prbs_any # (
76    .CHK_MODE    (1),
77    .INV_PATTERN (0),
78    .POLY_LENGHT (7),
79    .POLY_TAP    (6),
80    .NBITS       (32)
81  ) prbs_any_chk_inst (
82    .RST      (example_checking_reset_sync),
83    .CLK      (gtwiz_userclk_rx_usrclk2_in),
84    .DATA_IN  (checked_prbs),
85    .EN       (prbs_any_chk_en_int),
86    .DATA_OUT (prbs_any_chk_error_int)
87  );
88
89
90
91
92
93  // the prbs_any block indicates a match of the parallel prbs data when all
94  // data_out bits are 0. register the result of the nor function as the prbs
95  // match indicator.
96  always @(posedge gtwiz_userclk_rx_usrclk2_in) begin
```

```
97      if (example_checking_reset_sync)
98        prbs_match_out <= 1'b0;
99      else
100       prbs_match_out <= ~(|prbs_any_chk_error_int);
101   end
102 endmodule
```

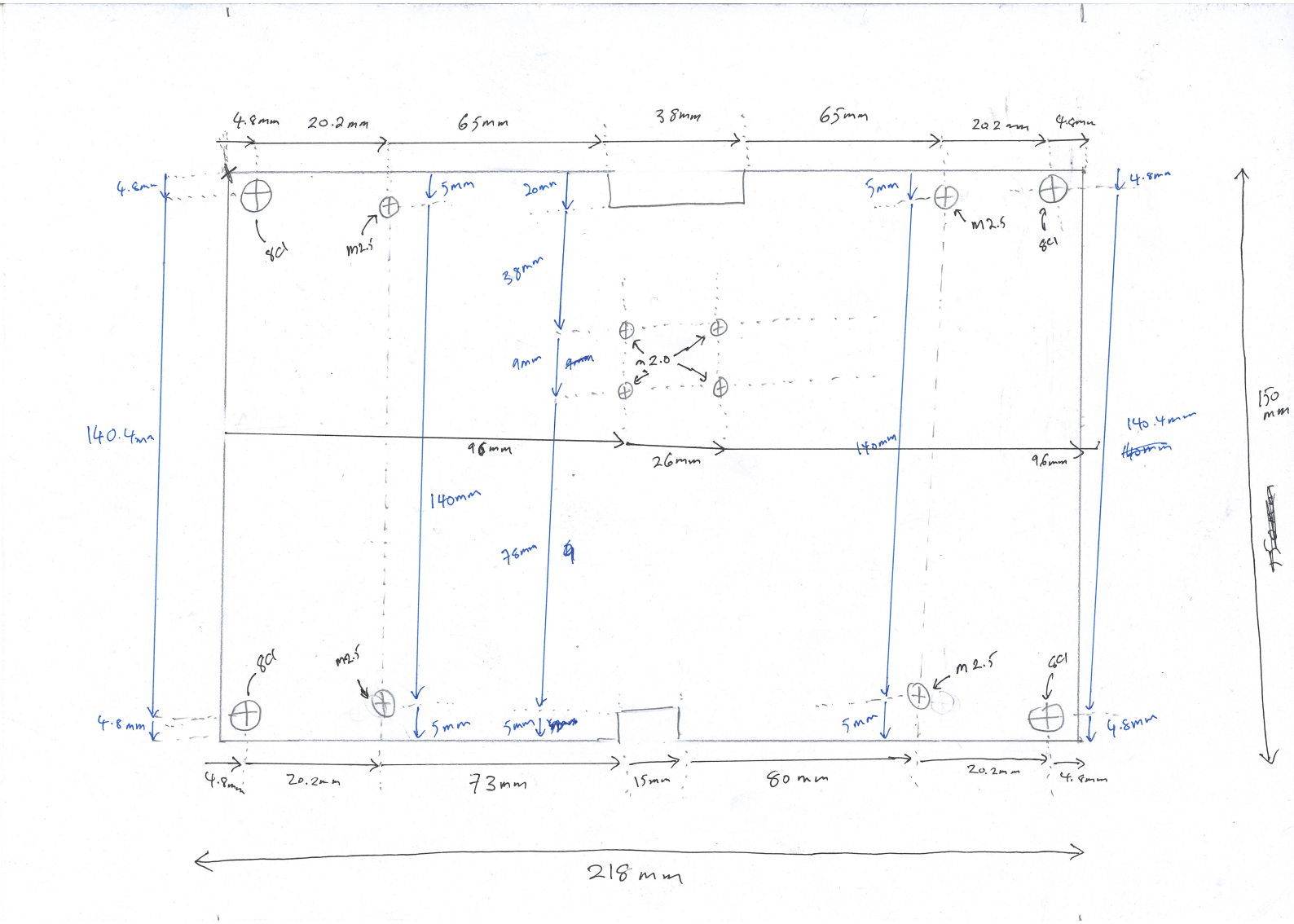
Appendix C

SOA PCB Design



Appendix D

Substrate Design



DECLARATION

I have read and understood the College and Department's statements and guidelines concerning plagiarism.

I declare that all material described in this report is all my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures and computer programs.

Name: Ammar Bin Shaqeel Ahmed ……

Signature:  ……

Date: 09/05/2020 ……