CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Waveform Interface for the Fiber-on-Chip Emulation Environment

Master's Thesis in Embedded Electronic System Design

Rafael Romón Sagredo

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's Thesis 2022

Waveform Interface for the Fiber-on-Chip Emulation Environment

Rafael Romón Sagredo

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Waveform Interface for the Fiber-on-Chip Emulation Environment
Rafael Romón Sagredo

Supervisor: Erik Börjeson, Department of Computer Science and Engineering
Examiner: Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Waveform Interface for the Fiber-on-Chip Emulation Environment
Rafael Romón Sagredo
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Testing digital signal processing (DSP) implementations for telecommunications is often conducted using digital emulation environments that generate synthetic transmissions based on channel models. However, digital emulation of channel models presents a series of limitations when testing DSPs and may not represent their actual performance in a working environment. This project aims to develop a waveform interface capable of storing and reconstructing optical fiber transmissions, allowing a real-time digital emulation environment to test DSP implementations with experimental data.

To this end, this project was based on the Chalmers Optical Fiber Channel Emulator (CHOICE), a VHDL implementation of the Fiber-on-Chip approach; which offers a way of testing DSP implementations using field-programmable gate array (FPGA) emulation of fiber-optic channels. This project resulted in a series of modules that enable the CHOICE environment to store experimental data on external double data rate (DDR) modules and generate continuous waveforms by stitching together short fiber transmissions.

The resulting system offers a novel way of testing DSP implementations for fiber optic communications systems with experimental data captured from optical transmissions, evaluating metrics like bit-error rate for large sample sizes at a higher speed than traditional fixed-point emulation environments.

Keywords: FPGA, DDR, DSP, embedded design, telecommunication, optical fiber, real time emulation.

Acknowledgements

I would like to thank my supervisor Erik Börjeson for his guidance and general support throughout this project. As well as my examiner Per Larsson-Edefors who originally had the idea behind this project, and helped me along the way.

I would also like to thank Ali Mirani and Magnus Karlsson from the Microtechnology and Nanoscience at the Chalmers University of Technology. For providing me with the experimental data I used to test the system, and the valuable discussions we had on the possible applications of the final implementation.

Rafael Romón Sagredo, Gothenburg, July 2022

Contents

1

Introduction

As the field of telecommunications moves towards higher throughput systems, the development of efficient digital signal processing (DSP) algorithms has become critical for embedded system designers [1]. The resulting implementations need to be thoroughly tested to ensure they comply with strict performance requirements, study their behavior over extended periods of time and produce accurate power estimations. This testing is traditionally conducted either with traditional fixed-point simulation environments or by conducting real-time optical communication experiments. However, fixed-point simulations are time consuming, while optical experiments require expensive equipment and suffer from a low degree of repeatability [2]. To circumvent these issues, there has been an increasing interest in developing real-time digital emulation environments [3].

One example of a real-time digital emulation environments is the fiber-on-chip (FoC) approach described in [2], and its VHDL implementation, the Chalmers Optical Fiber Channel Emulator or CHOICE environment, which serves as the basis for this project [4]. The FoC approach proposes testing DSP implementations using FPGA emulation of fiber-optic channels, allowing for faster DSP testing than traditional fixed-point simulation environments, while maintaining a higher degree of repeatability than optical experiments.

1.1 Problem Statement

Digital emulation presents a series of limitations when testing DSPs and may not represent their actual performance in a working environment. Real transmission data are not constrained by a fixed-point sampled environment and may be affected by additional fiber-effects, leading to slight differences in metrics like bit error rate (BER) and power dissipation.

To circumvent these limitations, this project aims to develop a waveform interface for the CHOICE environment that would enable it to store and reconstruct waveforms obtained in optical experiments to test DSP circuit implementations. The intended outcome of this project is a series of modules for the CHOICE environment enabling it to receive data from optical experiments from a PC, and perform DSP evaluations analogous to test runs performed using synthetic data generated in digital an emulation environment such as FoC.

1.2 Objectives

To achieve its aim, the development of this project was divided into three main objectives, each in turn divided into several sub-goals:

1. Transmit optical experiment data to an FPGA board, storing it in an onboard memory module.
   (a) Develop a program to read files with optical experiment data and apply any preprocessing techniques needed to use these data inside the emulation environment.
   (b) Modify the program to transmit the resulting data to an FPGA board.
   (c) Evaluate the performance of the current FoC communication interface between FPGA and PC.
   (d) Develop a memory controller to store the received data into an onboard memory module.
2. Read the data stored in the onboard memory module and perform a test run on the target DSP.
   (a) Modify the memory controller to read the optical experiment data from the onboard memory module.
   (b) Integrate the memory controller in the testing pipeline of the CHOICE environment.
3. Integrate the memory module into the CHOICE environment allowing it to transmit the reconstructed waveform to the target DSP.
   (a) Implement a waveform stitching module to generate a continuous waveform from the data bursts captured during optical experiments.
   (b) Apply the preprocessing techniques needed to perform the waveform stitching operation with the stored optical experiment data.
   (c) Integrate the waveform stitching module in the testing pipeline of the CHOICE environment.

Once the three main objectives have been reached, different DSP functions will be tested using randomly generated on-chip data, data obtained from optical experiments and data generated using MATLAB simulations. This evaluation aims to compare DSP performance when using different versions of the CHOICE environment and to ascertain the impact of using synthetic data when testing for DSP performance.

1.3 Limitations

- This project does not aim to modify any of the core components of the CHOICE environment. The resulting modules will be designed to be implemented with minimal impact to the rest of the FoC project.
- This project does not aim to replace but to complement the synthetic waveform generation module currently used in the CHOICE environment. The implemented module will offer an alternative to using randomly generated waveforms to test DSPs.

1.4 Thesis Outline

This thesis starts by describing the background theory for some of the concepts behind this project in Chapter 2, followed by giving some context to the development of this project in Chapter 3. Moving on, Chapter 4 describes how the waveform interface was designed and implemented with Chapter 5 presenting the results of this implementation, in the way of a brief performance evaluation and a use-case example. Finally, Chapter 6 concludes with a brief discussion on the outcome of this project, some of the major design decisions that were taken and the future work that should be conducted on the waveform interface.

2

Theory

This chapter describes the background theory of this project; mainly optical fiber communication, FPGAs, digital emulation environments and the Fiber-on-Chip approach that serves as the basis of this project.

2.1 Optical Fiber Communication

Optical fiber communication refers to the method of transmitting information by sending pulses of light through an optical fiber. Optical fiber communications as we know it is the result of two scientific developments [5]; the invention of laser technology in the 1960s, allowing unguided optical communication to transmit large amounts of information in zero noise environments, and the invention of pure silica fibers in 1970, allowing optical communication to become a viable alternative to copper cable systems. Thanks to its advantages over electrical transmissions, optical fiber systems have become the backbone of the telecommunications industry [5].

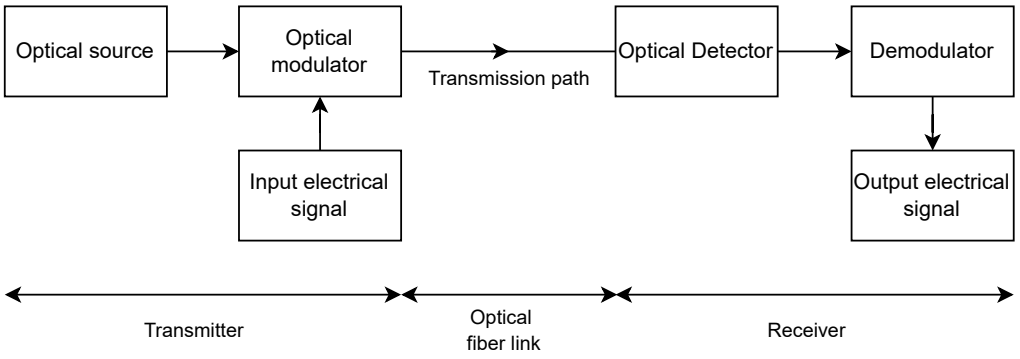


Figure 2.1: Basic optical fiber communication system. Adapted from [6].

Fig. 2.1 shows a block diagram describing a basic optical fiber communication system. The transmitter is comprised of an optical modulator that takes an electrical input signal and converts it into optical pulses. These pulses are then sent through an optical fiber link to an optical receiver. The optical receiver uses photo detectors to convert the optical signal back to electrical pulses [6].

2.1.1 Modulation

Modulation is the method of embedding information within a carrier signal by modifying some of its properties before sending it over a physical transmission channel. Most modulation methods used in the field of telecommunications are based on the variation of the amplitude, frequency, or phase of the carrier signal [7]. In this subsection, we will introduce quadrature amplitude modulation (QAM) since it is the format used in this project.

QAM is a combination of Phase Shift Keying (PSK) and Amplitude Shift Keying (ASK), where constellation points are arranged in the form of a square in a polar coordinate plot. For example, 16-QAM represents 4 bits in the form of 16 possible constellation points, see Fig. 2.2. QAM is a higly popular method due in part to the simplicity of its demodulation, since decision boundaries can be easily drawn in the constellation diagram [7].

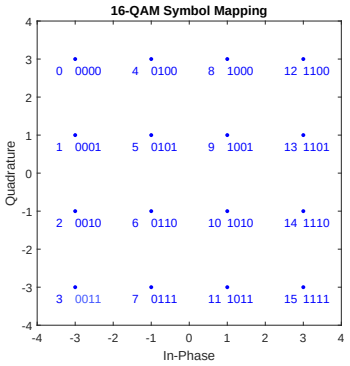


Figure 2.2: Constellation diagram for 16-QAM.

2.1.2 The Optical Fiber Medium

Optical fibers are made up of two layers of different types of glass or plastic; an inner layer called core surrounded by an outer layer called cladding. Usually, a protective coating surrounds the cladding to improve its resistance to mechanical stresses during deployment [5]. There are two main ways to classify optical fibers; based on their refractive index profile or the number of modes of light they allow to pass through [5], see Table 2.1 and Table 2.2.

Table 2.1: Fiber classification based on their refractive index.

Type	Characteristics
Step-index	• Characterized by a uniform refractive index of the core. • Light propagates as meridional rays that cross the fiber axis on every reflection with the cladding boundary.
Graded index	• Refractive index designed to vary in a parabolic manner, with its maximum value at the center of the fiber core. • Light propagates in the form of helical rays that do not cross the fiber axis.

Table 2.2: Fiber classification based on the number of modes of light.

Type	Characteristics
Single-Mode (SMF)	• Single strand of glass fiber. • Narrow core diameter between 8.3 and 10 µm. • Allows a single mode of light at 1310 or 1550 nm. • Low transmission loss and dispersion. • High transmission bandwidths.
Multi-Mode (MMF)	• Diameter greater than on SMF, usually 50, 62.5, or 100 µm. • Allow for multiple modes to be transmitted simultaneously. • Significant distortion at more than 1 km in length. • Lower cost of implementation.

Fig. 2.3 shows a graphical representation of the characteristics mentioned in Table 2.1 and Table 2.2. Fiber classification is not limited to these two criteria. Other types of fibers are used for specific applications, for example, to mitigate the effects of dispersion in optical communication systems with dispersion-shifted fibers or dispersion compensating fibers.

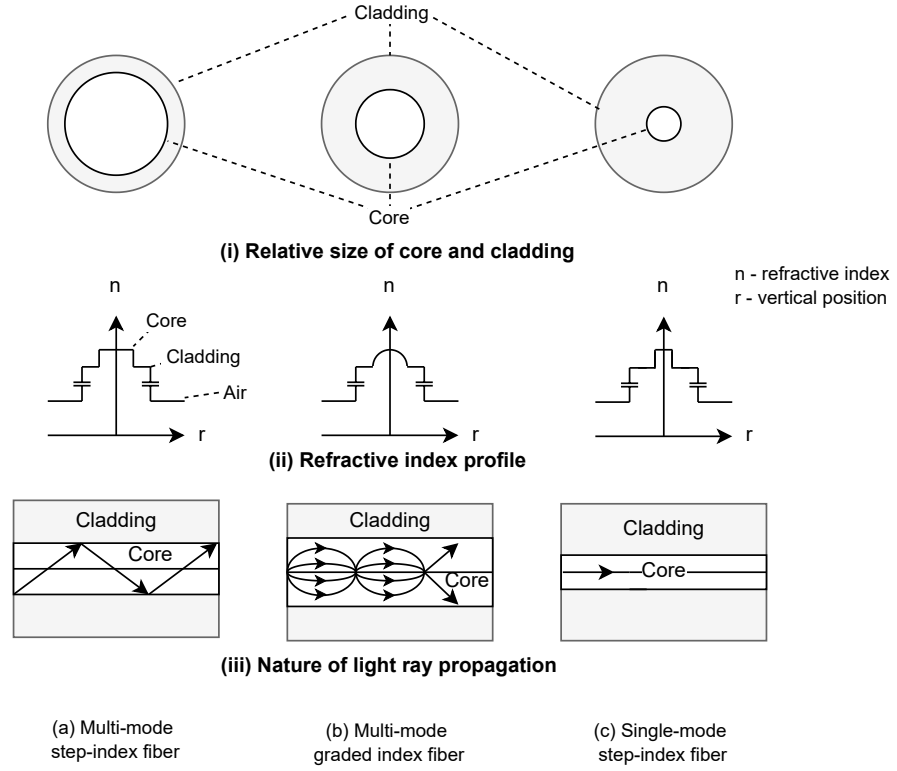


Figure 2.3: Characteristics of different types of fibers. Adapted from [6].

2.1.3 Noise and Channel Impairments

As with most analog systems, a signal transmitted through an optical fiber communication system is subjected to noise caused by optical and electronic phenomena, see Fig. 2.4.

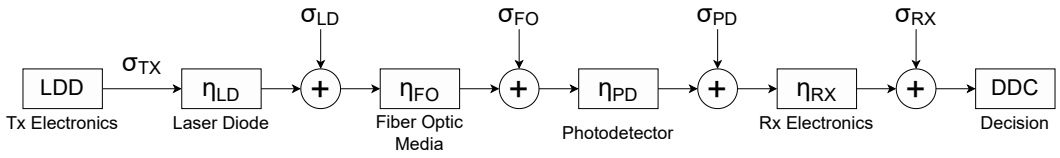


Figure 2.4: Signal path in optical fiber communication systems. Adapted from [8].

The primary source of noise in optical fiber communication systems is the light produced by spontaneous emissions and amplified by optical amplifiers, often referred to as amplified spontaneous emissions or ASE. ASE noise and most of the noise generated by the electronic components can be adequately modeled as additive white Gaussian noise (AWGN) [9].

Another important source of performance degradation is the attenuation of the light traveling through an optical fiber [5]. The primary sources of attenuation are:

- **Absorption**:
  Photons are absorbed by the fiber media and converted to heat, resulting in a loss of signal energy.
- **Scattering**:
  Microscopic variations in the transmission medium redirect the light beam onto the cladding, resulting in a loss of signal energy due to unnecessary reflections.
- **Dispersion**:
  Dispersion broadens the pulse width of signals sent through optical fibers, resulting in an overlap at the receiver and reducing signal quality, as shown in Fig. 2.5. Dispersion is caused by the dependence of the refractive index of the fiber material on the wavelength of the carrier [6].

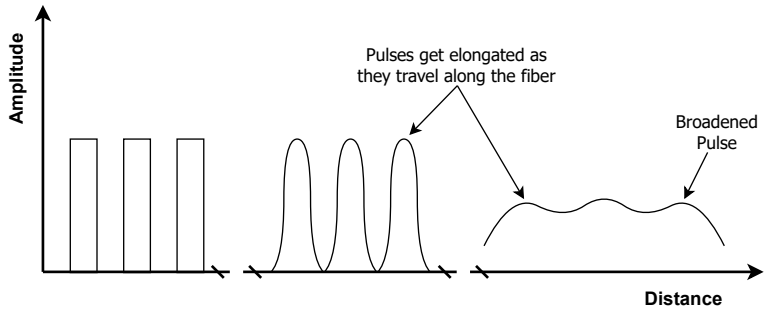


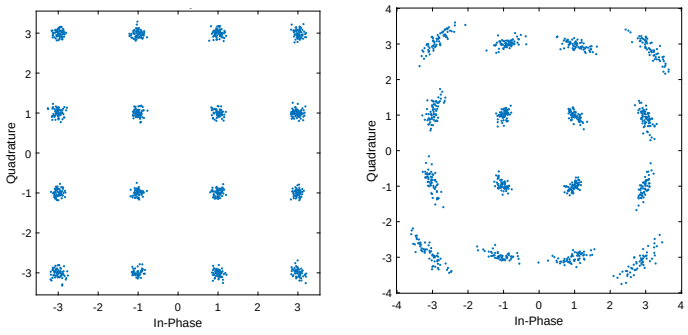
Figure 2.5: Effects of dispersion on signal integrity. Adapted from [5].

Dispersion mechanisms are divided into two categories, intermodal and intramodal dispersion. Intermodal dispersion, also called multi-mode dispersion, occurs due to the difference in group velocity between modes at a single frequency and results in different modes arriving at the exit point of the fiber at different times [5]. One of the most common forms on multi-mode dispersion is polarization mode dispersion and occurs due to imperfections in the fiber media [10].

On the other hand, intramodal dispersion can take place in all types of fiber and is caused by two phenomena:

- **Material dispersion**:
  Material dispersion is caused by the variation of the refractive index of the core material with the wavelength or frequency of the light source.
- **Waveguide dispersion or chromatic dispersion**:
  Waveguide dispersion is only relevant in single-mode fibers and is caused by the difference in velocity between the rays of light traveling within the fiber core and fiber cladding. Different wavelengths travel at a different speed within the optical fiber, causing the light pulse to spread at the receiver side [10]. Waveguide dispersion can be mitigated by using dispersion-shifted fibers that enable zero dispersion at a specific wavelenght by inserting dispersion compensating [5] or compensated by DSPs in the receiver.

Finally, another important source of performance degradation in optical communications is non-linear phenomena. One example would be the introduction of unwanted phase fluctuations or phase noise, by the coupling between Kerr nonlinearities of the fiber and ASE [11], as shown in Fig. 2.6. Modeling non-linear noise sources is one of the biggest challenges for noise analysis and simulation for optical fiber communications [8].



(a) Original signal

(b) Rotated symbols

Figure 2.6: Effects of phase noise on a 16-QAM signal.

With the adoption of coherent optical communication schemes at the beginning of the century, many of these transmission impairments are compensated at the receiver by electronic DSP components [12]. For instance, a carrier phase recovery algorithm like blind phase search (BPS) can be used to remove the phase noise introduced by phase fluctuations of the carrier and local oscillators, non-linear effects, etc.

In BPS each input symbol is rotated with a number of test phases, with the recovered phase being the one that results in the minimum distance between the rotated input symbols and the closest constellation point. In order to reduce the impact of white noise, a sliding-window average is used to select the minimum distance. Finally the transmitted symbol is recovered by using the complex conjugate of the recovered phase to rotate the input symbol [13]. Although BPS offers a non-data-aided solution for CPR, it is susceptible to cycle-slip (CS) errors where the received symbols are rotated by multiples of $\pi/2$ [14].

2.2 Field-programmable Gate Array

A field-programmable gate array or FPGA is a programmable logic device, a type of integrated circuit that can be used to implement digital logic circuits. Fig. 2.7, shows an overview of an island-style FPGA. FPGAs are comprised of three major components:

- Logic blocks (LB) that perform logic operations
- Input/output blocks (IOB) that serve as the interfaces of the FPGA board
- Wiring elements in the form of switch blocks (SB) and connection blocks (CB) that route the other components based on a netlist to achieve the user-defined logic circuits.

Most commercial FPGAs also include additional circuits for specific functions such as block memories or multiplexers [15].

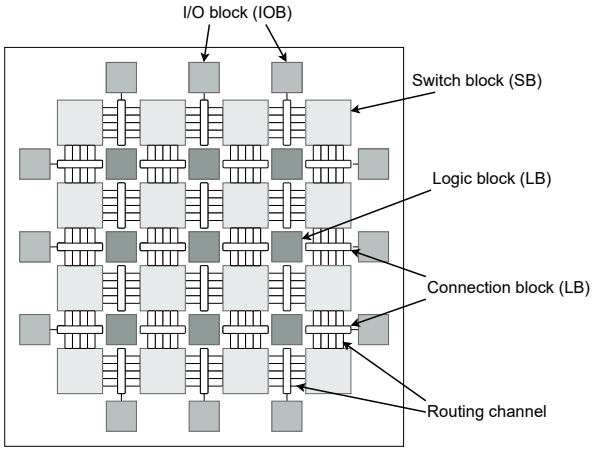


Figure 2.7: Overview of a island-style FPGA. Adapted from [15].

In a typical FPGA design flow a designer first defines the different logic circuits as register transfer level (RTL) abstractions described in a hardware description language (HDL), and a computer program then generates logic and sequential circuits adapted from the RTL description in a process called logic synthesis, producing a netlist. The netlist is tailored to the specific hardware resources available in the FPGA, in a process called technology mapping, using a constraint file. Finally, after place and route operations, a bitstream file is generated and uploaded into the FPGA.

Thanks to recent improvements in semiconductor technology many industries are starting to see FPGAs as an alternative to application-specific integrated circuits (ASIC), since they allow for a higher-level of flexibility after fabrication [16]. Nowadays, FPGAs are used in fields like high-performance computing, network processing, big data processing, etc.

2.2.1 Memory Resources

Since memory is an important aspect of this project, this subsection will give a brief overview of the two main memory types used in FPGA implementations: block memory and double data rate synchronous dynamic random-access memory (DDR SDRAM).

Block memory refers to a continuous chunk of memory. There are two main types of memory that will be considering for this project, block read-only memory (BROM) and block random-access memory (BRAM). BROM memory cannot be modified, and it is normally used to store system variables or lookup tables (LUTs). Since BRAM is volatile, meaning it is cleared when powered off, it is usually used for run-time variables.

The first memory design choices are word and memory size. Word size refers to the number of bits in every individual unit of storage given a memory address. Memory size is determined by the word size (width) and number of words (depth), see Fig. 2.8.

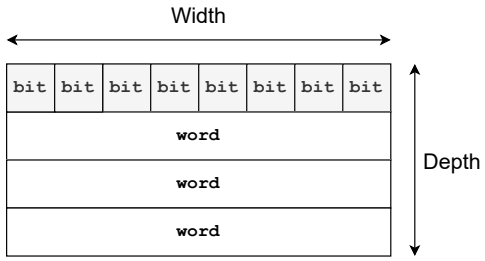


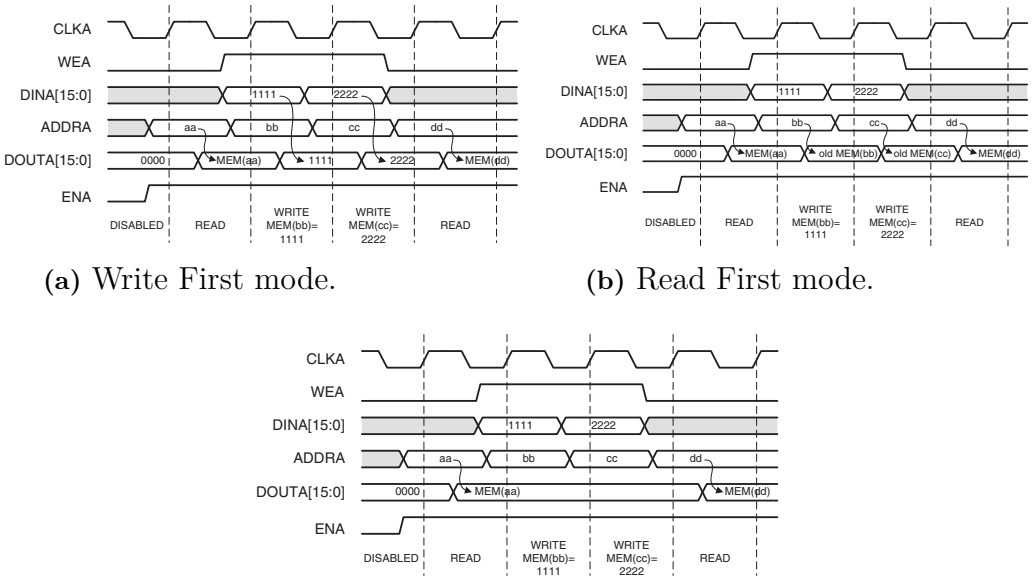
Figure 2.8: Block memory structure.

The second important design choice that must be made is the port configuration. Both BRAM and BROM can be implemented as single-port, simple dual-port or true dual-port. Single-port and dual-port denote the number of ports of the memory, while the difference between simple and true dual-port memory is the operations allowed on each port.

Simple dual-port has one port for read operations and the other one for write operations, whereas in true dual-port RAM both ports support both read and write (R/W) operations. Both types of dual-port BRAM are often present onboard FPGAs as look-up table (LUT) RAM.

One last thing to consider when using BRAM is to select the appropriate operating mode. This last decision is only important for systems in which we want to read and write to the BRAM simultaneously, as it determines the behavior of the `READ` port during a write operation, potentially helping to avoid writing collisions. Xilinx 7 series FPGAs offer three different operating modes [17], as shown in Fig. 2.9:

- Write First Mode: Data written to memory is at the same time driven out the `READ` port.
- Read First Mode: The `READ` port outputs the data stored in the memory before the `WRITE` operation.
- No Change Mode: Data being driven out of the `READ` port remains unchanged during the `WRITE` operation.



(a) Write First mode.

(b) Read First mode.

(c) No Change mode.

Figure 2.9: Timing diagram for the different modes of operation [18].

DDR SDRAM refers to double data rate synchronous dynamic RAM, double data rate means that the SDRAM transmits data along the rising and falling edge of clock. This results in DDR SDRAM having double the performance of SDRAM memory at the same core operating frequency. Subsequent versions of the DDR standard like DDR3, further increase the performance of DDR SDRAM over SDRAM [19]. Unlike block memory, DDR SDRAM is an external component located on the circuit board outside of the FPGA, so its implementation will require using some of the IO ports of the FPGA.

Table 2.3: Relative bandwidth and storage capacity by type of memory resource.

Resource	Bandwith	Storage Capacity
LUT RAM	high	low
BRAM	medium	medium
DDR SDRAM	low	high

Table 2.3 shows a comparison of bandwidth and storage capacity between each memory type. When implementing high-throughput memory-intensive systems, DDR SDRAM is often used as a low-bandwidth bulk data storage, with BRAM and LUT RAM as higher-bandwidth cache memories throughout the system [20].

2.3 Real-time Digital Emulation Environments

One method to test hardware implementations is MATLAB-HDL co-simulation, where experimental data is generated using a software model of the hardware, and then imported into MATLAB for analysis [21]. This method is able to produce accurate results, however, the simulations become extremely time-consuming when testing with large sample sizes.

There has been an increasing interest in developing real-time digital emulation environments to reduce the time needed to test hardware implementations. These environments are often implemented using FPGAs since their reconfigurability and fast computing speeds make them well suited for real-time applications. The following are some examples of real-time digital emulation environments for telecommunication applications:

- In [22], a Wireless Open-Access Research Platform (WARP) based testbed for DSP applications is presented. The WARP is an FPGA-based radio platform for prototyping radio networks. The architecture presented in [22] turns WARP into a capture and playback device for RF transmissions, allowing the designer to offload them into a computer for further analysis or to be processed by custom DSP algorithms implemented in the FPGA.
- In [23], Zhao et al. introduce a waveform playback system for radar applications. The proposed architecture makes use of two FPGAs connected to a disk array server for storage. Waveform data stored in the disk array server are used by the FPGAs for waveform playback, with one board dedicated to wide-band transmissions and the other one to narrow-band transmissions. The waveforms reconstructed by the FPGAs can then be used for real-time digital signal processing tests.
- Val et al. [3] present an FPGA-based wide-band channel emulator intended for early evaluation of wireless sensor and actuator networks. In the proposed architecture, an FPGA board serves as the bridge between two RF transceivers emulating a wireless sensor network (WSN). The channel model used within the board can simulate static and dynamic channels using AWGN, multipath, and Doppler fading generators.

2.3.1 Fiber-on-Chip Approach

The Fiber-on-Chip or FoC approach introduced in [2] proposes using ASICs or FPGA circuits to emulate real-time optical communication experiments. This approach offers a faster analysis of DSP implementations than traditional fixed-point simulation and a higher degree of repeatability than optical fiber testbeds.

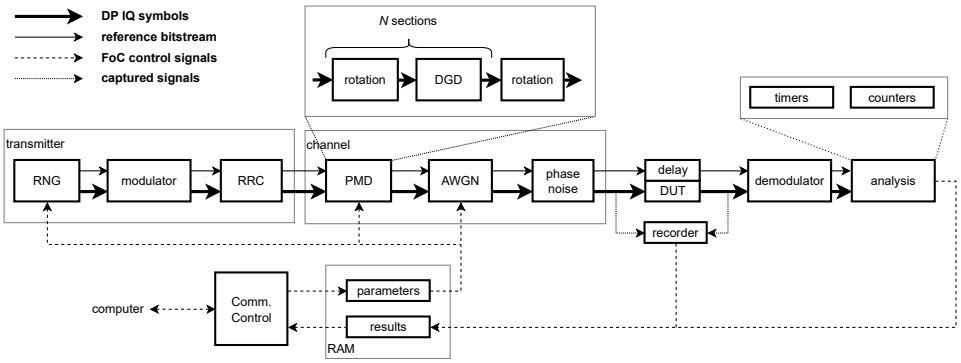


Figure 2.10: Block diagram of an example FoC system used to evaluate the DSP under test (DUT). Adapted from [2].

Fig. 2.10 shows the block diagram of an example system. In the transmitter block, an input bitstream is generated by a random number generator (RNG), modulated to the selected modulation scheme, interpolated, and shaped in a root-raised cosine (RRC) filter. The resulting signals are then passed through digital channel models that introduce noise and simulate phenomena like polarization-mode dispersion. The signals are then passed through an HDL implementation of the DSP and get demodulated.

The *channel* block shown in Fig. 2.10 introduces three types of channel impairments:

- AWGN generated by a Gaussian noise generator (GNG) IP Core, based on [24].
- Phase noise is modelled as a Wiener process $\theta_i = \theta_{i-1} + \Delta_i$ and makes use of the same GNG to generate Δ_i [1].
- Polarization-mode dispersion (PMD) generated using the digital PMD emulator introduced in [25].

Finally, in an analysis block, the resulting bit-stream is compared to the original to detect bit errors, cycle slips, etc. This series of test circuits allows the FoC system to perform continuous and autonomous evaluation of a DSP implementation. Additionally, the resulting data can be transmitted to a computer through an UART interface.

3

Methods

This chapter will give some context for the development of this project; going over the acquisition of experimental data and other resources used.

3.1 Data Acquisition

To provide the implemented waveform interface with data for testing and validation, the Department of Microtechnology and Nanoscience (MC2) at Chalmers University of Technology provided a series of fiber transmissions captured from a back-to-back link in an optical testbench.

These transmissions consist of a series of frames made up of 4QAM pilots and a 16QAM payload. Each transmitted frame consists of 2^{15} symbols, starting with a 1024 symbol 4QAM pilot sequence followed by the 16QAM payload, with a 4QAM phase pilot introduced every 32 symbols, see Fig. 3.1.

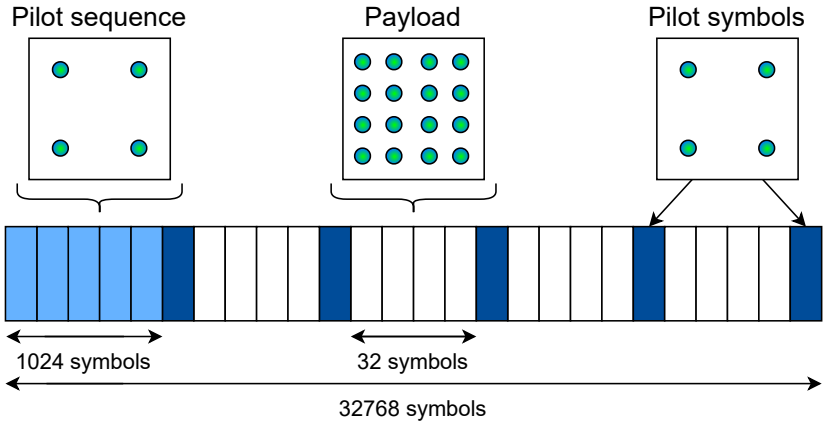


Figure 3.1: Example of a transmitted frame.

Fig. 3.2 shows a diagram of the testbench used to capture the transmissions: A transmitter using a dual-polarization in-phase and quadrature (DP-IQ) optical modulator sends electrical pulses generated in an arbitrary waveform generator (AWG) through polarization-maintaining fibers (PMF) to a coherent receiver. The receiver shifts any phase and amplitude fluctuations on the optical carrier to a carrier at an electronic frequency, while a set of photoreceivers converts them to electrical pulses. Finally, a Tektronix DPO733045K optical oscilloscope captures the resulting electrical signals.

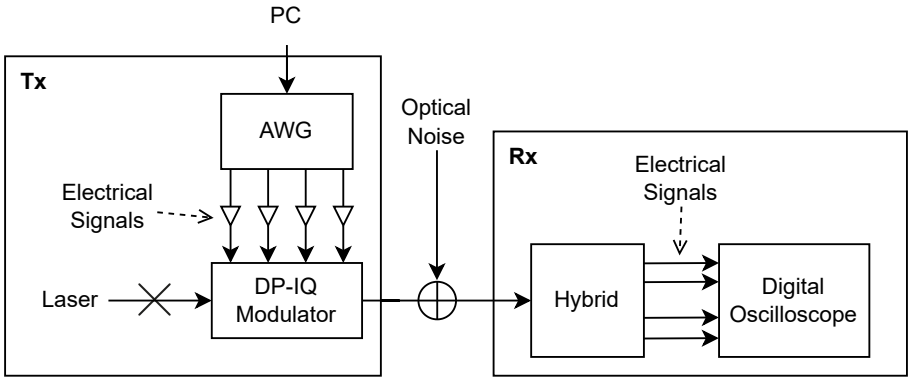


Figure 3.2: Optical testbench diagram.

The digital oscilloscope is then connected to a PC to retrieve the captured data stored in '.mat' binary data container format. This setup produces transmission with a relatively high signal-to-noise ratio (SNR), with the main sources of noise being ASE from the optical amplifiers and the electrical components in the transmitter and receiver. Since these transmissions are captured from a back-to-back link, signal attenuation and noises caused by non-linear phenomena are negligible.

Since the recorded data may present features not expected by the DSP tested in the waveform interface, some preprocessing may need to be handled off-board. For example, for the use-case example presented in Section 5.3; resampling, IQ imbalance compensation, frame synchronization and equalization were all performed using the QAMPy Python module developed by MC2 [26] before sending the data to the FPGA.

3.2 Target FPGAs

The KC705 evaluation board [27] was chosen to implement the developed waveform interface, with the VC709 evaluation board [28] selected as a higher performance alternative.

Both boards contain Xilinx 7-series FPGAs and offer similar features, the main distinction being that the KC705 is from the Kintex family of FPGAs, designed to offer a good price-performance ratio. In contrast, the VC709 belongs to the Virtex family, optimized for high system performance, and has a higher amount of FPGA resources available [17].

Table 3.1: Memory resource available for each board.

Memory Type	KC705	VC709
36 Kb BRAM Blocks	445	1,470
DDR3 Memory	1 GB	2x4GB
BPI flash memory	128 Mb	128 Mb
SPI flash memory	128 Mb	-

The main advantage the VC709 board offers to this project is a higher quantity of memory resources, as shown in Table 3.1. This allows, the waveform interface to store more data in DDR memory and to instantiate a larger number of cache memories, enabling longer continuous tests on the target DSP.

4

Design and Implementation

This chapter will go over the design decisions that were taken for this project, and describe how the system was implemented for the FPGA boards introduced in Section 3.2.

4.1 System Overview

Since the waveform interface is not intended to fully replace the synthetic data generation of the CHOICE environment, its functionality will be implemented as part of a new mode of operation, referred to as *memory* mode. In contrast, the system as described in Section 2.3.1 will be referred to as *synthetic* mode.

In order to encapsulate the new functionality introduced to the CHOICE environment, two modules have been implemented: a *control* module that houses control entities and memory controllers; and a *testbench* module that houses the DSP component along with other entities to be used during testing. Fig. 4.1 shows the block diagram for an example FoC system running exclusively in *memory* mode:

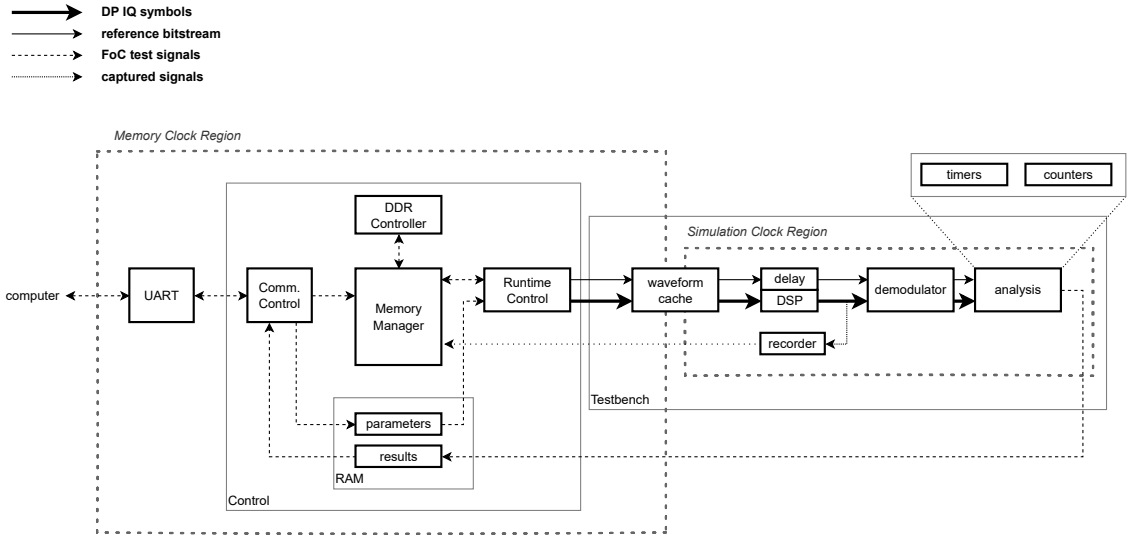


Figure 4.1: Block diagram of an FoC system in *memory* mode.

An important thing to note in Fig. 4.1 is the presence of two clock regions. This is a result of the need to run any entity interacting with the DDR memory at a 200 MHz clock frequency, see Section 4.2. Running the whole system at a fixed frequency would limit the DSP components that the system can test, since some designs may cause timing violations at this frequency. In order to get around this issue, a separate clock region with a configurable clock frequency was introduced for the DSP and other testbench entities.

4.2 Control Module

The control module, marked in Fig. 4.1, governs the behavior of the FoC system, and it is made up of the following entities:

- A DDR controller to enable read and write operations from the external DDR module.
- A communications controller that enables remote operation through a UART interface.
- A testbench controller used to load the cache memory used during testing, and control the DSP tests.
- A memory manager in charge of handling the memory requests issued by the rest of the components, ensuring proper flow control with the DDR controller.

4.2.1 DDR Controller and MIG

Using BRAM memory is one of the most common ways of storing data in FPGA implementations. As a result, BRAM primitives often end up being a limiting factor when implementing memory intensive designs. The implemented waveform interface relies instead on external DDR modules to store the full-length data transmissions, since they would otherwise require most of the BRAM primitive available on our target FPGAs. In addition to this long-term storage module, smaller BRAM entities will be used as cache memory in the testbench module, described in Section 4.3.

The Memory Interface Generator or MIG IP block from Xilinx was selected to connect the user logic with the available DDR module, improving portability and reducing the complexity of the implementation [29]. The MIG CORE generates a pre-engineered controller and a physical layer (PHY) to interface 7-series FPGA user designs with DDR3 SDRAM devices. The designer can configure the target FPGA and other DDR parameters through the Vivado software suite, allowing the implementation of a generic memory controller that works on multiple FPGA boards, see Fig 4.2.

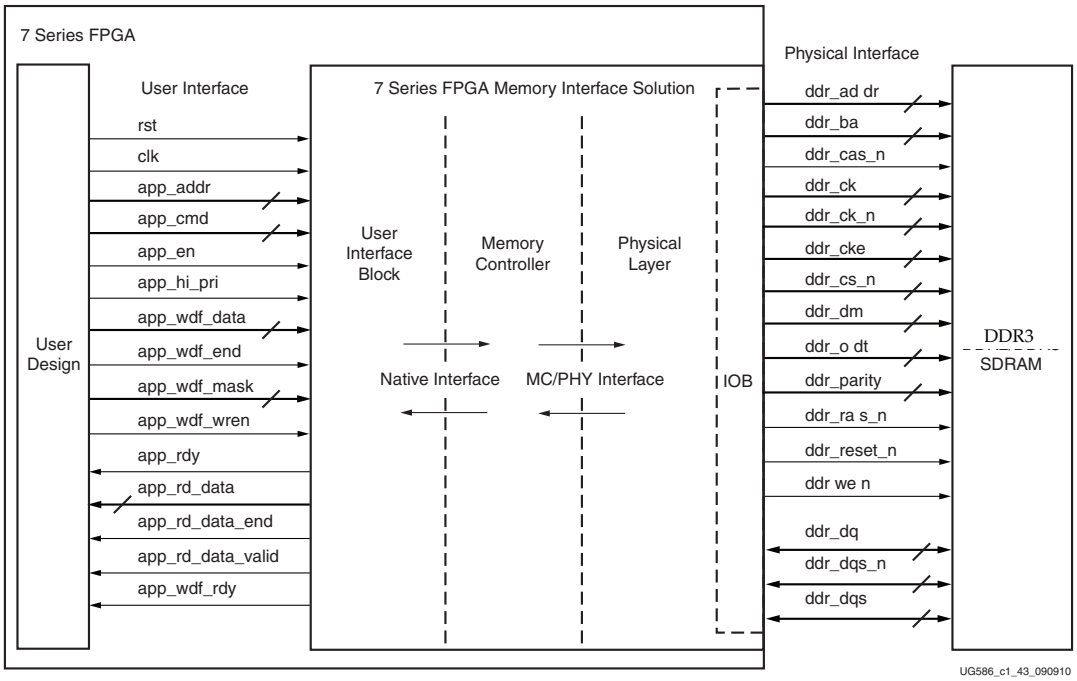


Figure 4.2: Xilinx 7-series FPGAs memory interface solution [29].

The generated PHY requires that a phase-locked loop (PLL) module generates the clocks used by the FPGA user logic and memory module. The MIG interface will take a clock signal either 2x or 4x the memory clock frequency to use as reference when generating these clock sources. A 2:1 ratio will result in lower latency, while a 4:1 ratio is required to achieve the highest data rates [29]. In this design a 200 MHz clock signal was used, achieving a 4:1 clock ratio with the 800 MHz needed by the DDR modules on the selected boards.

The MIG offers two ways of communication with the onboard DDR module, a slave interface using AXI4 protocol and a native user interface. For this implementation, the native user interface was selected, avoiding further reliance on external IP blocks. To simplify addressing the external DDR module, the user interface (UI) presents a flat address space, where physical memory row, bank, and column addresses are assigned different sections of a single address signal [29], see Fig. 4.3.

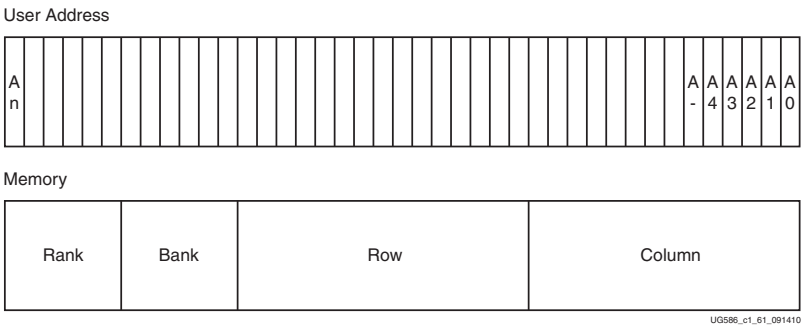


Figure 4.3: Memory address mapping in the UI [29].

In order to send write or read commands, the UI offers a command path along with an address and enable signal. The controller will accept a new command when the `app_rdy` signal is high. If the `app_rdy` signal is low when a command arrives the corresponding `app_addr` signal must be maintained until the `app_rdy` signal is asserted, Fig. 4.4.

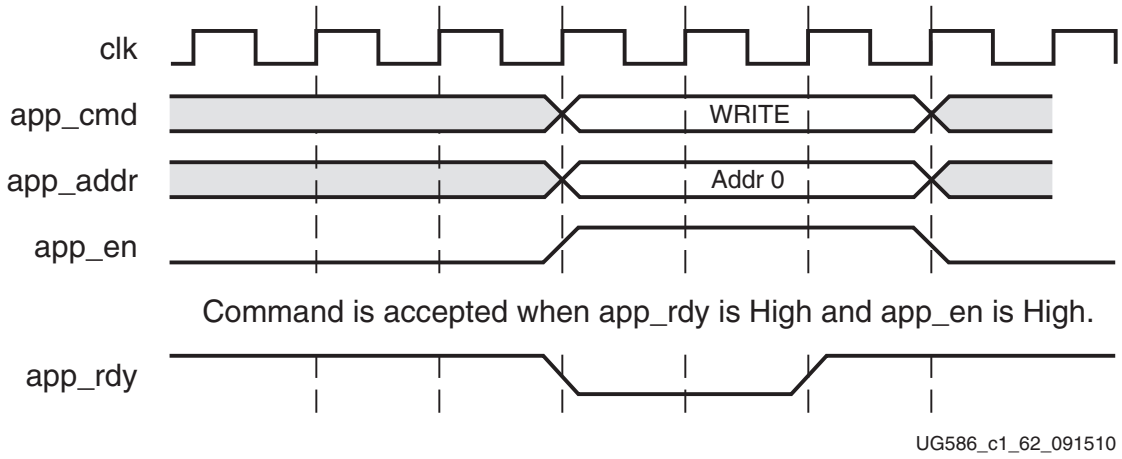


Figure 4.4: Timing diagram for the UI command path [29].

Read commands are issued by setting the `mig_cmd` signal to `001`. Read data is returned by the UI order requested and valid when `app_rd_data_valid` is asserted, as shown in Fig. 4.5.

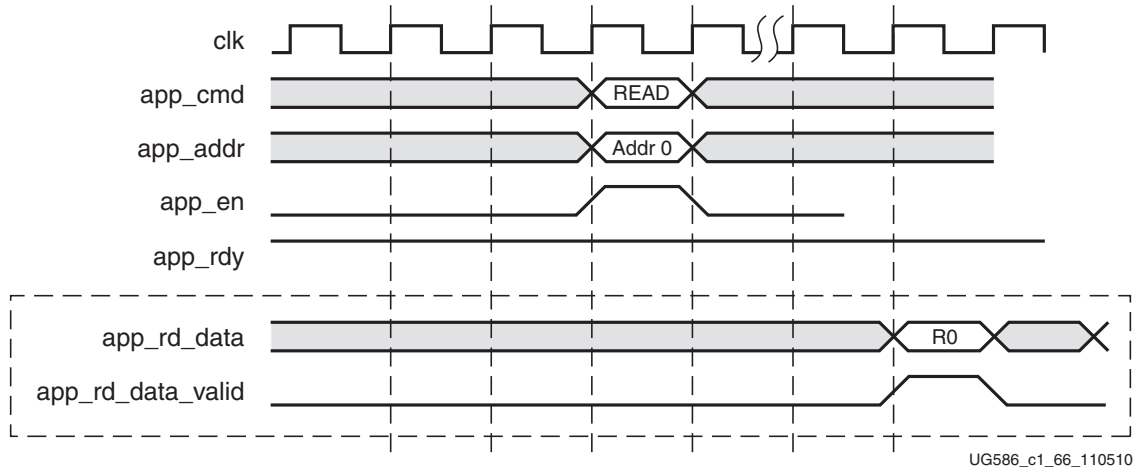


Figure 4.5: Timing diagram for the UI read path [29].

Write commands are issued by setting the `mig_cmd` signal to `000`. Written data will be stored in a FIFO when the `app_wdf_ready` signal is high and `app_wdf_wren` is asserted. The `app_wdf_end` is used to indicate the last cycle of data in the `app_wdf_data` signal. However, when using 4:1 clock ratio this signal is redundant since the `app_wdf_wren` signal will be asserted on the last cycle of data.

In a similar way to the `app_addr` signal, this `app_wdf_wren` must be maintained until `app_wdf_ready` is high. The controller will accept the write command under the following circumstances, as shown in Fig. 4.6:

- Write data presented along with the corresponding write command.
- Write data presented before the corresponding write command.
- Write data presented after the corresponding write command, not exceeding a limitation of two clock cycles.

MIG also supports back-to-back writing, by keeping the `app_wdf_wren` and `app_wdf_end` asserted while changing the `app_addr` every clock cycle [29]. However, this option is not used in the implemented design.

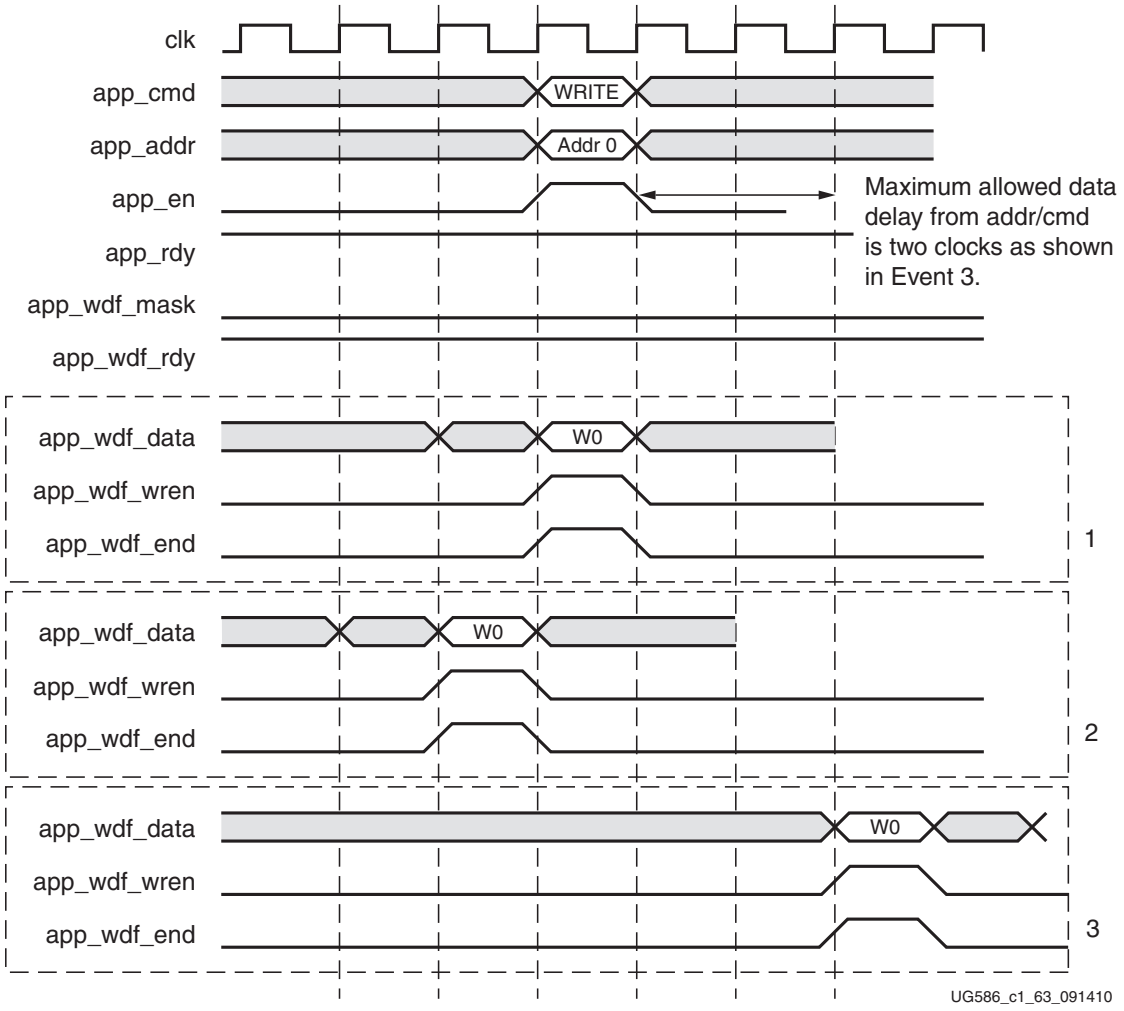


Figure 4.6: Timing diagram for the UI write path [29].

To simplify the interface between the rest of the waveform interface and the controller generated by MIG, a generic DDR controller was implemented. This controller encapsulates all the logic required to perform DDR operations through MIG; streamlining memory operations to modifying a 1-bit `start` and `cmd` signals for the rest of the components, as the DDR controller will handle the strict timing enforced by MIG.

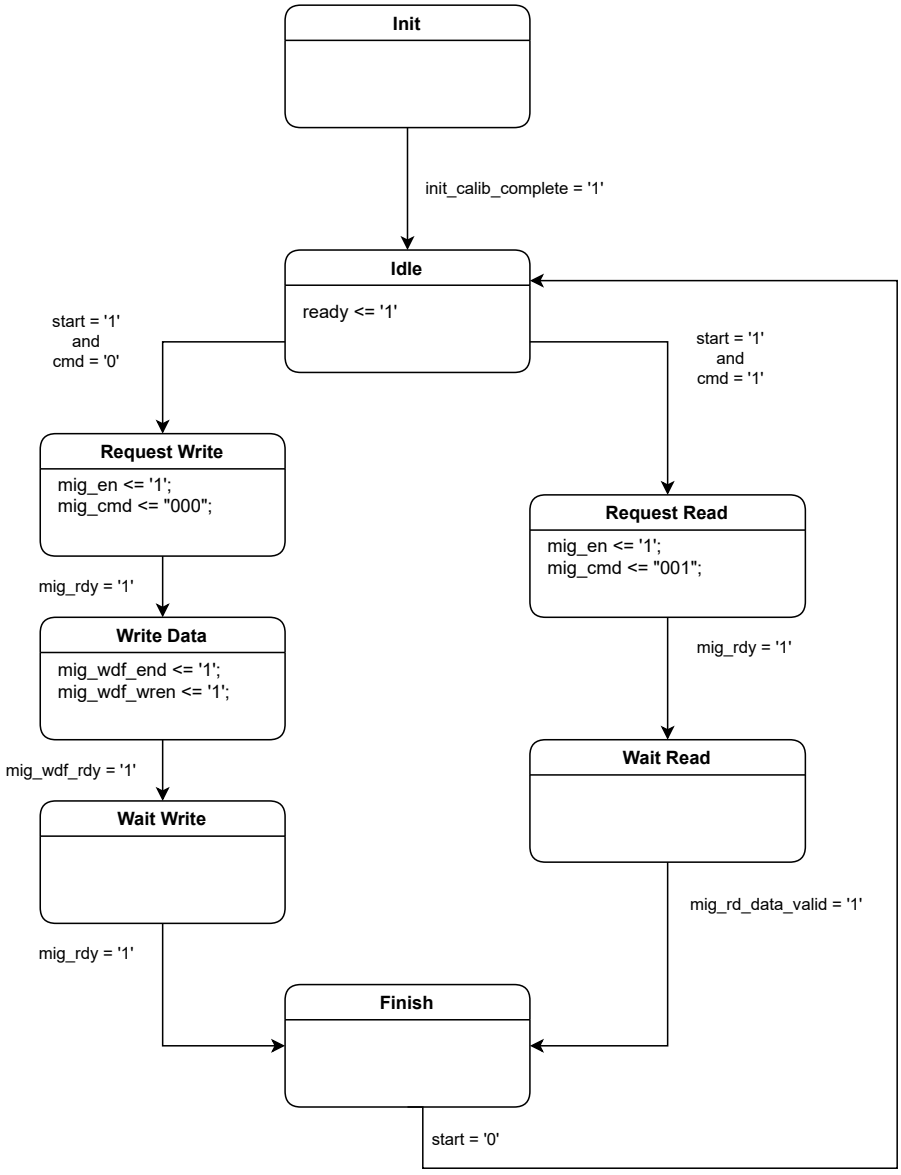


Figure 4.7: State machine for the DDR controller.

Fig. 4.7 shows the state machine that controls MIG flags within the DDR controller. The usual control flow is as follows:

- `Init:` The controller will wait in this state until the MIG interface has finished calibrating the DDR module.
- `Idle:` The controller drives the `ready` signal high to indicate its availability, and waits for a new command to be issued with the `start` signal.

- DDR instruction: The controller will take one of two branching paths depending on the requested command.
  - Write Operation
    - `Request Write:` The controller requests a write command until the `mig_rdy` signal is high for a clock cycle.
    - `Write Data:` The controller will provide the data to be written to the MIG interface.
    - `Wait Write:` The controller will wait for the MIG interface to finish the write instruction.
  - Read Operation
    - `Request Read:` The controller requests a read command until the `mig_rdy` signal is high for a clock cycle.
    - `Wait Read:` The controller waits for the operation to finish, capturing the resulting data when it does.
- `Finish:` With the DDR instruction finished, the controller will update its output but remain on this state until start is drive low to ensure proper flow control.

4.2.2 Communications Controller

Since most FPGA development boards include a USB-to-UART bridge, the CHOICE environment contains modules to support UART communication. The most notable one is a controller that enables an operator to perform the operations shown in Table 4.1.

Table 4.1: Current UART commands supported by the CHOICE environment.

Name	UART Code	Description
Reset	0x00	A reset signal will be asserted for one clock cycle, restarting the current DSP test.
Store results	0x01	The current results will be captured in the results memory.
Get results	0x02 0xXX	The controller will return with the data at address 0xXX in the results memory.
Set parameters	0x03 0xXX 0xYY	Store 0xYY in the parameter memory at address 0xXX.
Empty recorder	0x04	The controller will stream out the contents of the recorder memory over UART starting from address 0.

Since the implementation of the waveform interface results in larger amounts of data being transmitted through the UART interface, the communication controller was expanded to support the following operations:

- Write data (`0xD1`): The controller receives a series of bytes with the experimental data to be written into DDR memory.
- Read results (`0xD2`): The controller returns data corresponding to a window of the DSPs output.
- Dump data (`0xD3`): The controller returns the experimental data currently stored in the DDR memory. This operation enable the verifying the integrity of the experimental data once written into memory.

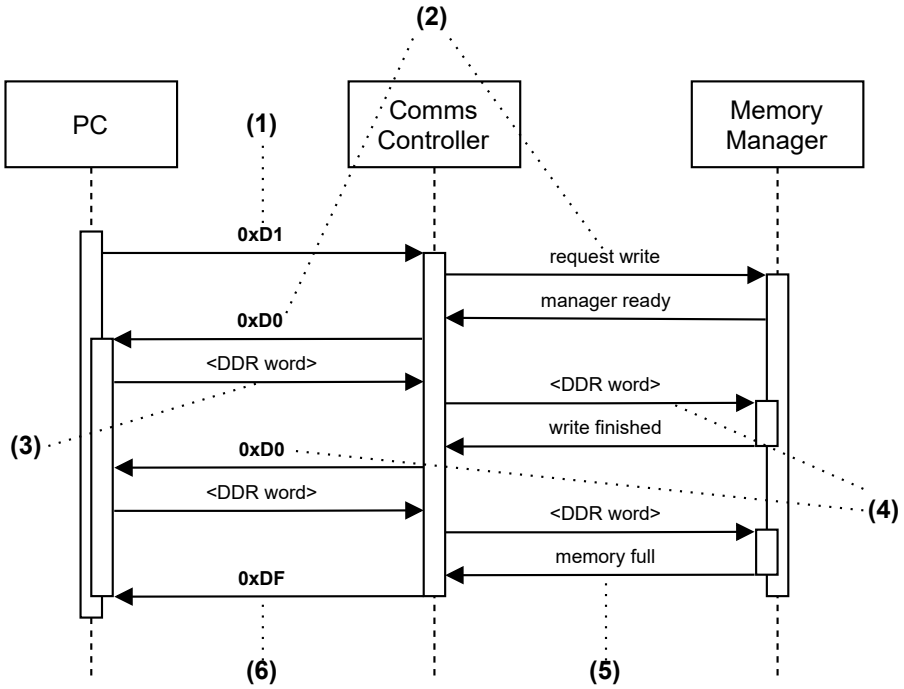


Figure 4.8: Sequence diagram for a *write data* operation.

Fig. 4.8 shows a sequence diagram representing a *write data* operation:

- The operation starts with the controller receiving the `0xD1` instructions through UART.
- The controller requests a *write data* operation from the memory manager, returning `0xD0` through UART once the memory manager is ready.
- The PC sends a single word of experimental data through UART.
- The controller tells the memory manager to write the received word, returning `0xD0` through UART when the operation is finished.
- Step 3 and 4 repeat themselves until the memory manager confirms that the memory is full.
- The controller returns `0xDF` through UART announcing that the operation has been completed.

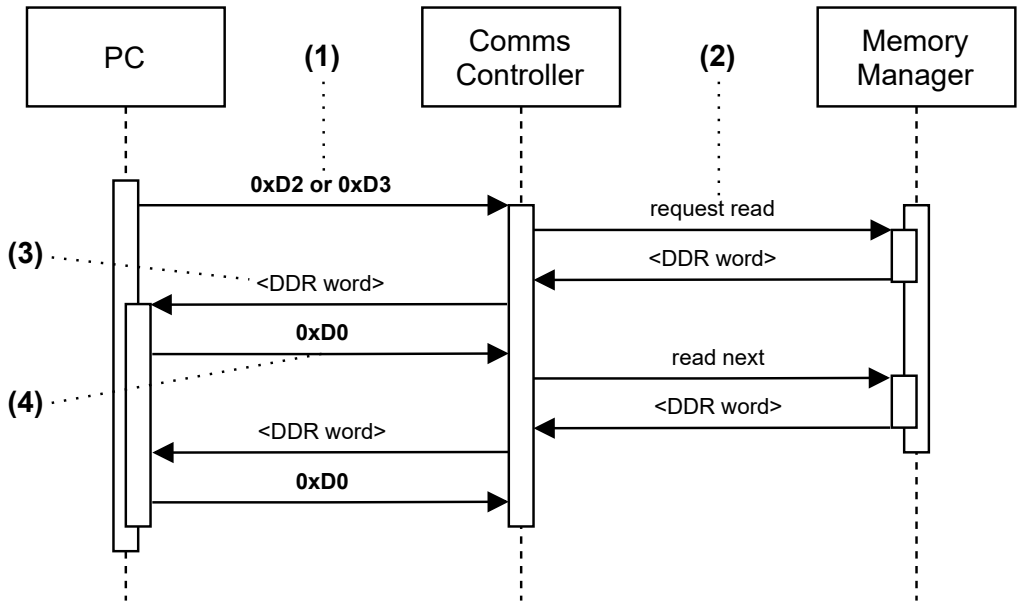


Figure 4.9: Sequence diagram for a *read results* or *dump data* operation.

Fig. 4.9 shows a sequence diagram representing a *read results* or *dump data* operation. The only difference between them is the command sent to the memory manager entity:

- The operation starts with the controller receiving the `0xD2` or `0xD3` instructions through UART.
- The controller requests the corresponding read operation from the memory manager.
- Once the memory manager finishes reading from memory, the controller returns the resulting data through UART.
- The controller then waits for the PC to confirm its reception by sending `0xD0`.
- Step 2 to 4 repeat themselves until the PC confirms that the last word has been received.

4.2.3 Testbench Controller

The testbench controller is in charge of slicing each word stored in DDR memory into the corresponding number of samples or symbols of experimental data and loading them into the cache memory used in the testbench entity. It essentially acts as a bridge between the memory manager and testbench cache entities, controlling the flow of the DSP tests. Its functionality has been split into two state machines, allowing for parallel operations and increasing the overall throughput of the system.

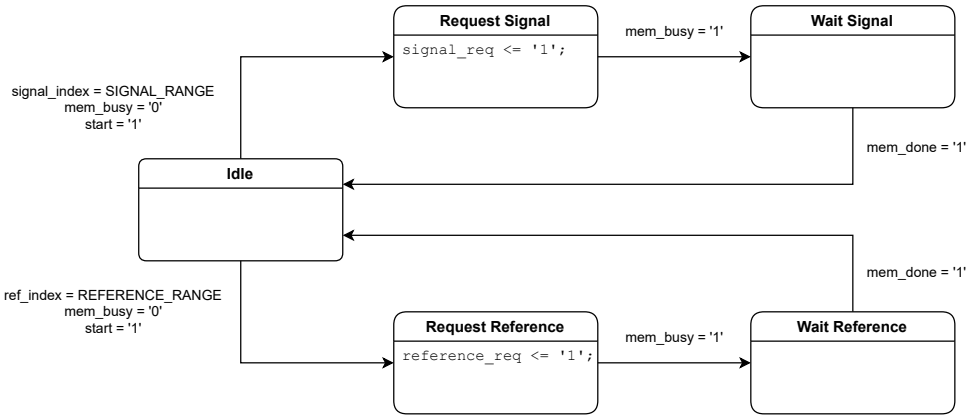


Figure 4.10: State machine that interacts with the memory manager.

Fig. 4.10 shows the state machine that controls the interaction between this controller and the memory manager entity. This state machine keeps track of the indices used to slice the received words from memory into their corresponding samples or symbols, requesting new data to the memory manager when needed.

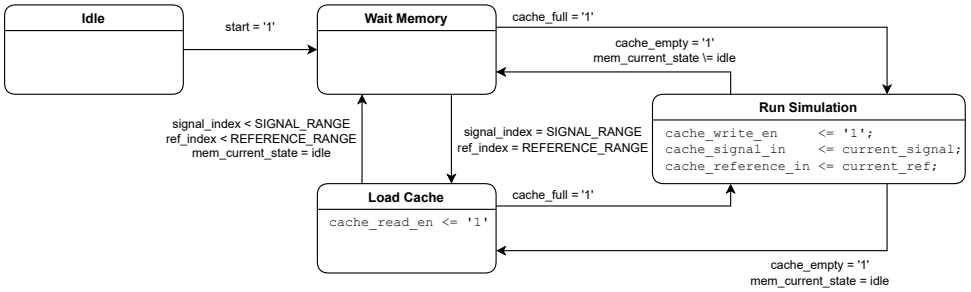


Figure 4.11: State machine that interacts with the testbench cache.

Fig. 4.11 shows the state machine that controls the interaction between this controller and the testbench cache entity. Once started, the controller will load up the FIFO buffer used as cache memory with the current section of data to be processed by the DSP when it's empty; and begin the DSP test when it's full. Depending on the clock ratio between the control and the testbench module, the DSP could be stalled to wait for memory operations to finish.

4.2.4 Memory Manager

This entity handles all memory operations requested by the other components in the waveform interface, interacting directly with the DDR controller. This entity also keeps track of the different sections of the DDR module used to store sample, symbol and result data; essentially turning the DDR memory into a series of circular buffers.

The operations performed by the memory manager can be classified in two groups: communication operations and runtime operations. Communication operations are needed by the communication controller, while runtime operations are the ones needed by the components involved with the testbench module:

- Read samples: the memory manager returns a single word from the samples section of the DDR memory.
- Read symbols: the memory manager returns a single word from the symbol section of the DDR memory.
- Write results: the memory manager writes a single word in the result section of the DDR memory

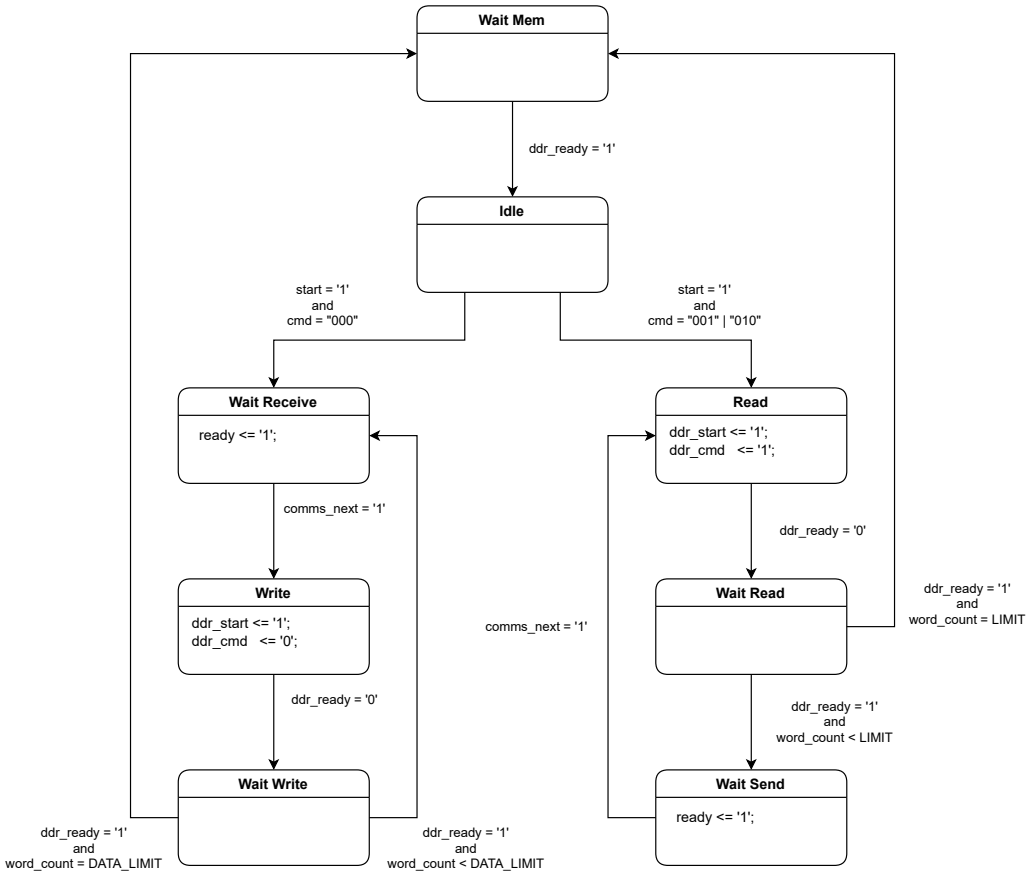


Figure 4.12: Simplified state machine for the memory manager.

Fig. 4.12 shows a simplified version of the state machine used to control the memory manager. The usual control flow of the communication operations is as follows:

- `Wait Mem`: The memory manager will wait in this state until DDR controller is ready.
- `Init`: The memory manager will wait in this state for an instruction, giving priority to comms operations.
- Instruction: The memory manager will take one of two branching paths depending on the requested command.

  - Write Operation
    * `Wait Receive:` The memory manager waits for the communication controller to receive a full word of data.
    * `Write:` The memory manager will provide the data to be written to the DDR controller.
    * `Wait Write:` The memory manager will wait for the DDR controller to finish the write instruction.
  - Read Operation
    * `Read:` The memory manager requests a read command from the DDR controller.
    * `Wait Read:` The memory manager waits for the operation to finish.
    * `Wait Send:` The memory manager waits for the communications controller to finish transmitting the data.

4.3 Testbench Module

The testbench module, marked in Fig. 4.1, contains the clock region where the DSP components are tested, and it is made up of the following entities:

- A small cache memory used to store experimental data.
- The DSP component to be tested.
- Output recorder to capture a rolling window from the output of the DSP.
- Demodulator and analysis blocks to evaluate the performance of the DSP.[1]

4.3.1 Testbench Cache

As previously mentioned, the testbench entity uses a small cache memory to store the section of experimental data currently being processed by the DSP. Even though its size can be configured, to ensure maximum throughput it is recommended that it matches the size of a DDR word, which is 512 bits for both of our development boards.

The cache memory is implemented as two FIFO buffers sharing read and write enable signals. One buffer stores signal samples and one reference symbol, with one sample or symbol per word. Since this entity interacts with both clock regions, the write and read interfaces of the FIFO buffer run on different clocks. Fig. 4.13 shows the block diagram for the cache entity.

[1] Neither of these components have been modified from the original CHOICE environment described in [2], and will not be described in this section.

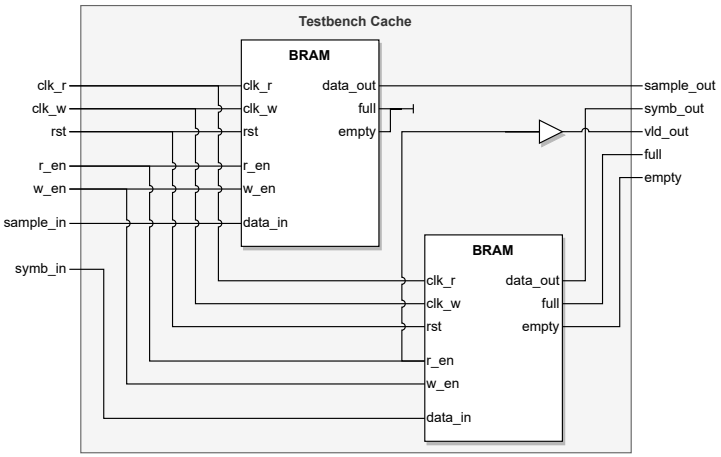


Figure 4.13: Block diagram for the DDR cache entity.

4.3.2 DSP Recorder

The current version of the CHOICE environment already includes a small BRAM entity able to record the output of the DSP during testing. However, after implementing the DDR controller it was expanded to allow the system to store longer windows of the DSP's output.

Every time the DSP produces a result, this is stored in a register until it reaches DDR word length. The resulting data is then stored in a FIFO buffer until the memory manager is ready to store it in the DDR memory. In a similar manner to the cache entity, the read and write interfaces of the FIFO buffer work on separate clock regions.

5

Results

The implemented waveform interface was successfully implemented on both the KC705 board and the VC709 board; and we were able to achieve the three main goals and each non-optional subgoal introduced in Section 1.2. Since the main difference between implementations is the size of DDR memory, this chapter will evaluate the results and performance of the KC705 implementation.

5.1 Storage Capacity and Resource Usage

The first important thing to evaluate on implementation is the maximum length of the experimental data it can store in DDR memory. Since this is entirely dependent on the data to be tested, we will be using the 16 bit signals and 4 reference bits from the use-case example in Section 5.3 as a reference and removing the results buffer from memory. This results in the KC705 board being able to store up to $4 \cdot 10^9$ samples which correspond to 0.04 s in a 10 Gbaud/s transmission, while the VC709 board can store up to $8 \cdot 10^9$ samples or 0.08 s of signal data.

Even though these values are greatly impacted by the size of the samples and the speed of the transmissions, $4 \cdot 10^9$ samples are enough to calculate the BER of most DSP implementations.

Table 5.1: Post-implementation resource utilization of the waveform interface.

Resource	Utilisation	[%]
LUT	16965	8.32
LUTRAM	2034	3.18
FF	16684	4.09
IO	119	23.80
BUFG	6	18.75
MMCM	2	20.00
PLL	1	10.00

Table 5.1 shows the post-implementation resource utilization of the implemented system in the KC705 board. As we can see, logic block utilization is quite small, and most notably makes use of no DSP slices which are often the limiting factor in complex DSP implementations.

5.2 Throughput

One of the main advantages of the FoC approach over digital simulations is its high throughput. Therefore it is essential to evaluate the throughput of the implemented waveform interface. The main bottleneck in the implemented system is loading up the cache memory used during testing. Since we can only perform either a read or write operation, the system needs to stall the DSP test while the cache memory is loaded with new data.

The length of this operation depends on the length in bits of the DP-IQ samples and the word size of the DDR module, see Chapter 4. Once again, for this evaluation, we used the 100 MHz emulation clock frequency, 16-bit DP-IQ samples, and 512-bit DDR words from our use-case example in Section 5.3. Furthermore, we will assume the cache memory does not need to wait to request data from the memory manager.

This results in needing $\frac{512}{16} = 32$ clock cycles to load and empty the cache memory. However, since there is a 2:1 ratio between our simulation and our memory clock region, from the perspective of the DSP loading only lasts 16 clocks. Therefore, 2 samples are processed every 3 clock cycles, resulting in a throughput of $\approx$1 Gbit/s. 1/3 slower than an FoC system running in *synthetic* mode where a sample is processed every clock cycle.

During testing, the system ran slightly under our theoretical estimation at a 0.8 Gbit/s throughput, meaning some other operations were keeping the memory manager busy while the cache memory waits to be loaded.

5.3 Use-Case Example

Finally, in order to verify the system, the waveform interface was used to test the performance of the BPS implementation shown in Fig. 5.1.

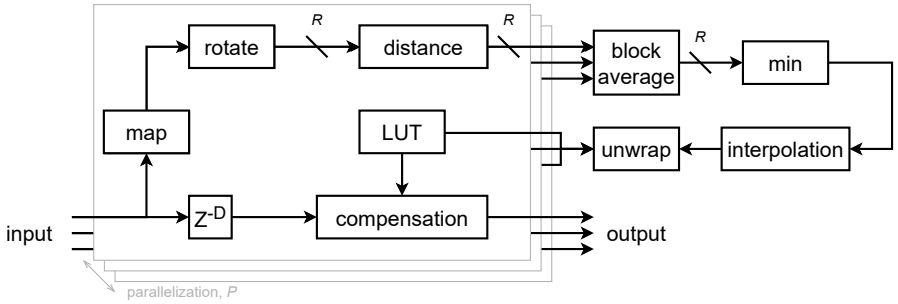
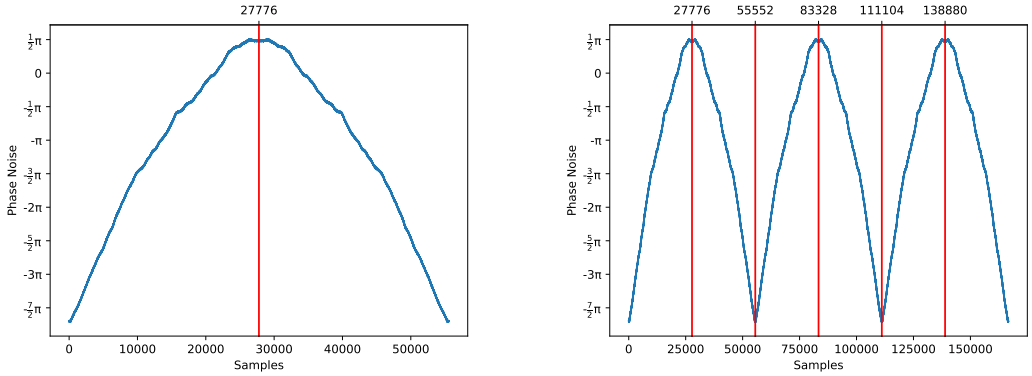


Figure 5.1: Block diagram for the BPS implementation. Adapted from [13].

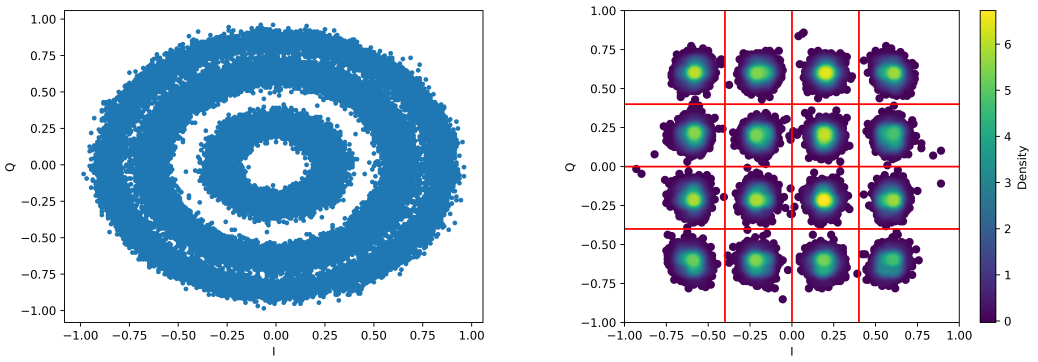
For this test, all impairments aside from phase noise were compensated before sending the data to the waveform interface, reducing the complexity of the DSP pipeline needed in the FPGA. Additionally, the data sent to the waveform interface need to generate a continuous waveform when put into circular buffers. To this end, the transmission was looped front-to-back to create a continuous phase change, as shown in Fig. 5.2.



(a) Data stored in memory

(b) Resulting continuous waveform

Figure 5.2: Phase noise of the data transmitted to the waveform interface.



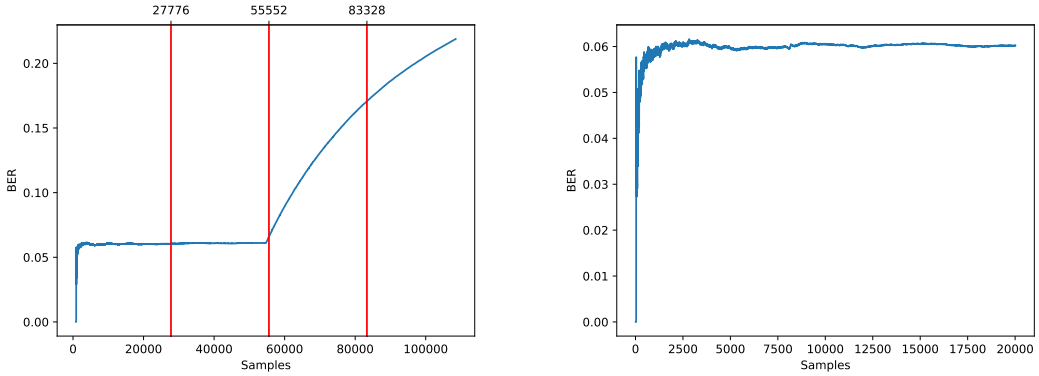
(a) Experimental data

(b) DSP output

Figure 5.3: Constellation diagrams obtained from the DSP test.

As we can see from Fig. 5.3, the BPS DSP is able to compensate the received samples back to the original 16-QAM constellation. Most of the points recovered appear centered within the decision regions of the demodulator, shown by the red lines in the diagram. Using the onboard analysis block to calculate the BER resulted in values of ≈ 0.25 or ≈ 0.5, which usually signifies unwanted phase rotations.

By calculating the BER of the DSP output recorded in DDR memory, we can see how the BER converges around 0.06 before reaching around 55000 samples, where it starts rising, see Fig. 5.4a. This change in BER consistently occurs at the loop-back point of the circular buffer.



(a) Full transmission **(b)** First 20000 samples

Figure 5.4: BER plots obtained from the DSP test.

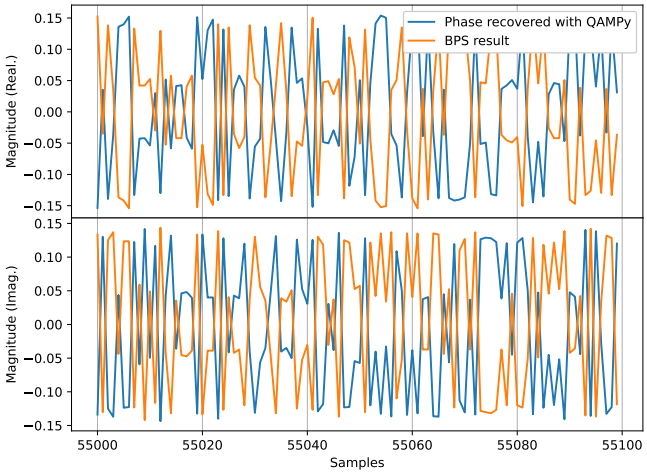


Figure 5.5: Comparison between DSP output and QAMPy phase recovery.

Fig. 5.5 compares the output of the DSP after the loop-back point with the symbols recovered using a pilot-based phase recovery algorithm in QAMPy[1]. The output of the DSP appears to be a flipped version of the correct symbols, denoting an unwanted phase rotation, which goes in line with the BER calculated by the onboard analysis module. Further testing needs to be conducted to identify what is causing the phase rotation. But since it is consistently located at the loop-back point, it is reasonable to assume that the sudden change in phase noise is causing cycle slips in the DSP.

Regretfully, a full comparison between the waveform interface and the original version of CHOICE will be postponed until this issue is fixed. However, this method of testing for deep BER using experimental data shows promise, since the waveform interface shows consistent results after the first cycle of experimental data.

[1]Magnitude values have been normalized since signal data is scaled-down when converted to fixed-point representation before being sent to the waveform interface.

6

Discussion

Even though the waveform interface was intended to test DSP components for fiber communications using recorded experimental data, the implemented system also opens up the following possibilities:

- **Using synthetic data generated in MATLAB to test DSP components**:
  By loading the waveform interface with data generated in MATLAB, it is possible to test DSPs with transmissions that contain channel impairments currently not supported by the CHOICE environment.
- **Evaluation of channel models implemented in CHOICE**:
  One can evaluate the fidelity of the channel models implemented in CHOICE against similar MATLAB implementations. This can be done by testing the same DSP using the waveform interface data generated in MATLAB; and comparing the results with tests conducted with the synthetic data generation in CHOICE.
- **Testing DSP components for fields other than optical fiber communications.**:
  There is nothing inherent to optical fiber communications in the resulting implementation. So as long as the correct data are loaded into the system, it is possible to test any DSP implementation.

6.1 Design Decisions

In this subsection, some of the main design decisions and the impact they had on the final implementation will be discussed:

- **Using DDR memory for bulk-storage**:
  Even though its low bandwidth means the system needs to make use of supporting cache memories to maintain a high throughput, its high storage capacity allows for storing the large amounts of experimental data required by this implementation. Implementing a DDR controller was also beneficial to the CHOICE environment as a whole; for example, the DSP recorder is now able to store a larger window of DSP results.
- **Using MIG for interfacing with the DDR module**:
  The decision to rely on an external IP block proved to be a correct one, as

the increased complexity of implementing a native DDR controller would have possibly resulted in the waveform interface not being fully implemented. However, relying on MIG requires the use of Xilinx boards for implementation.

- **Keeping the UART interface for communications**:
  Although its low speed was not an issue for the CHOICE environment, the large amounts of data transmitted by the waveform interface would benefit from a faster communication interface.

6.2 Future Work

Even though the implemented waveform interface shows promise as an alternative to the synthetic data generation in the CHOICE environment presented in [2], the cause for the unwanted phase rotations shown in Section 5.3 should be investigated. Once this problem is solved the performance comparison we mentioned in Section 1.2 should be conducted to fully evaluate the implemented waveform interface.

Although the performance of the resulting system has been deemed adequate, there are many possible improvements that could increase its performance, so that it more closely mirrors the CHOICE environment presented in [2]:

- Multiple DDR controllers could be implemented, allowing for simultaneous read and write operations to the DDR memory module. For example the DDR module in the KC705 board has three ranks, allowing the implementation of a separate controller to store sample data, reference symbols and DSP results.
- The testbench cache entity could be reworked to make use of FIFO buffers in a ping-pong configuration, completely obviating the need for the DSP to wait for the cache memory to be loaded with new experimental data.

Aside from increasing the throughput of the system, these modifications would also improve the potential parallelism of the tested DSP components. As it stands, the waveform interface is able to provide a single sample per clock cycle. By reducing the bottleneck caused by the DDR controller, multiple samples could be served at once; resulting in even higher system throughput for specific DSP implementations.

Finally, as we mentioned in Section 6.1, using the UART interface to transfer data to and from the system is not ideal. Therefore, the implementation of a faster communications interface should be explored. A faster interface could increase the length of the transmissions tested by the system since new data could be loaded into DDR memory, while tests are running.

Bibliography

[1] E. Börjeson, C. Fougstedt, and P. Larsson-Edefors, "Towards FPGA Emulation of Fiber-Optic Channels for Deep-BER Evaluation of DSP Implementations," *Advanced Photonics Congress*, 2019.

[2] P. Larsson-Edefors and E. Börjeson, "Fiber-on-Chip: Digital FPGA Emulation of Channel Impairments for Real-Time Evaluation of DSP," in *Optical Fiber Communication Conference (OFC) 2022*. Optica Publishing Group, 2022, p. W3H.3.

[3] I. Val, F. Casado, P. M. Rodriguez, and A. Arriola, "FPGA-based wideband channel emulator for evaluation of Wireless Sensor Networks in industrial environments," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–7.

[4] P. Larsson-Edefors and E. Börjeson, "CHOICE Chalmers Optical Fiber Channel Emulator," 2022. [Online]. Available: https://www.cse.chalmers.se/research/group/vlsi/choice/

[5] S. Warier, *ABCs of Fiber Optic Communication A Practical Handbook*. Artech House, 2017.

[6] M. Arumugam, "Optical fiber communication - An overview," *Pramana*, vol. 57, no. 5, pp. 849–869, Nov 2001.

[7] J. Penttinen, *The Telecommunications Handbook Engineering Guidelines for fixed, mobile and Satellite Systems*. Wiley, 2015.

[8] R. P. Dahlgren and B. Dahlgren, "Noise in fiber optic communication links." Silicon Valley Photonics, 2001.

[9] A. Demir, "Nonlinear Phase Noise in Optical-Fiber-Communication Systems," *Journal of Lightwave Technology*, vol. 25, no. 8, pp. 2002–2032, 2007.

[10] G. Chauvel, "Dispersion in Optical Fibers." Anritsu Corportation, 2008.

[11] S. Kumar, "Analysis of Nonlinear Phase Noise in Coherent Fiber-Optic Systems Based on Phase Shift Keying," *Journal of Lightwave Technology*, vol. 27, no. 21, pp. 4722–4733, 2009.

[12] J. Zhao, Y. Liu, and T. Xu, "Advanced DSP for Coherent Optical Fiber Communication," *Applied Sciences*, vol. 9, no. 19, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/19/4192

[13] E. Börjeson, C. Fougstedt, and P. Larsson-Edefors, "ASIC Design Exploration of Phase Recovery Algorithms for M-QAM Fiber-Optic Systems," in *2019 Optical Fiber Communications Conference and Exhibition (OFC)*, 2019, pp. 1–3.

[14] E. Börjeson and P. Larsson-Edefors, "Cycle-Slip Rate Analysis of Blind Phase Search DSP Circuit Implementations," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, 2020, pp. 1–3.

[15] H. Amano, *Principles and Structures of FPGAs* . Singapore: Springer, 2018.

[16] A. Podobas, K. Sano, and S. Matsuoka, "A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective," *IEEE Access*, vol. 8, pp. 146 719–146 743, 2020.

[17] *7 Series FPGAs Memory Resources*, Xilinx. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug473_7Series_Memory_Resources

[18] Xilinx Inc., *Block Memory Generator v8.4: LogiCORE IP Product Guide*, december 2019. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_4/pg058-blk-mem-gen.pdf

[19] Z. Zhou, X. Tang, L. Yan, L. Tang, S. Han, Y. Wan, C. Feng, and K. Xiong, "Reliability improvement scheme of DDR controller based on FPGA," in *2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, 2021, pp. 1269–1273.

[20] M. Milford and J. McAllister, "Valved dataflow for FPGA memory hierarchy synthesis," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 1645–1648.

[21] E. Börjeson and P. Larsson-Edefors, "Energy-Efficient Implementation of Carrier Phase Recovery for Higher-Order Modulation Formats," *IEEE J. Lightw. Technol.*, vol. 39, no. 2, pp. 505–510, 2021.

[22] J. L. Hershberger, E. A. Thompson, and T. S. Loos, "A real-time WARP-based data capture and playback test bed for DSP applications," in *2013 IEEE Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE)*, 2013, pp. 48–53.

[23] Y. Zhao, Y. Su, R. Huang, P. Hu, and Z. Chen, "Design and implementation of a radar waveform playback system for real-time digital signal processing test," in *2017 Sixth Asia-Pacific Conference on Antennas and Propagation (APCAP)*, 2017, pp. 1–3.

[24] R. Gutierrez, V. Torres, and J. Valls, "Hardware Architecture of a Gaussian Noise Generator Based on the Inversion Method," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 8, pp. 501–505, 2012.

[25] H. Kan, H. Zhou, E. Börjeson, M. Karlsson, and P. Larsson-Edefors, "Digital Emulation of Time-Varying PMD for Real-Time DSP Evaluations," in *Asia Communications and Photonics Conference 2021*. Optica Publishing Group, 2021, p. M4H.4.

[26] J. Schröder, M. Mazur, and M. Brehler, "QAMPy a DSP chain for optical communications," 2019. [Online]. Available: https://zenodo.org/record/2638956

[27] *KC705 Evaluation Board for the Kintex-7 FPGA*, Xilinx. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug883_K7_KC705_Eval_Kit

[28] *VC709 Evaluation Board for the Virtex-7 FPGA*, Xilinx. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug887-vc709-eval-board-v7-fpga

[29] *7 Series FPGAs Memory Interface Solutions*, Xilinx. [Online]. Available: https://docs.xilinx.com/v/u/1.7-English/ug586_7Series_MIS