The Role of FPGA Technology in the Design of Smart Transceivers for use in Optical Communications Systems

MICHAEL J ANDERSON

Doctor of Philosophy

ASTON UNIVERSITY

SCHOOL OF ENGINEERING AND APPLIED SCIENCE

December 2023

Aston University

The Role of FPGA Technology in the Design of Smart Transceivers for use in Optical Communications Systems

Michael J Anderson

Doctor of Philosophy

2023

Abstract

This thesis looks at where the recent advances in Field Programmable Gate Array (FPGA) technology have been or could be used to complement and further the development of smarter optical devices in both the in-service and product development environments.

One approach to the development of cost-effective integrated components is the utilisation of low cost, low power circuitry, built as part of the module, which is capable of being repurposed from providing automated manufacturing orientated functions, such as characterisation and calibration, to operational control functions. Although these individual functions are well known, computationally efficient and low-cost implementations are required to enable competitive module pricing. This work examines the case of an optical Mach Zehnder Modulator (MZM) where the relationship between bias voltages and output power requires determining. With this aim, this study investigates the self-characterisation of this relationship with low cost and low power components which could be readily integrated into an optical module. In particular, the behaviour and capabilities required for automatic digital bias characterisation functionality, implemented in small gate count, low-cost FPGAs, are developed. The suitability of highly efficient implementations of DSP functions within the bias measurement function, such as digital filters, is tested, by investigating experimentally the use of a computationally efficient algorithm for computing a single component of a discrete Fourier transform, as a demonstration of the viability of using low-cost digital hardware to implement a circuit capable of monitoring the MZM transfer function. The results of this experiment are then used to investigate whether a simple Machine Learning model can be trained to extract characterisation data for the modulator.

Smarter optical modules, deployed in optical networks, are being developed as part of the solution to the increasing demand for bandwidth. There is a growing interest in augmenting existing communication systems, to carry greater bandwidth over the existing installed infrastructure, by adopting more parallel transmission techniques, such as Spatial Division Multiplexing (SDM), and by the application of advanced Digital Signal Processing (DSP) methods, e.g. compensating for non-linear impairments, allowing a higher transmit power and thus improving the Signal to Noise Ratio (SNR) which is required to support higher order modulation formats. Here, the role which can be played by FPGAs is investigated, through the first-time realization in FPGA technology of a real time, recurrent machine learning solution addressing optical channel non-linearities.

This thesis reports on three first-time achievements:

- the real time demonstration and the novel use of a filter based on the computationally efficient Goertzel algorithm to monitor an MZM transfer function.
- the implementation in FPGA technology of a biLSTM based equalizer for non-linear effects in optical communications systems. This is a novel use of the biLSTM architecture for this application.
- the use of a small size Neural Network to perform the Vπ and V_{OFFSET} characterisation of an MZM optical component. In this work the novel application of ML, a NN with only 3 hidden layers of 8 neurons per layer and a small input layer of 82 inputs, is demonstrated as being able to determine the Vπ and V_{OFFSET} parameters of a real MZM.

The combination of the computationally efficient filter and the small NN capable of processing the output from the filter together show the ability of cost-optimised FPGAs to perform the analysis and processing of data required to carry out characterisation procedures on the photonic module, without the time overheads associated with large scale data transfers to a computer. Additionally, the implementation of a novel biSTLM equalizer demonstrates a practical design flow to realise the applications of NNs in FPGAs.

Keywords: Field Programmable Gate Array, Nonlinear Equalization, Cost-effective Characterisation, Manufacturing, Parallelism, Repurposable Design, Machine Learning, Neural Networks, High Level Synthesis.

ACKNOWLEDGEMENTS

I would like to sincerely thank my supervisor Prof. Wladek Forysiak for encouraging me to pursue my ambition to study for a PhD at Aston University. I am deeply grateful to him for his guidance, the remarkable patience and the great support.

I would also like to express my gratitude to my associate supervisors Dr Paul Harper and Dr Richard Nock.

I would like to thank my employer TerOpta Ltd for their support and flexibility in my undertaking of this venture.

I would like to acknowledge my fellow researchers and colleagues at AIPT, most particularly Dr Ian Phillips for his valuable support in the lab and for his help sourcing equipment for my experiments. I also would like to thank Dr Pedro J Freire for his help and advice on Machine Learning when I needed it.

Finally, a special thanks to my wife Annette for her continuous support and encouragement.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ADC – Analogue to Digital Convertor

AI – Artificial Intelligence

ALU – Arithmetic Logic Unit

ANN – Artificial Neural Network

ASIC – Application Specific Integrated Circuit

biLSTM – Bidirectional Long Short-Term Memory

CDC – Chromatic Dispersion Compensation

CNN – Convolutional Neural Network

CPU – Central Processing Unit

DAC – Digital to Analogue Convertor

DBP – Digital Backpropagation

DFT – Discrete Fourier Transform

DPR – Dynamic Partial Reconfiguration

DSP – Digital Signal Processing

DWDM – Dense Wavelength Division Multiplexing

FEC – Forward Error Correction

FPGA – Field Programmable Gate Array

FFT – Fast Fourier Transform

FIR – Finite Impulse Response (filter)

HDL – Hardware Description Language

HLS – High Level Synthesis

I2C – Inter Integrated Circuit Bus

IIR – Infinite Impulse Response (filter)

IoT – Internet of Things

IP – Internet Protocol

ISP – Internet Service Provider

LE – Logic Element

LUT – Look Up Table

ML – Machine Learning

MSA – Multiple Supplier Agreement

MZM – Mach Zehnder Modulator

NN – Neural Network

OWC – Optical Wireless Communications

PC – Personal Computer

PCB – Printed Circuit Board

PIC – Photonic Integrated Circuit

POF – Programmer Object File

ReLU – Rectified Linear Unit

RISC – Reduced Instruction Set Computer

ROSA – Receiver Optical Sub-Assembly

SDM – Spatial Division Multiplexing

SDN – Software Defined Networks

SDR – Software Defined Radio

SOA – Semiconductor Optical Amplifier

SPI – Serial Peripheral Interface

SoC – System on Chip

TOSA – Transmitter Optical Sub-Assembly

TEC – Thermoelectric Cooler

UART – Universal Asynchronous Receiver Transmitter

VHDL – Very High-Speed Integrated Circuit Hardware Description Language

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLICATIONS

Peer-reviewed

Anderson, M., Philips, I., Callan, P., & Forysiak, W. (2021). Repurposable hardware in smart photonic components. *Journal of Physics: Conference Series*, *1919*(1), [012010]. https://doi.org/10.1088/1742-6596/1919/1/012010

Freire, P. J., Srivallapanondh, S., **Anderson, M.**, Spinnler, B., Bex, T., Eriksson, T. A., Napoli, A., Schairer, W., Costa, N., Blott, M., Turitsyn, S. K., & Prilepsky, J. E. (2023). Implementing Neural Network-Based Equalizers in a Coherent Optical Transmission System Using Field-Programmable Gate Arrays. *Journal of Lightwave Technology*, [10113728]. https://doi.org/10.1109/JLT.2023.3272011

Freire, P. J., Napoli, A., Ron, D. A., Spinnler, B., **Anderson, M.**, Schairer, W., Bex, T., Costa, N., Turitsyn, S. K., & Prilepsky, J. E. (2023). Reducing Computational Complexity of Neural Networks in Optical Channel Equalization: From Concepts to Implementation. *Journal of Lightwave Technology*. https://doi.org/10.1109/jlt.2023.3234327

Conference proceedings

Freire, P. J., **Anderson, M.**, Spinnler, B., Bex, T., Prilepsky, J. E., Eriksson, T. A., Costa, N., Schairer, W., Blott, M., Napoli, A., & Turitsyn, S. K. (2022). Towards FPGA Implementation of Neural Network-Based Nonlinearity Mitigation Equalizers in Coherent Optical Transmission Systems. In *2022 European Conference on Optical Communication, ECOC 2022* (2022 European Conference on Optical Communication, ECOC 2022). IEEE.

1 INTRODUCTION

With continuous advances being made in FPGA technology, this thesis examines where these devices have been used and could be used to complement and further the development of smarter optical transceivers in a product development environment. In the context of this thesis, the definition of smart optics is intended as meaning 'having the capability of autonomously performing complex analytical functions which may normally be performed by an operator', for example to characterise a control surface or transfer function, or to find optimum operating points. It is essential, however, that this does not add excessive cost overheads to the module and this is where the concept of repurposable hardware is introduced – by being able to reuse elements of the module which are present for normal operation, potentially multiple times, additional costs can be minimised.

Both FPGAs and optics are fast moving technology fields and, as such, the majority of papers concerning FPGAs contributing to this review have been published in the time period beginning five years prior to the start of this work, in respected international industry journals and conferences, and have been identified through searches of online facilities such as Google Scholar and ResearchGate. Older literature is also included as it does provide an insight to the diversity of applications of FPGAs and of the rapid developments in optical component bandwidth.

During the course of this work, three of the largest FPGA vendors have been acquired – Altera by Intel, Xilinx by AMD and MicroSemi by MicroChip. Consequently, the names are used interchangeably throughout this thesis and the referenced literature.

Contributions of the Thesis

This thesis reports on three first-time achievements:

- the real time demonstration and the novel use of a filter based on the computationally efficient Goertzel algorithm to monitor an MZM transfer function. This algorithm enables a digital filter to be designed with lower complexity in terms of multiplications and additions per input sample and significantly fewer stored weighting factors when compared with conventional digital filter structures with the same parameters. The use of computationally efficient methods is essential when investigating low-cost, self-characterising hardware.

- the implementation in FPGA technology of a biLSTM based equalizer for non-linear effects in optical communications systems. This is a novel use of the biLSTM architecture for this application. A study of the steps required to implement a large, recurrent neural network in an FPGA was performed as part of an investigation into the viability of NN based equalizers for commercial applications. The real-time implementation of this biLSTM equalizer was shown by experiment to offer a 1.7 dB improvement in Q-factor when compared with a standard DSP based Chromatic Dispersion Compensation block for only a 2.5x increase in estimated FPGA capacity at a throughput of 400 Gbit/s.
- the use of a small size Neural Network to perform the Vπ and V_{OFFSET} characterisation of an MZM optical component. In this work the novel application of ML, a NN with only 3 hidden layers of 8 neurons per layer and a small input layer of 82 inputs, is demonstrated as being able to determine the Vπ and V_{OFFSET} parameters of a real MZM, in conjunction with the Goertzel filter, using experimentally obtained data, with electronic system noise, from an automated voltage sweep of the DC bias electrode. The determined parameters are shown to match those obtained from a manual measurement and voltage sweep to within 2% of Vπ. The complexity of implementing this small scale NN solution is compared with the complexity of implementing a ‘soft’ IP microcontroller to analyse whether such an approach is compatible with the use of cost optimised FPGAs to create smarter, self-characterising photonic components.

1.1 Thesis Structure

This thesis has been broken down into 7 chapters:

The following parts of this chapter present the motivation for this piece of work, by analysing the implications on the cost of manufacturing high bandwidth optical transceivers and what approaches could make such a transceiver 'smarter' and able to reduce costs by incorporating self-test capabilities.

Chapter 2 introduces the need for parallelisation in optical communication systems by exploring the increasing requirements for optical bandwidth which is being forecast by multiple industry sources. The effects of the changing nature of the traffic and the subsequent need for flexibility in the network presents further challenges to the transceiver manufacturer. The adoption of multichannel optical modules, using tightly spaced Superchannels and implemented using arrays of components, is shown as one possible approach to satisfying this appetite for data. This presents new challenges for the transceiver manufacturer when trying to control the test and assembly costs associated with packaging arrays of components together in a single transceiver entity.

Chapter 3 offers a background on FPGA technology, introducing the basic architectural building blocks, or logic elements, which form these devices. The advantages and disadvantages of FPGAs over custom integrated circuits are underlined, particularly in the light of the development costs associated with the latter. By looking at the development of new features in these devices over time and also at the broad range of device capabilities – in terms of the number of resources available – the chapter evaluates the potential advantages of FPGAs in this application by inspecting two areas of operation:

1. The traffic path of an in-service multichannel transceiver

2. The set up and control of multichannel transceiver optics during manufacture.

In chapter 4, the operation of the Mach Zehnder Modulator is examined. Within the subject of this thesis these components are a compelling and relevant example to study. Initially, the behaviour of the Mach Zehnder is well described arithmetically and the theory behind this is presented. Then, the design of digital filters, used as a means of monitoring the Mach Zehnder harmonically, is discussed and a resource efficient algorithm for implementing filters is introduced and compared with the more commonly used FIR and IIR structures. Finally, this chapter reports on an experiment assessing the suitability of using such a filter, implemented using this resource efficient method, of monitoring the transfer function, via harmonic analysis, of these devices.

Chapter 5 examines the design process involved in the implementation of a small microcontroller based system in an FPGA. Such a system commonly forms part of an integrated design. Firstly, this chapter provides an assessment of the complexity of the design process by implementing the steps involved in building the embedded microcontroller system using commercially available 'soft' cores as well as creating a bespoke, embedded build of the Linux operating system. Then, the design itself is examined, by analysing the FPGA resources – logic and memory – consumed by its implementation and where this may be optimised.

Chapter 6 is focussed on the potential of Machine Learning to be used in FPGAs. Firstly, the concepts of Machine Learning are introduced and a brief review of the current uses of Machine Learning in optical communications is given. The mapping of Machine Learning functionality onto the fundamental building blocks of an FPGA is analysed. A design flow, starting with a high level description of a Machine Learning model, is examined in detail, through the implementation of a machine learning based solution capable of solving non-linear effects, and the optimisation approaches required during this activity are discussed. Finally, to answer the question about whether it is possible to implement self characterisation functions in an FPGA the application of machine learning to the task of characterising an MZM is investigated.

The final chapter brings together the conclusions from the preceding chapters and sets out some suggestions for the activities required to further pursue this study into FPGAs as a route to smarter optical transceivers.

Appendix A gives further details about the Tensorflow method and the Python code used in the design of the Neural Networks used in Chapter 6.3.

1.2 Motivation

In the light of the growing trend for data transfer capacity across all areas of the optical network, from the edge to the optical core, as a result of both high bandwidth consumer services such as IP TV and low bandwidth but high-volume applications such as the Internet of Things, coupled with tightening cost, power and footprint requirements, the integration of multiple optical components into transceiver modules has become standard practice. Transceiver modules are now complex micro system assemblies built from many discrete components [1] [2] [3].

There is an ongoing need for cost efficiency in the manufacture of these modules in order to service ever tightening capital budgets and 'cost per bit' targets. One area of device production which can be examined for cost reduction opportunities is the unit test and characterisation step which occurs before and after packaging. Packaging operations can account for a substantial proportion of the module production costs, with an estimate of almost 40% of the cost of a 400Gb/s pluggable transceiver being due to the test, assembly and packaging activities as indicated in Figure 1-1 [1]. Throughout the manufacturing process there are multiple stages involved in the packaging operation. At each stage, test and characterisation procedures are required to be repeated to ensure that failed assemblies are removed from the process.

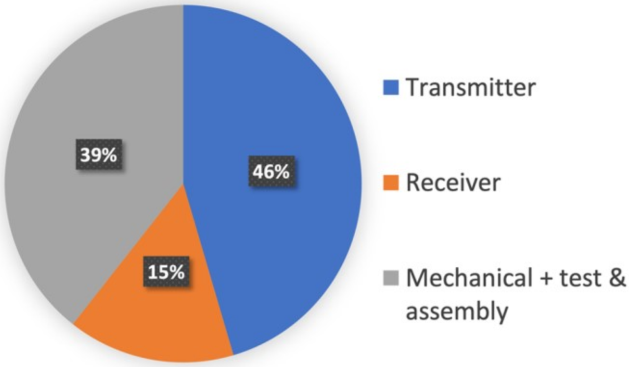


Figure 1-1 Breakdown of the cost structure of a 400 Gb/s pluggable transceiver [1].

The implementation of self-contained, intelligent functions within an optical module to perform aspects of test and characterisation can potentially reduce costs by allowing for reduced floorspace in the factory, reduced operator time and reduced equipment infrastructure. By embedding such functions in the module, it then also becomes possible to repeat these functions after deployment in the network for maintenance purposes or to increase operational life, for example; through periodic recalibration, without the need for engineer site visits. This remote repeatability of tests will take on greater importance as optics find newer applications and markets such as in satellites [3]. The concept of creating cost effective, repurposable hardware is essential for implementation and inclusion in optical devices deployed in communications networks and beyond.

1.3 Requirements Study

The first part of this study has been dedicated to examining the practical requirements associated with the implementation of a smart transceiver equipped with a calibration engine. When reporting on the design of a calibration and control engine, we first need to define the functions required of such an engine and any limitations placed on its implementation.

Based on commercial experience and on the literature reviewed in the early sections of this report, it is proposed that such an engine would need to conform to the following requirements:

- The chosen solution must demonstrate the capacity to process at least 16 tuneable lasers plus modulators and other components forming part of a 16 channel multiple wavelength tuneable module; and must be easily scalable for future use with larger scale integrated arrays.
- The capability to carry out automated measurements and recording of data. This data should be stored in on-unit non-volatile memory and should be available to be transmitted through the test management interface to an external host.
- The calibration engine must be able to perform the computations associated with the calibration routines for both transmit and receive functions in order to provide on-unit analysis of the measured data; for example, sample mean, standard deviation, z score, identification of peaks, troughs and points of maximum slope.
- The solution must be able to use a standard optical module management interface for communications – this is typically a low rate low pin count

serial protocol such as I2C or IEEE 802.3 MDIO [4] and is defined by a multiple supplier agreement (MSA) which standardises the user side interface. There are two drivers towards the use of the standardised management interface. Firstly, the cost and complexity of implementing a high speed serial interface, e.g. USB with the relevant software stack and hardware physical layer, can be avoided. Secondly, by retaining the standardised management interface, existing system level management solutions can be easily modified to provide access to this calibration functionality while the module is 'in the field'. There is currently no ratified physical layer agreement in place for modules with a transmission capacity > 800 Gb/s (Figure 1-2) although this activity is ongoing within the industry standards bodies as two separate groups, one (IEEE P802.3df [5]) defining 800 Gb/s Ethernet based on 100 Gb/s lane technology and the second (IEEE P802.3dj [6]) defining 800Gb/s and 1.6 Tb/s using lanes of 200Gb/s and above. A set of MSAs for 800Gb/s pluggable modules targeting datacentres [7] [8] has been released however, and the current lack of a physical layer standard is not an issue to the pursuit of this study. The management interface is defined within the MSA covering the module form factor, but as the physical layer standards evolve this could also change. There is no requirement for the interface to be the same between factory and field. Taking advantage of the reconfigurable nature of FPGAs, the interface protocol can be changed to suit the requirements of the application. Complex higher rate protocols such as Ethernet and USB would not generally be chosen for the calibration engine due to the increased hardware overhead of implementing the physical layer of the interface and the software overhead associated with these protocols which would not usually be required in the control application. As these serial interfaces tend to be low rate – in the region of 2.5 Mb/s in the case of MDIO and 1 Mb/s for I2C, some processing of the data on module is essential to prevent this interface becoming a bottleneck to the factory process.

- The calibration engine must be able to perform autonomous test and calibration routines with minimal operator interaction. The upload and implementation of new test routines should be simple. The test solution development needs to be separate from the engine design activity. Skilled FPGA designers and embedded software engineers are a scarce resource more effectively used in product development, designing FPGAs and software, whereas skilled test engineers do not necessarily have the specific skills required to develop FPGA based solutions. The requirement for significant upskilling of engineering staff in order to adopt this method of working would not be viewed favourably by industry, therefore this project must create a base solution to which new test scripts or test application software can be downloaded without requiring an FPGA redesign.

- The additional cost of the electronics required to implement the calibration engine must be significantly less than the cost of the optical assembly. Some initial tests of the optical components are performed at the wafer production stage to identify faulty optics. These tests, from experience, tend to be repeated throughout the subsequent manufacturing steps. There is scope for cost savings not only from automating these tests, but from the possibility of reducing the number of reiterations by eliminating some tests with the risk that faults will be discovered during the calibration activity later in the process. As a result, the calibration and control electronics need to be considered disposable. The cost of the control-related electronic components, including power supplies and TEC related components, should be minimised. The controller device, used to implement the calibration engine, should have multi-channel ADCs and/or DACs integrated into the device in order to save costs, both monetary and board space. The ADCs and DACs used for calibration, generally, will only require a low sample rate of, perhaps, 1 MSps once allowances for the settling times of the electronics to an input change are made. This cost consideration excludes the PCB and the high speed traffic parts of the design. These additional costs can be amortised if there are increases in throughput and reductions in the cost of manufacturing through reductions in test plant equipment. If the cost of the calibration engine is kept very low with respect to the cost of the optical assembly then, even if repurposing is not implemented, the potential of cost savings by integrating test and calibration functions on the module still exists.

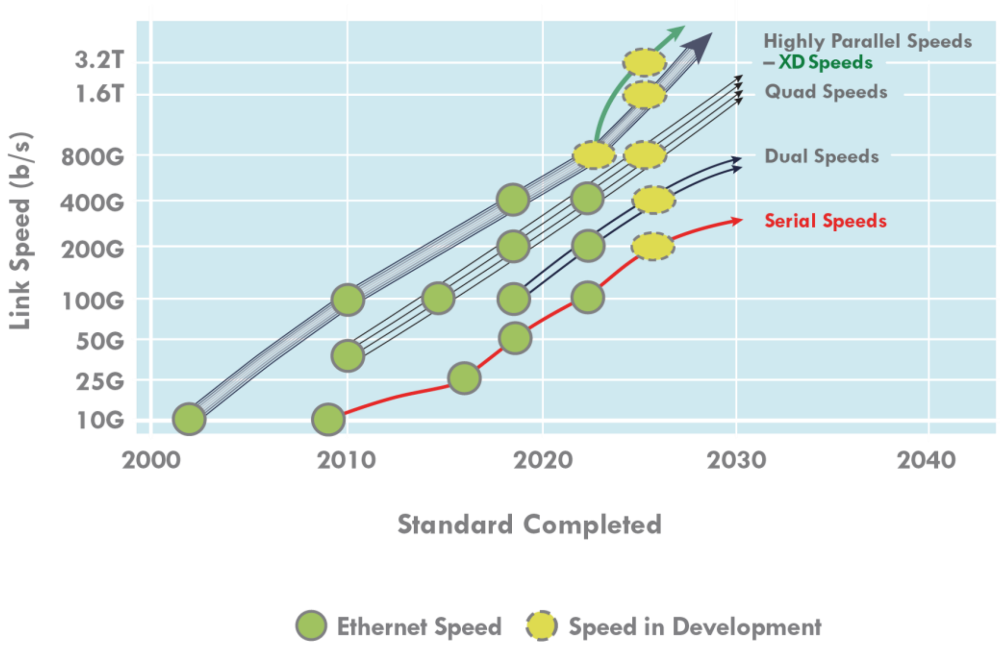


Figure 1-2 Ethernet standards evolution – The path to single lane [9].

The cost sensitive component area is illustrated in Figure 1-3 . The dashed line encloses the items to be included in the cost minimised design.

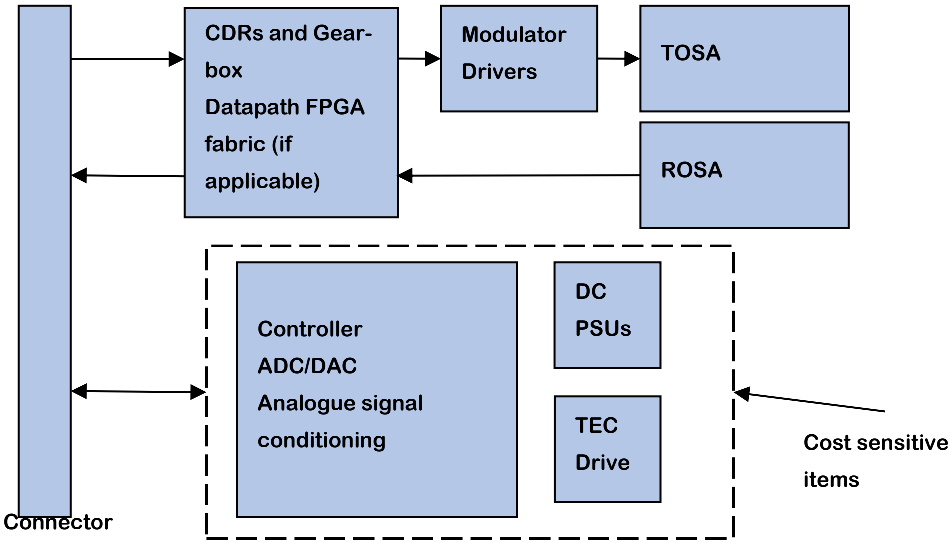


Figure 1-3 Illustration of the cost sensitive components of the calibration engine within an optical module.

The options available for implementing such circuits are:

- FPGA alone, all the test routines and processing would be coded as hardware modules. This would provide a solution with a high degree of parallelisation, able of performing multiple processing tasks simultaneously, but, it would prove harder to make subsequent test routine changes without involving the FPGA design team. Also, although FPGAs are very effective at performing high speed repetitive actions with deterministic timing, they can be inefficient at implementing functions such as arithmetic dividers which become hardware resource costly, or functions which may only be required once or twice in a test and otherwise sit dormant. During in-field use an FPGA based solution can prove to be very powerful – as it is able to parallel process multiple control loops in a deterministic time.
- Microcontroller alone, using a low level operating system and Embedded C code – this is a feasible route but requires the product design team to assist in the development of test routines. Although new programming files can be written; the hardware itself is dedicated to being a microcontroller. Many small microcontrollers execute processes in an interleaved manner, leading to non-deterministic execution times.
- An embedded high-level operating system such as Linux would offer the ability to use scripts written in standard scripting languages such as shell script which is designed to be run using the command line interpreter built into Linux. Other high level, interpreted languages such as Python could be proposed but the impact of including support for these in the operating system image will need to be analysed.

The availability of embedded, high level and customisable yet compact operating systems such as Linux is providing an accessible route for non-programmers into these embedded devices. After the initial effort of creating an operating system build tailored to the hardware in question, subsequent activities become no less familiar than file manipulation on a regular PC and executing commands from a command line interface.

The optimum solution is expected to be a mix of FPGA (for deterministic timing and parallelisation of functions) and embedded microcontroller or other processor functionality for controlling sweeps and processing data. The FPGA fabric can be used to implement both virtual instrumentation functions and coprocessors such as an ALU for enhanced software performance.

An example of a laser controller device functional architecture, showing the main functional components is given below in Figure 1-4. In this example, based on the architecture of the Intel MAX10 devices [10], communication off the device is via a low speed I2C interface, the DAC is a discrete component accessed over a serial interface whereas the ADC, in this device family, is integrated onto the chip. 4 parameters are being controlled - optical power, optical frequency, optical modulation depth and unit temperature – using a state machine.

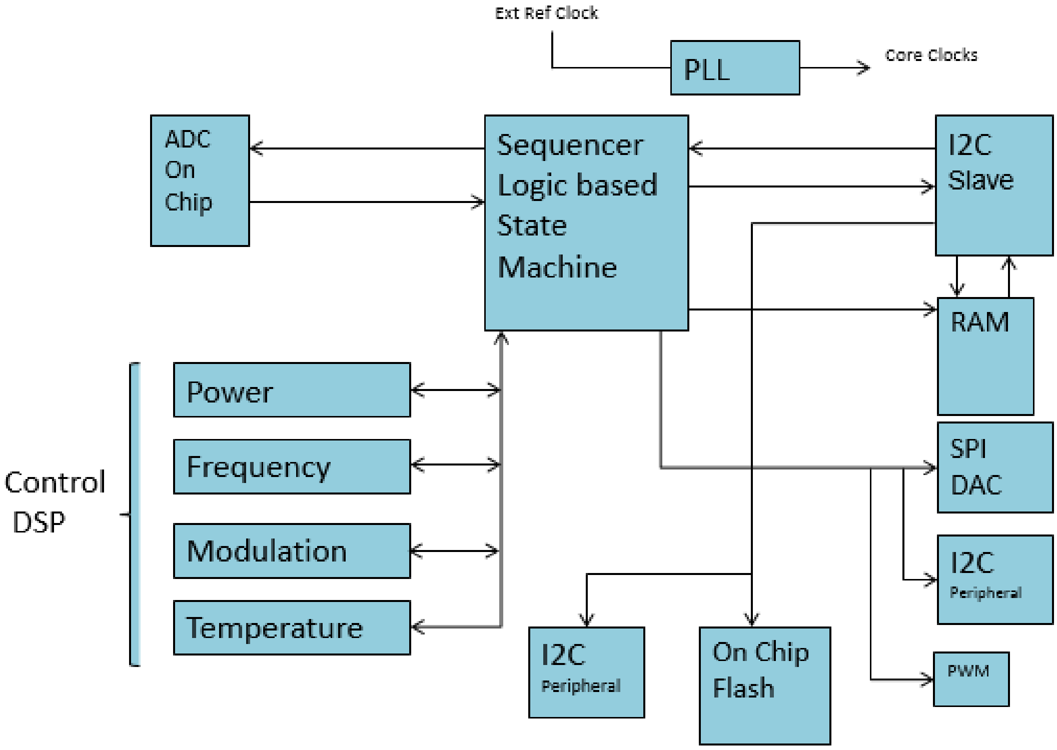


Figure 1-4 Example design of a laser controller based in an Intel MAX10 FPGA.

By inspecting the common driver functions between the controller and the calibration functions it becomes possible to propose the calibration engine architecture shown in Figure 1-5. In this case, the ADC and DAC peripherals remain the same. Communication off the device is either by the same I2C or the pins and logic can be repurposed to an alternative 2 wire interface standard for increased throughput or compatibility with the test environment. The state machine controller logic has been reused by a soft microcontroller with a separate Arithmetic Logic Unit (ALU).

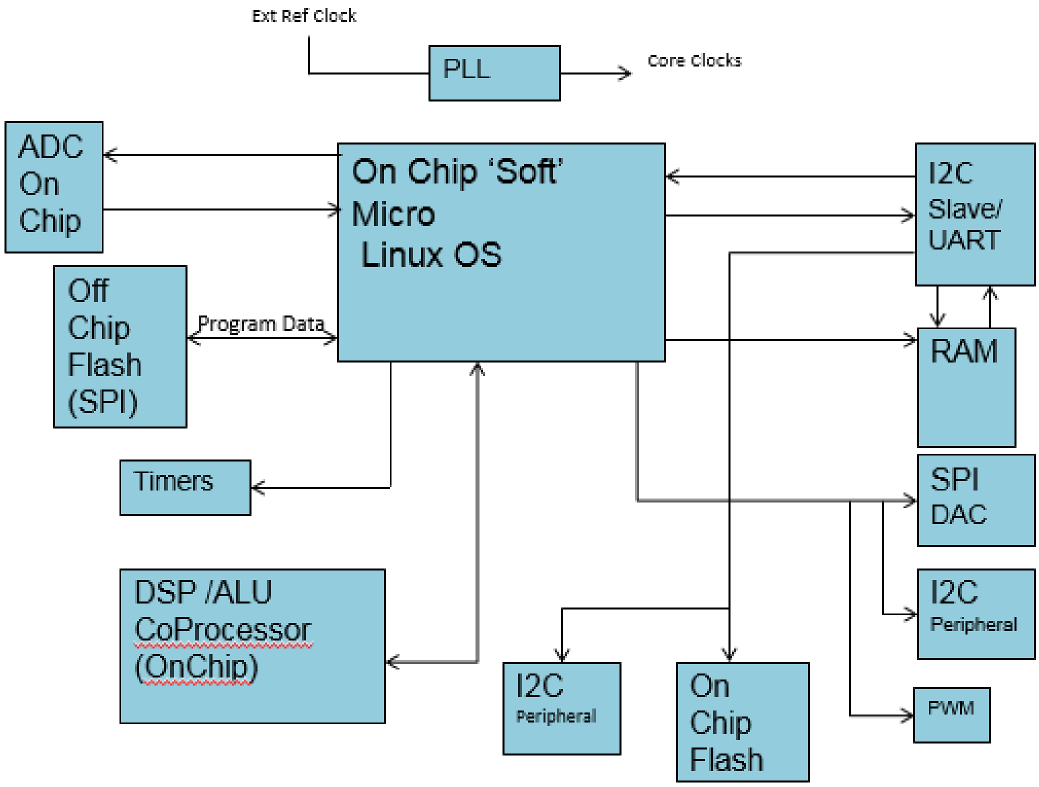


Figure 1-5 Example design of a laser calibration engine based in an Intel MAX10 FPGA.

1.4 Hypothesis

For all the reasons given above, it would appear to be a worthwhile endeavour to investigate what would be required in a smart photonic module to achieve the goal of increasing test efficiency when manufacturing the device, and enabling remote retest with the aim of increasing service life, by embedding the appropriate functionality in the module. The following hypothesis is therefore put forward:

'That it is possible and practical to design repurposable hardware functions, hosted in a photonic module and based on FPGA technology, to enable test and characterisation activities to be performed in a cost efficient manner in the factory or laboratory and remotely after installation.'

2 PARALLELISATION, SUPERCHANNELS AND THE NEED FOR ARRAYED MULTICHANNEL COMPONENTS

In this chapter the requirement for arrayed components is introduced. Such arrays can be a means of improving optical channel density in optical communications equipment. The increasing requirement for optical bandwidth, and the nature of the traffic driving this increase is examined and the principle of Superchannels, constructed using arrays of components, is discussed as one possible solution to this growth. Finally, the implications on the manufacturing process of packaging arrays of components together in a single transceiver entity are examined.

2.1 The Growth of Optical Network Traffic

In an invited tutorial for the Journal of Lightwave Technology [11] Peter J Winzer and David T Neilson examine the current trends in network growth which has been seen as increasing across all segments of the network. They identify this growth as coming from the ongoing development of newly emerging and largely unpredicted digital services. These new services are creating new sources of traffic which are beginning to overtake the previously dominant sources. For example, data traffic overtook voice traffic in the early 2000s [12], only to be then overtaken by video streaming and now machine to machine traffic, which dominates the IP traffic of operators today.

At the beginning of this study, in 2016, smartphone traffic was being predicted to exceed personal computer (PC) traffic over the period 2016-2021, rising from 13% to 33% of global IP traffic [13], and with the emergence of the Internet of Things (IoT) expected to dominate mobile data traffic in the near future [14]. By 2021, the number of IP connected devices was forecast to be 3.5 per capita globally [13].

Winzer and Neilson presented their view of a network consisting of relatively few, large, datacentres connected together by long haul networks. These connect to a greater number of regional metro networks to terminate at edge servers where content from many sources is brought together and handed off to ISPs. The regional networks would require a higher degree of connectivity since the number of metro areas would likely exceed the number of large datacentres. The metro space is where they expect to see the need for most switching and reconfiguration: users will switch between the kinds of applications or content they use; the location of users changes as they move through the metropolitan space during the day and week; and the type of access changes with them, fixed or wireless.

They conclude that there was a, then, current requirement for 1Tb/s interfaces and that by 2024 there would be the need for 10 Tb/s optical interfaces as a part of optical transport systems operating in the 1 Pb/s range in order to avoid an optical network capacity crunch. Given the then trends in the industry, where interoperable 400Gb/s interfaces were still a work in progress [15], they deemed it unlikely that 1Tb/s and 10 Tb/s single carrier interfaces will be available on schedule to meet the identified growth in demand.

This theme of growth through new services and new traffic sources is continued in a further, more recent, invited paper investigating capacity scalability issues and options [16]. Here, a network traffic growth figure equating to a doubling every 1.5 to 2 years is presented [12] [17], and is attributed to the uptake of cloud computing and video streaming services. However, in contrast to the earlier literature, 1Tb/s per wavelength capacity has become commercially available [18] due to advances in DSP ASIC and ADC technology in the intervening time between the two studies. This later paper concludes that parallelism (in this case spatial parallelism through SDM) is key to answering this traffic growth.

Cisco Networks concur with this growth viewpoint in their Visual Networking Index 2016 to 2021 [13]. It was predicted in their report that globally, by 2021, total IP traffic will experience a 3-fold growth (24% compound growth year on year), internet traffic (defined as IP traffic crossing an internet backbone) will increase by 26% year on year with the peak demand for internet traffic increasing by 35% p.a. and the proportion of this traffic being carried metro to metro will increase from 22% in 2016 to 35% by 2021 (see Table 2-1).

It was this anticipated increase in metro traffic that is one of the motivating factors of this study. The need to develop and deploy cost effective bandwidth increases in metro market transceivers, and how to assist this cost saving through manufacturing cost reduction based on repurposable hardware, together with the anticipated peak demand increases highlighting the need to adopt flexibility and reconfigurability in transceivers to cope with surges in demand [19] provides this motivation.

Subsequent white papers concerning network traffic growth produced by Cisco have changed focus from the volume of network traffic to use the number of connected devices and users as well as connection speeds as a metric [20]. The reported trend is still one of large scale growth. Between 2018 and 2023 the number of internet users is expected to have increased from 3.9 billion to 5.3 billion (6% compound growth year on year), the number of total connected devices is increasing at a compound annual rate of 10% and of these devices, M2M or IoT connections are expected to increase at the greatest rate reaching a predicted 14.7 billion connections by 2023 (see Table 2-2).

IP Traffic, 2016–2021	**2016**	**2017**	**2018**	**2019**	**2020**	**2021**	**CAGR 2016–2021**
By Type (Petabytes [PB] per Month)							
Fixed Internet	65,942	83,371	102,960	127,008	155,121	187,386	23%
Managed IP	22,911	27,140	31,304	35,226	38,908	42,452	13%
Mobile data	7,201	11,183	16,646	24,220	34,382	48,270	46%
Total (PB per Month)							
Total IP traffic	96,054	121,694	150,910	186,453	228,411	278,108	24%

Table 2-1 Predicted increase in IP traffic to 2021 [13].

Connection speeds, 2018–2023	**2018**	**2019**	**2020**	**2021**	**2022**	**2023**	**CAGR (2018–2023)**
By Type (Mbps)							
Fixed Broadband	45.9	52.9	61.2	77.4	97.8	110.4	20%
Mobile	13.2	17.7	23.5	29.4	35.9	43.9	27%
WiFi (inc public hotspots	30.3	36.3	50.8	58.9	72.9	91.6	25%

Table 2-2 Predicted increase in network connection speeds to 2023 [20].

Following from Ciscos predictions of network growth, Nokia are predicting a continuing picture of increasing growth to 2030 [21]. Based on 4 possible scenarios, termed Conservative (least growth), Moderate, Aggressive and Disruptive (most growth), using CAGR estimates between 17% and 32% they predict global internet traffic will reach between 1730 EB and 4819 EB per month through 2030.

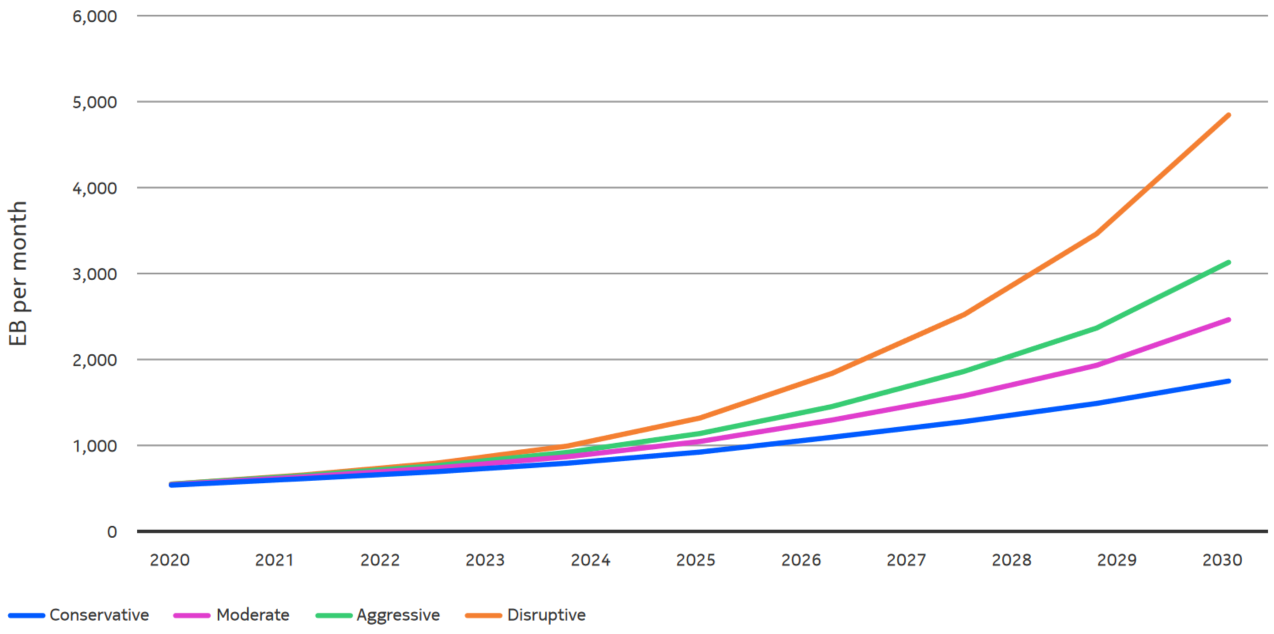


Figure 2-1: Predicted growth in global internet traffic to 2030 under 4 possible scenarios with CAGR ranging from 17% to 32% [21]

The major network equipment suppliers, such as Cisco and Nokia, are not the only source predicting massive data expansion. In a Seagate sponsored study titled 'Data Age 2025' [22], the market intelligence company IDC predict a 'Datasphere' (the sum of all data created, captured and replicated) which will increase 10-fold from 16.1 ZetaBytes in 2016 to 163 ZetaBytes by 2025 (Figure 2-2). Their predicted growth is based on IoT and the emerging cloud-based AI data analytic systems.

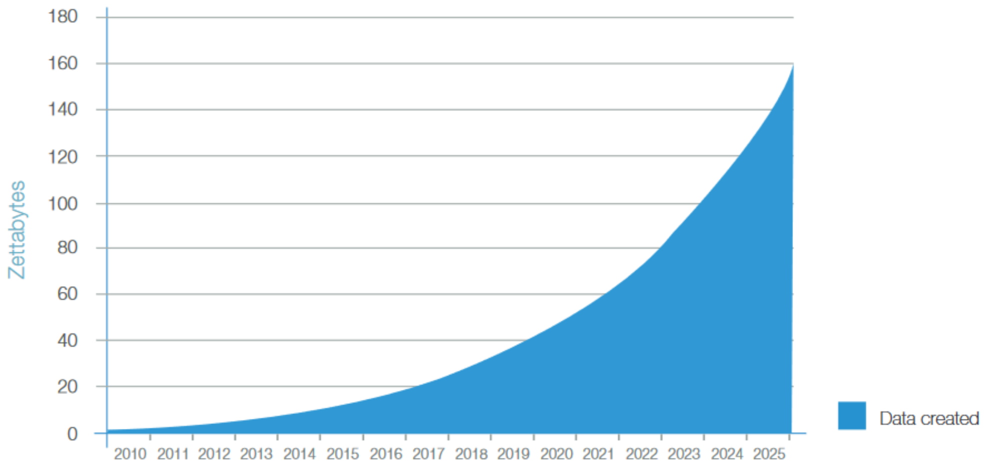


Figure 2-2 Annual size of the Global Datasphere [22].

The diversity of sources contributing to this data growth is clearly summarised in Figure 2-3 from The Ethernet Alliance industry consortium.

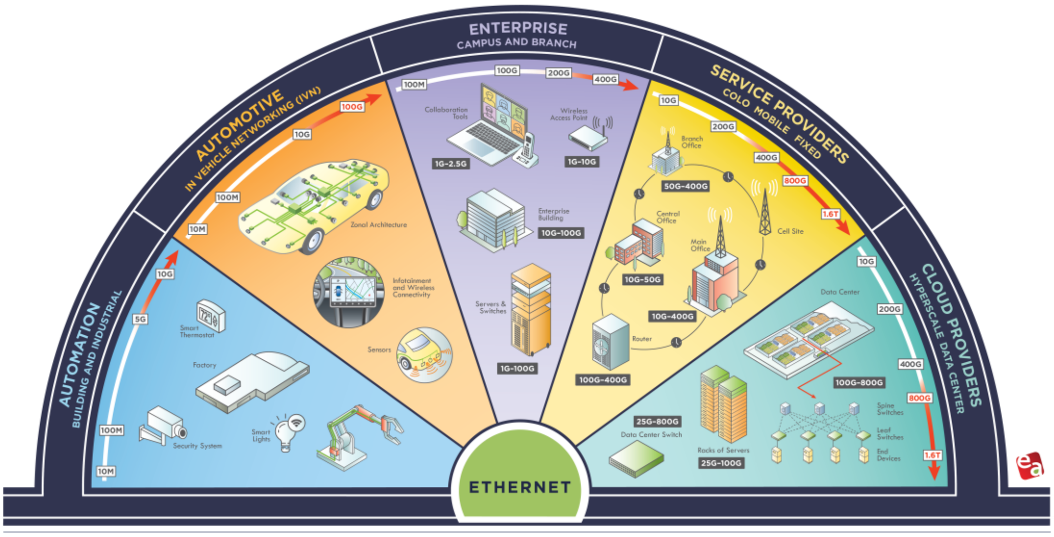


Figure 2-3 The diversity of Ethernet traffic sources [9].

2.2 Increasing Bandwidth through Parallelism

Driven by a relentless demand for ever increasing amounts of optical network capacity, generated by advances in web based services, cloud computing and media content quality, the long established optical multiplexing techniques of Dense Wavelength Division Multiplexing (DWDM) have evolved over recent years to define what was termed a ‘Superchannel’ in 2009 by S.Chandrasekhar and X.Liu [23]. Faced with finite fibre bandwidth and including the cost of installing new fibres, improving the spectral efficiency of DWDM systems has been recognised as an effective way of addressing the growing demand [24].

In order to address the requirement to improve the utilisation of the available optical spectrum and a need by the network service providers to improve operational efficiency, the Superchannel proposes combining multiple optical carriers or subchannels together into a single unified entity which is brought into service in a single commissioning cycle, hence improving the bandwidth increase to operational effort ratio, and is transmitted through the network as a single entity. This allows the gap between optical subchannels within the Superchannel to be reduced compared to conventional DWDM, as shown in Figure 2-4, reducing filtering penalties and resulting in increased spectral packing and improved utilisation of the available spectrum [25] [26].

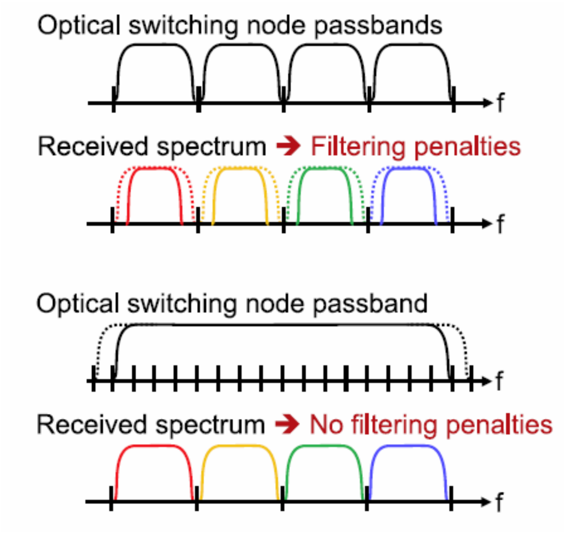

Optical switching node passbands
f
Received spectrum → Filtering penalties
f
Optical switching node passband
f
Received spectrum → No filtering penalties
f

Figure 2-4 Spectral Superchannels avoid filtering penalties [11].

In their tutorial Winzer and Neilson [11] explore a range of possible bandwidth scaling options including improved fibre, which is dismissed as requiring extensive research effort to make a moderate gain, and advanced modulation which is already in use in QAM systems. However, there are trade offs between higher order modulation schemes and top end reach and laser phase noise and ADC/DAC resolution at the lower reach end, which puts a limit to advancing both single carrier interface rates and WDM system capacities through higher-order modulation.

They propose a solution based on wavelength parallelism using both wavelength (spectral) and space (spatial) multiplexing to form a hybrid Superchannel as one possible solution which would be applicable in some network scenarios.

Spatial multiplexing in the optical domain is a mechanism whereby multiple data streams are carried in multiple transverse modes of a fibre that is not single mode. Multiple modes in multimode fibre, multiple cores in multicore fibre or a combination of the two can be used to implement spatial multiplexing [27] [28] [29]. These multiple data streams may all use the same optical carrier frequency hence increasing the data bandwidth transmitted over each carrier. A comparison of the architectures of a spectral and spatial multiplexed transceiver is shown in Figure 2-5. This clearly shows the reduced number of lasers required by spatial multiplexing compared to spectral multiplexing, for the same number of channels. One of the drivers behind this approach is to make best use of the available wavelengths across the C + L bands in the low loss fibre transmission window and the challenges of using frequencies outside of these bands due to OH absorption (~1400 nm in the E band) and interference from Raman pumps placed in the S Band in order to provide gain to the L band. The adoption of spatial multiplexing as part of the optical network solution is initially most likely to occur where the required multiple parallel fibre paths either already exist or new fibre would be deployed with the system (e.g. submarine links) rather than in the metro market where there is already an existing fibre base.

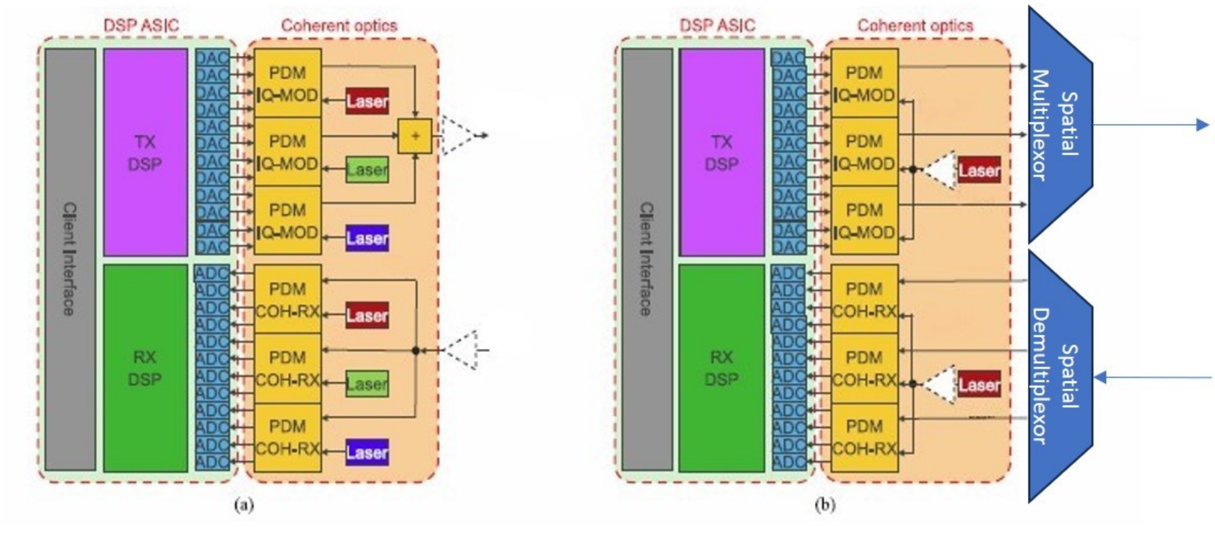


Figure 2-5 Structure of (a) spectral vs (b) spatial Superchannel transceiver – From Scaling Disparities to Integrated Parallelism [30].

The hardware of a spatial Superchannel transceiver is similar in concept to that required by a spectral Superchannel transceiver, consisting of laser sources, receivers, modulators and possibly amplifiers. When examining the possibility of manufacturing such a device, the complexity of the calibration activity and the in-service control of the optical components can be thought of as being broadly similar.

The use of spectral super channels as a path to evolve into hybrid Superchannels is considered a favourable path from current technology [11].

However, Winzer and Nielson's views on the downside of higher order modulation are not universally held. In a paper published in 2014 [31], Charles Laperle of Ciena Corporation explores the requirements of next generation transceivers. In his literature Laperle reports on the needs of both the long haul/submarine sectors and the metro/data centre market, identifying that a significant cost reduction is required and, in his opinion, that there was a need to carry 400Gb/s on a single wavelength. He supports his position by examining advances in ADC/DAC and DSP technology as enablers for advanced multi-dimensional modulation [32], together with compensation for optical fibre non-linearities [33] and stronger FEC algorithms to improve coding gain, in order to address the limitations identified by Winzer.

The methods proposed by Laperle are brought together to present an agile, flexible rate transceiver solution where factors such as reach, noise tolerance, data rate and carrier occupancy may be reprogrammable – possibly, even by the transceiver itself by means of machine learning.

Whichever approach is taken the benefits of having a reconfigurable transceiver for the network operator remain the same, namely:

- The ability to respond quickly to service requests.
- Cost reduction for supporting short term or busy hour increased bandwidths.

Writing in an article for the Lightwave journal [34], Geoff Bennett from Infinera draws a good analogy between Superchannels and the development of multicore microprocessors (Figure 2-6) where, in both cases, it is recognised that the parallel architecture has allowed for a greater step improvement in operational performance than the underlying technology would otherwise have allowed in the same time period, but this is hidden from the user who only sees a single unit of capacity.

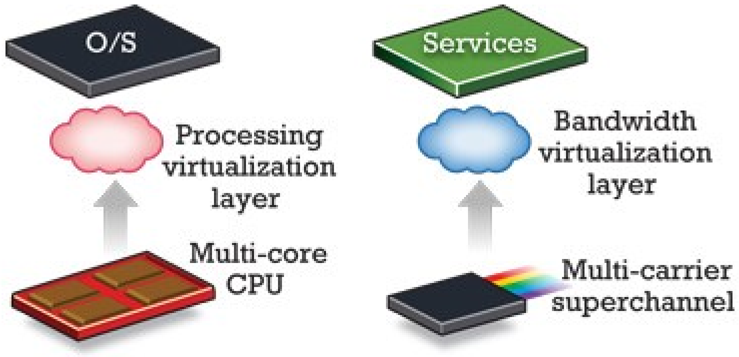

O/S
Processing virtualization layer
Multi-core CPU
Services
Bandwidth virtualization layer
Multi-carrier superchannel

Figure 2-6 Parallel processing in the CPU and Superchannel World – Superchannels to the Rescue [34].

So far, the commercial use of Superchannels, and the majority of research projects, have primarily been targeted at the Long-Haul market, where there is a need for high bandwidth connections; and the unit of capacity required to be brought into service is many multiples of 100Gb/s, even beyond 1Tb/s. This technology is now rolling back into the metro network arena, where high bandwidth user connections, for example High Definition Television broadcast, ever-evolving network standards and the interest in flexible Software Defined Networks (SDNs) are presenting new challenges for the network operator. The metro market is far more sensitive to cost, size and power consumption when compared to long haul. Two independent and highly influential US industry reports predict the concentration and localisation of traffic within the metro network as the demand for IP traffic increases. In one, the anticipated proliferation of data-centres for video delivery and cloud services was expected to drive a 560% increase in metro traffic by 2017 such that it was increasing at double the long haul traffic rate by 2017 [35], while the other predicted metro traffic would surpass long haul traffic, and grow at twice the long haul rate to 2017 [13].

While, in the long-haul market, a unit of 1Tb/s or above capacity may be appropriate, in the metro market a customer may not require all this capacity at day one. They may also not know what protocol they will wish to connect with in the future and they certainly will not want to pay for capacity which is not being used, instead preferring to take a ‘Pay as you Grow’ approach. This day one cost is presenting an obstacle to the take up of Superchannels, yet in order to provision for future growth and to make use of network operational efficiencies, a 1Tb/s +

Superchannel transceiver may be appropriate in the longer term.

Irrespective of the type of Superchannel generated (Spectral or Spatial), the architecture of future Superchannel transceivers will tend towards the integration of components into large scale (in the context of this thesis, large scale is defined as being 16 elements or greater) arrays in order to benefit from reduced cost, energy and footprint. Such transceivers will likely contain arrays of IQ modulators and SOAs, as well as tuneable laser sources per spectral subcarrier [36].

Tuneable laser sources require control algorithms to manage their phase and bias electrodes as well as temperature. Optical modulator bias currents require more complex control functions based around sinusoidal pilot tones and 1st/2nd order harmonic detection via bandpass filters, or using the calculated ratio of monitored input and output power levels. Both these functions can now be performed digitally using digital frequency synthesis and DSP digital filtering techniques.

All control functions will require factory calibration during manufacture to set optimum operating parameters for maximum stability over the expected operational life of the transceiver. From the industrial experience of the author, due to the complex nature of the optical techniques employed (particularly in the tuneable laser and Mach Zehnder modulator), the verification, calibration and set-up process for such devices requires a significant amount of external test equipment and is time-consuming. In particular, the capital cost of the required test equipment significantly increases the business overheads and also the incremental cost of scaling to higher production volumes. Floor space used by the equipment also contributes to overheads. This situation is exacerbated by the fact that several batches of tests are repeated a number of times throughout the manufacturing cycle, requiring further equipment and test time.

For the single tuneable laser, these costs are manageable. However, for modules containing arrays of lasers and modulators, the simple replication of an identical process across all of the laser-modulator transmit paths would potentially result in a direct multiplication of the costs and effort involved.

Some of these scalability issues have been examined in the context of a typical ‘Rack and Stack’ type factory test solution for a 4x25 Gb/s optical transceiver [37]. Here, the test time for 4 characterisation tests (referred to as Optical DC, Optical AC level, Optical AC level margin and Optical AC total jitter) at a single wavelength, performed using a typical factory test approach, was presented as being 311 seconds. Including the additional testing activities required for a Transmit Optical Sub Assembly (TOSA) featuring a single full band tuneable laser source, SOA and Mach Zehnder modulator a test duration of 30 minutes, as suggested in private communications and derived from factory process records, seems to be reasonable. This activity is repeating tests previously completed at the wafer level, so faulty components have already been filtered out, on a complete assembled device after assembly bakes and thermal cycling. For the mode test the SOA and Mach Zehnder controls are fixed and scans are performed to collect data for mode analysis. Analysis of the data is carried out on a PC using custom software. The checks and analysis carried out include:

- Estimation of mode centre lines
- Average optical power for each mode (used to calculate output SOA settings)
- Identification of areas of phase current hysteresis
- Estimation of mode hysteresis-free centre lines
- Estimation of mode forward sweep centre lines

For the Mach Zehnder modulator the characterisation techniques normally involve numerous steps performed, as detailed below, with no RF modulation in order to calculate the z-score of characterised values. The z-score is a statistical measure of variability in the batch under test, which indicates the difference between a value and the mean of the value measured across a batch:

1. Sweep DC and AC Electrode currents at discrete points
2. Take multiple samples of output power at each point
3. Calculate the mean of these samples at each point
4. Calculate the standard deviation (from each sample subtract the mean, square that result, find the mean of the squared differences. The square root of the mean squared difference is the standard deviation)
5. Normalize the samples by calculating the number of standard deviations from the mean of the sample (z-score)

The results of these tests are stored off chip in a large database.

This activity is repeated throughout subsequent test steps as small changes to most parameters e.g. determining optimum SOA settings for each tuning point, cause a change to previous calibrations.

Considerations towards automated testing have been presented by main stream component vendors such as Oclaro [38] who implemented a full set of integrated waveguide detectors along with phase controls for use in set-up, testing and ongoing monitoring of a dual modulator (Figure 2-7), although the primary aim of that work was to examine possible routes to package miniaturization and co-packaging with a tuneable laser [39].

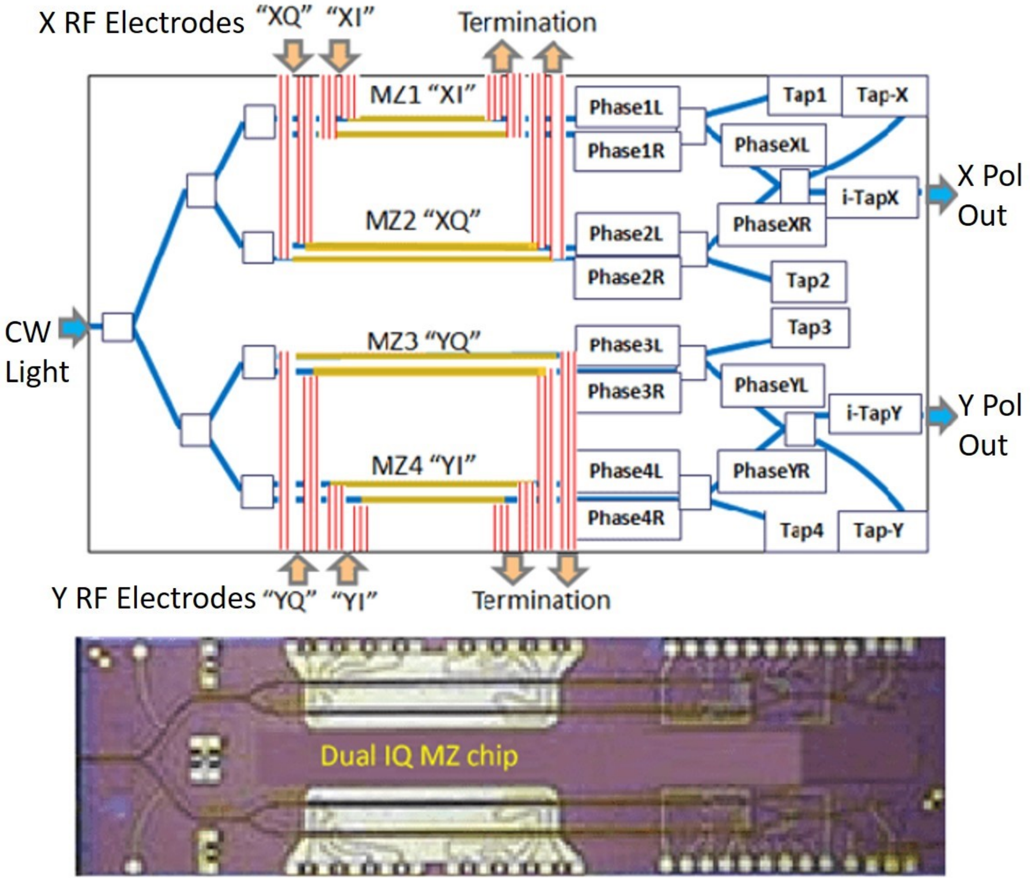


Figure 2-7 Schematic functionality of the Dual IQ Modulator Chip together with an optical image [38]. Phase control is by thermal elements 'Phase(n)L/R'

2.3 Summary

The pressure on the capacity of optical networks is increasing at an ever-increasing rate. In this chapter the growth in network traffic has been examined using data from a number of different published reports, from different sources, covering the period from the beginning of this study in 2016 through to 2030 and all echo this continuing growth prediction. Not only is the capacity increasing, but the nature of the traffic responsible for this growth is changing from one led by the metro market to one dominated by many low bandwidth IoT devices communicating with centralized datacentres. Also driving this growth is an increase in cloud based services such as streaming high-definition TV on demand. This shift in the nature of the traffic drives a need to offer network flexibility e.g. through reconfigurable transceivers to manage 'busy hour' bursts throughout the network.

The use of arrays of co-packaged components to build future generations of optical transceivers, utilising parallelism and arrays of components to generate Superchannels, with the bandwidth to address the continuing growth in network traffic has been examined. Two specific methods of adopting parallelism were discussed – spatial and spectral multiplexing – in the context of the hardware architecture of devices implementing these methods and some of the steps in the manufacturing process have been introduced to highlight where costs will increase.

In conclusion, the challenges presented during the manufacturing of such devices, in order to keep costs under control (as discussed in Section 1.2), will call for innovative solutions to be found.

3 FIELD PROGRAMMABLE GATE ARRAYS

This chapter introduces the principal technology of the Field Programmable Gate Array (FPGA). The history of this technology, from the basic architecture, features and advantages of the devices to the trend towards the introduction of more specialised cells, dedicated to a single high performance function is presented. A comparison with Application Specific Integrated Circuit (ASIC) technology is made to highlight where FPGAs are finding an advantage. Finally, the dual role this technology can play in the operation and manufacturing of arrayed or multichannel optical devices, leveraging the reconfigurable nature of this technology, is assessed.

3.1 Introduction to FPGAs

FPGA technology has been commercially available since the mid-1980s. The FPGA is a user configurable integrated circuit which consists of a matrix of low level or primary functions such as Logic Blocks containing Look Up Tables (LUTs) for storing Boolean logic equations and storage elements such as latches or memory connected through a configurable interconnect or routing layer (Figure 3-1). In a short paper examining the benefits of FPGAs in the context of the implementation of Network Interface controllers using programmable hardware [40], Gordon Brebner of AMD/Xilinx Inc. examines the benefits of FPGAs and identifies a key feature of FPGAs as being that they offer the performance of hardware with the flexibility of software. This is an important point. FPGAs should not be considered to be just hardware or software but, when coupled with higher level description languages such as VHDL or System C interfaces into tools such as MATLAB™ and high level abstraction design entry methods using C++, known as HLS, they can form a flexible tool accessible to a wide range of designers and engineers.

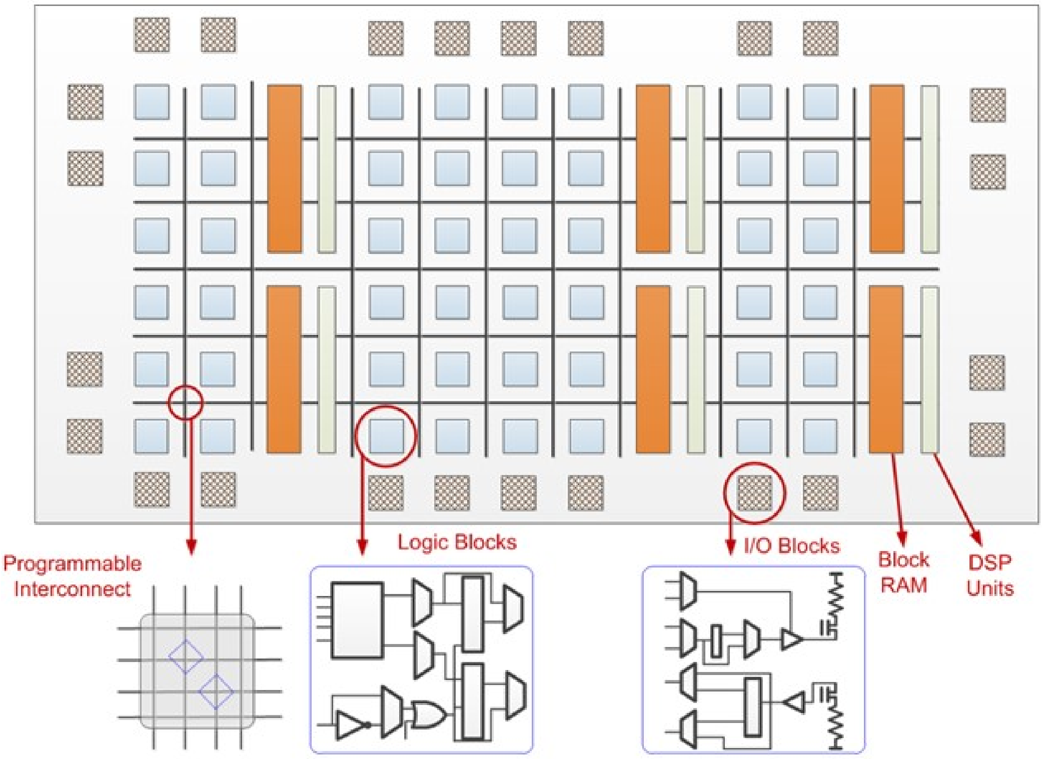


Figure 3-1 The basic architecture of a FPGA [41].

Initial devices were able to offer in the region of 128 3-input LUTs in a device with which to implement a design. In the subsequent 30 years, FPGA technology has seen many significant advances. As semiconductor geometries have shrunk to 14nm and below, internal clock speeds have increased, power has been reduced, features such as embedded high performance microcontrollers and high speed serial interfaces have been added and LUT counts (giving a measure of digital logic capability) have increased in size and complexity to several millions of 6 input LUTs (see Table 3-1), giving unprecedented reconfigurable logic capacity [42].

Process Technology (Device family)	28 nm TSMC (Cyclone V)	20 nm TSMC (Arria 10)	14 nm Intel Tri-Gate (Stratix 10)
Processor	Dual-core ARM Cortex-A9 MPCore	Dual-core ARM Cortex-A9 MPCore	Quad-core ARM Cortex-A53 MP Core
Maximum Processor Performance	1.05 GHz	1.5 GHz	1.5 GHz
Logic Core Performance	300 MHz	~500 MHz	1 GHz
Power Dissipation	1X	0.6X	0.3X
Logic Density Range	350 – 462K logic element (LE)	160 – 660K LE	500K LE - 5.5M LE
Embedded Memory	23 Mb	39 Mb	229 Mb
18 x 19 Multipliers	2,136	3,356	11,520
Maximum Transceivers	30	48	144
Maximum Transceiver Data Rate (Chip to Chip)	10 Gbps	17.4 Gbps	30 Gbps

Table 3-1 Intel/Altera SoC family comparison of performance and logic capacity (Compiled from [43], [44], [45]).

FPGAs, however, have long been thought of as the ‘poor cousin’ to custom or semi-custom ASICs due to the latter’s general higher performance and lower power. Subsequently a number of factors are now considered to be changing this dynamic:

- Integrated circuit costs are rising – see Figure 3-2 – mask costs alone at 40nm can run into millions of dollars [46] and 100s of millions of dollars at the latest 7 nm and smaller process nodes [47] [48] which are enabling the Terabit rate pluggable optical modules [18]. This reduces the number of developments which can be committed to by a commercial entity.
- ASIC complexity has lengthened development time.
- R&D resources and headcount are decreasing.
- Revenue losses for slow time-to-market are increasing – standards are evolving rapidly (see Figure 1-2).

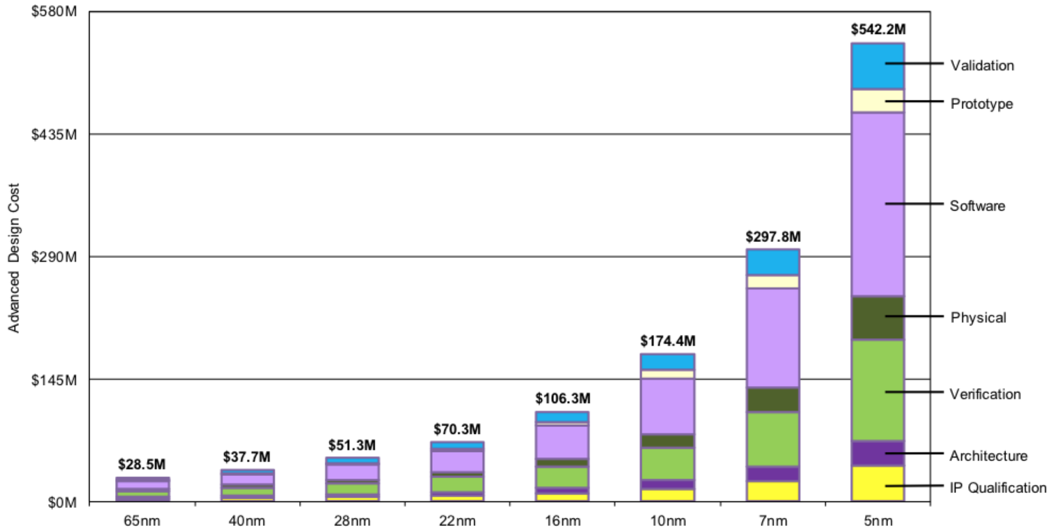


Figure 3-2 ASIC design costs comparison [48].

These trends are encouraging FPGAs to be looked at as an alternative to ASICs for applications where they previously may not have been used.

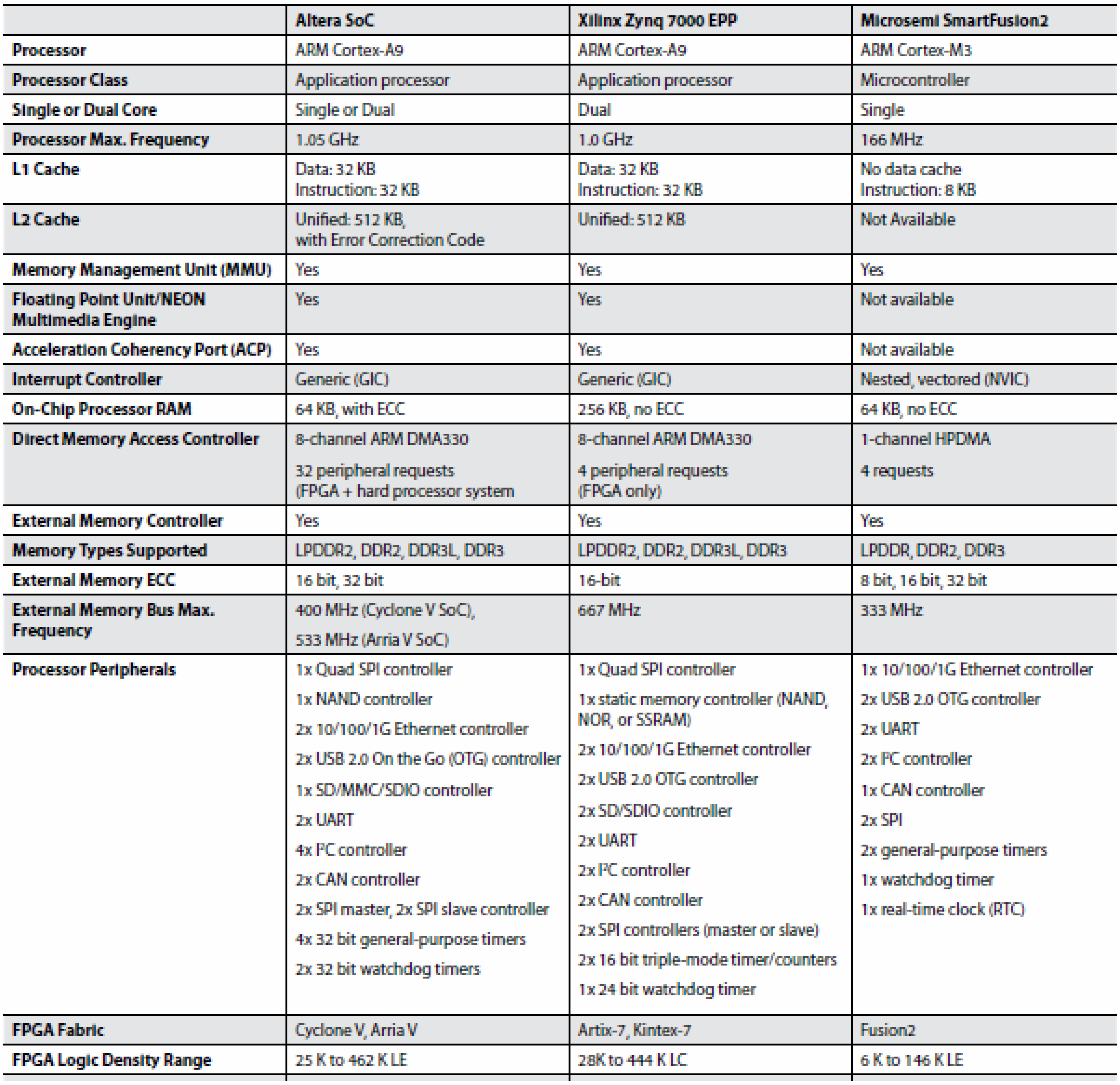
Advances in FPGA technology in recent years have seen the features usually reserved for high end, high cost devices roll down into the mid-range, lower cost families, while the high end devices get larger. For example, the current mid-range Intel/Altera Arria 10 family [49] features a large number (‘000s) of high speed DSP multipliers, the capability to support floating point arithmetic in a non-microcontroller, concurrent processing environment and up to 28 Gb/s serial IO; AMD/Xilinx Inc. have developed another enhancement by including multiple 5 GS/s 12 bit ADCs and 9.85 GS/s DACs on a chip, initially aimed at the 5G radio market, and Intel/Altera have announced a further development of a new device family with the availability up to 8 64 GS/s 10 bit ADCs and DACs on a single device [50] Together with high speed DSP capability [51], this supports the possible development of transceivers using reconfigurable higher order modulation schemes while benefiting from a lower power consumption due to the removal of the high bandwidth electrical interface between data convertor and DSP.

Low power, embedded microcontrollers can now be found even in some of the lower end devices (See Table 3-2), where the ARM M class processor has proven itself to be very powerful [52] – running applications as complex as hosting webpages. In the mid and high end devices, ARM A class processors appear to dominate [53]. System performance can be optimized by developing hardware parallel processing engines using FPGA logic and integrating with software algorithms running on the processor system of a SoC FPGA. SoC is the mixing of multiple, diverse system elements such as digital logic/FPGA, microprocessor, volatile and non-volatile memory into a single integrated package in order to reduce board footprints and save power due to reduced IO counts. Finding a balance between FPGA and software implementations has enabled functions to be handed off which previously would have been very slow in software. The nature of FPGA technology and design methodology is intrinsically parallel down to a very low level. This parallel structure allows the time spent on a processing function to be determined. Unlike processors FPGAs do not have a fixed design structure, functions can be designed so they do not have to compete for the same resources, each independent function is allocated to a dedicated area of the FPGA and can function without influence from other logic functions owing to the presence of multiple clock and routing resources.

It is this parallel structure which presents another design advantage of FPGAs. Where functions can operate autonomously it is possible to design reusable blocks or modules which can then be carried across designs once proven, so reducing subsequent design effort and ensuring design repeatability.

The increase in hardened (i.e. dedicated, non-reconfigurable functional cells) DSP multiplier availability and in the available width of these multipliers (up to 36x36 bits) has enabled high speed DSP arithmetic to be implemented efficiently in

FPGAs rather than traditional DSP processors. The advantage of this approach is that the FPGA can be configured and indeed reconfigured to perform other tasks as well.



	Altera SoC	Xilinx Zynq 7000 EPP	Microsemi SmartFusion2
Processor	ARM Cortex-A9	ARM Cortex-A9	ARM Cortex-M3
Processor Class	Application processor	Application processor	Microcontroller
Single or Dual Core	Single or Dual	Dual	Single
Processor Max. Frequency	1.05 GHz	1.0 GHz	166 MHz
L1 Cache	Data: 32 KB Instruction: 32 KB	Data: 32 KB Instruction: 32 KB	No data cache Instruction: 8 KB
L2 Cache	Unified: 512 KB, with Error Correction Code	Unified: 512 KB	Not Available
Memory Management Unit (MMU)	Yes	Yes	Yes
Floating Point Unit/NEON Multimedia Engine	Yes	Yes	Not available
Acceleration Coherency Port (ACP)	Yes	Yes	Not available
Interrupt Controller	Generic (GIC)	Generic (GIC)	Nested, vectored (NVIC)
On-Chip Processor RAM	64 KB, with ECC	256 KB, no ECC	64 KB, no ECC
Direct Memory Access Controller	8-channel ARM DMA330 32 peripheral requests (FPGA + hard processor system	8-channel ARM DMA330 4 peripheral requests (FPGA only)	1-channel HPDMA 4 requests
External Memory Controller	Yes	Yes	Yes
Memory Types Supported	LPDDR2, DDR2, DDR3L, DDR3	LPDDR2, DDR2, DDR3L, DDR3	LPDDR, DDR2, DDR3
External Memory ECC	16 bit, 32 bit	16-bit	8 bit, 16 bit, 32 bit
External Memory Bus Max. Frequency	400 MHz (Cyclone V SoC), 533 MHz (Arria V SoC)	667 MHz	333 MHz
Processor Peripherals	1x Quad SPI controller 1x NAND controller 2x 10/100/1G Ethernet controller 2x USB 2.0 On the Go (OTG) controller 1x SD/MMC/SDIO controller 2x UART 4x I^{2}C controller 2x CAN controller 2x SPI master, 2x SPI slave controller 4x 32 bit general-purpose timers 2x 32 bit watchdog timers	1x Quad SPI controller 1x static memory controller (NAND, NOR, or SSRAM) 2x 10/100/1G Ethernet controller 2x USB 2.0 OTG controller 2x SD/SDIO controller 2x UART 2x I^{2}C controller 2x CAN controller 2x SPI controllers (master or slave) 2x 16 bit triple-mode timer/counters 1x 24 bit watchdog timer	1x 10/100/1G Ethernet controller 2x USB 2.0 OTG controller 2x UART 2x I^{2}C controller 1x CAN controller 2x SPI 2x general-purpose timers 1x watchdog timer 1x real-time clock (RTC)
FPGA Fabric	Cyclone V, Arria V	Artix-7, Kintex-7	Fusion2
FPGA Logic Density Range	25 K to 462 K LE	28K to 444 K LC	6 K to 146 K LE

Table 3-2 Cross vendor comparison of low/mid range SoC capabilities [54].

Perhaps the most important feature to have been added to the FPGA design flow over recent years is Partial Reconfiguration. As the name suggests, Partial Reconfiguration is a means by which a subset of the FPGA matrix can be reprogrammed while keeping other parts of the matrix operational. In an evaluation of the technique when used in a Software Defined Radio (SDR) environment [55] the advantages of Dynamic Partial Reconfiguration (DPR), when part of the device is kept operational during reconfiguration, are discussed using the example of differing convolutional encoders required for 3G and WiFi communication systems; in the experiment one encoder is loaded into the FPGA at a time. The convolutional encoder is a large building block and a major function of the SDR transceiver. DPR is experimentally shown to add flexibility to the hardware design for the cost of a relatively small additional overhead to the logic area and the requirement to floorplan the design to reserve a suitable size partition for the encoders to occupy. This addition is, however, easily offset by removing the necessity to implement all identified encoders in one design. An additional benefit also afforded by DPR is a reduction in static power consumption by reducing the area or logic block count of the implemented design .

The study highlights the growth of SDR and the development of new standards, leading to the likelihood of current hardware designs becoming obsolete in a short period. Similarities can be seen between SDR in this case and the current developments in optical networks where both are undergoing a rapid evolution of standards.

In an earlier paper from 2009 [56], J. Huang et al investigate Discrete Cosine Transformation (DCT) computation for image processing applications using a scalable FPGA architecture. This paper focusses on reasonably small blocks implementing fundamental, low level mathematical functions of the DCT using memory based computations and the possibility of reusing these areas for other mathematical functions at the expense of reduced DCT performance. In this example an empty architecture is also defined to allow for a region to be unconfigured, resulting in a reduction in power. Although this paper is older than other literature reviewed in this work, and deals with relatively small and less complex reconfigurable blocks, it is included here as it reports the interesting case where partial reconfiguration may be applied during a calculation without interruption.

3.2 Multichannel Transceivers – The role of FPGAs in their Operation

FPGA technology can prove to be applicable in the operation of arrayed component transceivers. One of the core components identified as part of such a transceiver is the 'Traffic Path ASIC'. Functions of this ASIC could include traffic path related features such as a dynamic modulation format encoder/decoder, FEC or dispersion compensation on the transmission line facing interface, or protocol definition (OTN, Ethernet etc.) on the client facing interface, and device control processes such as control loops for optical power (per channel or balancing across all channels), optical frequency, modulation depth and temperature.

Given the point-to-point nature of a Superchannel it may be desirable for an operator to group the wavelengths available on the transceiver as one high capacity Superchannel utilising all wavelengths for maximum spectral efficiency or as two or more lower capacity Superchannels using fewer wavelengths for maximum flexibility. The advantage of FPGAs in this area is that they can be reprogrammed and repurposed as requirements and standards evolve, for instance the modulation scheme can be flexible, allowing the support of different networks (LH, Metro, etc.) without needing to build all options in or even fix the transmission spectrum at day one. As previously discussed in Section 2.2, the concept of an agile and software configurable transceiver, able to adapt as required by adjustments to modulation scheme, FEC, number of subcarriers etc. is seen as being a key element to future networks, and FPGAs, particularly using partial reconfiguration, fit well into this picture. FPGAs are also, as previously stated, intrinsically parallel in their nature, allowing many tasks to be performed simultaneously.

The concept of a reconfigurable Superchannel Transponder has appeared in a series of top scored papers presented to ECOC [57] and the Journal of Lightwave Technology [58]. These papers examine the development of a Bandwidth Variable Transponder by considering an Ethernet only based system. The papers recognise the need to aggregate variable client signals from across different domains. The bandwidth variability comes from a software defined reconfigurability of the line side modulation format and symbol rate with options of 40 GBaud PM-QPSK/PM-16QAM or 10 GBaud PM-QPSK depending on transmission link requirements. Whilst highlighting the advantage of being able to reconfigure the line side characteristics for capacity and transmission distance, these papers do not discuss the methods used to reconfigure the FPGA in the transponder and limit themselves to Ethernet clients and protocols avoiding the ITU OTN standards on one interface due to concerns over fixed aggregation ratios, modulation format and number of used carriers in the Superchannel on the line side. The aggregation ratio concerns are currently being addressed by the standards bodies [59]. A further evaluation would be whether there is a power advantage to reducing capacity away from peak usage times by not having unused functions present and whether this could this be practically achieved by dynamic reconfiguration.

The Bandwidth Variable Transponder concept evolved as part of the FP7 funded IDEALIST project [36] which was a collaboration between industrial and academic partners including some Tier 1 telecommunications equipment suppliers. This project expands on the concepts of the previous papers by adding a slicing function; whereas previously a transponder would aggregate onto a single optical path, in this paper slicing is defined as the ability to allow Superchannels to be generated across multiple optical paths for routing towards one or more destinations. This is considered by IDEALIST as being an important feature for expanding metro networks which may already have a partially allocated spectrum. The paper uses the example of splitting a high bitrate signal into multiple lower rate channels in the scenario of there not being sufficient contiguous space on the optical spectrum due to other, pre-configured, carriers. It is also possible to imagine this feature being of value in the case of a network operator wishing to divide the total bandwidth of the transceiver, in this case 1.2Tb/s, between multiple clients, although this is not discussed specifically. This theme of flexibility is continued in more recent papers [19] and is still seen as an essential feature of modern optical networking.

The IDEALIST paper reviews a possible architecture of such a sliceable transceiver (Figure 3-3). However, while examining the Optical Transport Layer (OTN) functionality and the 'Multiflow' optical modules, client mapping and the OTN fabric are not considered and so no conclusions are drawn on the reconfigurability of the client ports or how this might be managed in a Software Defined Network.

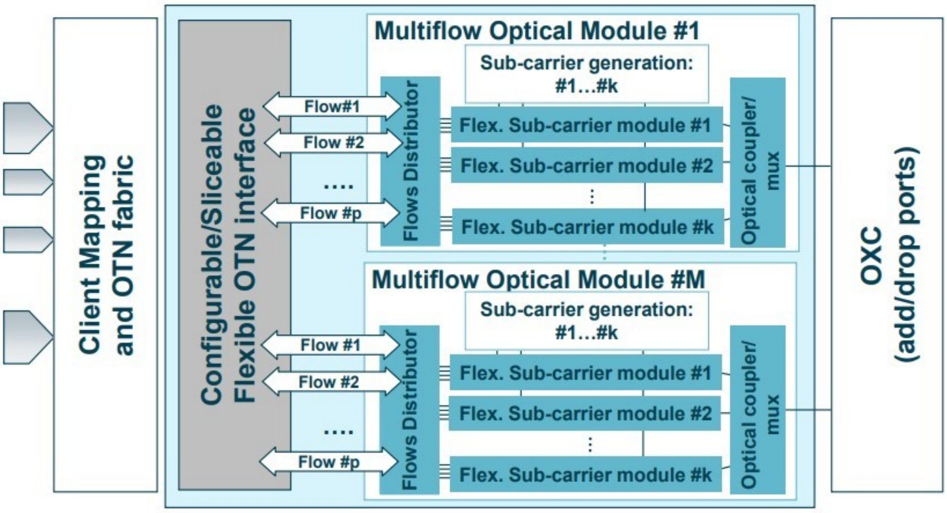

Client Mapping and OTN fabric
Configurable/Sliceable Flexible OTN interface
Multiflow Optical Module #1
Sub-carrier generation: #1…#k
Flow#1
Flow #2
....
Flow #p
Flows Distributor
Flex. Sub-carrier module #1
Flex. Sub-carrier module #2
Flex. Sub-carrier module #k
Optical coupler/ mux
Multiflow Optical Module #M
Sub-carrier generation: #1…#k
Flow #1
Flow #2
....
Flow #p
Flows Distributor
Flex. Sub-carrier module #1
Flex. Sub-carrier module #2
Flex. Sub-carrier module #k
Optical coupler/ mux
OXC (add/drop ports)

Figure 3-3 Architecture of a Sliceable Transceiver - Sliceable Variable Bandwidth Transponder: The IDEALIST Vision [36].

The proposed approach to the ‘Pay as You Grow’ question involves the installation of subsequent client and Multiflow optical modules to provide extra subcarriers. While this approach certainly addresses one aspect of ‘Pay as You Grow’ – i.e. the high initial cost of having a 1.2Tb/s capable system before the network can generate revenue from the full capacity- it does require extra operational activities, e.g. site visits by a commissioning engineer. The proposed architecture does not address the requirements of a dynamic network, where the total bandwidth required can change frequently, requiring the full transceiver bandwidth to be available at short notice and for short periods, nor does it leverage the strides forward being made in the Long Haul market where 6 or 12 optical subcarriers in a single optical module (Figure 3-4) are being manufactured by some vendors in order to achieve 1.2Tb/s [60].

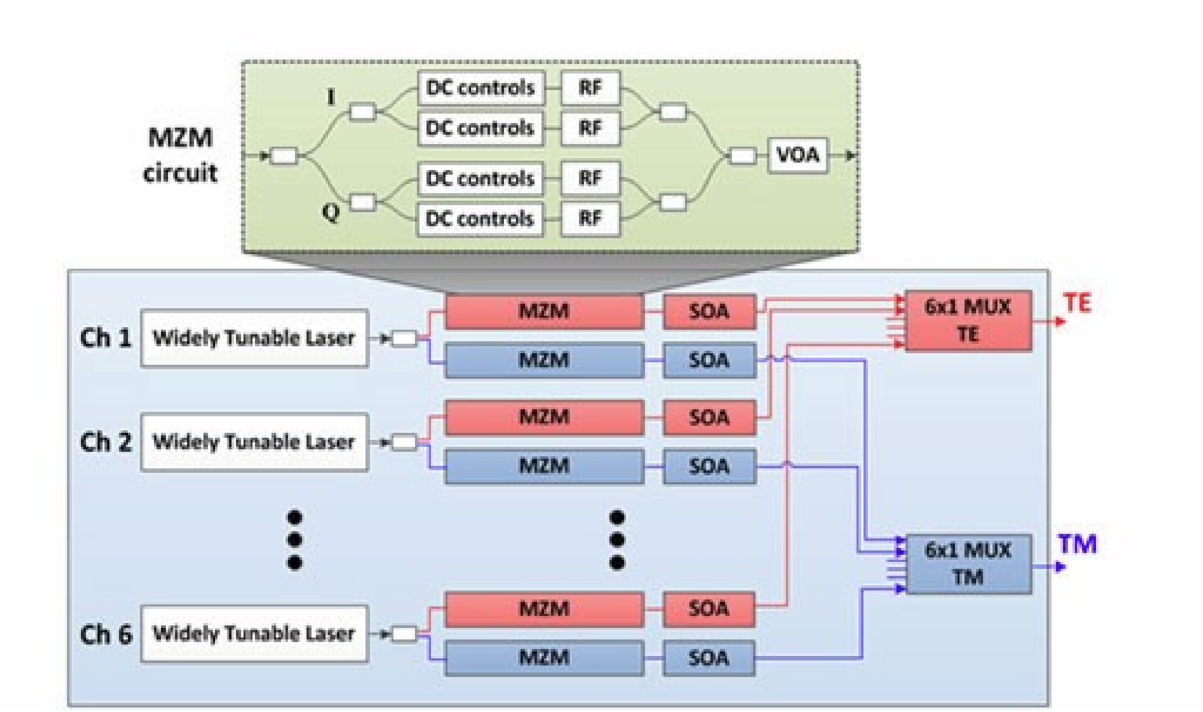


Figure 3-4 1.2Tb/s PIC architecture [60].

As previously stated, the long haul market is still seen as the driver for Superchannel transceivers. This presents an opportunity to examine if there is a suitable, repurposable and cost effective architecture which could be adopted and whether the FPGA technology now exists to support this.

3.3 Superchannel Transceivers - Manufacturability

By their very nature, Superchannel transceivers with multiple optical carriers can be expected to require more time, and hence cost, to manufacture and test, if the current working model is applied. Where a transceiver has an increased number of optical assemblies, then a logical assumption would be that setup and test would take longer than for a single assembly, so factory time can present a significant overhead in the device cost. By careful analysis, it should be possible to identify areas where time savings could be made, and where bottlenecks in the manufacturing process exist. However, a search of the literature for previous published works based on this area has provided few results. In the industrial experience of the author such analysis is ongoing with the component manufacturers; one, highly likely, reason for the lack of publications is the commercially sensitive nature of the manufacturing process and the device yield data involved in examining this question. Looking a bit wider into the area of virtual instrumentation, a block based, open source approach to FPGA based instruments has been discussed [61] as a means of generating custom instrumentation to support experimental physics. This paper acknowledges the issue of design time and specialist skills needed to initiate the design, but once this effort has been made far less effort is required to build new systems based on this platform. This leveraging of block based design and reuse methodology is a good approach for constructing a scalable solution and supports the use of partial reconfiguration.

By use of reprogrammable technology, together with embedded software, it is possible to save time and cost by performing operations on all channels in parallel through the FPGA. Furthermore, by allowing the transceivers to be 'smarter' and perform self-test and calibration, it will be possible to reduce infrastructure required in the factory (e.g. PCs as test controllers and data processors) and hence floor space required. Integrated Analogue to Digital convertors and, in some devices, Digital to Analogue convertors further enhance the opportunities to create an embedded test and calibration System on Chip architecture.

For automated test and setup activities, a balance between hardware FPGA features and software can enable complex routines to be run autonomously 'on module', with the additional advantage that the FPGA/processor combination can be repurposed via reconfiguration later for normal day to day operation. The results of these test routines could then either be passed off module for further processing and storage, or kept in non-volatile storage on the module.

A typical laser tuning algorithm will consist of a series of nested sweeps across the control currents of the device under test be it a laser or modulator to form a grid of output power vs applied control currents and is illustrated in Figure 3-5.

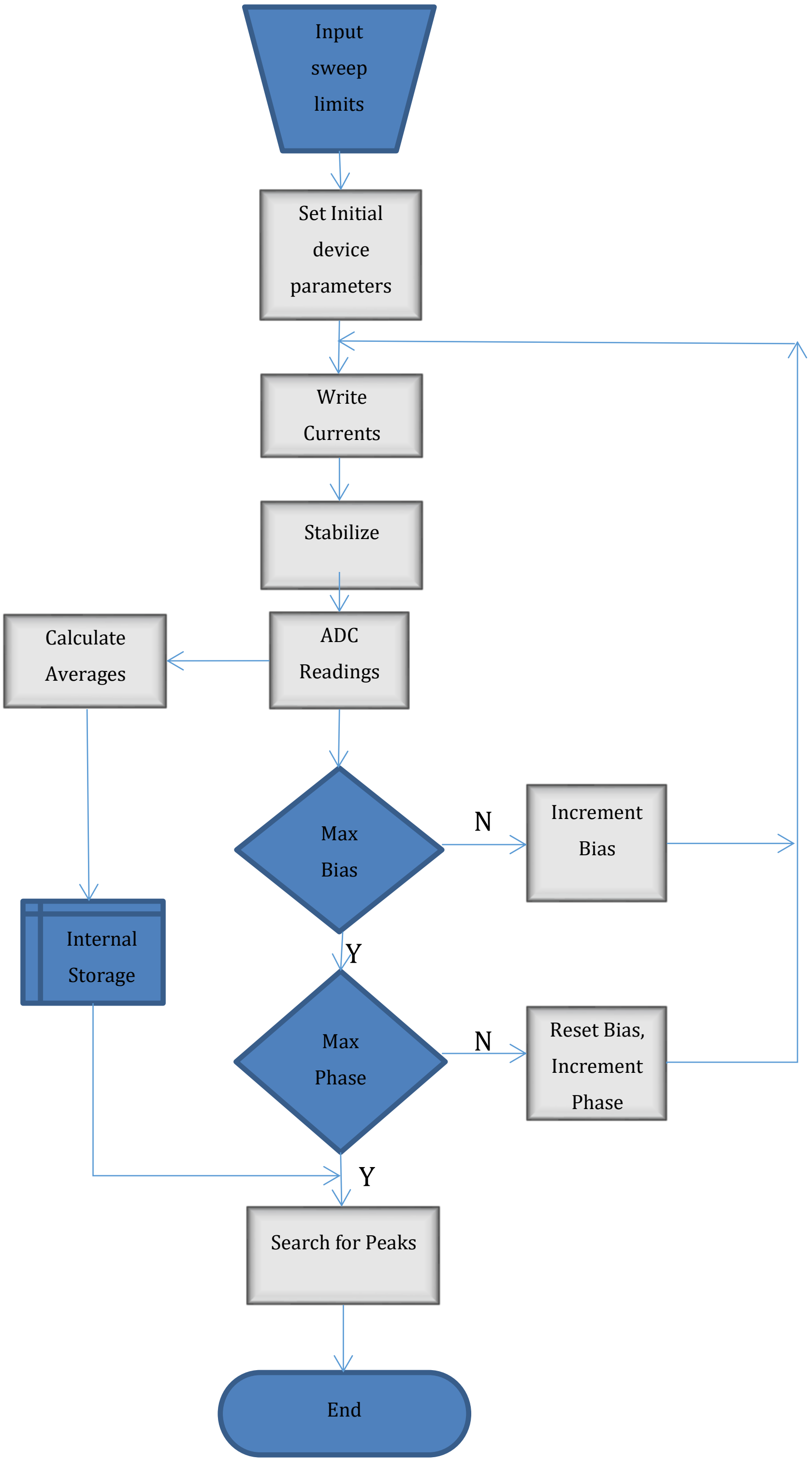


Figure 3-5 Flowchart of a simplified tunable laser calibration sweep.

The set of currents swept may be a subset of the full ranges possible. At each current step a settling time is required before sampling starts. At each stage the output power is monitored either using an external power monitor or ideally using the hardware which will be used to control the device in the field, and the result is averaged across multiple samples to find the mean and standard deviation. This data is logged along with the unit temperature. Mode hops can be seen as discontinuities in the power readings whereas desirable operating points can be seen as peaks/troughs in the power reading (zero slope or null points) or points of maximum slope in the power readings (quadrature points in the transfer function). In a traditional manufacturing flow, control of the currents and processing of the resulting data would normally be carried out using a PC at a test station. If this can be performed using an embedded processor coupled with repurposed logic capability from the in-service control circuits, then many units could be calibrated in a smaller bench area and without the additional overhead of a PC. Some post work with a computer and test equipment may be required to identify frequencies vs peak power (although a US Patent [62] proposes a calibration method of using a comb filter to generate digital pulses which could be counted by the FPGA) but this will require less time as the points of interest are already known.

In Figure 3-6 the axes represent the control value being applied to the on-board DAC rather than a direct measurement of the applied current. The DAC control words are controlled by the FPGA implemented control loops in normal operation. Variations in the external circuit are taken out. Variations in the output power monitor current are visualised as changes in colour from dark blue (low) to red (higher).

For a single optical device, the total time taken may seem short, but as arrayed devices become more widely used to implement Superchannel transceivers, the time taken to qualify a single device will increase linearly with the complexity. If a possible result of this process includes the rejection of the device due to calibration failures, then the less time and cost spent identifying this the better.

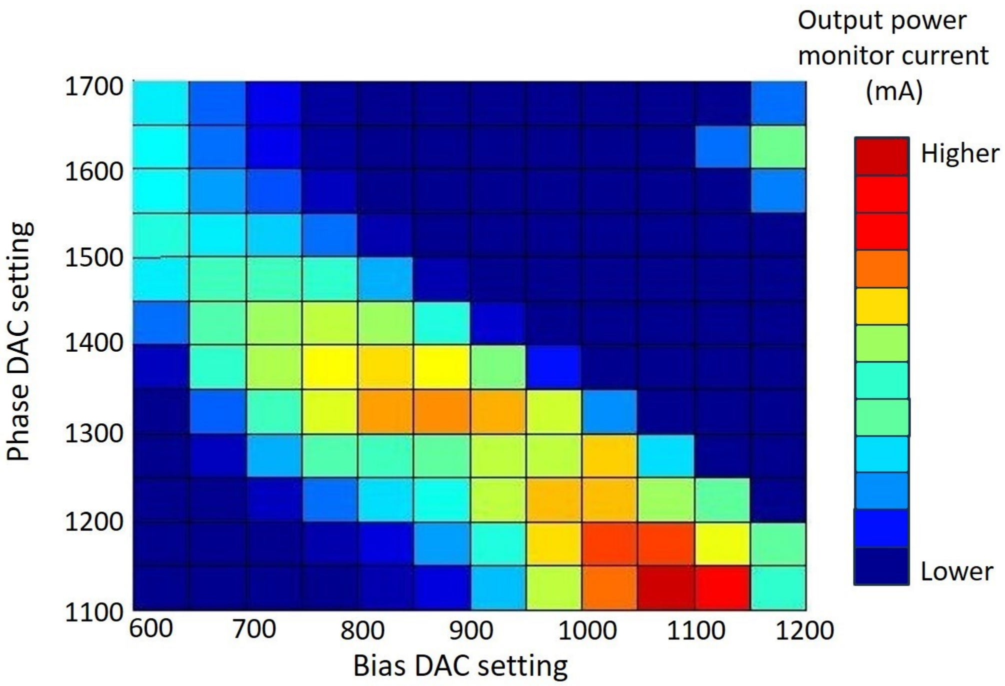


Figure 3-6 Example result of a laser tuning sweep showing the variation of output power monitor current with Phase and Bias DAC setting.
(Original image shared in confidence)

3.4 Summary

The FPGA is a user programmable hardware device which can be programmed and reprogrammed during the lifetime of the unit it is built into. As a means of providing a background knowledge of the capabilities of these chips, this chapter has offered an introduction to the technology of FPGAs, looking at the architecture and the building blocks or functions which make up these devices, from the early FPGAs, which were able to perform simple logic tasks, to the latest devices supporting a wide breadth of applications. The development of FPGAs has, over time, come to include high performance, hardened blocks with dedicated functionality as a means of increasing performance and facilitating expansion into new emerging markets, challenging the position of ASICs.

The advantages of FPGAs over ASICs lie in both their lower development costs and their reconfigurability. The advantages over microcontrollers are the inherent parallel processing capabilities available to designers.

The role FPGAs can play in the operation of multichannel transceivers has been illustrated through the example of the IDEALIST transceiver and the possibility of offering a pay as you grow, flexible transponder.

Also examined more closely are some of the problems arising during the manufacturing process once arrays of components are introduced, and where FPGAs could play a role in mitigating these problems. This could be done by creating reusable blocks for use during the manufacturing process, and potentially after deployment, capable of performing operations in parallel and autonomously. These blocks could then subsequently be reprogrammed and the logic area, plus the supporting external components, repurposed for operational functions.

In summary, the FPGA is a versatile tool capable of addressing the issues presented relating to the bandwidth growth and flexibility requirements and the problems which arise during the manufacturing of the solutions to these issues.

4 MACH ZEHNDER MODULATORS

4.1 Introduction

Analysing the previously identified constituent parts of an arrayed transceiver product, the Mach Zehnder modulator (MZM) makes an excellent example to study. MZMs find applications in many areas of photonics such as optical signal modulation, optical beam steering [63] and quantum computing [64]. The MZMs behaviour is well defined by an arithmetic transfer function but it is also prone to drifting away from the set operating point under external influences such as temperature, ageing, charge accumulation [65] or a variation in wavelength, hence changing the relationship between applied bias voltage and phase shift which leads to degraded performance of the modulator.

The ability to automatically determine the bias voltages corresponding to the key operating points discussed in Section 4.2 below, using a function contained within the optical module, would be advantageous for the reasons set out in Section 3.3 earlier.

4.2 Theory of operation

The MZM provides a means of controlling the amplitude of an optical wave. A simplified MZM structure is shown in Figure 4-1. In the device an input waveguide is split into two equal arms via a Y junction. If a voltage is applied across one of the arms an electric field is generated across that arm which varies the refractive index of the substrate material and thereby introduces a phase shift in the optical wave passing through it. When the two arms are then recombined, the phase shift between them is converted into an amplitude difference. If there is no phase difference then the optical waves in the two arms combine constructively and maximum output is seen. If the two arms are in complete antiphase then the waves combine destructively and a minimum output is seen. The difference in the voltage applied for no phase shift and the voltage required for a phase shift of π is referred to as the ½ - wave voltage or Vπ.

The MZM has 2 voltage electrodes:

- $V_{DC,}$ also sometimes referred to as V_{BIAS} is used to set the position on the transfer curve.
- $V_{AC,}$ also sometimes referred to as V_{RF} is used to apply the electrical signal which is to be used to modulate the output.

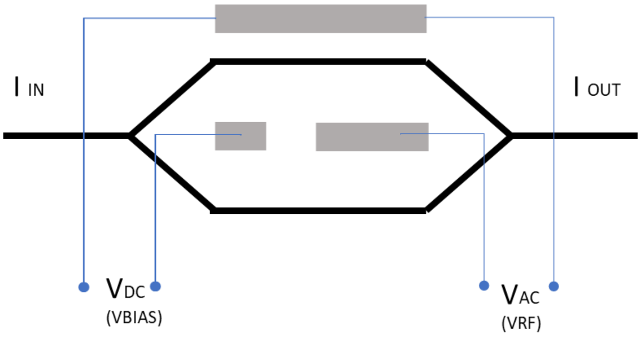


Figure 4-1 Simplified MZM structure.

The basic transfer function of the modulator, when driven by a single time dependent voltage *v(t)*, is

$$I_{OUT}\,(t)\;\propto \frac{I_{In}}{2}\left[1+\cos\left(\frac{\pi v(t)}{V_{\pi}}\right)\right] \tag{4.1}$$

where
$I_{OUT}\,(t)$ is the output Optical Intensity,
I_{In} is the input Optical Intensity

and is plotted in Figure 4-2. The output power has been normalised to the input power assuming an ideal modulator with no insertion loss.

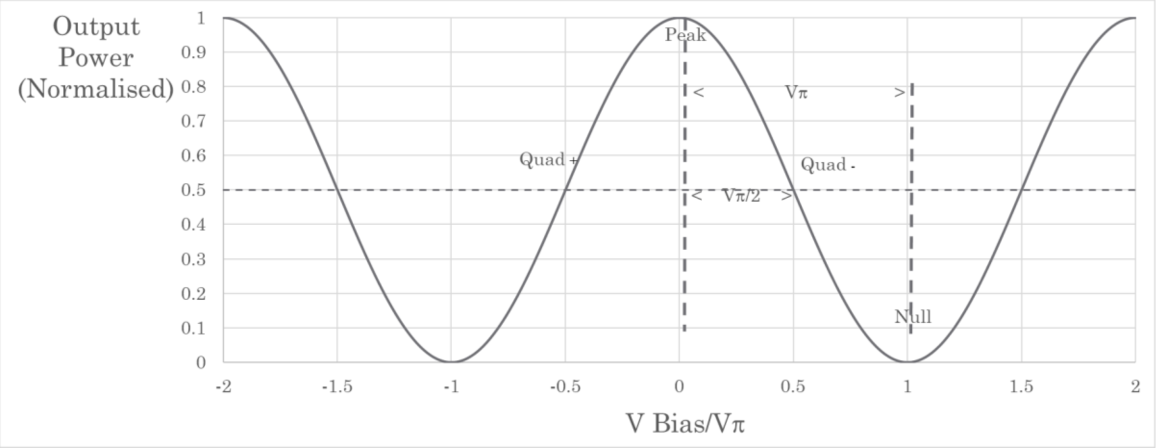


Figure 4-2 Transfer function of a Mach Zehnder Modulator showing the key operating points.

Analysis of the transfer function using Bessel functions has been discussed in the literature [66] [67] and the key equations are highlighted here for clarity. It has been shown that operating at odd integer multiples of $V\pi/2$, i.e. quadrature points as shown in Figure 4-2, forces even order harmonic distortion terms to be zero; in contrast, operating at integer multiples of Vπ (peak/null bias points) forces the odd order harmonic terms including the fundamental to be zero. Furthermore, it has been calculated ([68] Section IV) that the amplitude of the 2nd harmonic of a pilot tone applied to the bias input electrode of a modulator is directly proportional to the 2nd derivative of the optical output power.

Examine the effect on a simple sinusoid superimposed onto a DC bias voltage

$$v(t) = v_{DC} + v_{AC}\, sin(\omega_{AC}\, t)$$

Substituting this into the transfer function equation (4.1) gives

$$I_{OUT}\,(t) \propto \frac{I_{In}}{2}\left[1 + \cos\left(\frac{\pi v_{DC} + \pi v_{AC}\ \sin(\omega_{AC}\, t)}{V_\pi}\right)\right] \tag{4.2}$$

and expanding equation (4.2) gives

$$I_{OUT}\,(t) \;\propto \frac{I_{In}}{2}+\frac{I_{In}}{2}\cos\left(\frac{\pi v_{DC}}{V_\pi}\right)\cos\left(\frac{\pi v_{AC}\ \sin(\omega_{AC}\,t)}{V_\pi}\right)-\frac{I_{In}}{2}\sin\left(\frac{\pi v_{DC}}{V_\pi}\right)\sin\left(\frac{\pi v_{AC}\ \sin(\omega_{AC}\,t)}{V_\pi}\right)\ (4.3)$$

By using Bessel function identities (J_n) to express the following

$$\cos(xsiny) = J_0(x) + 2\sum_{k=1}^{\infty} J_{2k}\,(x)\cos(2ky)$$

$$\sin(xsiny) = 2\sum_{k=1}^{\infty} J_{2k-1}\,(x)\sin\,[(2k-1)y]$$

it is possible to expose equations for specific harmonics from (4.3)

$$I_{OUT\ odd} \propto= \left|I_{in}sin\left(\frac{V_{DC}}{V_\pi}\pi\right)J_n\left(\frac{V_{AC}}{V_\pi}\right)\pi\right| \qquad (4.4)$$

$$n \in \langle 1,3,5\rangle$$

$$I_{OUT\ even} \propto= \left|I_{in}cos\left(\frac{V_{DC}}{V_\pi}\pi\right)J_n\left(\frac{V_{AC}}{V_\pi}\right)\pi\right| \qquad (4.5)$$

$$n \in \langle 2,4,6\rangle$$

By implementing equations (4.4) and (4.5) in Python using the SciPy Jv Bessel function [69], the variation of the power of the fundamental and 2nd harmonic frequencies with V_{DC} (VBias) can be plotted as in Figure 4-3.

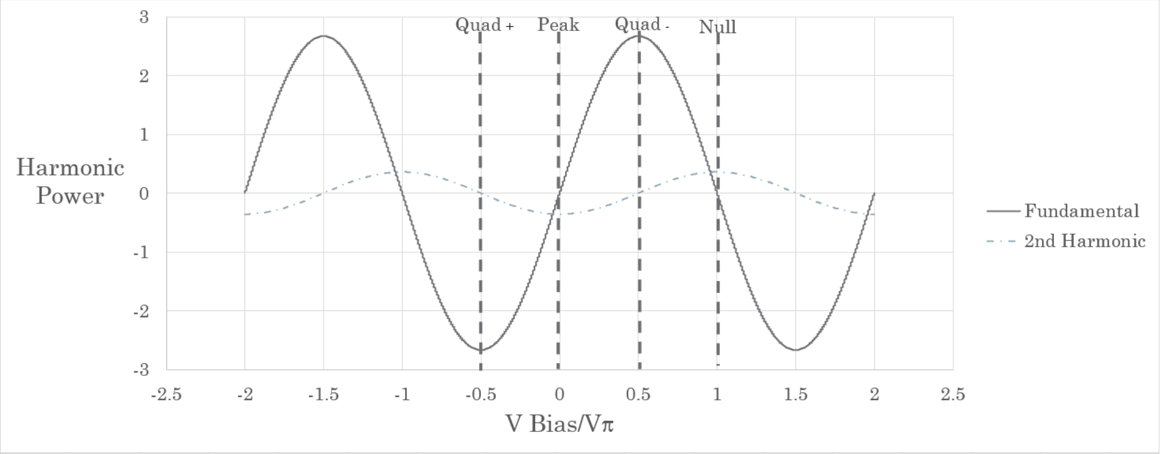


Figure 4-3 Variation of harmonic power with V_{Bias}.

It is this harmonic variation with V_{BIAS} which leads to a degradation in performance of the optical link, but it can also be used as a means of determining and controlling the operating position on the transfer curve.

Bias control algorithms tend to fall into two categories – dither (applying a low frequency pilot tone whose modulation depth is expressed as a % of Vπ) which follows the fundamental and second harmonic behaviour shown in Figure 4-3, and ratiometric (based on a characterised input to output power ratio) which follows the transfer function shown in Figure 4-2. The pros and cons of each method are widely discussed in the literature [70], [71], [72].

A dither-based approach to maintain a bias point, where either primary or 2nd order distortion is zero, is to add a low frequency sinusoid to the bias electrode. This moves the bias point causing harmonics of the dither frequency to appear on the modulated optical signal, so if a bias point where the 2nd harmonic distortion is zero (quadrature point) is required, the dither frequency 2nd harmonic should be detected and using a feedback loop the bias should be adjusted until the 2nd harmonic is at a minimum. Conversely, to operate at a point where the primary is zero (peak or null) the dither frequency should be detected and the bias voltage should be adjusted to minimise the signal power at the dither frequency [66]. When using this approach, it is necessary to have correctly characterised the modulator first in order to determine a suitable starting value for the control loop.

It should be noted that the use of a dither frequency can enable a high degree of precision in the bias point control but the presence of a dither signal may not be tolerable in the end application (analogue, RF over Fibre, orthogonal frequency division multiplexing) due to the presence of intermodulation products [70].

The dither-based approach consists of a local oscillator or other source, for example an FPGA or microcontroller for digital frequency synthesis, which generates a low-frequency pilot tone, normally in the kHz range, and a single fiber-optic coupler that taps a small percentage of the MZM's optical output power and feeds it to a single photodetector. From a manufacturing perspective it can be seen that this pilot tone-type bias controller will cost less to assemble than a ratiometric-type controller, because it requires only one tap coupler, which does not operate in polarization-maintaining fibre; it requires only one photodetector in the control circuit; and the proper feedback circuit settings do not depend on the specific performance parameters of the modulator, tap coupler, or photodetector.

A dither-based circuit requires the implementation of digital filters to monitor the power of the fundamental and/or 2nd harmonic of the pilot tone frequency. In keeping with the low cost, low power ethos followed in this study, this circuit needs to be as efficient as possible in terms of FPGA internal resource usage.

4.3 Filter Implementation

Analysing the possible implementations of the digital filters, typical architectures for Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filters are shown in Figure 4-4 and Figure 4-5.

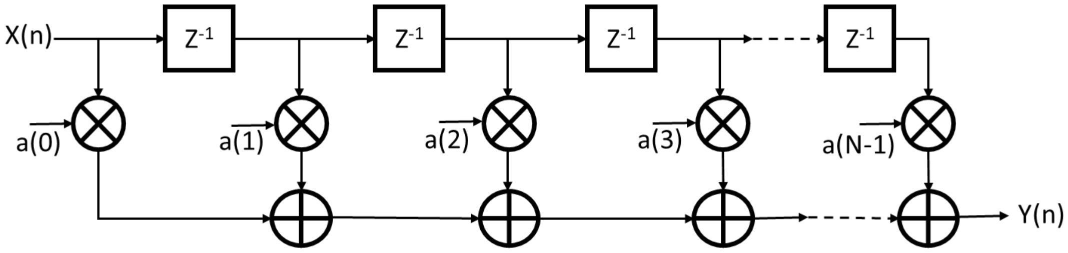


Figure 4-4 FIR filter architecture with N taps.

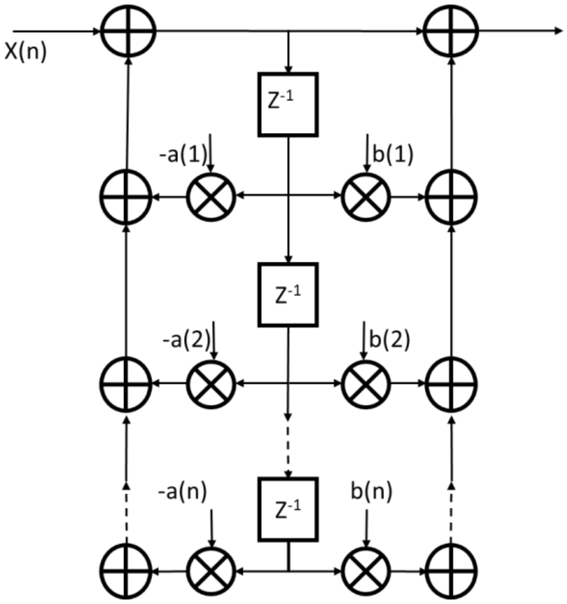


Figure 4-5 IIR Filter architecture with N taps.

Each filter has its advantages and disadvantages. The FIR filter is inherently stable and can be designed to have a linear phase response. There is also no feedback path so any rounding errors introduced in the arithmetic are not summed over multiple iterations. The FIR filter tends to require more taps and hence more coefficients to achieve a sharp cut off and a narrow passband whereas the IIR filter tends to require substantially fewer taps than an FIR filter of equivalent performance but sometimes can be unstable as the impulse response doesn't become zero but continues indefinitely, and the phase/delay response is not constant across all frequencies,

Both have many coefficients which need calculating and storing. The FIR implementation in particular requires a very large number of coefficients. Larger filters will require more resources from the FPGA and more clock cycles to execute, additionally long chains of multiply and accumulate functions are required which leads to increased intermediate product storage in the FPGA. During execution the intermediate values calculated will exhibit bit expansion as they accumulate and may require truncation to manage this, leading to cumulative rounding errors.

An alternative is to evaluate a mathematical sequence proposed by Gerald Goertzel in 1958 [73]. This effectively behaves as a single bin Fast Fourier Transform (FFT).

Goertzel's algorithm has historically found uses in applications which require the presence of a tone to be determined such as dual tone dialling in telephony where pairs of discrete, orthogonal sinusoid frequencies are used to encode button presses. The Goertzel algorithm was well suited to this application compared to a FFT when constrained in an 8 bit microcontroller as it required less memory and could be targeted at just the eight dual tone frequencies with a narrow bin width. Its main advantage is that it uses very few coefficients which need calculating up front and storing. The performance of the filter when used as part of an MZM monitoring function, where detecting tone absence is also an important factor, needed to be verified experimentally.

When implementing the Goertzel algorithm the processing is performed on blocks of data; however, as the numerical calculations are very simple, and as FPGAs are fast and inherently parallel by design, it is not necessary to store all the samples in a block before computing the result, although the final result will not be available until after the last sample in the block is received. Four parameters are required to be defined, the sample rate, the block size and a pair of precomputed coefficients (one sine the other cosine). In this experiment, this is reduced further as only the cosine term was used.

When determining the sampling rate ($f_{s)}$, ideally the frequencies of interest need to be integer factors of sample rate/ block size. In this case the sample rate was also determined by the available sample rates which could be configured for the on-chip ADC function in the Max10 FPGA. Finally, Nyquist rules must also be obeyed when setting the sampling rate.

The block size (N) controls the bin width or frequency resolution and is analogous to the number of points in an equivalent FFT. The bin width is given by sample rate/block size

There is a trade-off to be judged here. A large block size will give a narrow bin width but will take longer to calculate the filter response. For the frequencies of interest to be centred in their respective filters they need to be integer multiples of the bin width.

To define and calculate the two fixed coefficients (sine and cosine), the starting point is the basic Discrete Fourier Transform (DFT) equation shown in equation (4.6). The target frequency is in bin k out of a total of N bins, W^k is the weighting factor which is the complex term $e^{-i2\pi\frac{kn}{N}}$.

$$X_k = \sum_{n=0}^{N-1} \left(W^k\right)^n . x_n$$
$$k = 0,1,2 \dots, N-1 \tag{4.6}$$

Finally, as discussed in [74], the equivalent bin number (k) can be calculated:

$$\mathrm{k} = (\mathrm{int})\left(0.5 + \frac{\text{Block Size(N)} * \text{Target Frequency}}{\text{sample rate}}\right) \tag{4.7}$$

Then the equivalent weighting factor (W) can be determined. For applications such as this where the input samples are real-values, the complex-valued calculations are reduced to real-valued calculations, leading to the sine and cosine terms shown below.

W = (2*π/N)*k
cosine = cos W
sine = sin W
coeff = 2 * cosine.

In this experiment a 4 kHz sine wave was used as the fundamental pilot tone giving the following parameters:

Block Size: N = 1500 samples
Sample Rate: f_s = 20 ksps
Bin width: f_s/N = 13.333 Hz

Finally, the bin width was used to verify the target frequencies were the centre frequencies of the filter by checking the value of target frequency/bin width was an integer:

$4\text{kHz}*N/f_s = 300$

$8\text{kHz}*N/f_s = 600$

These parameters produced calculated cosine terms of 0.309021 for the fundamental tone filter and -0.809021 for the second harmonic filter.

When implementing the filter a three stage calculation with two pipeline stages is defined: Q0, Q1, and Q2 (see Figure 4-6). For every input sample n, the following three equations are calculated:

$$Q0(n) = coeff * Q1(n-1) - Q2(n-1) + n$$

$$Q1(n) = Q0(n-1)$$

$$Q2(n) = Q1(n-1).$$

After running the per-sample equations for the whole block of n = N samples, the complex output X_k can be calculated as follows:

$$\text{Output } (X_k) = (Q0(N) - Q1(N)* (\text{cosine}+i*\text{Sine}))$$

Also:

$$\text{Realterm} = (Q0(N) - Q1(N)* \text{cosine})$$

$$\text{Imgterm} = (Q1(N) * \text{sine})$$

$$\text{magnitude}^2 = \text{Realterm}^2 + \text{Imgterm}^2$$

The relationship between these variables is shown in Figure 4-6

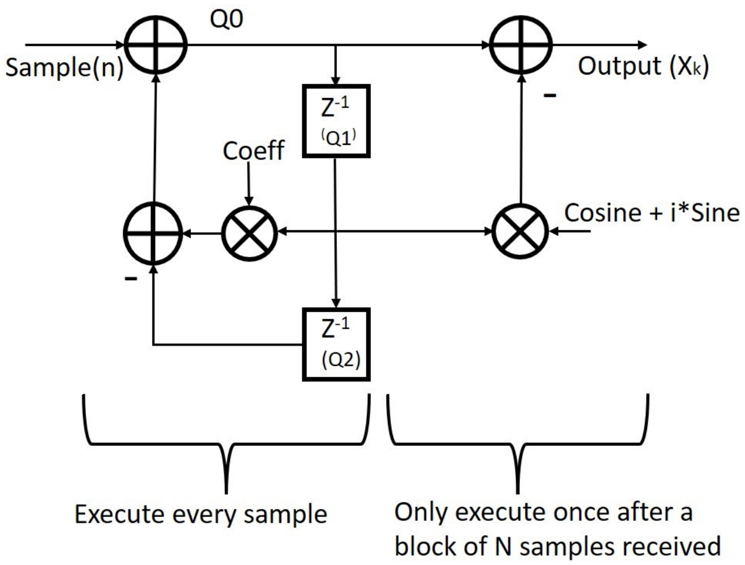


Figure 4-6 Standard form Goertzel filter structure showing relationship of Q0, 1, 2, Coeff, Sine and Cosine.

The simulated response of a Goertzel filter centred on 4kHz, when presented with a sweep of sinusoids at frequencies between 3 kHz and 5 kHz is plotted in Figure 4-7

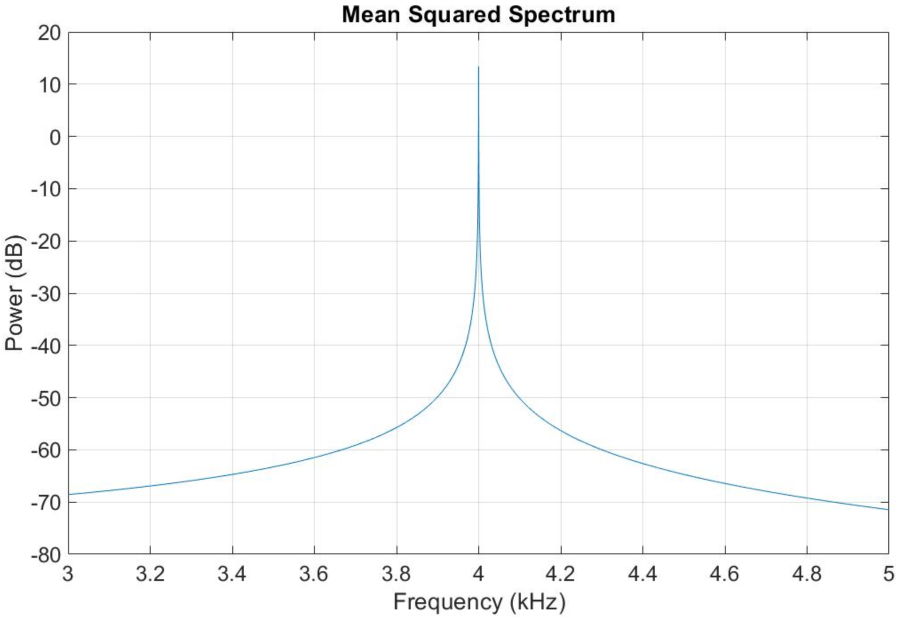


Figure 4-7 Simulated power response of the Goertzel algorithm based 4 kHz frequency filter to a range of sinusoids between 3 kHz and 5 kHz.

For the implementation of the fundamental and second harmonic filters a modified filter structure was adopted as shown in Figure 4-8.

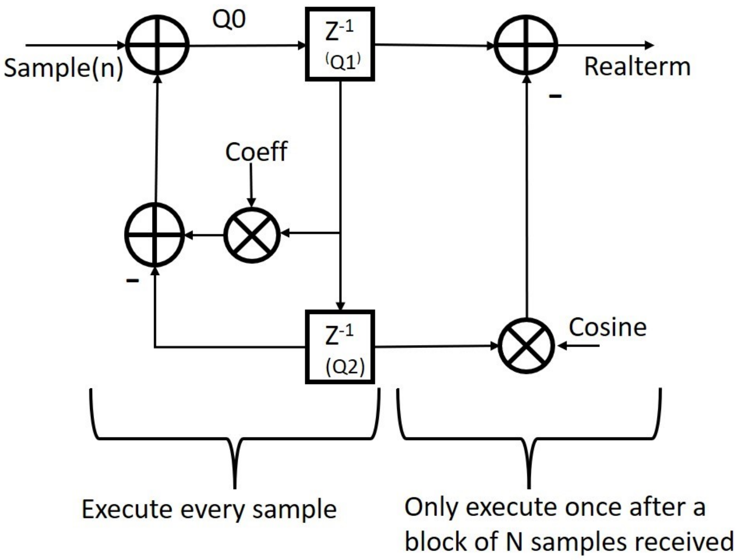
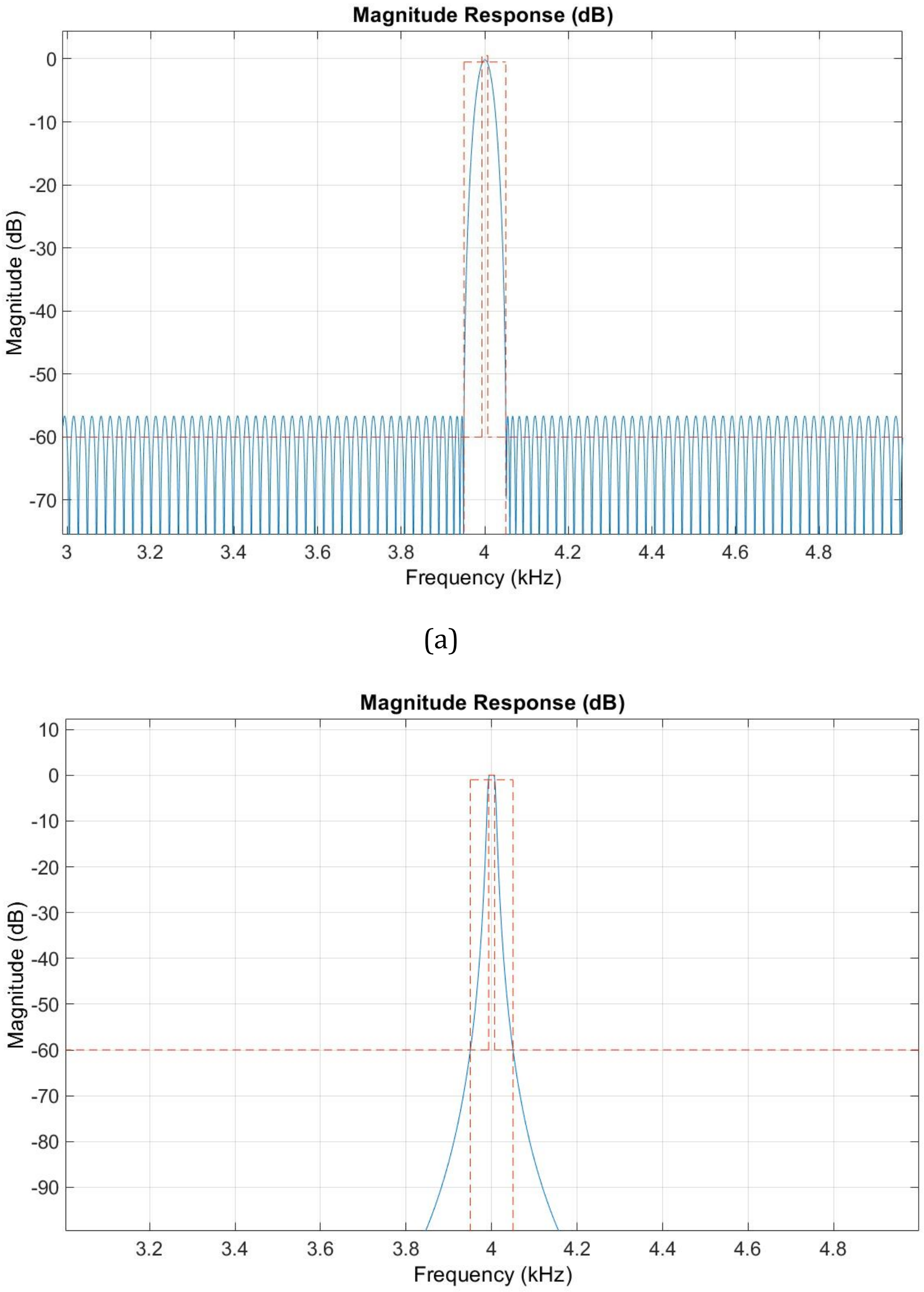


Figure 4-8 Simplified form Goertzel filter structure as implemented

The modified structure has two differences to the structure of the standard form shown in Figure 4-6. Firstly, as the input samples are real-valued only, then only the Realterm output based on the cosine weighting factor is evaluated. Secondly, the implemented filter has an additional delay before calculating the Realterm output but numerically behaves identically to the standard form, with the 'per sample' calculation being identical and the 'per block' calculation having the additional delay which was accounted for in the generated design. This was done to minimize the length of the Q0 to Realterm combinatorial logic paths in the implemented design in order to avoid any potential timing problems. The reuse of the Q1 and Q2 registers to add a clock stage to the output calculation avoided adding any additional registers to the design.

Implementing a filter using this method, with a 13 Hz bandwidth centred on the target frequency, requires very few resources when compared with traditional IIR and FIR designs. Using MatLab to generate comparable functions, centered on 4kHz with 20 kHz sampling rate, a passband from 3.993 kHz to 4.007 kHz and a stopband attenuation of 60 dB shows the complexity of an equivalent FIR implementation would be 918th order, whereas an IIR implementation would be 4 x 2nd order sections. The responses of these FIR and IIR filters are plotted in Figure 4-9.



(a)

(b)

Figure 4-9 Magnitude responses of equivalent FIR (a) and IIR (b) filters to the Goertzel based filter used in this experiment.

These would both involve the calculation and storage of many coefficients and the implementation of large logic trees in the FPGA in order to realise the filter, limiting internal FPGA operating speeds and potentially causing routing congestion. The implemented filter based on the Goertzel algorithm requires by comparison, 2 coefficients (one sine, one cosine implemented as 16 bit signed fixed point numbers). In this experiment only the real term was evaluated, and, as the value ‘coeff’ equals 2x cosine term, it was only necessary to hardcode a single coefficient, further reducing complexity and FPGA resource use. By using only the real term we see a positive and negative answer from the filter which could be used to indicate the direction in which a closed loop controller should move the bias voltage. In keeping with the target of demonstrating practical repurposable hardware, the FPGA used was a low cost, low power MAX10™ 10M08 device from Intel. This has a small number of DSP specific macrocells capable of performing multiplication and accumulation actions so the filter algorithm has been chosen for efficiency. The design itself was implemented using the hardware description language VHDL, using Intel’s Quartus Prime development tools. The implementation of the filter has been adapted to the proposed FPGA platform whilst aiming to achieve the best possible accuracy. To test the effectiveness of the filter implementation a VHDL simulation was performed and the result compared with that predicted by calculation.

The resource utilisation, post fitting stage, is given in Table 4-1

Figure 4-10 shows the filter following the technology fitting stage of the FPGA implementation. The wide pipeline stage for Q1 to Q2 (ringed in green) and a relatively long full adder chain (orange) can be seen, however this design is still capable of operating with a clock of 53 MHz. The 2 multipliers each constructed from 2 18x18 bit multiplier hard macrocells are ringed in red and the sample block counter in blue.

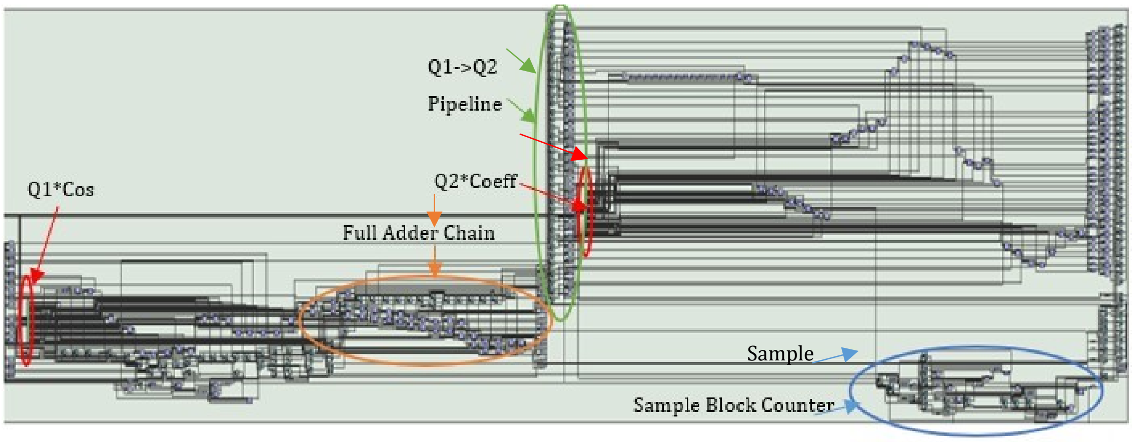


Figure 4-10 Filter implementation in Intel MAX10 primitive cells.

Goertzel filter implementation- MAX 10 resource usage (post fitter)	
Logic Cells	391: {242 Logic LUT only 14 Register only }
Single Bit logic registers	147
18x18 bit DSP multipliers	4
Memory Blocks	0

Table 4-1 Goertzel algorithm based filter - Post Technology Fitting resource utilisation.

Fixed point arithmetic was used throughout using a 16 bit representation of the fractional part of the numbers and a single multiplier stage (35x17 bit signed binary representation).

The representation of the Cosine term in this format gave an approximation which was correct to better than 5 decimal places.

4.4 Experimental Setup

In [66] bias control methods using harmonic analysis are examined using calibrated laboratory equipment. The question then is whether it is practical to adopt features of these methods as part of a characterisation procedure implemented with commercially available, low cost and low power components which could be integrated into an optical module.

The purpose of this experimental activity was therefore to validate the behaviour of such a control system when implemented using low cost components and computationally efficient algorithms. To this end it was proposed to digitally generate a 4 kHz pilot tone to be applied to the DC electrode of an MZM while applying digital pseudo random (PRBS) traffic at 10 Gbps to the RF electrode, and examine the performance of digital filters based on the algorithm described by Goertzel [73] when determining the presence and absence of the applied pilot tone and its 2nd harmonic. To carry out these investigations the experimental setup shown in Figure 4-11 was implemented.

The analogue electronics (filters, photo diode (PD)), digital components (FPGA, ADC, DAC) and user interface blocks (USB) were implemented on a custom designed circuit board by TerOpta Ltd. The optical components were connected together on the workbench. A $LiNbO_3$ MZM (JDSU OC192) was connected to a 10

Gb/s Anritsu data source generating a PRBS31 sequence, through a JDSU H301 modulator data driver. The laser source used was a C Band tuneable IDPhotonics laser set to 1530.0 nm and using a launch power of 15 dBm.

The amplitude of the pilot tone was set to 85 mV peak to peak which is approximately 2% of the bias electrode Vπ for this MZM. Increasing this value may be beneficial in the short term if the pilot tone harmonic method is only to be used during characterisation of the device; however, as stated previously, it may not be desirable if a pilot tone control method is to be employed [75]. As discussed in [71], the effect of the pilot tone is not only due to the amplitude of the applied electrical signal but also to the intensity of the optical signal at the receiver; this can be adjusted by changing the percentage of the optical signal routed to the controller through the tap coupler. In this experiment a 5% tap was used together with an attenuator to limit the optical power of the signal at the controller anti-aliasing filter to be within the operating range of the receiver circuit, in this instance -19 dBm. The application of this attenuation limits the electrical signal being monitored by the ADC within the FPGA. Initially no data was applied to the RF or data electrode as this may introduce further noise to the system.

The first step was to determine the value of Vπ for the modulator used in this test. This was achieved by manually altering the value of the DC bias voltage by setting the DAC control word through the user interface shown in Figure 4-11 while monitoring the output power of the MZM directly, after the 5% tap, using an optical power meter.

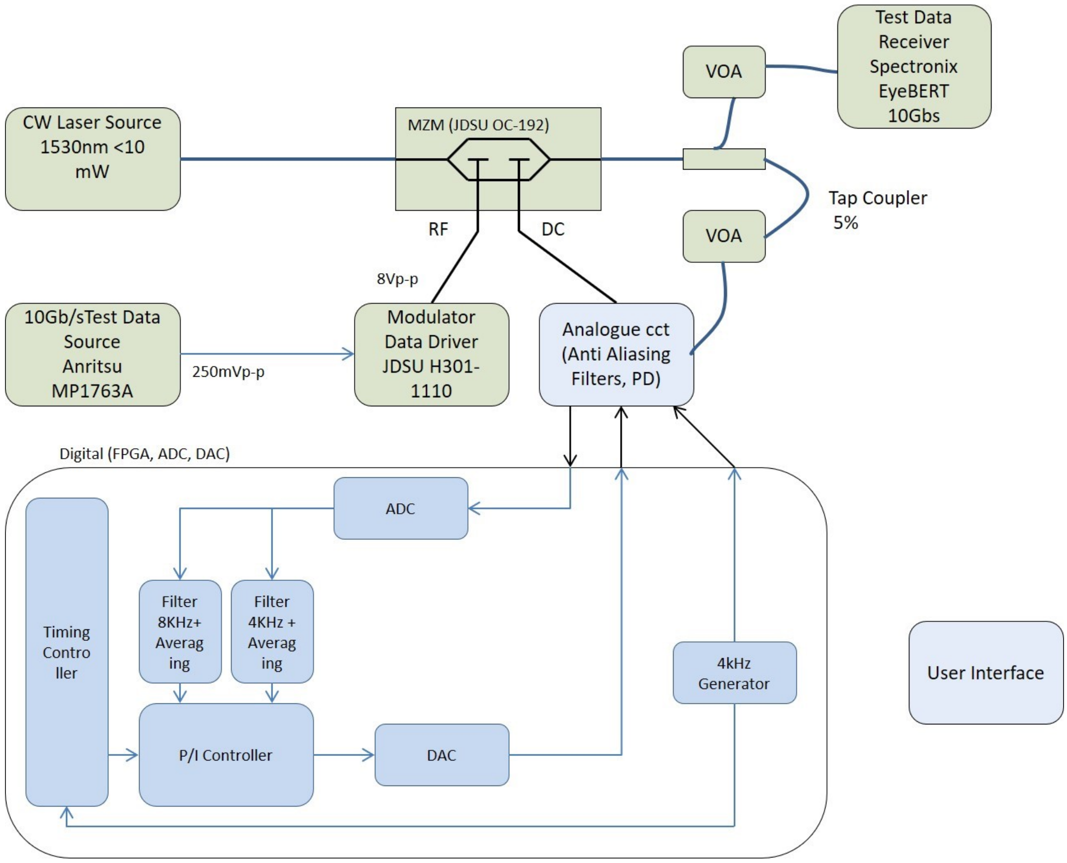


Figure 4-11 Experimental setup for filter response DC bias sweep.

The behaviour of the analogue circuitry was then tested by applying a pseudo random binary sequence (PRBS15) at 10 Gb/s and manually adjusting the DC bias voltage searching for Peak, Null and Quadrature operating points while simultaneously monitoring the FPGA ADC input pin with a low speed oscilloscope performing a FFT and the MZM output with a high speed oscilloscope to examine the optical data eye.

Finally, a coarse voltage sweep on the DC electrode was performed using the hardware model. The applied DC bias voltage DAC was varied across its range representing -10V to +10V in small discrete steps by a software control function running on a computer; that voltage was then held for 100 seconds. At each step the output power, filter outputs and averaged filter outputs, calculated using an exponentially weighted moving average (EWMA), were recorded at 1 second intervals by the control software. The voltage steps were defined as a fixed increase of 50 on the 12 bit DAC control word. A total of 82 DAC words were used.

4.5 Results

The results are presented Figure 4-12 as a plot of optical power vs bias voltage.

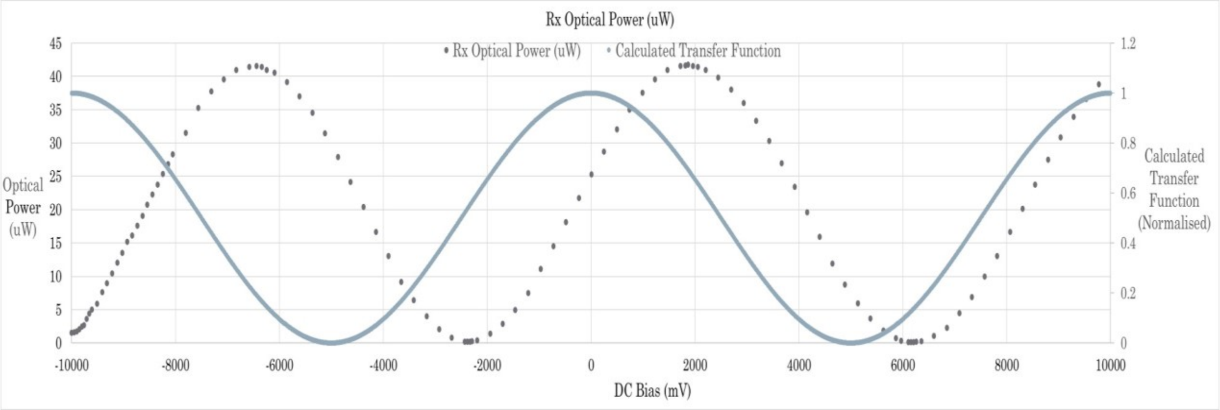


Figure 4-12 Optical power output of the MZM vs DC Bias compared with the calculated theoretical transfer function.

Figure 4-12 shows the result of the manual sweep of output power versus DC bias voltage. This response agrees with the sinusoidal form of the ideal, calculated transfer function shown in Figure 4-2, and overlaid here, with a horizontal offset of 1960 mV due to a combination of thermal drift and manufacturing variations of the MZM. The small difference in Vπ is due to calculation differences, the calculated transfer function had used a Vπ of 5000 mV . Analysis of this sweep gives a Vπ value for this modulator as ~4400 mV.

Figure 4-13 and Figure 4-14 show the FFT and eye traces recorded, at the MZM output, during the test of the analogue circuits. As predicted by equations 4.4 and 4.5 at quadrature the 4 kHz pilot tone was clearly visible and the 2nd harmonic (8kHz) was absent. Quadrature was confirmed by the good, open optical data eye which was observed.

In Figure 4-15 and Figure 4-16, the two filter responses are shown separately as a plot of the recorded output from the FPGA implemented filter directly against the applied DC bias voltage in mV. The height of each vertical bar plotted shows the range of the results recorded at that voltage. It can be seen that the filter centred on the pilot tone fundamental frequency of 4 kHz exhibited a strong response with large amplitude and relatively little variation, whereas the 2nd harmonic filter centred on 8 kHz produced a much weaker, less distinct result. Although it is expected from theory that the 2nd harmonic will be of considerably lower amplitude than the fundamental, the large variation in the filter output has been determined to be predominantly due to interference from electrical noise introduced by switching convertors and digital circuitry on the test board. In order to improve the 8 kHz filter response by allowing for a greater analogue gain to be applied to the feedback signal, an internal attenuation was required to be applied digitally, in the FPGA, on the input to the 4 kHz filter so as to prevent saturation, which is seen as a digital rollover in the filter output value. This analogue gain also emphasised any noise present in the detector chain or on the optical signal from elsewhere in the system. Even in the presence of this noise the output of this resource efficient filter implementation still clearly demonstrates the predicted sinusoidal response of the pilot tone fundamental (Figure 4-15) and 2nd harmonic (Figure 4-16) amplitudes as a sweep across the MZM transfer function is performed.

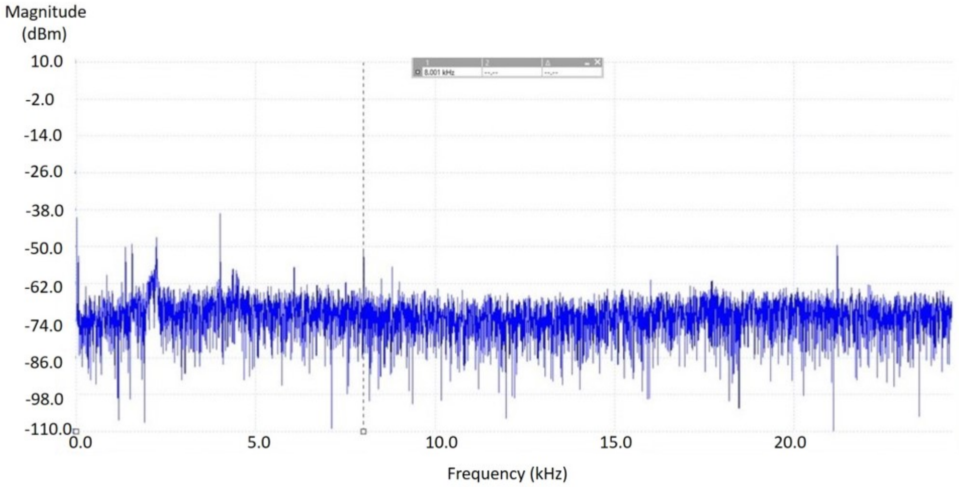
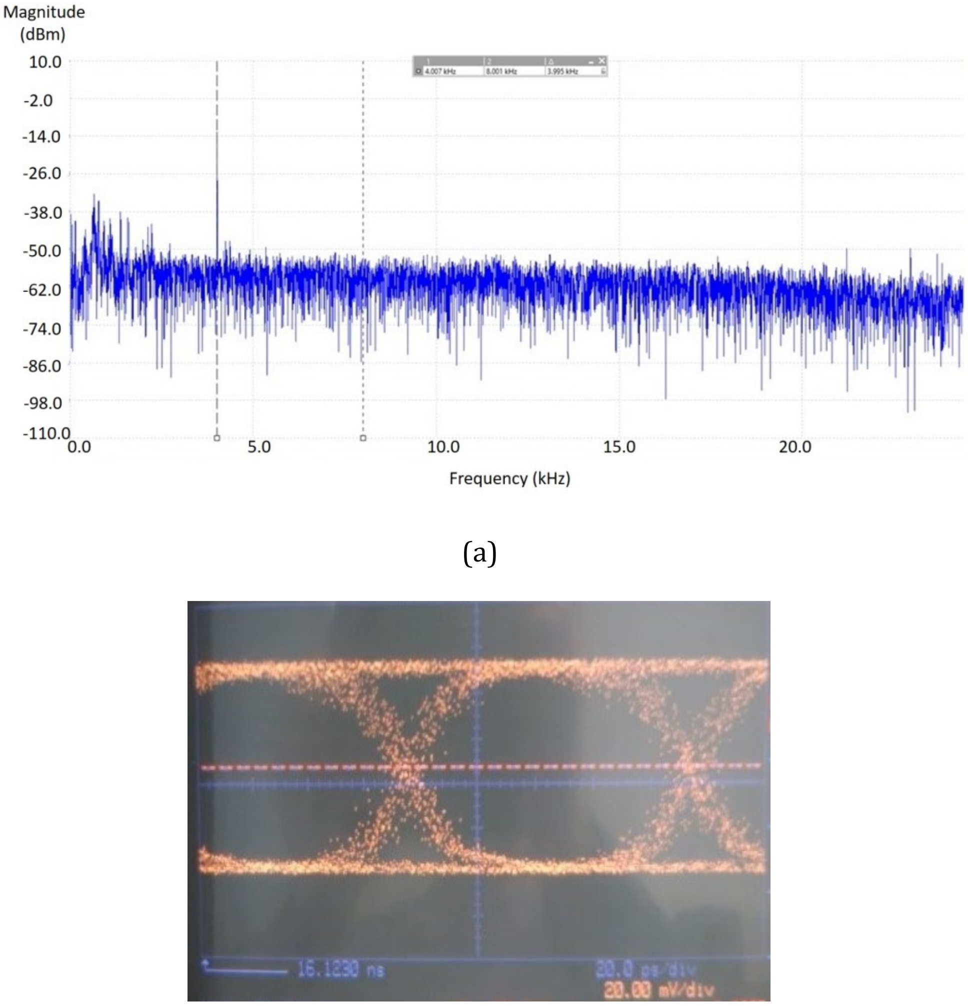


Figure 4-13 FFT of ADC input taken at Null operating point with marker at 8 kHz.



(a)

(b)

Figure 4-14 FFT of ADC input signal with markers at 4 kHz and 8 kHz(a) and Optical eye (b) at Quadrature.

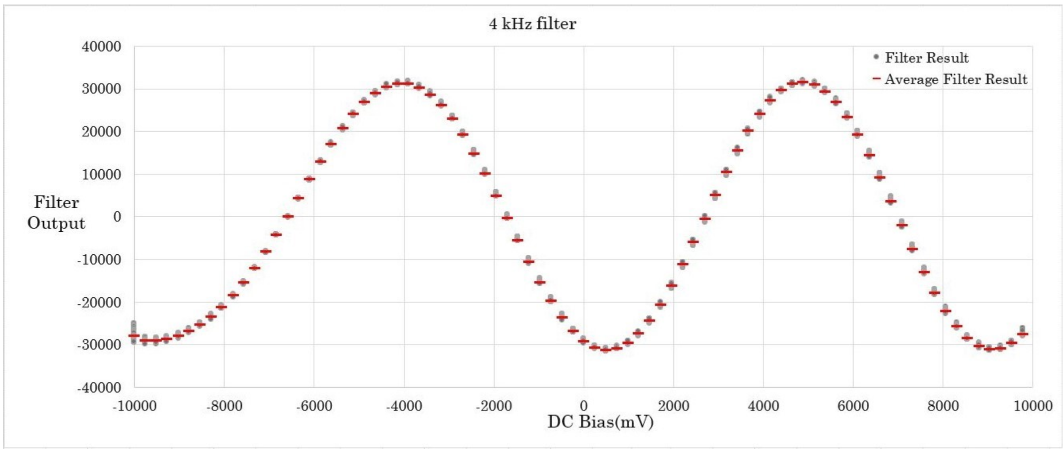


Figure 4-15 Fundamental frequency filter output with averaging vs Bias voltage.

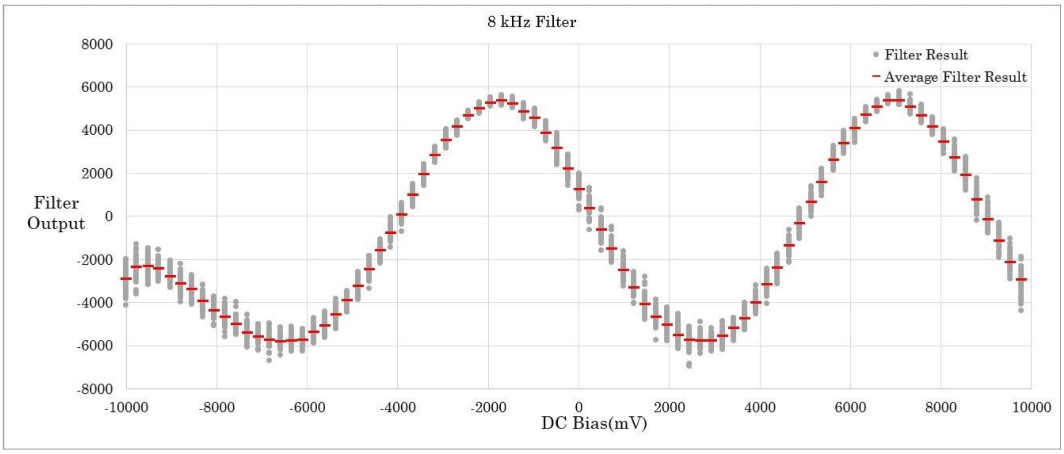


Figure 4-16 2nd harmonic frequency filter output with averaging vs Bias voltage.

Both Figure 4-15 and Figure 4-16 indicate a Vπ of ~4400 mV based on the theoretical responses plotted in Figure 4-3, with an offset of ~2500 mV. This strongly agrees with the Vπ figure obtained by examining the modulator output power in Figure 4-12, which also shows a Vπ of 4400 mV. The offset is a variable figure due to the horizontal drift of the MZM transfer function over time.

4.6 Summary

This experiment has demonstrated the viability of using a low-cost FPGA to implement in digital hardware a circuit capable of monitoring the transfer function of a Mach Zehnder Modulator. Specifically of note is that this is the first reported use of logic area efficient Goertzel algorithm based filters in this MZM application. The use of such small size implementations, when compared to the traditional filter designs, is important as it helps minimise the overheads associated with embedding test functions within the optical module assembly.

The implemented filters were able to track the harmonic response of the MZM and indicate a figure for Vπ which agrees, within the limits of the chosen DAC step size, with that seen by monitoring the MZM output power. This filter implementation has been shown to have a reduced implementation complexity, when compared to similarly specified FIR and IIR filters. The main complexity saving is in the reduction of the number of stored weighting factors from 918, in the case of the FIR filter or 17 in the case of the IIR filter down to 2 for the Goertzel filter. The number multiplication and addition operations performed per input cycle is also reduced. For the FIR filter, 918 multiplications and 917 additions are required for each input sample, the IIR filter requires 17 multiplications and 12 additions per input sample and the Goertzel filter implementation demonstrated here only requires 1 multiplication and 2 additions per input sample plus a further 1 multiplication and 1 addition per block of samples. The trade-off is the number of samples required before an answer is returned, In this experiment, the Goertzel filter requires 1500 samples, the FIR filter 918 samples and the IIR filter 17. In this experiment an external computer was used as a controller and multiple readings were recorded every second. One second was the minimum time the control software required to poll all required readings. This presents a bottleneck in the characterisation process as discussed earlier in Section 2.2. The total time required to perform a DC bias sweep in this case was 136 minutes. This can be addressed by moving the sweep functionality into the FPGA. Taking the sample rate used in this experiment of 20kS/s an entire sweep of 82 x 100 samples would require 0.41 seconds.

5 Embedded Soft Microcontroller

Following the earlier example of a microcontroller based calibration engine as shown in Figure 1-5; a soft microcontroller core is, in many instances, considered to be the first port of call to developing an integrated solution, either to perform all of the test procedure or a subset of less performance critical steps towards the final characterisation. Despite the rolldown of features into cost optimised devices, the availability of 'hard', i.e. dedicated functional block, IP microcontrollers in cost optimised FPGA's is not universal so a 'soft' implementation is required. The disadvantage of a 'soft' microcontroller is that it uses the reconfigurable logic cells of the FPGA, increasing the chip size to contain the additional functionality and hence adding to the cost. In this chapter the implementation of a simple microcontroller as a 'soft' piece of IP is evaluated. Here the design process metrics of complexity, logic and memory requirements relating to a simple microcontroller design, based around the Intel NIOS soft processor, are assessed in order to set a baseline for the comparison with the work of the following chapter.

5.1 Design Flow Analysis

When analysing the calibration engine it is important that the design flow is easily separated. As identified earlier it is desirable to separate the design environment from the test environment due to the different technical skills required in these stages.

Initial activity has been based around identifying a suitable FPGA platform to assess the capabilities of an embedded Linux system and test suitable ways of working. The method of achieving this was to initially examine a simple case using a single ADC and a single DAC.

The development board chosen for this stage of the experiment was an Altera Development Kit fitted with a device from the MAX10 family [76]. This board was chosen as the MAX10 is a low cost, low power device equipped with embedded multichannel ADCs, embedded high speed arithmetic multiplier cells for offloading calculations or building a custom arithmetic unit and has sufficient logic capacity to implement an embedded microcontroller. The board is equipped with a DAC device plus memory (volatile and non-volatile) and a UART interface to represent low bandwidth access. The FPGA device fitted to this board is one of the larger devices (10M50) offering around 50,000 LUTs and registers plus 1600 Kb of memory.

The processor in this instance is a NIOS II microcontroller developed by Altera. The NIOS II is termed a ‘soft core’ meaning that the logic used to implement the microcontroller and peripherals is not dedicated or hardwired to that function and can subsequently be reprogrammed and reused for another function. The NIOS II uses a Reduced Instruction Set (RISC) for process execution efficiency and supports a memory management unit as required by the Linux kernel. There is support, through an active, open source developer community and by the chip vendor, to be able to run an embedded version of the Linux operating system, referred to as the Golden System Reference Design (GSRD) for the Linux operating system which has been built for this board and the Golden Hardware Reference Design (GHRD) for the NIOS II FPGA core design presented in Figure 5-1 [77]. This is a reference design intended to give an introduction to developing embedded operating systems for field programmable hardware and, although not optimised for the application required here, gives a starting point from which to gauge the complexity of creating a generic programmable design for calibration activities. The GSRD is based on the BuildRoot open source repository and provides a scripted build environment using the Makefile language.

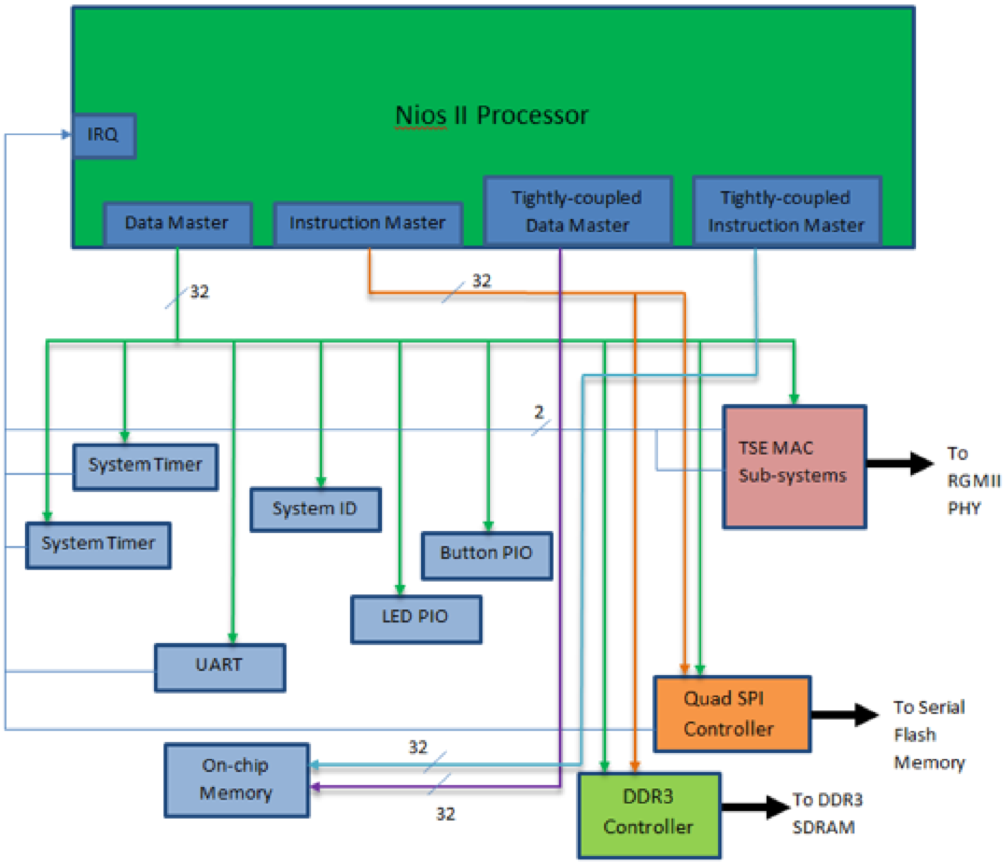
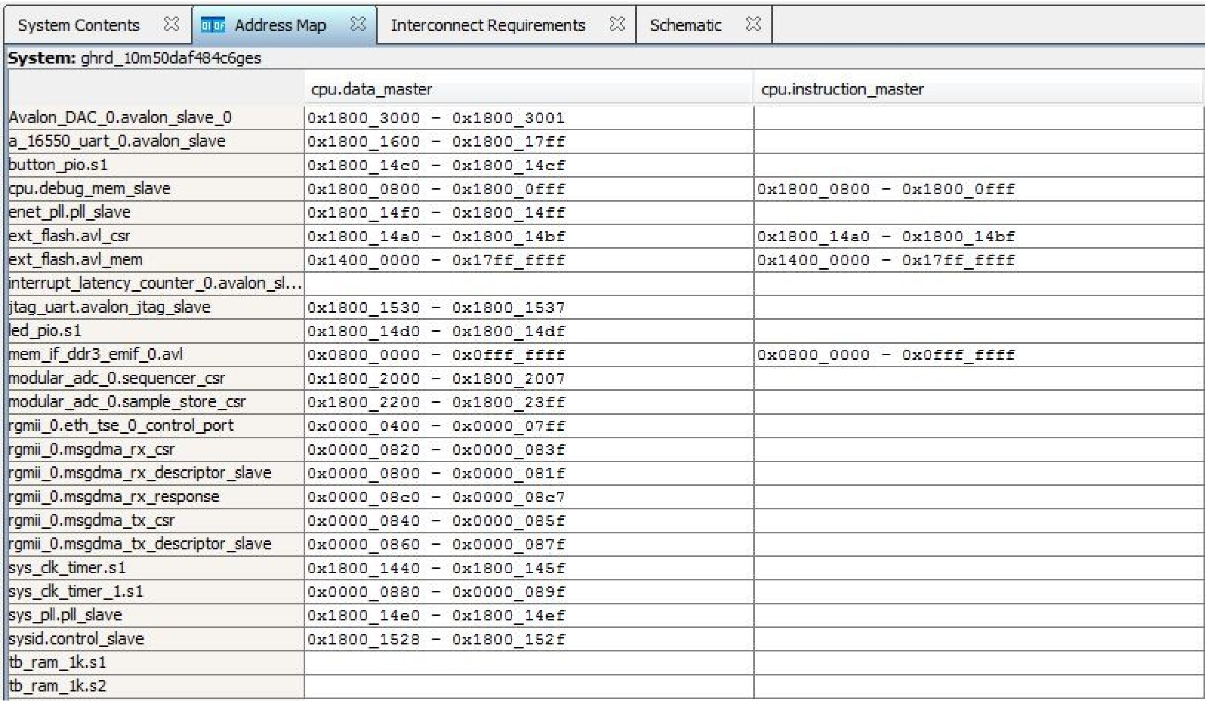


Figure 5-1 Initial Golden Hardware Reference Design architecture [77].

The stages of the OS build process are detailed in [77] but in summary involve:

- Building the basic root filesystem for a NIOSII core. At this stage it is possible to store test scripts in the root file system but they can be downloaded later using a Linux terminal window.
- Building a customised NIOS II Linux kernel for the development kit hardware. At this stage we include the board memory map or static device tree file in order to define the connection between hardware and software. An example of the hardware memory map is shown in Figure 5-2.



System: ghrd_10m50daf484c6ges

	cpu.data_master	cpu.instruction_master
Avalon_DAC_0.avalon_slave_0	0x1800_3000 - 0x1800_3001	
a_16550_uart_0.avalon_slave	0x1800_1600 - 0x1800_17ff	
button_pio.s1	0x1800_14c0 - 0x1800_14cf	
cpu.debug_mem_slave	0x1800_0800 - 0x1800_0fff	0x1800_0800 - 0x1800_0fff
enet_pll.pll_slave	0x1800_14f0 - 0x1800_14ff	
ext_flash.avl_csr	0x1800_14a0 - 0x1800_14bf	0x1800_14a0 - 0x1800_14bf
ext_flash.avl_mem	0x1400_0000 - 0x17ff_ffff	0x1400_0000 - 0x17ff_ffff
interrupt_latency_counter_0.avalon_sl...		
jtag_uart.avalon_jtag_slave	0x1800_1530 - 0x1800_1537	
led_pio.s1	0x1800_14d0 - 0x1800_14df	
mem_if_ddr3_emif_0.avl	0x0800_0000 - 0x0fff_ffff	0x0800_0000 - 0x0fff_ffff
modular_adc_0.sequencer_csr	0x1800_2000 - 0x1800_2007	
modular_adc_0.sample_store_csr	0x1800_2200 - 0x1800_23ff	
rgmii_0.eth_tse_0_control_port	0x0000_0400 - 0x0000_07ff	
rgmii_0.msgdma_rx_csr	0x0000_0820 - 0x0000_083f	
rgmii_0.msgdma_rx_descriptor_slave	0x0000_0800 - 0x0000_081f	
rgmii_0.msgdma_rx_response	0x0000_08c0 - 0x0000_08c7	
rgmii_0.msgdma_tx_csr	0x0000_0840 - 0x0000_085f	
rgmii_0.msgdma_tx_descriptor_slave	0x0000_0860 - 0x0000_087f	
sys_clk_timer.s1	0x1800_1440 - 0x1800_145f	
sys_clk_timer_1.s1	0x0000_0880 - 0x0000_089f	
sys_pll.pll_slave	0x1800_14e0 - 0x1800_14ef	
sysid.control_slave	0x1800_1528 - 0x1800_152f	
tb_ram_1k.s1		
tb_ram_1k.s2		

Figure 5-2 GHRD Memory Map.

- Generation of an Altera proprietary format binary file (POF) for programming into on-card flash, this is required by the toolchain to be a 2 stage process involving the conversion of the NIOS II Linux kernel into an intermediate format, then into HEX. The built root file system also needs to be converted into Hex format before both files are combined into a single POF. The size of the resulting POF is 64 MB.

The initial activity was familiarization with this build flow which unfortunately highlighted a number of compatibility and feature deprecation issues within the scripted build process and tools. These included command options no longer being supported when sourcing design code from the reference design repositories. This led to software build process failures which prevented even the original GSRD image being rebuilt. These, unfortunately, proved time consuming to track down and resolve before the reference design software build process was repeatable and provided a stable base on which to proceed.

The reference design supplied only makes use of a single 1 MS/s ADC which is embedded into the FPGA and which is controlled by its own sample sequencer, running in parallel in FPGA fabric to the main microcontroller software and passing the results into on chip memory. DAC functionality on the development board is provided externally to the FPGA with a separate chip.

As a simple trial of the proposed block based, hardware/software hybrid, virtual instrumentation method, an interface to the on-card DAC device was added. An interface module was required to be developed in VHDL and inserted into the reference design to drive this device.

The chosen implementation for the DAC interface was to create a memory mapped peripheral with the DAC control interface protocol implemented in FPGA fabric. This approach was chosen, firstly to simplify the design at the software level by taking the interface protocol into hardware and secondly to leverage the parallel nature of FPGA fabric, freeing up the software to perform other tasks during the DAC write. This approach is representative to the method which would be required as more complex, time intensive features, such as voltage sweeps, are added later. The DAC fitted to the card is a DAC 8551 from TI and uses a 3 wire SPI connection. To support the block based approach the module was designed to look like a memory mapped location using the Avalon bus standard adopted by Altera's NIOSII system and the SPI protocol was implemented in VHDL.

The trigger for a DAC write then simply becomes a single write to a memory mapped location which is a trivial matter to implement in a control shell script.

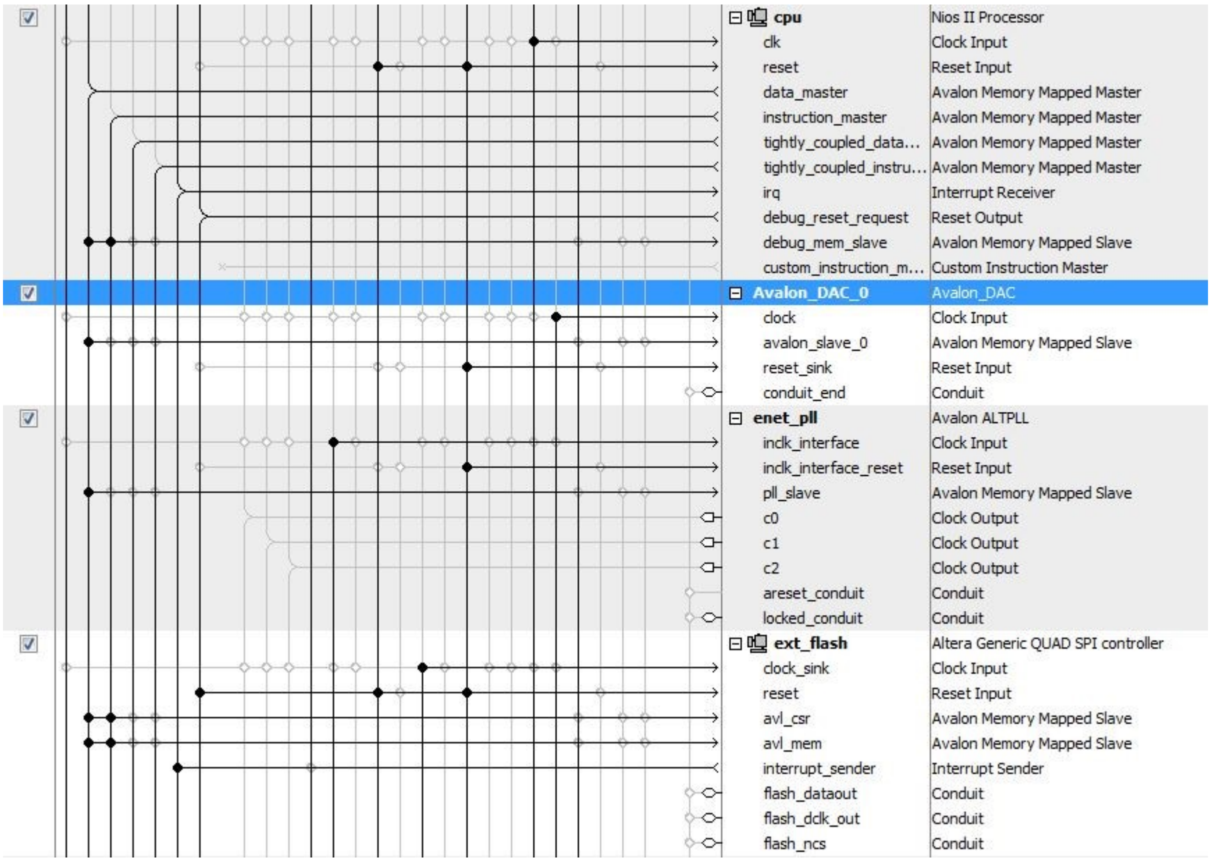
The FPGA hardware build process uses the Altera QSYS system generator tool to define connections between functional blocks, then makes use of Altera Quartus toolchain for synthesis, routing and timing check activities. Functional blocks can be created and simulated using any supported Hardware description language, in this case VHDL was used.

The steps taken to implement this were:

- Design, code and test the DAC interface using VHDL and ModelSim simulator.
- Import the VHDL design into QSYS as a component and define the microcontroller side interface as being an Avalon bus or conduit. Connect these interfaces into the NIOS II system. The result is shown in Figure 5-3. A schematic representation of the QSYS connections is shown in Figure 5-4.
- Allocate an address range in the device memory map to the DAC interface, which is shown in Figure 5-2. The resulting memory map could then be used

to derive a new static device tree file, to be used in the operating system design flow, to generate a new NIOS II Linux kernel with the DAC interface visible to the operating system.

- Synthesise the new hardware design using Quartus and map onto LUTs. The results are shown in Figure 5-5.



☑ ⊟ **cpu**	Nios II Processor
clk	Clock Input
reset	Reset Input
data_master	Avalon Memory Mapped Master
instruction_master	Avalon Memory Mapped Master
tightly_coupled_data...	Avalon Memory Mapped Master
tightly_coupled_instru...	Avalon Memory Mapped Master
irq	Interrupt Receiver
debug_reset_request	Reset Output
debug_mem_slave	Avalon Memory Mapped Slave
custom_instruction_m...	Custom Instruction Master
☑ ⊟ **Avalon_DAC_0**	Avalon_DAC
clock	Clock Input
avalon_slave_0	Avalon Memory Mapped Slave
reset_sink	Reset Input
conduit_end	Conduit
☑ ⊟ **enet_pll**	Avalon ALTPLL
inclk_interface	Clock Input
inclk_interface_reset	Reset Input
pll_slave	Avalon Memory Mapped Slave
c0	Clock Output
c1	Clock Output
c2	Clock Output
areset_conduit	Conduit
locked_conduit	Conduit
☑ ⊟ **ext_flash**	Altera Generic QUAD SPI controller
clock_sink	Clock Input
reset	Reset Input
avl_csr	Avalon Memory Mapped Slave
avl_mem	Avalon Memory Mapped Slave
interrupt_sender	Interrupt Sender
flash_dataout	Conduit
flash_dclk_out	Conduit
flash_ncs	Conduit

Figure 5-3 DAC interface interconnect into NIOS II using QSYS.

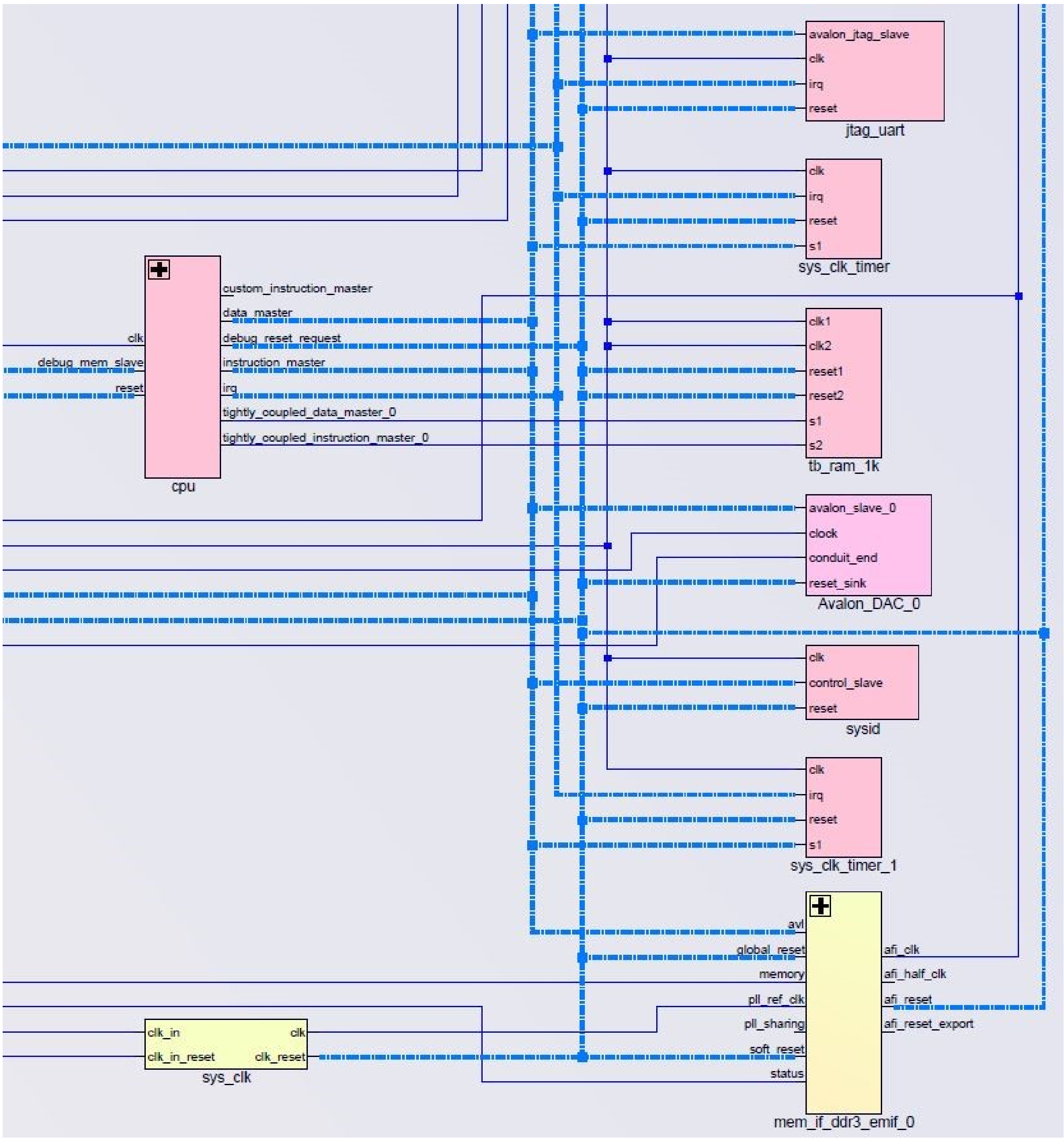
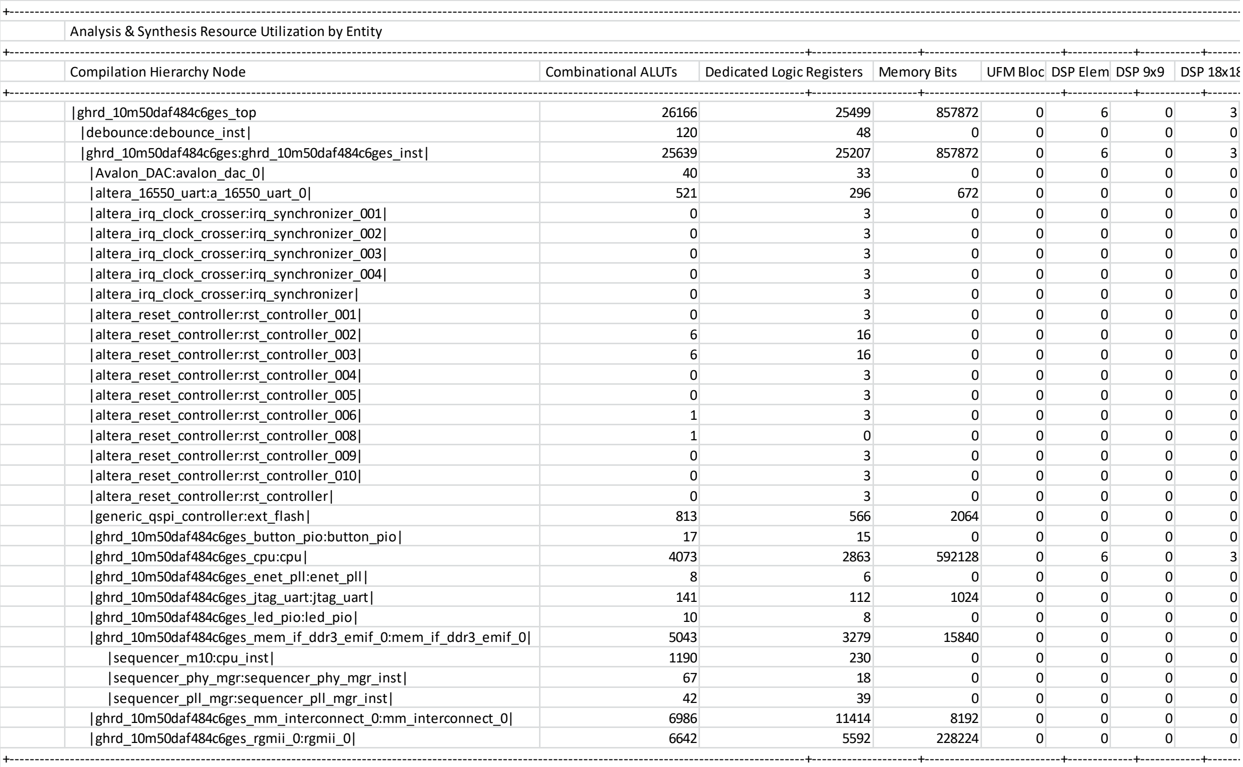


Figure 5-4 Block Schematic representation of the QSYS system with DAC.



Analysis & Synthesis Resource Utilization by Entity

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Bloc	DSP Elem	DSP 9x9	DSP 18x18
|ghrd_10m50daf484c6ges_top	26166	25499	857872	0	6	0	3
|debounce:debounce_inst|	120	48	0	0	0	0	0
|ghrd_10m50daf484c6ges:ghrd_10m50daf484c6ges_inst|	25639	25207	857872	0	6	0	3
|Avalon_DAC:avalon_dac_0|	40	33	0	0	0	0	0
|altera_16550_uart:a_16550_uart_0|	521	296	672	0	0	0	0
|altera_irq_clock_crosser:irq_synchronizer_001|	0	3	0	0	0	0	0
|altera_irq_clock_crosser:irq_synchronizer_002|	0	3	0	0	0	0	0
|altera_irq_clock_crosser:irq_synchronizer_003|	0	3	0	0	0	0	0
|altera_irq_clock_crosser:irq_synchronizer_004|	0	3	0	0	0	0	0
|altera_irq_clock_crosser:irq_synchronizer|	0	3	0	0	0	0	0
|altera_reset_controller:rst_controller_001|	0	3	0	0	0	0	0
|altera_reset_controller:rst_controller_002|	6	16	0	0	0	0	0
|altera_reset_controller:rst_controller_003|	6	16	0	0	0	0	0
|altera_reset_controller:rst_controller_004|	0	3	0	0	0	0	0
|altera_reset_controller:rst_controller_005|	0	3	0	0	0	0	0
|altera_reset_controller:rst_controller_006|	1	3	0	0	0	0	0
|altera_reset_controller:rst_controller_008|	1	0	0	0	0	0	0
|altera_reset_controller:rst_controller_009|	0	3	0	0	0	0	0
|altera_reset_controller:rst_controller_010|	0	3	0	0	0	0	0
|altera_reset_controller:rst_controller|	0	3	0	0	0	0	0
|generic_qspi_controller:ext_flash|	813	566	2064	0	0	0	0
|ghrd_10m50daf484c6ges_button_pio:button_pio|	17	15	0	0	0	0	0
|ghrd_10m50daf484c6ges_cpu:cpu|	4073	2863	592128	0	6	0	3
|ghrd_10m50daf484c6ges_enet_pll:enet_pll|	8	6	0	0	0	0	0
|ghrd_10m50daf484c6ges_jtag_uart:jtag_uart|	141	112	1024	0	0	0	0
|ghrd_10m50daf484c6ges_led_pio:led_pio|	10	8	0	0	0	0	0
|ghrd_10m50daf484c6ges_mem_if_ddr3_emif_0:mem_if_ddr3_emif_0|	5043	3279	15840	0	0	0	0
|sequencer_m10:cpu_inst|	1190	230	0	0	0	0	0
|sequencer_phy_mgr:sequencer_phy_mgr_inst|	67	18	0	0	0	0	0
|sequencer_pll_mgr:sequencer_pll_mgr_inst|	42	39	0	0	0	0	0
|ghrd_10m50daf484c6ges_mm_interconnect_0:mm_interconnect_0|	6986	11414	8192	0	0	0	0
|ghrd_10m50daf484c6ges_rgmii_0:rgmii_0|	6642	5592	228224	0	0	0	0

Figure 5-5 Extracted LUT counts for GHRD with DAC interface.

Analysing the utilisation figures shown in Figure 5-5 it is apparent that the soft IP core microcontroller is a large piece of logic. The FPGA device required (MAX10 M50) is the largest in the family with 50,000 LUTs and DFFs. This NIOS design requires a little over 50% of these in which to be implemented. For comparison, the FPGA fitted to the board used in the experiment of section 4.4 is a MAX10M08 with only 8000 LUTs and DFFs, barely one third the number needed for the basic microcontroller reference design. When considering the memory required to run the microcontroller the comparison between devices shows the same result. The whole NIOS design requires 51% of the available memory of a M50 device (857872 bits out of 1677312) which is 2.3 x the available memory of the smaller device used in section 4.4. If the soft microcontroller is part of a design which is subsequently reconfigured into the control logic, such that the microcontroller is no longer required, then these additional redundant memory blocks, LUTs and DFFs are adding cost which would need to be examined for value.

The GHRD contains functions and modules which would not ultimately be required by a project based on the requirements stated in section 1.3. The Ethernet module of the GHRD alone accounts for 26% of the present design. Similarly, the GSRD Linux build is a generic demonstration platform and contains features of the operating system which may not be necessary, so there is significant room for optimisation.

5.2 Summary

In this section the complexity of implementing a functionally simple microcontroller system in an FPGA based on a commercially available piece of soft IP has been evaluated. This design used a custom embedded Linux operating system, with the aim of attempting to separate the creation of tests, using a scripting language, from the low level OS interactions with peripherals such as the ADC and DAC. The build process required to obtain the custom Linux image proved to be lengthy, obfuscated and complex. The complete processor system hardware itself occupied approximately 50% of the available memory, logic and registers in the targeted MAX10 device. Optimisation of the microcontroller system will reduce this as the bare NIOS II, without peripherals, is only approximately 10% (4000 LUTs) of the available resources in this device, however the NIOS II alone accounts for a substantial percentage of the used memory and approximately 35% of the total memory available on the chip (592128 bits used out of 1677312 bits available in this case), and this would likely be the limiting factor in any optimisation. This will have the effect of increasing the minimum device resources required for this approach and, hence, increasing the cost.

Using an embedded microcontroller is a valid and commonly used approach to inserting test functionality into an optical module, particularly when used alongside co-processing blocks implemented in FPGA logic cells which are able to circumvent the more sequential nature of the microcontroller. However, logic and, in particular, memory requirements can push the target device size up to allow for sufficient space. Given this, the question remains as to whether there is an alternative approach, potentially better suited to the FPGA's unique advantages.

6 MACHINE LEARNING

As an alternative to using an embedded microcontroller, in this section the use of Artificial Intelligence (AI) and in particular the sub category of Machine Learning (ML) is studied as a solution to the characterisation and test tasks discussed previously. The concepts of ML are introduced and current applications of ML in optical communications are presented. The application of ML using FPGAs is discussed through the example of the first-time implementation of an Artificial Neural Network (ANN) using a recurrent bidirectional architecture. Finally, the novel use of a small ANN in conjunction with the filter described in Section 4.3 is presented as a means of characterising an MZM.

6.1 Concepts of Machine Learning

Machine Learning is a broad field which, in general, covers the application of numerical analytic methods to find answers to problems without having an explicit set of preprogrammed instructions. Instead, a ML model is taught by finding the solution to a mathematical optimisation problem by way of exploring sets of parameters in order to minimise the values of a defined cost function e.g. a mean squared error. By this definition learning is specifically related to mathematical optimisation. Classic ML tasks have included classification, curve fitting and anomaly detection. The training of a ML model can fall under one of 4 general categories:

1. Unsupervised – using unlabelled training data to find general patterns
2. Supervised – using labelled training data to make predictions or classification of input data
3. Semi-supervised – a combination of labelled and unlabelled training data
4. Reinforcement learning – training via feedback from interactions with the environment.

While some applications are ‘Cloud’ based, making use of the massive computing power available with an internet connection, and in parallel with the growth of IoT services (as discussed in section 2.1), there has been a push to move ML applications towards the system edge, nearer to the point of data gathering and with more limited computing resources e.g. spoken keyword detection on smart household devices, looking for anomalies in sensor readings [78] or enabling the use of low cost atmospheric pollution sensors by providing correction functions, [79] as implementation in low cost processing devices becomes possible. This has been in order to reduce the volume of data sent over the internet and to remove any network latency involved in the connection back to a cloud server. There is also a need for the ML algorithms to be available even when network connectivity is lost in order to be effective constantly.

In these applications ANNs have shown themselves to be an effective tool in identifying patterns in data which otherwise may be obscure. Neural networks were inspired by a simple functional model of an animal brain. Layers of artificial neurons are interconnected to form a network (Figure 6-1).

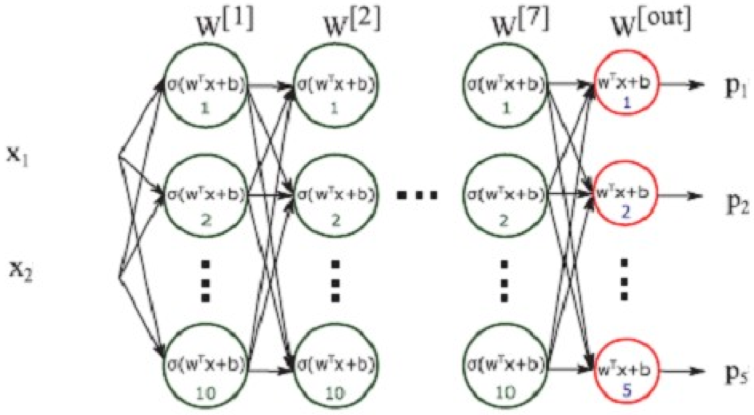


Figure 6-1 An Artificial Neural Network [80].

Each artificial neuron is built from a linear sum of products and a non-linear activation function (Figure 6-2) as shown in equation 6.1 below.

$$y = \sigma\left(\sum_{0}^{n} W_n x_n + b\right) \tag{6.1}$$

Where: σ is the activation function

W_n is the linear gain value or weight for each input x is the input value to the neuron

b is a fixed offset or bias

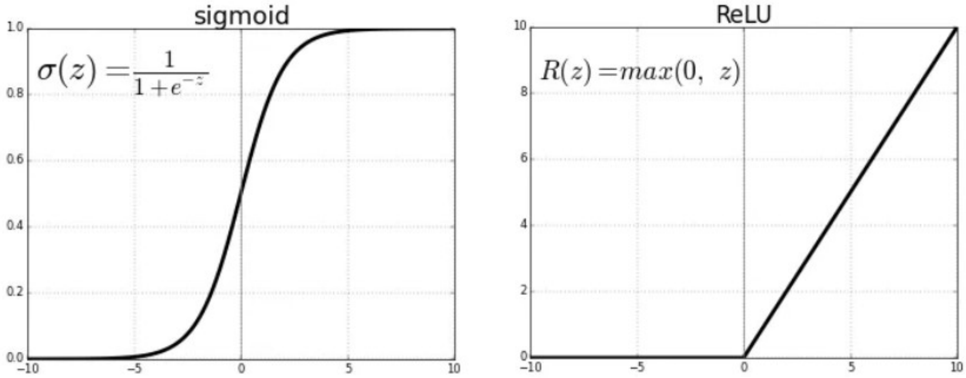


Figure 6-2 Common non linear activation functions include the sigmoid function and the Rectified Linear Unit (ReLU) [81].

Over the course of this study the field of Machine Learning across all fields (for example: speech recognition in smart home devices, image recognition in autonomous vehicles and drug discovery for clinical trials) has been developing at speed. In the field of Optical communications, Machine Learning is now proving to be a useful tool in the optical researchers toolkit [82] [83] [84] [85] (see Figure 6-3) .

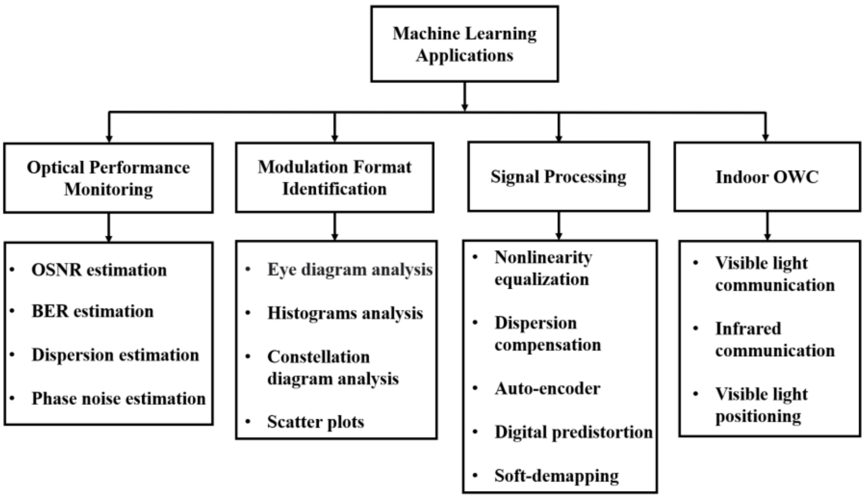


Figure 6-3 Machine learning applications in optical communications [84].

6.1.1 FPGAs and ML

To date the majority of published research concerning the use of Machine Learning in optical communications is focussed on larger scale DNNs used, for example, for correcting non-linear transmission effects or calculating the optimum gain settings for Raman amplification. Many of these papers have concentrated on the offline development of the NN and have used external computers for implementation.

In Schaedler [86] a Machine Learning approach to correcting for the non-linearity of a Indium Phosphide MZM transfer function based on an adaptive digital predistortion algorithm is presented. This experiment makes use of what is termed the 'Extreme Learning Machine architecture' to demonstrate adaptive estimation and compensation of MZM transfer functions albeit with offline DSP processing.

The use of a trained Neural Network to calculate the pump powers and wavelengths required to generate a desired Raman Gain profile is proposed by Zibar [80], with experimental evidence to show that this is quicker to execute than a traditional Raman Solver program and allows for almost realtime adjustments to be calculated.

However, practical implementations are being shown to be possible aided by the use of FPGAs [87] [88]. When implementing a ML algorithm using programmable logic, there is a potential performance advantage over a CPU based solution which comes from parallelism in the device. For example, an operation requiring 100 functionally similar, calculation operations running sequentially on a CPU using a 1 GHz clock would require 100 cycles giving a throughput of 10M operations per second. An FPGA, leveraging parallelism, could complete the entire operation in a single cycle. Assuming a clock speed of 250 MHz in the FPGA this would give a throughput of 250 M operations per second.

An example of a simple neuron with multiple inputs, following equation 6.1, being mapped into FPGA 'cells' is shown in Figure 6-4. With the availability of DSP cells to perform the multiplication and addition a highly efficient implementation is possible, using a small RAM block to store the associated weights and another RAM to implement the activation function as a table. A number of possible methods to implement the non-linear activation function have been compared and the implications of each assessed [88]. Using a RAM to implement a table is a commonly used method [89] due to the high clock speeds/short access time offered by these blocks. However, FPGA RAM blocks generally only have a maximum of 2 read ports so can present a bottleneck in the operation of the neuron and careful analysis of the sequence of events is required. Using this method an approximation of the activation function can be constructed from a number of evenly separated points. The architecture of the FPGA allows for the creation of many small, distributed RAMs using the LUT memory of the LEs which facilitates parallel calculations, although with the cost of decreased approximation resolution as the number of bits in each of the distributed RAM blocks is limited.

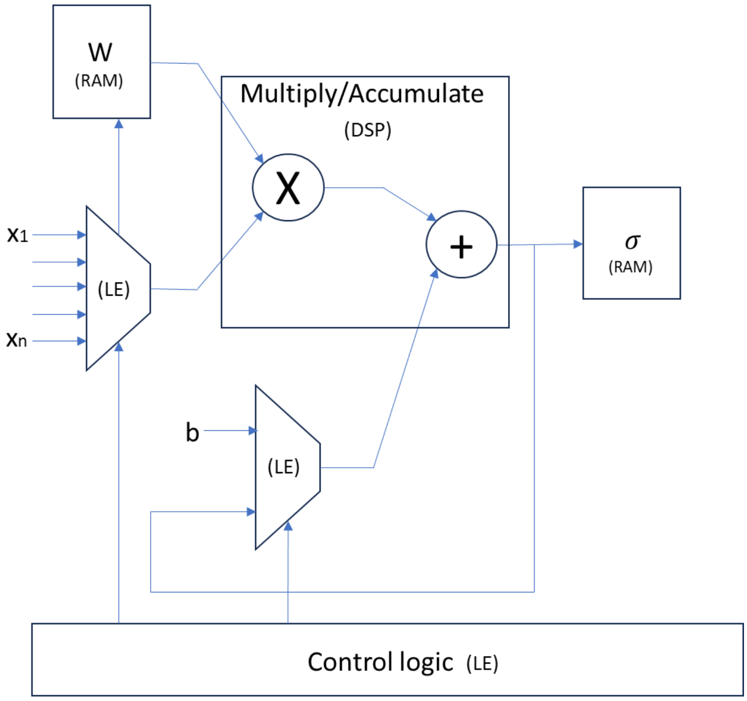


Figure 6-4 Neuron function mapped into generic FPGA function categories.

With the introduction of higher level languages such as OpenCL and C++ into the FPGA design flow, toolchains are now becoming available which can utilise these High Level Synthesis (HLS) methods, integrated with proven physical design flows, to design the logic associated with ANNs [90] [91]. The steps involved in the realization of an FPGA design are illustrated in Figure 6-5 and are discussed further in section 6.2.

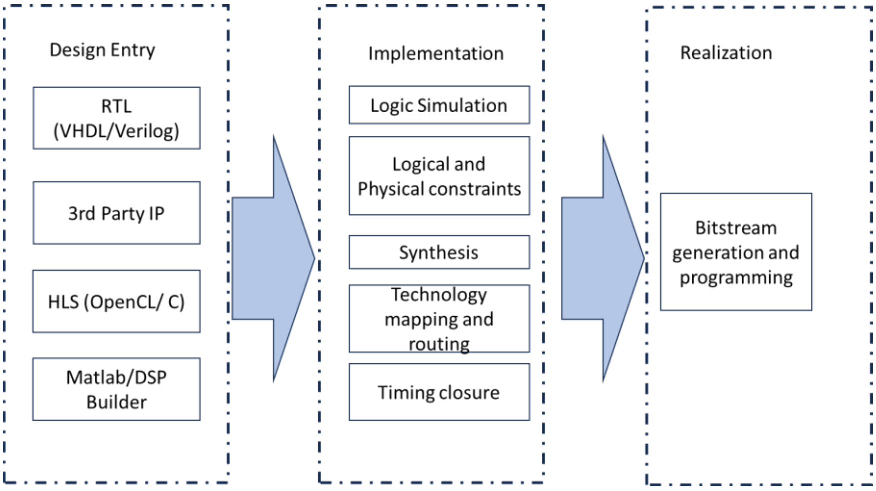


Figure 6-5 FPGA design flow to device realization.

6.2 Challenges when implementing a neural network based equalizer in an FPGA.

The aim of this section is to discuss some of the steps, considerations, problems and challenges encountered during this first-time implementation, in an FPGA, of a recurrent NN (RNN) using the bidirectional Long Short-Term Memory (biLSTM) architecture to solve the problem of non-linearity mitigation in coherent optical networks.

While the details of non-linearity correction are discussed in detail elsewhere [92], within the context of this thesis the implementation challenges are presented in more detail. Specific focus is given to the HLS design flow and the motivation behind the use of this method in general. This is followed by a brief description of the tool chain and the necessary steps to create a NN for an FPGA. Solving the design challenges presented while investigating the viability of implementing such NN based solutions in an FPGA or ASIC is a key question when investigating the commercial application of such solutions.

The challenge presented by this piece of work was to achieve the required throughput using the RNN, as the overall purpose of the experiment was to assess the possibility of using the biLSTM architecture instead of the more commonly found feedforward Chromatic Dispersion Compensation (CDC) algorithm [93]. Although the biLSTM RNN can be shown to give a Q factor improvement [94], the feedback paths in the RNN architecture result in higher latency than the purely feedforward CDC.

The LSTM cell architecture (as described in section 6.2.1 and illustrated therein in Figure 6-9) is built from 3 gates – an input gate, an output gate and a forget gate – which control the flow through the cell. The function of the forget gate is to discard information from a previous state, thus setting a memory time interval. The input gate performs the same function on the new input data and the output gate controls what is forwarded from the cell. Weights and biases for each gate are determined in the training process. The general equation of an LSTM gate is expressed in equation 6.2.

$$h_t = \sigma(Wx_t + Uh_{t-1}) \tag{6.2}$$

The output state of the current cell at time t (h_t) is dependant on a weighted function of the current input (Wx_t) added to a weighted function of the previous output state (Uh_{t-1}) multiplied by an activation function σ.

The biLSTM is constructed from 2 LSTMs, one taking the input in the forward direction and the other taking it in the reverse direction (Figure 6-6). The output Y of each biLSTM cell is calculated as a function (σ) combining the outputs of both forward (h_{FWD})and reverse (h_{REV}) LSTM cells.

$$Y_t = \sigma(h_{tFWD} + h_{tREV}) \tag{6.3}$$

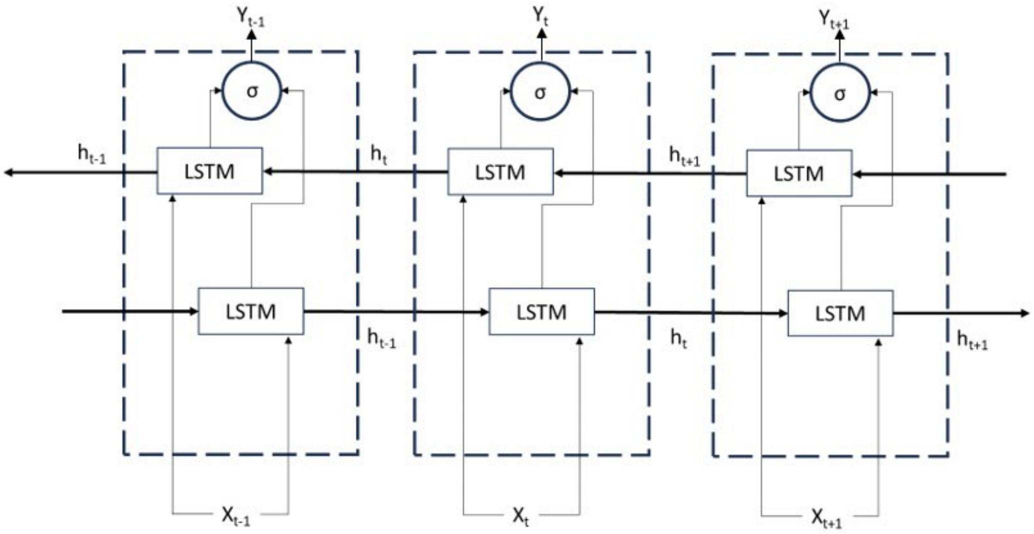


Figure 6-6 3-cell section of a biLSTM.

The starting point for this realization is a trained Python model of the biLSTM network. The proliferation of ML based applications and solutions has generated a number of readily available software packages and libraries to build and train ML models, with Python based tools such as Google's open source platform – Tensorflow [95] – being commonly used for this task. The Python source code generated is not directly synthesizable into hardware gates so an additional step is required where the Python is converted into C++ which can be used as a design entry point as shown in the design flow depicted in Figure 6-5.

A more detailed description of this model and of its training phase have been published in [88] but in summary the biLSTM layer consists of 35 hidden units (equating to h_t in equation 6.2) taking 81 inputs and, after a convolutional output layer, a total of 61 symbols are output (Figure 6-7). It is this parallelisation of the outputs which allows the FPGA to have a high throughput. Hyperbolic Tangent activation functions were used in the hidden layers whereas the output used a linear activation function. The Mean Square Error (MSE) loss function and the Adam algorithm for stochastic optimisation [96] were used during training.

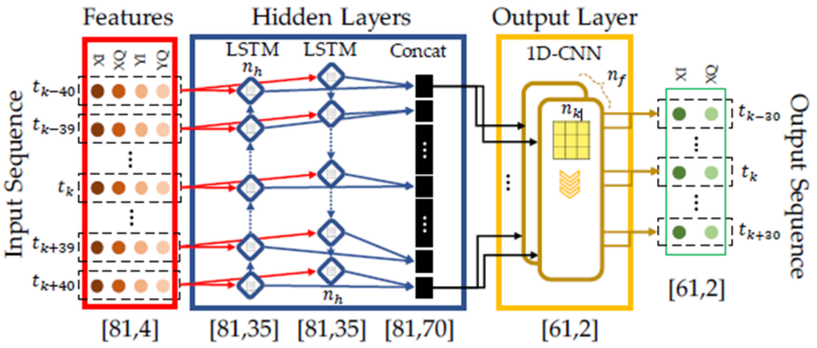


Figure 6-7 Recurrent equalizer using biLSTM Hidden Layers [88].

The targeted evaluation board for this equalizer implementation was the AMD Versal VCK190 [97] which is equipped with an AMD XCVC1902 Adaptive Computing SoC, a high performance device built using a 7nm process technology. The associated AMD toolchain as part of the overall design process is shown in Figure 6-8.

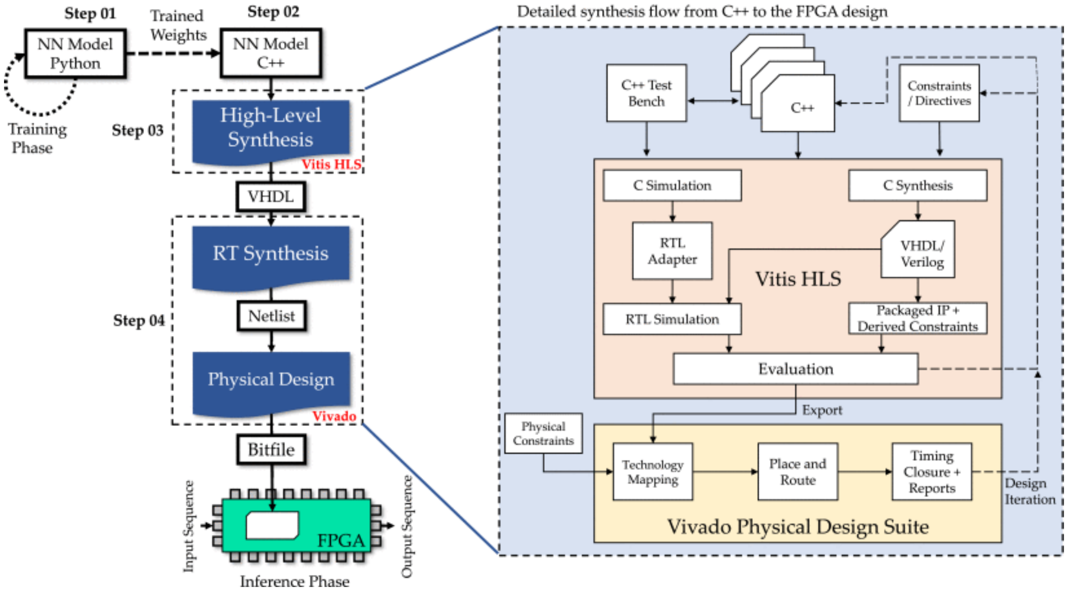


Figure 6-8 AMD Toolchain and the steps used to realize a NN [88].

6.2.1 C++ Implementation of Neural Networks and Generation of VHDL through High Level Synthesis

The HLS design flow allows a functional code block to be described using a high level of abstraction in C++ code and then automatically converted into synthesisable VHDL for use in a normal FPGA tool flow. The C++ model is untimed, that is, it has no internal clocked storage in its functional description and clocked elements are added later in the VHDL conversion process. The advantage of this method is that it separates the functional behaviour of the block, in this instance a biLSTM NN, from the technology specific features of the FPGA. Working at a higher level of abstraction, the intended functionality can be the focus of attention and be described more easily in fewer lines of code [98] bypassing the lower level Register Transfer Level (RTL) constructs associated with languages such as VHDL, e.g. process sensitivity lists and clock statements, which define the hardware data flow. As the C++ model is untimed, functional verification in C++ is much faster than functional simulation in VHDL where a timed model approach is taken i.e. the state of every node in the model is evaluated (sequentially controlled by the aforementioned sensitivity lists and clock statements) on every simulation ‘tick’. This makes it quicker to test and debug the design [99].

Ultimately, the model being described in C++ for HLS is to be targeted at a hardware device and it is important to recognise this throughout the conversion. It is important to note that, although HLS supports an extensive range of the C++ language, not all can be synthesized into an FPGA. The FPGA has a different architecture from a CPU. For example, during the course of NN implementation, memory allocation is a key part of writing the C++ code that needs to be properly assessed. In hardware, such as an FPGA, the memory allocations are static and are assigned to specific instances of RAM during the technology mapping phase of the physical design flow, meaning that the designer must allow for sufficient memory before execution. Dynamic memory allocations, as found in many C++ standard library functions e.g. calloc(), which allocate memory at runtime, cannot be supported and must be avoided or replaced with functions optimized for implementation in an FPGA by using the libraries provided by the HLS tool supplier, in this case, AMD. Similarly, high level functions which would relate to the operating system such as file or date and time operations have no equivalent meaning in hardware and all data exchanged with the FPGA block must use simple digital input/output ports.

As shown in Figure 6-8, two pieces of C++ code are required by the HLS flow. The first is the testbench, which will not be converted into VHDL, while the second is the function to be converted and implemented in VHDL. The testbench code contains the entry point (by convention labelled as the main() function) to the model and also any functions which do not form part of the synthesizable code, for example, in this case the testbench was used to read the previously saved signal inputs and weights derived through Python training and convert them to a fixed-point 32 bit format. This format was selected to take advantage of the simpler implementation of integer arithmetic in hardware. The function contained in the second piece of C++ code was the C++ translation of the Python NN biLSTM equalization architecture, using fixed-point arithmetic operations. It is this piece of code which was subsequently converted into VHDL. The purpose of the testbench was then to take the outputs from the biLSTM code after equalization has been performed and to verify the performance of the C++ model by calculating the MSE.

The recursive structure of the implemented LSTM cell is illustrated in Figure 6-9. The recursion is indicated by the dashed line. Expanding on equation 6.2 the equations for each of the LSTM data gates at time t are given as:

$$i_{t=}\sigma(W_i x_t + \ U_i h_{t-1} + b_i) \tag{6.4}$$

$$f_{t=}\sigma\left(W_f x_t + \ U_f h_{t-1} + b_f\right) \tag{6.5}$$

$$o_{t=}\sigma(W_o x_t + \ U_o h_{t-1} + b_o) \tag{6.6}$$

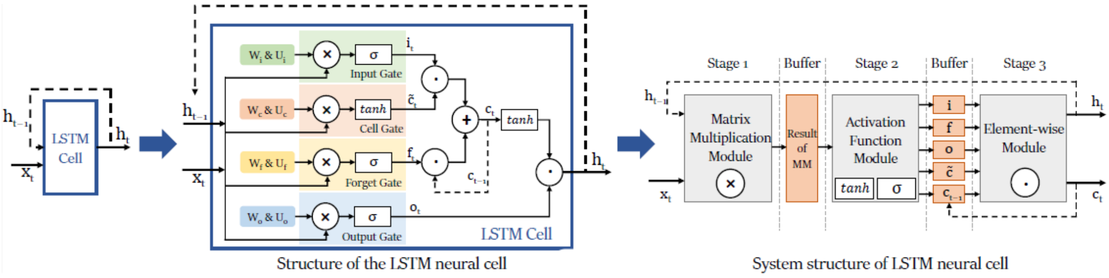
$$\tilde{c}_{t=}\emptyset(W_c x_t + \ U_c h_{t-1} + b_c) \tag{6.7}$$

$$c_{t=}f_t \odot c_{t-1} + \ i_t \odot \tilde{c}_t \tag{6.8}$$

$$h_{t=}o_t \odot \phi(c_t) \tag{6.9}$$

∅ is usually the hyperbolic tangent activation function (tanh), σ is the sigmoid activation function, x_t is the input vector at time t, W and U represent the trainable weight matrices, and b is the bias vector. i_t; f_t; o_t; $\tilde{c}_t$, c_t, and h_t denote input gate, forget gate, output gate, cell input, cell state, and hidden state vectors, respectively. The $\odot$ symbol represents the elementwise (Hadamard) multiplication.

For the implemented hardware, the input data h_{t-1}, x_t, and weight matrices W and U are read for each gate, and the systolic array technique is used to do the matrix multiplication. The results are temporarily stored in memory on the chip. Next, the activation function module reads the data from this temporary buffer and calculates the output vectors corresponding to each gate; i, f, o, and c, as shown in Figure 6-9. Each gate's output is then buffered in memory. Finally, the element-wise computation module reads the data - i, f, o and c from the buffer, completes the element-wise computation and then obtains the output h_t and cell state c_t, which will be used in the next time step. After all required time steps have been completed, the final output is written back to the memory.



Structure of the LSTM neural cell

System structure of LSTM neural cell

Figure 6-9 Structure of an LSTM cell showing the recursive path and the implementation of the LSTM showing buffers [88].

When using HLS some human design intervention is still necessary: even though the conversion to VHDL is automated, engineering decisions, based on an understanding of the targeted FPGA device architecture and the intended implementation of the NN, must be taken to achieve the desired performance. This presented one of the major challenges in achieving a physical implementation which met its target performance parameters. HLS supports a set of directives, or pragmas, that can be used to modify and guide the behaviour of the HLS C++ to VHDL synthesis stage to support these interventions [100]. By utilizing these pragmas, it is possible to investigate several design structures without re-coding the model in order to discover the best implementation. Although there are a variety of different pragmas, here those relating to pipelines, loops and arrays have been used.

Pipelines allow the parallel execution of operations within a function, lowering the number of clock cycles between commencing loop iterations; these clock cycles are referred to as the Iteration Interval (II). If there are no dependencies each loop iteration does not need to end before the next one begins, i.e., the iterations can overlap. The number of pipeline stages can be controlled by setting the value of II using the HLS pipeline pragma. Setting the II to 1, as was done in this project, enables each cycle to begin with a new iteration.

An example of pipelining in action is illustrated in Figure 6-10. A loop function has been split into 5 operations. Operations A-E each take 1 clock cycle to execute. With no pipelining (i) a new iteration can only start every 5 clock cycles. If there are no conflicts or shared resources, applying pipelining with an II of 1 allows a new iteration to start every clock cycle (ii)

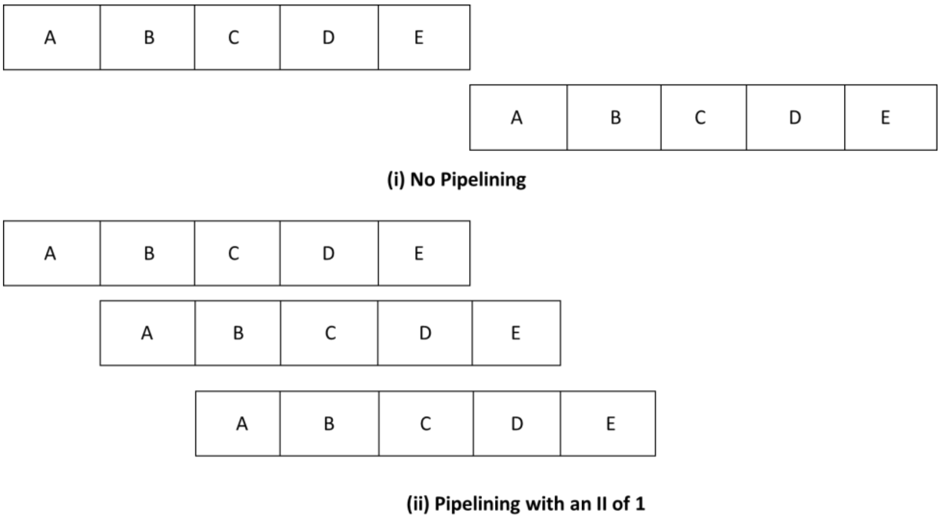


Figure 6-10 Pipelining of loop operations.

Loops can be manipulated by unrolling and flattening. By default, the VHDL conversion process favours using minimum circuit resources and so leaves loops within a function rolled up, which means that the loop body is executed sequentially, utilizing a single set of logic resources. The minimum loop delay is then equal to the number of loop iterations. By unrolling the loop multiple copies of the loop body logic are created, enabling parallel execution and reduced latency at the expense of additional circuit resources. Loops can be entirely or partially unrolled, resulting in either one copy of the loop body for each iteration, giving optimum throughput, or fewer copies for reduced resource cost. The flatten directive transforms a hierarchy of nested loops into a single loop, which eliminates a clock cycle delay while traversing between higher and lower nested loops and can help with better optimization of the loop logic. Flattening can only occur if the loops are functionally perfect. In a perfect loop there are no dependencies between loop iterations.

In Figure 6-11 the rolled loop (i) would take a minimum of 3 cycles to execute the function on the 3 inputs. Note the multiplexor on the input to Fn which can lead to a long path. Unrolling the loop (ii) allows execution to be completed in a single cycle at the cost of 3 implementations of Fn. While unrolling the loop can lead to a single cycle implementation, as the output of one function is dependent on the output of the previous function, the overall speed increase is limited and the cycle time may need to increase to accomplish the additional processing within one cycle.

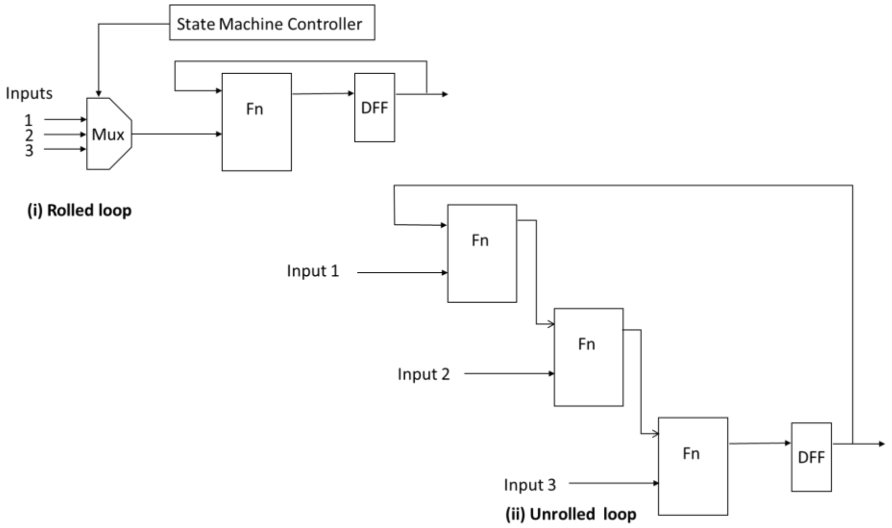


Figure 6-11 Unrolling of looped functions.

Given this limitation, the loops in the feedforward NN layers were able to be flattened; however, the loops in the recurrent NN layers were imperfect and could not be flattened due to their memory dependence. In the recurrent layer, therefore, only the matrix multiplication function in each LSTM was able to be flattened.

Array structures can be partitioned and reshaped. HLS will implement arrays defined in the C++ code as RAM blocks in the VHDL model. This can cause a restriction in the design concurrency as FPGA RAM blocks only have 2 access ports, which, if the designer has also used the unroll and pipeline pragmas, will need to be shared between all instances of the loop body. By reshaping and partitioning the array, the size and number of these memory blocks can be controlled. To enable the successful flattening of the loops in the NN architecture, the non-equalized signal (input signal) and the weights of the NN architecture required partitioning to spread the implemented array across multiple RAMs and increase the number of visible access ports.

The final stage of the HLS step was to export the generated VHDL and the HLS derived constraints for use in the physical design step.

6.2.2 The Physical Synthesis Step to the FPGA Realization

During the HLS C++ to VHDL conversion process, targets relating to the performance of the design can be set. These area and timing reports generated by the HLS process are, however, only estimates of the final design performance based on the technology-specific data libraries for each FPGA; the actual performance cannot be determined until the physical implementation is complete. For this work, the Vivado Design suite from AMD was utilized [101]. Vivado uses design entry through Hardware Description Languages (HDL) such as VHDL. The physical implementation of the design into the FPGA device is performed by Vivado in three steps: technology mapping, placement and routing, and timing analysis.

Technology Mapping: Within this step, the VHDL source code is translated into primitive logic gates and functions described by Boolean logic equations, followed by mapping these gates and equations onto FPGA programable logic blocks containing D-type FFs (DFFs) and RAM-based LUTs or more specialized functional cells, such as DSPs, as covered in Section 3.1. During the technology mapping phase, the design is optimized eliminating duplicated or unnecessary logic.

The next two steps in the physical implementation constitute an iterative process executed automatically by the tool based on design constraints, such as a clock frequency. These limitations can be inherited from the HLS stage or defined in Vivado. The Vivado tool provides sets of pre-configured options through established optimization strategies, which are discussed in detail in the AMD user manuals

[102], and which the user can apply depending on the design goals. The optimal technique is determined by balancing computer runtime and outcomes. In [103] we can find all potential pairings of synthesis and optimization procedures in Vivado using a high-speed pulse width modulation circuit as a target design, as well as a comprehensive evaluation of the runtime versus performance of the different Vivado optimization methodologies. Since the target here was to increase throughput, solutions that targeted a reduction in chip size, power, or runtime were discarded and the Vivado configuration called "Performance ExtraTimingOpt" was adopted, since it effectively optimizes throughput by reducing timing slack.

Placement and Routing: This stage positions the logic blocks, developed during the mapping phase, onto the specific elements of the FPGA cell array (Figure 3-1), and configures the signal routing switches between them. The placement algorithm starts from a random seed position and then moves functions around the cell array based on the degree of placement congestion observed in parts of the die and the fanout of the driving function.

Time Analysis: This stage compares the design with the provided timing constraints to determine whether the overall performance requirement has been met. Timing analysis, in particular, requires a solid understanding of the FPGA structure and how the design has been mapped onto the array. It may be necessary to return to the HLS phase, as indicated by the iteration path in Figure 6-8, to apply more directives or even adjust the function's architecture to achieve timing closure. The time for a data (signal) to travel between two points is determined by a variety of factors, including DFF switching time, setup requirements (the time at which the signal must arrive at the destination before the capturing clock edge), logic and routing path delays, and clock edge uncertainty due to jitter and clock path skew. The critical metric here is the timing slack.

A negative timing slack indicates that the total delay in the data path between two DFFs is greater than the requested clock period. In this work the presence of negative slack, only reported once the physical implementation was complete, presented a challenge in successfully demonstrating the realization of the biLSTM in an FPGA. The NN had many nested loops and, as discussed in the previous Section 6.2.1, unrolling these loops would lead to a larger design consuming more logic area; but leaving large loops, i.e., the loops with a high number of iterations rolled up for area efficiency, can produce long logic multiplexer paths, as the inputs to the loop logic are selected (as illustrated in Figure 6-11 case (i)). Long logic paths i.e ones which pass through many LUTs' without any DFF retiming stages, accumulate long routing delays as the design is distributed during the placement operation. As previously stated, the HLS conversion phase can only make an estimate of timing paths as the tool will not be aware of all influencing factors at that stage, e.g further design features such as 3rd party IP, not being introduced through the HLS flow. Using Vivado timing analysis reports and an annotated netlist viewer, it was possible to identify the multiplexor paths which were responsible for the negative slack. In this case, the solution was to return to the C++ source and, by understanding how the C++ signal names had been mapped onto VHDL signal names, reorder the nested loops so that the outer loop, which was not being unrolled, had fewer iterations; this approach reduced the size of the logic chain in the multiplexer path leading to a positive timing slack result.

6.2.3 Conclusions from the implementation of a biLSTM NN equalizer

The first conclusion from this implementation is reached when analysing the resource utilisation in the FPGA. The biLSTM based equalizer required Block Random Access Memory (BRAM) to store future/past recurrent states, while the CDC does not need such blocks. BRAM is used to store the hidden states (h_t)of an LSTM cell, which can then be fed back into the NN at the next time step to maintain its memory. The structure of the system of an LSTM cell is shown in Figure 6-9, and the buffers, shown between stages, used in the implementation were synthesized as BRAMs on the chip. The number reported in the Table 6-1 is the number of BRAM blocks, which was the automatic result reported after the technology mapping step using Vivado. By using BRAM, the hidden state information can be stored in a dedicated memory block, separate from the other resources in the FPGA. This can lead to improved performance, as memory accesses are timing optimized and dedicated resources are used for memory storage. However, the size of the BRAM blocks and the memory requirements for the recurrent connections should be carefully scrutinized when designing an LSTM on an FPGA. The available BRAM resources may be limited, and it may be necessary to trade off memory size for performance, depending on the throughput requirements of the specific LSTM application. BRAM on the Versal FPGA device used here, has a maximum of 2 read and 2 write ports available per instance [104], limiting the number of neural cells which can simultaneously access the memory block. This can impact the pipelineing behaviour discussed previously, preventing pipeline stages operating in parallel and increasing latency. The resource usage of DSP slices, LUT, and FF in the biLSTM is shown in Table 6-1. This equates to 64% of DSP slices and 13% of LUTs and FFs in the Versal VC1902.

biLSTM implementation- Versal VC1902 resource usage (post fitter) and performance	
Logic Cells	113532 (13% of total available)
Single Bit logic registers	224386 (13% of total available)
DSP engines	1260 (64% of total available)
Memory Blocks (36Kb BRAM)	164
Clock Frequency (MHz)	270

Table 6-1 Post Fit resource utilisation biLSTM based equalizer.

The mapping of the biLSTM function onto the VC1902 device is show in Figure 6-12. The columnlike characteristic of the layout is related to the high number of DSP slices required in the design and the columnar architecture of the FPGA (see Figure 3-1).

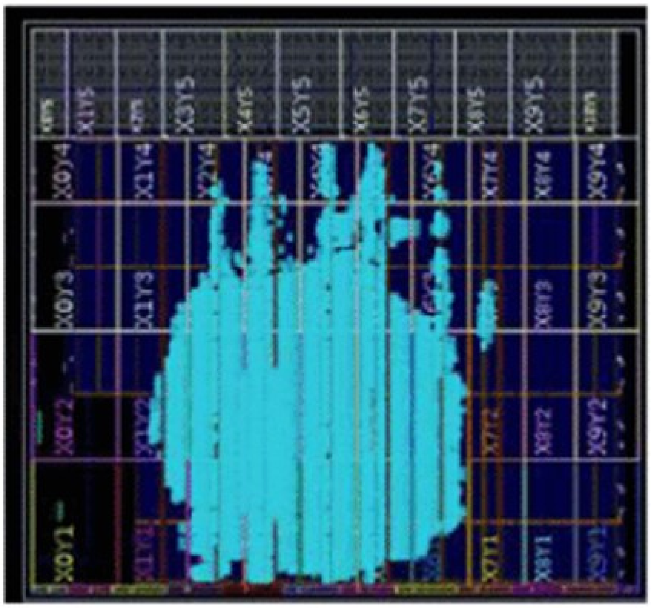


Figure 6-12 Design placement biLSTM based equaliser.

Next, in terms of throughput, the clock frequency is the maximum that the implementation can handle to comply with a zero-negative slack design. The feedback paths in the LSTM neuron cells caused a bottleneck in the design, limiting the clock speed.

The performance of this biLSTM based equaliser when addressing channel non-linearity has been analysed thoroughly, in the experiment described in [94], in comparison to a traditional one step per span digital backpropagation (DBP)solution, a deep convolutional neural network (CNN) solution and a DSP based Chromatic Dispersion Compensation (CDC) block. The Q-factor with respect to launch power for each solution was plotted as shown in Figure 6-13. The biLSTM based solution was determined to have given a 1.7 dB improvement in Q-factor when compared to a CDC solution, which was also better than the gains seen using a CNN or DBP solution. The HLS approach enabled the biLSTM implementation to be adapted quickly in order to address timing performance issues due to excessively long paths. From the utilisation figures reported in [94], comparing the biLSTM, CNN and CDC implementations, it is observed that, once the solution is scaled up to a throughput of 400 Gbit/s, the biLSTM only requires 2.5x the FPGA processing capacity when compared to a CDC.

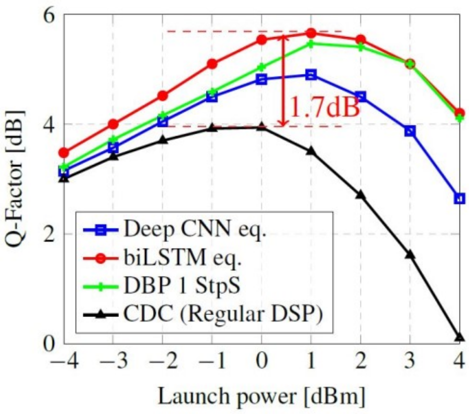


Figure 6-13 Q-factor versus launch power for the different types of equalizer (described in the legends) by experiment [94]

Finally, with regards to the design flow, the HLS method proved to be efficient and capable when converting a C++ model of an ANN into VHDL, allowing the initial model to be developed without extensive knowledge of the FPGA architecture. The ability to quickly experiment with loop architectures in order to minimise estimated latency was exceptionally beneficial even though a path timing violation, shown as negative slack, still occurred after layout. The downside to the HLS approach was that the model signal names did not persist transparently through the conversion process. When a timing problem was reported post layout the reverse engineering of that path to the original C++ model was frustrating but achievable.

This work concludes that NNs are an excellent subject for exploiting the benefits of HLS, as their nested architecture consisting of multiple layers and several multiply and accumulate functions can make good use of the loop and pipeline directives to investigate the trade-off between area and latency to meet the design requirements.

6.3 MZM and ML

One of the main tasks when characterising a Mach Zehnder modulator is the determination of the transfer function by finding Vpi. From this value, the initial positions of the principle operating points (defined in Section 4.2)can be estimated. The steps involved in the characterisation of an MZM are described in Section 2.2 and involve lengthy calculations which are normally done on a PC connected to a testbed. Following the example of 6.2 the question is could a ML algorithm be simpler to implement than a solution based on an embedded soft core microcontroller and small enough to be realisable in low cost hardware? Hence, enabling the task to be self contained on the optical transceiver. The key to Machine Learning is to be able to find features in the data which can be used to train the ANN model. There is a clear relationship between harmonics and the transfer function, as predicted by theory and demonstrated here in Chapter 4, which can potentially be exploited by a small ANN to identify features relating to the main operating points on the transfer function for characterisation.

Initial research in the published literature has shown a curve fitting NN is one possibility [105] [106] utilising a feedforward ANN with a small number of hidden layers. Following this approach, Figure 6-14 shows an attempt by a deep ANN to find the transfer function sinusoid in a set of noisy data taken from a bias sweep recorded during the experiment of section 4.4. The data recorded in the sweep and the applied bias voltage were both normalised to lie between 1.0 and -1.0, as shown in Appendix A-1 step 3, as part of the ANN training procedure. The data for this exercise was deliberately chosen to be a noisy dataset to test the ability of an ANN to find the underlying sinusoid amongst the noise. In this instance the response of the 2nd harmonic filter was used. As demonstrated in Figure 4-15 and Figure 4-16, the 2nd harmonic filter response is not as strong as that of the fundamental filter and does show a greater noise variation. In Figure 6-14 the noise is most apparent at the trace minima, which would correspond to the null point on the transfer function.

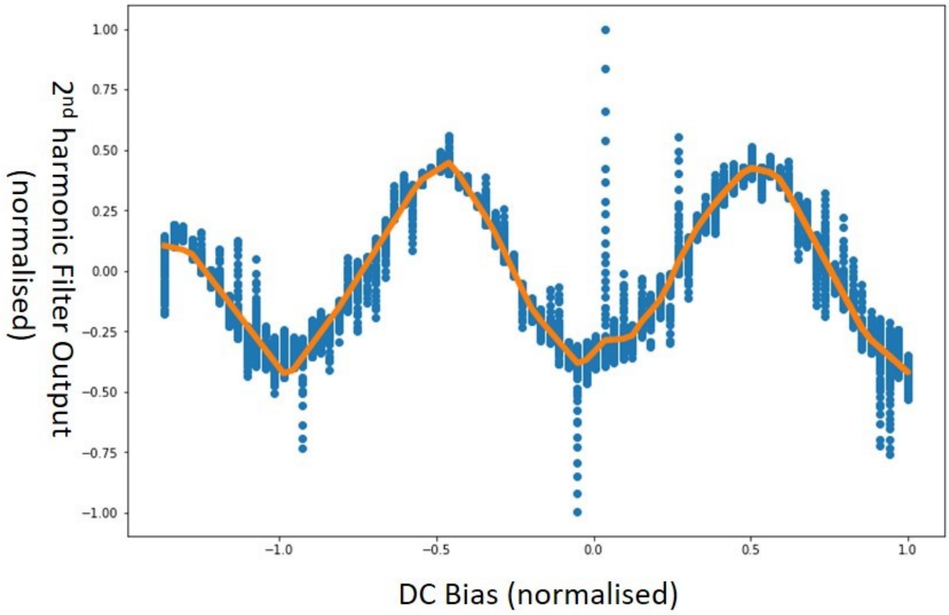


Figure 6-14 ANN curve fitting to a filter response obtained during the experiment of Section 4.

The ANN used to obtain this result was constructed with an Input layer with a single input, 2 densely connected hidden layers, each of 16 neurons using a ReLU activation function, and an output layer giving a single result as presented in Figure 6-15. The total number of trainable parameters i.e. weights and bias is shown in Table 6-2. The Python code used to generate and train this ANN is given in Appendix A.1.

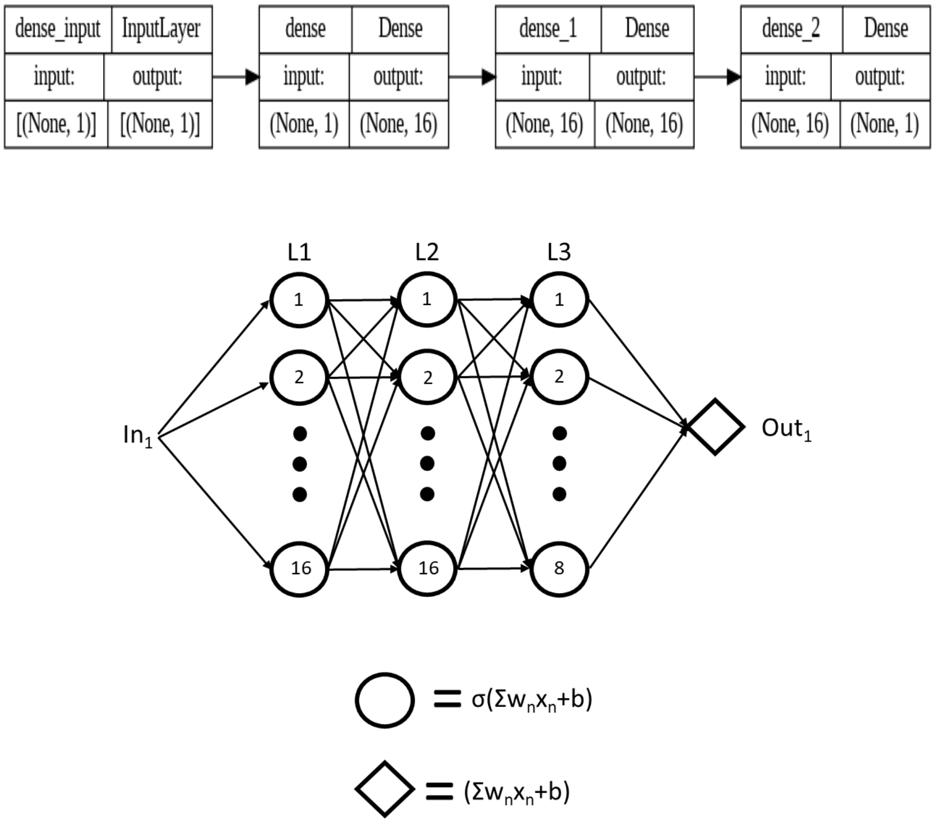


Figure 6-15 Visualisation of the ANN used to find the transfer function of Figure 6-14.

Layer (type)	Output Shape	Number of Parameters
dense (Dense)	(None: 16)	32
dense_1 (Dense)	(None: 16)	272
dense_2 (Dense)	(None: 1)	17
Total params:	321	
Trainable params:	321	
Non-trainable params:	0	

Table 6-2 Parameter Count of the ANN used to find the transfer function in Figure 6-14.

Training the ANN in this manner is unsupervised learning as previously defined in this chapter. The training data is not labelled and the ANN is finding the trend connecting the applied Bias voltage (X axis) and the corresponding filter output (Y axis) in the presented data. In the case being reported i.e. a model capable of finding Vπ in a manufacturing scenario, the model must be transferrable between modulators no need for retraining. The issue with this curve fitting approach is that the ANN is trained to look for a single relationship between Bias voltage and filter output, and as such this is not suitably transferrable between modulators.

As an alternative, finding Vπ for an MZM from a set of data can be considered very similar to identifying the period of a waveform working with bias voltage instead of time. This appears to be a better approach to the problem, one which is more likely to generate a transferable solution, so is the one which is explored here.

The ANN architecture was determined by experimentation and has an input layer of 82 input samples, matching the experimental data obtained in Section 4.4, connected to three densely connected hidden layers of eight neurons each using a ReLU activation function, before a final output layer gives a single numeric answer. Training data was generated using a python program (see Appendix A.2), using the equations determined in Section 4.2 a large number (1000) of filter response curves could be calculated for values of Vπ ranging from 2000 mV to 8000 mV in small but equal increments. Each filter response curve was labelled with its Vπ value as a supervised learning approach was to be adopted. For this data generation a V_{bias} step size equal to the step size used in the filter characterisation experiment of Section 4.4 was chosen giving 82 input values from a single simulated voltage sweep. The ANN was implemented in a Python Notebook using the Tensorflow Keras libraries and was trained using Google's Colab server.

During training it became apparent that the V_{bias} offset was a significant factor in obtaining a good, repeatable, result. The training data was then enhanced by adding another step where, for each Vπ value generated a series of 500 evenly spaced voltage offsets between -3000 mV and +3000mV were applied. This gave a total number of 500000 training waveforms.

The ability to determine the voltage offset of the transfer function from the ideal position is very useful when characterising an MZM, so another output was added to the ANN to give a voltage offset as well as Vπ as outputs and another label, for the offset, was added to the training data.

The final ANN used to predict the values of Vπ and Voffset was constructed with an input layer taking 82 inputs ,3 densely connected hidden layers of 8 neurons using a ReLU activation function in each layer and 1 densely connected output layer of 2 neurons returning values for Vπ and Voffset. The loss function used during training was Mean Square Error (MSE). This resulted in an ANN with 664 trainable parameters in the 1st hidden layer, 72 in 2nd hidden layer and also in the 3rd layer, then 18 in the final output layer. Giving a total of 826 Weights and biases as shown in Figure 6-16 and Table 6-3.

Once trained, the ANN model was saved and tested using real data obtained in the experiment of Section 4.4.

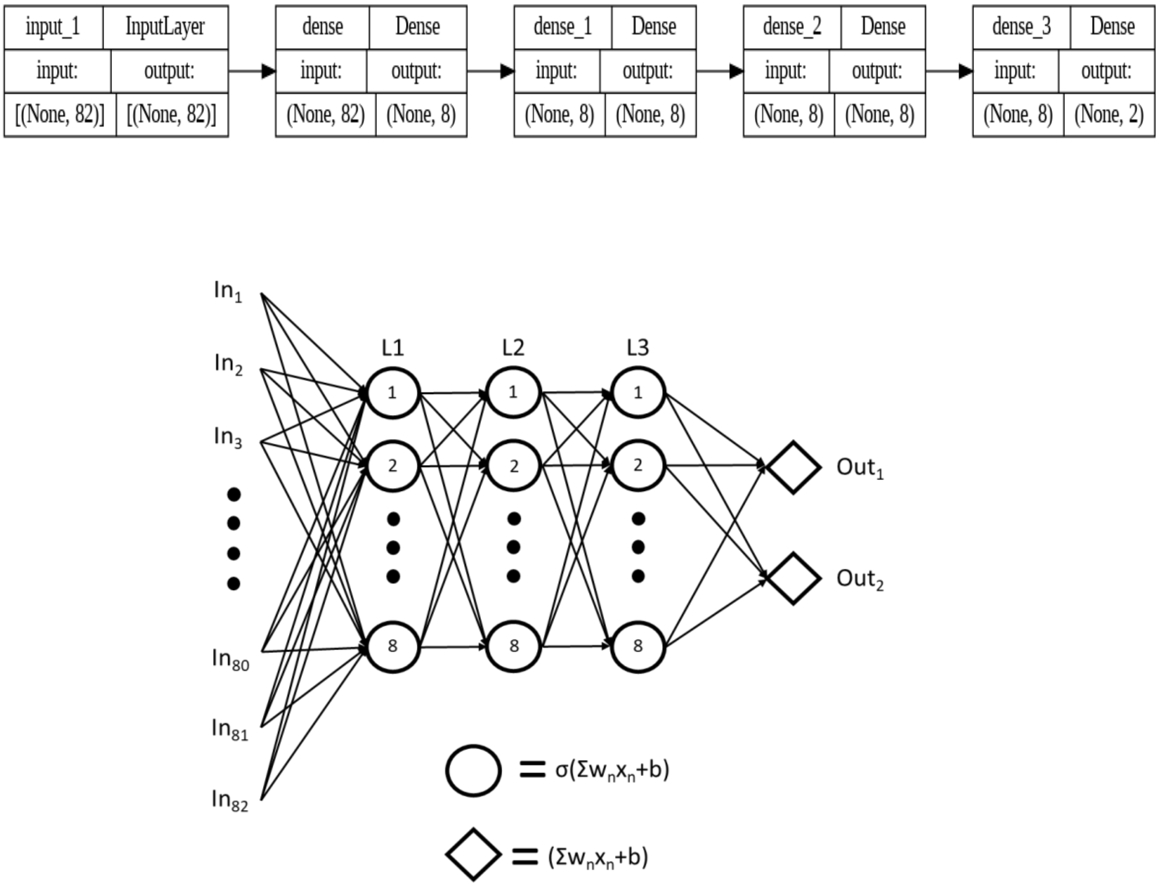


Figure 6-16 ANN structure used to Determine Vπ and Offset.

Layer (type)	Output Shape	Number of Parameters
input_1 (InputLayer)	(None: 82)	0
dense (Dense)	(None: 8)	664
dense_1 (Dense)	(None: 8)	72
dense_2 (Dense)	(None: 8)	72
dense_3 (Dense)	(None: 2)	18
Total Params		826
Trainable Params		826
Non-trainable Params		0

Table 6-3 Parameter count of the ANN used to determine Vπ and Offset.

6.3.1 Results

To test the behaviour of the trained neural network another set of data was generated and was input into the model to generate a prediction. A further two sets of data taken from the results of the experiment in Section 4.4 was also applied.

A summary of the results is presented in Table 6-4 and plotted in Figure 6-17. The plot shows that the trained ML model is performing the characterisation task with a good degree of accuracy. It is only at the higher values of Vπ that the performance drops significantly. This is most likely due to the limit on the upper bounds of the training data which was chosen.

Target Vπ (mV)	Target Offset (mV)	Predicted Vπ (mV)	Predicted Offset (mV)	Vπ Error (mV)	Offset Error (mV)
5220	2100	5213	2098	7	2
4400	-2700	4365	-2658	35	-42
3140	700	3113	770	27	-70
3140	-700	3182	-727	-42	27
3140	1700	3120	1690	20	10
3140	-1700	3080	-1708	60	8
1250	100	1268	111	-18	-11
7340	-1300	6950	-1166	390	-134
1250	200	1276	225	-26	-25
1000	0	1037	-130	-37	130
7000	0	6794	78	206	-78

Table 6-4 generated target values vs predicted testpoints with error (mV).

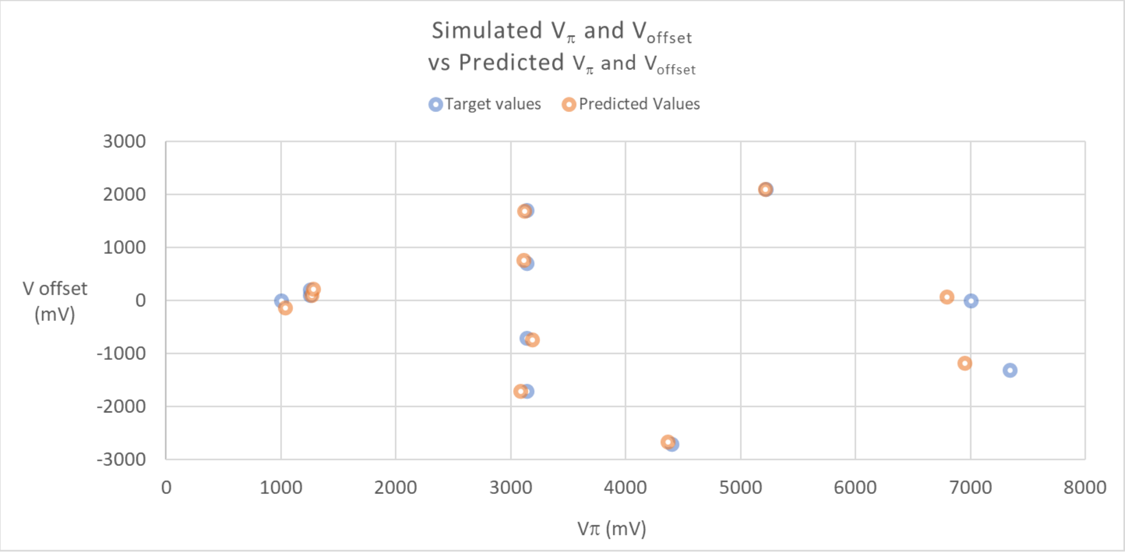


Figure 6-17 Plot of generated target vs predicted data points.

The recorded experimental data obtained from Section 4.4 is plotted in Figure 6-18 and Figure 6-19 below. The derived values in each case, for Vπ (≈ 4390 mV and ≈ 4150 mV) and the absolute value for V_{OFFSET} (≈2700 mV and ≈ 1720 mV), have been annotated onto the graphs. When the data was presented to the trained ANN model a Vπ of 4439 mV and V_{OFFSET} of -2768 mV were predicted for the first set of data and a Vπ of 4102 mV and V_{OFFSET} of -1836 mV were predicted for the first set (see Table 6-5). The negative value for V_{OFFSET} indicating that the transfer function was shifted to the right of the nominal position. The overall magnitude of the prediction errors is in line with the simulated test data, even in the presence of 'real world' noise on the filter output. The prediction errors recorded are all less than one DC bias 'sweep step' (250 mV) of the target value, which is an acceptable starting point for a subsequent bias point controller to begin managing the transfer function drift. A reduction in the error could, potentially, be achieved by reducing the sweep step size and gathering more datapoints from the calibration sweep. The cost of this would be twofold. Firstly, the complexity of the ML model would be increased and the number of trainable parameters would also increase. Halving the step size and gathering 164 input samples, while keeping the hidden layer structure the same, would increase the number of first layer trainable parameters from 664 to 1320. This is still substantially less memory than was required by the soft microcontroller demonstrated in Chapter 5. Secondly, the time required to perform the calculation would increase. This is the factor which affects the potential cost saving benefits of using a smarter transceiver for self-characterisation. The additional time required to take the extra samples is almost totally mitigated by keeping the data processing on the device. Following the conclusions of Chapter 4, a sweep of 164 DC bias values, using that experimental setup, would be completed in 0.8 seconds.

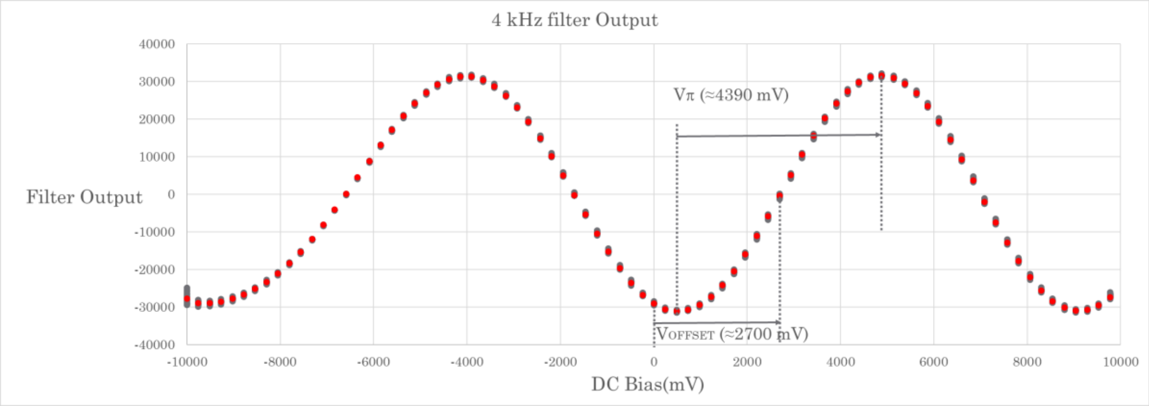


Figure 6-18 Experimental data used as input to ANN

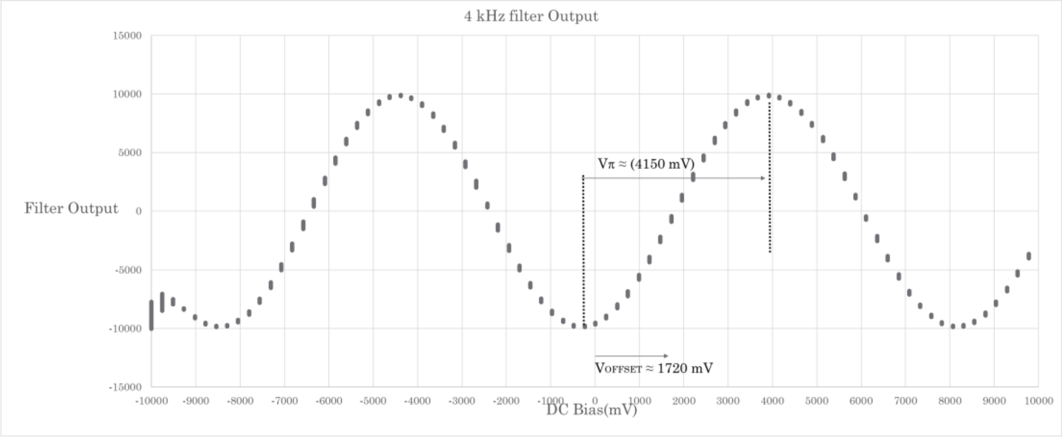


Figure 6-19 2nd experimental sweep data used as input to ANN

Target Vπ (mV)	Target Offset (mV)	Predicted Vπ (mV)	Predicted Offset (mV)	Vπ Error (mV)	Offset Error (mV)
4390	2700 (to right)	4439	-2768	-49	68
4150	1720 (to right	4102	-1836	48	116

Table 6-5 Experimental recorded values vs ANN predicted values

The developed ML solution to determining Vπ and V_{OFFSET} has been shown to be of significantly lower complexity than a soft embedded microcontroller capable of performing the same task, particularly when comparing the memory requirements. As demonstrated in Figure 5-5, the RAM requirements are substantially reduced from 592128 bits, used mostly for program space and operating system storage by the NIOSII , down to 13216 bits based on 826 x 16 bit implementations of the trainable parameters. This lower complexity and reduced memory requirement support the use of a cost-optimised, computationally limited FPGA.

While proceeding to the implementation step, following the steps set out in Section 6.2, two issues were found with the toolchain which prevented a successful implementation. While replicating the high-level design and simulation flow demonstrated during the creation of the biLSTM solution, the Intel HLS compiler was found to have been deprecated from the Quartus lite toolchain. As the target board used in the experiment of Section 4.4 used an Intel MAX10 device, no other solution was readily available. Section7.2 of this thesis presents some future options to progress. The quantisation and subsequent simulation of the Python model to use lower precision fixed point weights and biases was also impacted by unexpected errors from the Tensorflow lite tool. This prevented the generation of validated quantised weights and biases which could then have been used to manually code such a small NN.

6.4 Summary

Chapter 6 has provided an introduction to the developing field of Machine Learning and introduced some of the wide variety of applications using ML and also some of the concepts relating to the design and training of ML models. The push to move ML to the edge to reduce latency and the volume of data transmitted has been recognised. The suitability of FPGAs to implement ML has been shown by examining how an ANN can be efficiently mapped onto FPGA logic structures to implement the neurons of a trained inference model and where this can leverage the advantages of parallel processing in the FPGA.

The design flow of an ANN, from a high-level functional model to FPGA realisation, has been demonstrated, through the first time implementation of a biLSTM based equalizer for non-linear effects, the potential issues this can raise have been identified and ways of addressing them discussed.

Finally, in addressing the original concept of this thesis, the novel use of a small ANN to perform a characterisation task on an MZM, as a possible solution to the problem of increasing test time during manufacture of optical assemblies, has been examined. With the results presented showing an acceptable degree of accuracy – generally better than 2% of Vπ and within one bias voltage sweep step- even with real data with noise. The time saving comes from being able to calculate sufficiently accurate values for Vπ and V_{OFFSET} on-device, removing the bottleneck of passing the large quantity of acquired, unprocessed data off-device to an external computer for analysis. Furthermore, by showing that it is possible to embed the test intelligence in the module, and without needing external measurement equipment, it can be seen that it becomes possible to repeat tests away from the manufacturing factory, potentially without an operator being present.

7 CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis the need for creating smarter optical transceivers using FPGA technology has been established from a number of perspectives, starting from the wider pictures of both the increasing demands on optical capacity and the cost pressures of manufacturing, before focussing on the specific example of adding built-in characterisation intelligence to a transceiver using optical MZMs. ML has been demonstrated as a method of adding this intelligence through implementation in FPGAs.

As reported in Chapter 2, the nature of network traffic is changing and the pressures on bandwidth are increasing. This presents the double challenges of network flexibility and increased bandwidth. Higher bandwidth is now required further towards the network edge as connected devices and cloud based streaming services come to dominate the datasphere. The transceivers required to address these challenges are becoming more complex to build, featuring arrays of co-packaged components to enable densely packed multichannel optics to be created. The cost of packaging and testing these arrays of components is a significant portion of the overall device cost. Components must be tested and retested throughout the manufacturing process to eliminate failures at as early a stage as possible. Performing these tests automatically is of financial benefit.

The potential of FPGAs to offer a solution to increasing test times has been discussed in Chapter 3. Following an introduction to the technology of FPGAs and a look at the advantages they can offer over ASICs, through their reconfigurability and low development costs, and microcontrollers, through their parallel processing capabilities, the issues affecting manufacturing are examined in detail using the specific examples of determining a laser tuning map and characterising an MZM. By leveraging their reconfigurability and parallel behaviour the concept of creating reusable blocks for use during the manufacturing process, capable of reducing test times by performing operations in parallel and autonomously, is introduced.

Through the experimental work of Chapter 4, the use of low cost digital hardware to implement a circuit capable of monitoring the transfer function of a

Mach Zehnder Modulator has been confirmed. The small resource requirements of the Goertzel algorithm based filter, particularly when compared to an equivalent FIR or IIR digital filter, are a distinct advantage when analysing the overheads of adding monitoring functionality. Yet, this filter was shown to be able to accurately reflect the transfer function of the MZM through its fundamental and 2nd harmonic responses to a bias voltage sweep. Moving the control of the bias sweep function into the FPGA, alongside the filters, has been calculated to reduce this portion of the test activity from over 100 minutes to, in the order of, a few seconds.

Chapter 5 explored the, more usual, approach of implementing a small microcontroller using FPGA cells, which was capable of hosting an embedded Linux operating system and running tests through scripts. This showed that, although this is a usable route, the resources required to implement this simple microcontroller required an FPGA six times larger than the one used to implement the Goertzel filters of Chapter 4, adding cost overhead to an embedded self-test solution.

Chapter 6 introduced the adoption of ML techniques, specifically ANNs, as part of a low cost on device characterisation solution. ML is finding new applications at the system edge, being used to preprocess data in situ, in order to reduce latency in the decision making process and allow ML to operate at the end of low bandwidth connections. The use of FPGAs to implement trained ML models has been established in this thesis, both by showing how an ANN can be mapped onto FPGA cells and through the demonstration of a high level design flow to realise a bidirectional RNN, based on a biLSTM architecture, in an FPGA. The issues, relating to negative timing slack, which occurred in this experiment have been included in the discussion.

Across this thesis three first-time achievements have been reported, the first being the demonstration of the use of efficient Goertzel algorithm based filters as part of the monitoring of an MZM transfer function. The second is the implementation in FPGA technology of a biLSTM based equalizer for non-linear effects in optical communications systems and finally, the third is the use of a small size ANN to perform the Vπ and Voffset characterisation of an MZM optical component.

In conclusion, this thesis can be regarded to have proved the hypothesis put forward in Chapter 1.4 – FPGAs are a flexible tool which have been shown to have the potential to be used in optical transceivers to provide cost-effective and repurposable embedded test functions in optical transceivers, utilising Machine Learning techniques, enabling built-in test and characterisation activities to be performed.

7.2 Future Work

The question of how to address all possible characterisation tasks for the different, constituent, parts of an integrated optical module, and indeed which are the most pertinent of these tasks, is quite possibly an entire domain of research in itself. Machine Learning is currently an ever-expanding field of research as we seek out new problems that it can solve for us.

As has been seen in this thesis, the availability of large scale, real and varied data for training and validation of Machine Learning models can present a challenge. The greatest source of this is likely to be the module manufacturers themselves, however, this data can be seen as commercially sensitive and not readily shared. To progress the work started here, an agreement with an industrial partner would be highly advantageous.

The work contained in this thesis has demonstrated some of what is possible by using real time micro-scale Machine Learning engines as a part of real world solutions, but there is still some way to go in order to encourage uptake of this technique outside of research. First, additional ML model architectures need to be investigated to see where the most efficient solutions lie. For example, the use of the 2nd harmonic response as an additional input to the Vπ ANN has not yet been simulated. Alternatively, consider the case of key word recognition. Here a pictogram is built from the time varying frequency patterns of the phrase (Figure 7-1 [107]). The pictograms turn this from a frequency detection problem into an image recognition problem, with the frequency map of a given phrase becoming the training data for a Neural Network. Based on the results presented earlier using the simple Goertzel filter, a frequency map based on 1st and 2nd harmonics could be generated. Also, the use of RNN architectures can be investigated to assess their potential, such as the biLSTM, for detecting the high and low points of the filter response. Looking beyond the MZM, ML solutions to the laser tuning characterisation, as shown in Figure 3-6, would seem to be useful, particularly if they can then be actioned remotely, when the transceiver is in-service.

Finally, there remain outstanding questions on the implications of quantisation and the impact on quality of results. The migration of the trained model's weights and biases to 16 or 8 bit fixed point or integer representations and the use of input and output values also in the fixed point or integer form that these numbers would be represented in the FPGA is essential in order to achieve a practical implementation of the MZM characterisation. As reported in Section 6.3.1, the toolchain to support the HLS route to implement the design in the target Intel Max10 had become unavailable and a new experimental setup, targeting a different device (for example the Microchip IGLOO2) will provide a solution to this. While the ANN is small scale and has relatively few interconnections between neurons, the possibility of manually coding the ANN using a hardware description language such as VHDL remains as an approach to keep the established experimental setup. Verifying the quantised design using VHDL remains a challenge if manual coding is used, due to the relative slow speed of VHDL simulation compared with C/C++.

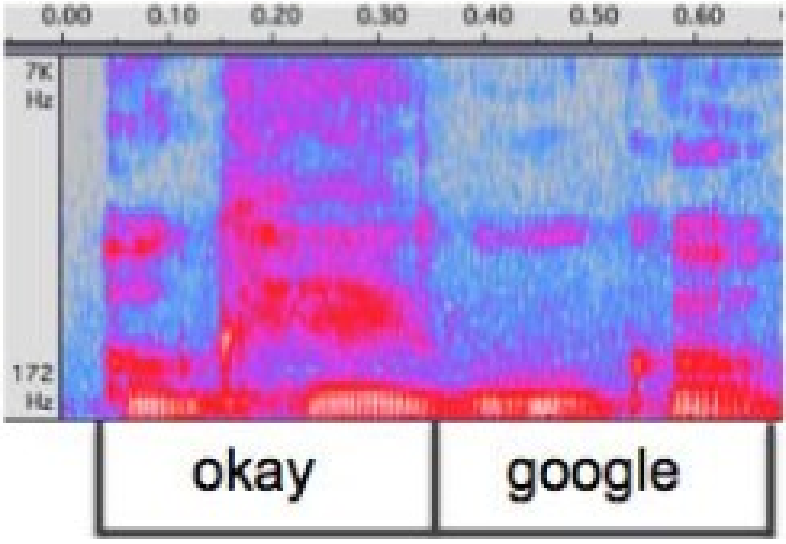


Figure 7-1 Example speech pictogram showing frequency magnitude vs time

Appendix A Tensorflow and Python code

A.1. Python code and steps used to generate curve fitting ANN.

1. Firstly, set up the Python environment by loading the relevant libraries. As the training was performed on Google Colab, a Google drive was also mounted and the dataset loaded from a file on this drive.

```
import numpy as np
import matplotlib.pyplot as plt
# Mount your Google Drive to access files in a folder
from google.colab import drive
drive.mount('/content/drive')
dataset=np.genfromtxt(r"/content/drive/My Drive/MZMLog_16 October 2019_112334.csv",delimiter=",")
# the leading r converts the path string to a raw string otherwise the / is seen as an escape character
# need to remove the initial message and date lines for this to work
```

2. Extract the relevant data from the columns of the dataset. Plot this data to confirm it is correct (Figure A-0-1).

```
train_x = dataset[1:,3].astype(np.float32)
train_y = dataset[1:,6].astype(np.float32)
train_x.shape
```

```
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(train_x, train_y, 'o') ;
plt.show()
```

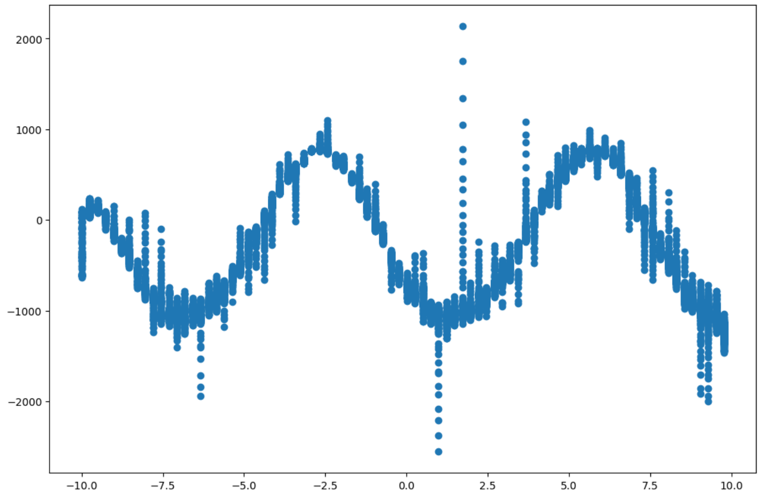


Figure A-0-1 Extracted training data for curve fitting ANN.

3. Normalize the data and replot to check it is still the same shape (Figure A-0-2)

```
train_x = train_x.reshape(-1,1)
train_y = train_y.reshape(-1,1)
train_x = train_x - np.mean(train_x, axis = 0)
train_x = train_x / np.max(train_x, axis = 0)
train_x.shape
train_y = train_y - np.mean(train_y, axis = 0)
train_y = train_y / np.max(train_y, axis = 0)
train_y.shape
```

```
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(train_x, train_y, 'o') ;
plt.show()
```

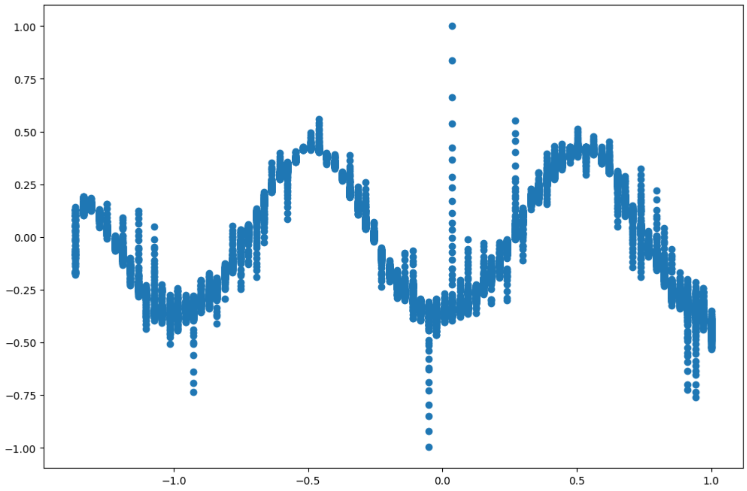


Figure A-0-2 Normalized training data for curve fitting ANN.

4. Import the Tensorflow libraries, specify the model, train with the model.fit command and plot the result (Figure A-0-3).

```
import tensorflow
from tensorflow.keras.models     import Sequential
from tensorflow.keras.layers     import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import plot_model

tensorflow.keras.backend.clear_session()
n_hidden = 16

model = Sequential()
model.add(Dense(input_dim=1, units=n_hidden, activation='relu'))
model.add(Dense(units=n_hidden, activation='relu'))
model.add(Dense(units=1))
model.summary()
model.compile(loss='mean_squared_error',
optimizer=SGD(learning_rate=0.01))

model.fit(train_x, train_y, epochs=20000, batch_size=m, verbose=0);

y = model.predict(train_x)
plt.plot(train_x, train_y, 'o')
plt.plot(train_x, y, linewidth=5)
plt.show()
```

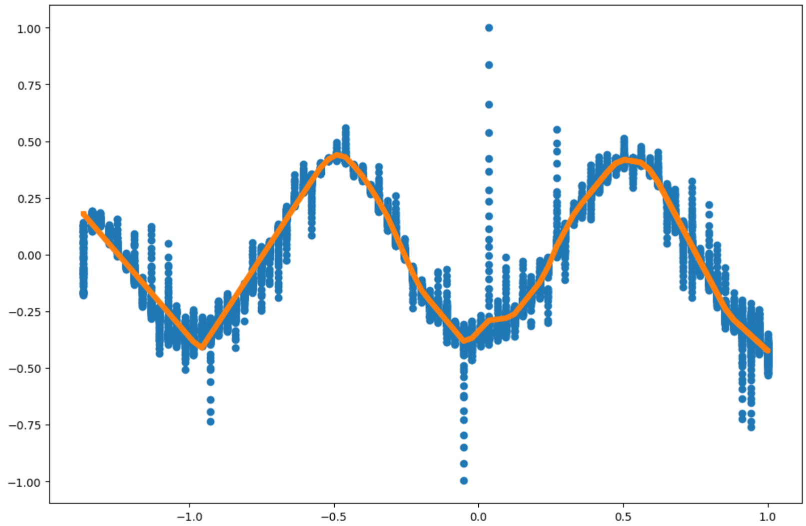


Figure A-0-3 predicted curve using trained ANN

A.2. Python code and steps used to generate ANN predicting Vπ and Voffset.

1. Firstly, set up the Python environment by loading the relevant libraries. As the training was performed on Google Colab, a Google drive was also mounted and the file paths set.

```
import pandas as pd
import numpy as np
from datetime import datetime
import tensorflow as tf
import scipy.special as sp
!pip install keras-visualizer
!pip install graphviz
!pip install pydot
!pip install pydotplus
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from keras_visualizer import visualizer
from tensorflow.keras.utils import plot_model

# Mount the Google Drive to access files in a folder
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
# Read the Excel file from the folder
filename1 = '/content/drive/My Drive/Mike/MZMLog_4 Apr_13hz bin85mVpilot
- 2nd no data sweep - 1530nm with avg.xlsx'
modelfile = '/content/drive/My Drive/Mike/MZM_Model'
```

2. Create a function to generate a set of test data from results recorded experimentally into an Excel spreadsheet. The column headings were used to identify the required data. The data is normalized using the 'max' normalization i.e. normalized value = value/max absolute value.

```
def data_generation(filename, v_pi, v_offset, frequency, n_bias_op=0):
    from sklearn import preprocessing
    df = pd.read_excel(filename, dtype={'Timestamp ': np.datetime64})
    # Get the header names of all columns
    header_names = df.columns
    Table_values = df.to_numpy()
    for i in range(len(Table_values[:,0])):
      Table_values[i,0] = np.datetime64(Table_values[i,0])
    index1 = np.where(header_names == ' Goertzel_Quad')
    index2 = np.where(header_names == ' Goertzel_Peak')
    index3 = np.where(header_names == ' Bias Loop Result')
    Goertzel_Quad = Table_values[:,index1].flatten()
    Goertzel_Peak = Table_values[:,index2].flatten()
    # if using the fundamental frequency 4kHz in this case
    if frequency == 4:
         x
=np.copy(preprocessing.normalize([Goertzel_Peak],norm='max'))[0]
    else:
         x
=np.copy(preprocessing.normalize([Goertzel_Quad],norm='max'))[0]

    Bias = Table_values[:,index3].flatten()
    Bias_values = np.unique(Bias)
    if n_bias_op==0:
       n_bias = int(len(Bias_values))
    else:
      n_bias = n_bias_op
    n_bias_ranges = np.zeros(n_bias)
    for i in range(n_bias):
      n_bias_ranges[i]= np.sum(Bias == Bias_values[i])
    data_size = int(np.min(n_bias_ranges))
    print (data_size)
    data_input = np.zeros([data_size,n_bias])
    data_label = np.ones([data_size,2])
    temp=0
    for j in range(n_bias):
        for i in range(data_size):
            data_input[i,j] = x[int(temp+i)]
            data_label[i,(0)] = v_pi
            data_label[i,(1)] = v_offset
        temp += n_bias_ranges[j]
    data_input.shape
    return data_input, data_label
```

3. Create a function to generate a set of samples from a given Vpi and Voffset. This is used to create test cases for validation.

```
def test_stimulus(Vpi, VOffset):
  from sklearn import preprocessing
  import random
  a = 1  ## insertion loss set to 1 for lossless
  Pin = 10000000  ## power in, use as scaling factor to get min/max

  DivScale = 9.5  ## dividing scaling factor to reduce magnitude of
Fundamental curve. Mimics a divisor in the FPGA design

  Stim_StepSize = 50  ## step between adjacent steps sizes when Sweeping
DC Bias DAC
  Vac = 85  ## pk to pk amplitude of the pilot tone in mV
  Stim_n_sample = 50  ## 50 samples at each V
  stim_x = np.arange(0,4100, Stim_StepSize)   ## use this to generate x =
DAC values with Stepsize between. the max value isn't included. This
generates values between 0 and 4050 inclusive at 50 increments
  stim_V = ((((stim_x*20.0)/4096.0)-10.0)*1000)  ## in mV
  stim_length_x = len(stim_x)
  data_input_stim = np.empty((stim_length_x,Stim_n_sample))
  data_label_stim = np.empty((1,2))
  temp_stim = np.empty((1,Stim_n_sample))

## These are the fundamental equations which give the harmonics
  for _j in range(stim_length_x):
    stim_Vdc = stim_V[_j] + VOffset
   ## generate an array of random float between these 2 values to use as
a noise multiplier. Use 1.0 and 1.0 for no noise
    Noise = np.array([random.uniform(1.0, 1.0) for _i in range
(Stim_n_sample)])
    stim_func1 = a*Pin*np.sin(np.pi*(stim_Vdc)/Vpi)
    stim_func2 = (Vac/Vpi)*np.pi
    stim_func3 = a*Pin*np.cos(np.pi*(stim_Vdc)/Vpi)
    data_label_stim[(0),(0)] = Vpi/1000   ## in mV
    data_label_stim[(0),(1)] = VOffset/1000   ## in mv
    # only deriving fundamental at the moment
    temp_stim = (Noise*((stim_func1*sp.jv(1, stim_func2))/DivScale))
    data_input_stim[_j:] = temp_stim

  data_input_stim_transpose=np.transpose(data_input_stim)
  data_input_stim_transpose.reshape(Stim_n_sample,stim_length_x)
  for _k in range (Stim_n_sample):
    data_input_stim_transpose.shape
    data_input_stim_transpose[_k] =
np.copy(preprocessing.normalize([data_input_stim_transpose[_k]],norm='max
'))[0]


  data_label_stim.shape
  data_input_stim_transpose.shape
  return data_input_stim_transpose, data_label_stim
```

4. Create a function to generate training data from the fundamental equations describing the harmonic response of the MZM transfer function.

```
# plot the first 2 harmonics using the derived equations
from sklearn import preprocessing
a = 1  ## insertion loss set to 1 for lossless
Pin = 10000000  ## power in, use as scaling factor to get min/max

DivScale = 9.5  ## dividing scaling factor to reduce magnitude of
Fundamental curve. Mimics a divisor in the FPGA design
StepSize = 50  ## step between adjacent steps sizes when Sweeping DC Bias
DAC
Vac = 85  ## pk to pk amplitude of the pilot tone in mV

x = np.arange(0,4100, StepSize)   ## use this to generate x = DAC values
with Stepsize between. the max value isn't included

V = ((((x*20.0)/4096.0)-10.0)*1000) ## in mV
y = np.linspace(1000,8000,1000)  # use this to generate multiple filter
responses. Y will be Vpi in mV linspace(a,b,c) creates c evenly
distributed values between b and c
z = np.linspace(-3000,3000,500)  # z will be Vdc offset in mV
length_x = len(x)
length_y = len(y)
length_z = len(z)

data_input_train = np.empty(((length_y*length_z),length_x))
data_label_train = np.empty(((length_y*length_z),2))
```

```
## These are the fundamental equations which give the harmonics
for i in range(length_y):
    Vpi = y[i]
    for j in range(length_z):
       Vdc = V + z[j]
       func1 = a*Pin*np.sin(np.pi*(Vdc)/Vpi)
       func2 = (Vac/Vpi)*np.pi
       func3 = a*Pin*np.cos(np.pi*(Vdc)/Vpi)
       data_label_train[((i*length_z)+j),(0)] = Vpi/1000   ## in mV
       data_label_train[((i*length_z)+j),(1)] = z[j]/1000   ## in mv

    # only deriving fundamental at the moment
       temp = ((func1*sp.jv(1, func2))/DivScale)
       data_input_train[((i*length_z)+j)] =
np.copy(preprocessing.normalize([temp],norm='max'))[0]

data_label_train.shape
data_input_train.shape

#aiming to have 500000 sets of 82 normalized numbers representing filter
response curves
```

5. Define the ANN.

```
tf.compat.v1.reset_default_graph()


def model(input_shape1):
    inputs = tf.keras.layers.Input(input_shape1)

    x = tf.keras.layers.Dense(8, 'relu')(inputs)
    x = tf.keras.layers.Dense(8, 'relu')(x)
    x = tf.keras.layers.Dense(8, 'relu')(x)

    output = tf.keras.layers.Dense(2)(x)

    model_created = tf.keras.models.Model(inputs=[inputs],
outputs=[output])

    model_created.compile(loss=["mse"],
optimizer=tf.keras.optimizers.Adam(0.001),

                          metrics=["accuracy"])

    return model_created
```

```
Model = model((data_input_train.shape[1],))
Model.summary()
plot_model(Model, rankdir='LR', show_shapes='true')
```

6. Train the ANN. Plot the loss figure during training to look for overfitting (Figure A-0-4)

```
# %% Training NN Model
%%time
History = Model.fit(
    data_input_train, data_label_train,
    validation_split=0.2,
    batch_size = 1024,
    verbose=0, epochs=100)
```

```
hist = pd.DataFrame(History.history)
hist['epoch'] = History.epoch
hist.tail()
```

```
# Draw a graph of the loss, which is the distance between
# the predicted and actual values during training and validation.
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 10])
  plt.xlabel('Epoch')
  plt.ylabel('Error')
  plt.legend()
  plt.grid(True)
```

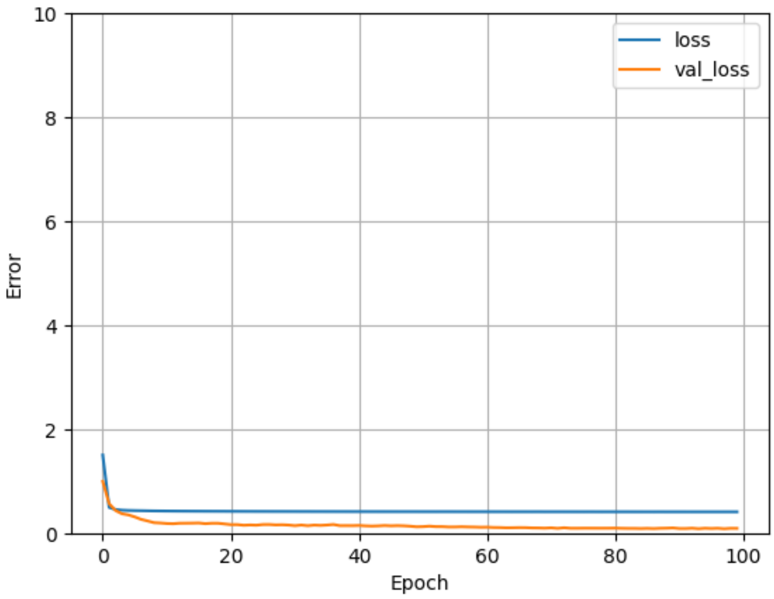


Figure A-0-4 Reduction of Loss parameter during ANN training

7. Save the Model for future reuse

```
Model.save(modelfile)
```

8. Generate a set of validation test data and plot its values (Figure A-0-5), then use this data to test the trained model

```
stim_input_test, stim_label_test = test_stimulus(Vpi=3140, VOffset=700)
prediction_v_pi = Model.predict(stim_input_test, verbose=0)
print('V_pi_predicted test', np.mean(prediction_v_pi[0,0]),
np.mean(prediction_v_pi[0,1]) ,'V_pi_label test',
np.mean(stim_label_test[0,0]), np.mean(stim_label_test[0,1]) )
plt.plot(stim_input_test[0])
```

```
[<matplotlib.lines.Line2D at 0x7950967d1ae0>]
```

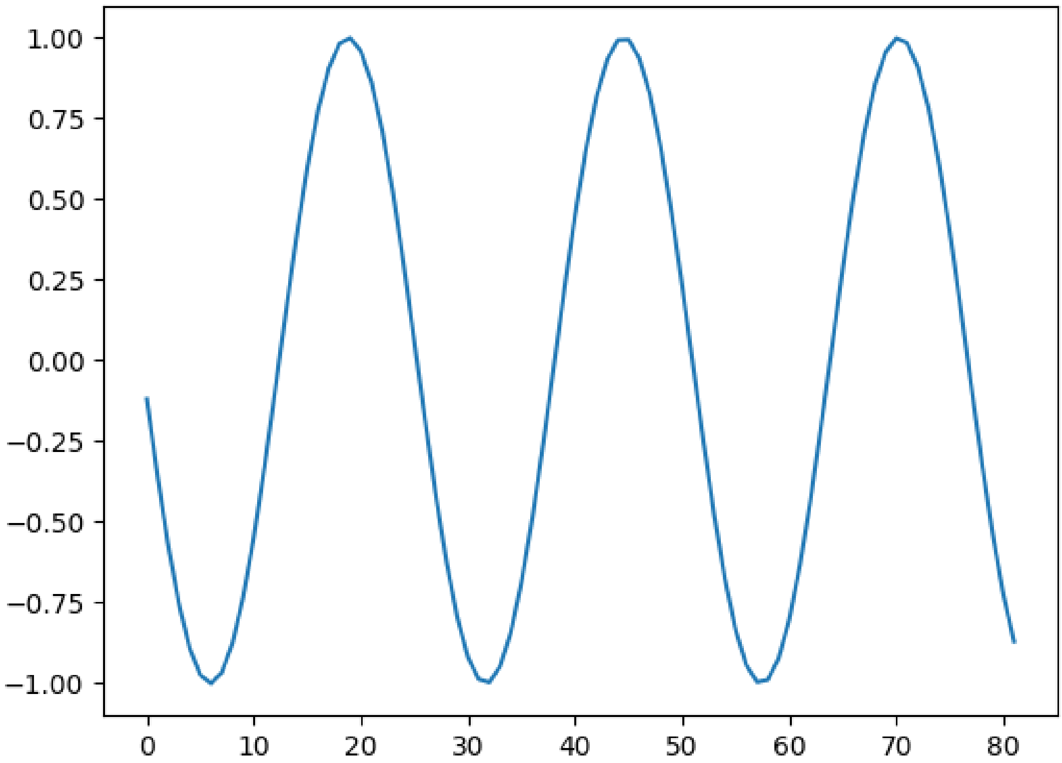


Figure A-0-5 Example generated validation test data.

```
V_pi_predicted test 3.103526 0.6494898 V_pi_label test 3.14 0.7
```

8 REFERENCES

[1] C. Minkenberg, R. Krishnaswamy, A. Zilkie and D. Nelson, “Co-packaged datacenter optics: Opportunities and challenges,” *IET Optoelectronics,* vol. 15, no. 2, pp. 77-99, 2021.

[2] T. Kobayashi, J. Cho, M. Lamponi, G. De Valicourt and C. R. Doerr, “Coherent Optical Transceivers Scaling and Integration Challenges,” *Proceedings of the IEEE,* vol. 110, no. 11, pp. 1679-1698, November 2022.

[3] R. T. Logan and D. Basuita, “Space-grade 1–10 Gbps parallel optical transceivers and fibre optic connectors for SpaceFibre datalinks,” in *2022 International SpaceWire & SpaceFibre Conference (ISC)*, Pisa, Italy, 2022.

[4] E. Turner and D. Law, “IEEE P802.3ae MDC/MDIO,” September 2001. [Online]. Available: https://www.ieee802.org/3/efm/public/sep01/turner_1_0901.pdf. [Accessed 18 July 2023].

[5] IEEE, “Adopted IEEE P802.3df Timeline (04 Oct 2022),” 4 October 2022. [Online]. Available: https://www.ieee802.org/3/df/proj_doc/timeline_3df_221004.pdf. [Accessed 15 November 2023].

[6] IEEE, “Adopted IEEE P802.3dj Timeline (16 Jan 2023),” 16 January 2023. [Online]. Available: https://www.ieee802.org/3/dj/projdoc/timeline_3dj_230116.pdf. [Accessed 15 November 2023].

[7] 800G Pluggable MSA, “800G-FR4 Technical Specification,” 8 June 2021. [Online]. Available: https://static.s123-cdn-static-d.com/uploads/2598123/normal_60bf88b173c87.pdf. [Accessed 2 November 2023].

[8] 800G Pluggable MSA, “800G PSM8 100M SPECIFICATION V1.0,” 28 August 2020. [Online]. Available: https://static.s123-cdn-static-d.com/uploads/2598123/normal_5f50dfed42c1e.pdf. [Accessed 2 November 2023].

[9] The Ethernet Alliance, “2023 Ethernet Roadmap,” February 2023. [Online]. Available: https://ethernetalliance.org/wp-content/uploads/2023/03/EthernetRoadmap-2023-Website-REV-March-17.pdf. [Accessed 14 November 2023].

[10] Intel Corporation, “MAX® 10 FPGA Device Overview,” 14 June 2022. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683658/current/fpga-device-overview.html. [Accessed 8 May 2024].

[11] P. J. Winzer and D. T. Neilson, “From Scaling Disparities to Integrated Parallelism: A Decathlon for a Decade,” *Journal of Lightwave Technology,* vol. 35, no. 5, pp. 1099-1115, 2017.

[12] R. Tkach, “Scaling optical communications for the next decade and beyond,” *Bell Labs Technical Journal,* vol. 14, no. 4, pp. 3-9, 2010.

[13] Cisco Systems Inc., “Cisco Visual Networking Index: Forecast and Methodology, 2016 - 2021,” September 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html. [Accessed January 2018].

[14] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang and J. Wang, “A first look at cellular machine-to-machine traffic: large scale measurement and characterization,” in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, New York, 2012.

[15] IEEE, “IEEE P802.3bs 400GbE Adopted Timeline,” September 2015. [Online]. Available: https://www.ieee802.org/3/bs/timeline_3bs_0915.pdf. [Accessed Sepetember 2023].

[16] W. Klaus, P. J. Winzer and K. Nakajima, “The Role of Parallelism in the Evolution of Optical Fiber Communication Systems,” *Proceedings of the IEEE,* vol. 110, no. 11, pp. 1619-1654, 2022.

[17] P. Winzer, D. Neilson and A. Chraplyvy, “Fiber-optic transmission and networking: the previous 20 and the next 20 years,” *Optics Express,* vol. 26, no. 18, pp. 24190-24239, 2018.

[18] Acacia Communications Inc., “Acacia Unveils Industry’s First Single Carrier 1.2T Multi-Haul Pluggable Module,” 23 September 2021. [Online]. Available: https://acacia-inc.com/blog/acacia-unveils-industrys-first-single-carrier-1-2t-multi-haul-pluggable-module/. [Accessed 29 October 2023].

[19] A. Lord, S. J. Savory, M. Tornatore and A. Mitra, “Flexible Technologies to Increase Optical Network Capacity,” *Proceedings of the IEEE,* vol. 110, no. 11, pp. 1714-1724, 2022.

[20] Cisco Systems Inc., “Cisco Annual Internet Report (2018–2023) White Paper,” 9 March 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed 1 November 2023].

[21] Nokia, “Global Network Traffic 2030 Report,” 2023. [Online]. Available: https://onestore.nokia.com/asset/213660. [Accessed 10 May 2024].

[22] International Data Corporation, “Data Age 2025,” April 2017. [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf. [Accessed November 2023].

[23] S. Chandrasekhar, X. Liu, B. Zhu and D. Peckham, “Transmission of a 1.2-Tb/s 24-carrier no-guard-interval coherent OFDM superchannel over 7200-km of ultra-large-area fiber,” in *35th European Conference on Optical Communication*, Vienna, Austria, 2009.

[24] A. A. M. Saleh and J. M. Simmons, “Technology and architecture to enable the explosive growth of the internet,” *IEEE Communications Magazine, vol.49, no.1,* pp. 126-132, January 2011.

[25] T. Zami, “Current and Future Flexible Wavelength Routing Cross-Connects,” *Bell Labs Technical Jouranl,* vol. 18, no. 3, pp. 23-38, 2013.

[26] J. M. Fabrega, M. S. Moreolo, L. Martín, A. C. Piat, E. Riccardi, D. Roccato, N. Sambo, F. Cugini, L. Potì, S. Yan, E. Hugues Salas, D. Simeonidou, M. Gunkel, R. Palmer, S. Fedderwitz, D. Rafique and T. W. H. D. &. N. A. Rahman, “On the filter narrowing issues in elastic optical networks,”

Journal of Optical Communications and Networking, vol. 8, no. 7, pp. A23-A33, 2016.

[27] P. J. Winzer, "Energy-Efficient Optical Transport Capacity Scaling Through Spatial Multiplexing," *IEEE Photonics Technology Letters,* vol. 23, no. 13, pp. 851-853, 2011.

[28] P. J. Winzer, "Modulation and multiplexing in optical communication systems," *IEEE LEOS Newsletter,* pp. 4-8, February 2009.

[29] D. J. Richardson, J. Fini and L. E. Nelson, "Space-division multiplexing in optical fibres," *Nature Photonics,* vol. 7, pp. 354-362, 2013.

[30] P. J. Winzer, "Spatial multiplexing in fiber optics: The 10x scaling of metro/core capacities," *Bell Labs Technical Journal,* vol. 19, pp. 22-30, 2014.

[31] C. Laperle and M. O'Sullivan, "Advances in High-Speed DACs, ADCs, and DSP for Optical Coherent Transceivers," *Journal of Lightwave Technology,* vol. 32, no. 4, pp. 629-643, 2014.

[32] A. D. Shiner, M. Reimer, A. Borowiec, S. Oveis Gharan, J. Gaudette, P. Mehta, D. Charlton, K. Roberts and M. O'Sullivan, "Demonstration of an 8-dimensional modulation format with reduced inter-channel nonlinearities in a polarization multiplexed coherent system," *Optics Express,* vol. 22, no. 17, pp. 20366-20374, 2014.

[33] Q. Zhuge, M. Reimer, A. Borowiec, M. O'Sullivan and D. V. Plant, "Aggressive quantization on perturbation coefficients for nonlinear pre-distortion," in *OFC 2014 Th4D.7.*, San Francisco, 2014.

[34] G. Bennett, "Superchannels to the rescue!," *Lightwave,* vol. 29, no. 2, 2012.

[35] Bell Labs, "Bell Labs Metro Network Traffic Growth: An Architecture Impact Study," 2012. [Online]. Available: https://docplayer.net/10745078-Bell-labs-metro-network-traffic-growth-an-architecture-impact-study.html. [Accessed 6 November 2023].

[36] E. Riccardi, A. Pagano, E. Hugues-Salas, G. Zervas, D. Simeonidou, M. Bohn, A. Napoli, D. Rafique, A. D'Errico, N. Sambo, P. Castoldi, T. Rahman, M. Moreolo, J. Fàbrega and M. Gunkel, "Sliceable Bandwidth Variable

Transponder: The IDEALIST Vision," in *2015 European Conference on Networks and Communications (EuCNC)*, Paris, 2015.

[37] K. Shirahata, T. Mizushima, T. Fujibe, H. Matsumura, T. Itakura, M. Ishida, D. Watanabe and S. Masuda, "An Optical Interconnection Test Method Applicable to 100-Gb/s Transceivers Using an ATE Based Hardware," in *IEEE 25th Asian Test Symposium*, Hiroshima, Japan, 2016.

[38] S. C. Heck, S. K. Jones, R. A. Griffin, N. Whitbread, P. Bromley, G. Kate Harris, D. Smith, L. N. Lloyd N. Langley and T. Goodall, "Miniaturized InP dual I&Q mach Zehnder modulator with full monitoring functionality for CFP2.," in *The European Conference on Optical Communication (ECOC)*, Cannes, France, 2014.

[39] A. J. Ward, V. Hill, R. Cush, S. C. Heck, P. Firth and Y. Honzawa, "Monolithic integration of AlInGaAs DS-DBR tunable laser and AlInGaAs MZ modulator with small footprint, low power dissipation and long-haul 10Gb/s performance," in *39th European Conference and Exhibition on Optical Communication (ECOC 2013)*, London, 2013.

[40] G. Brebner, "Optical Fiber Communications Conference and Exhibition (OFC)," in *Programmable hardware for high performance SDN*, Los Angeles, 2015.

[41] P. Biswas, "Introduction to FPGA and its Architecture," 20 November 2019. [Online]. Available: https://towardsdatascience.com/introduction-to-fpga-and-its-architecture-20a62c14421c. [Accessed 11 October 2023].

[42] Intel Corp., "Intel® Stratix® 10 FPGA and SoC FPGA," [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10.html. [Accessed 25 October 2023].

[43] Intel Corp., "Cyclone® V FPGA and SoC FPFA Family Overview Product Table," 25 9 2018. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/714207/cyclone-v-fpga-and-soc-fpfa-family-overview-product-table.html?wapkw=FPGA%20soc%20family%20comparison. [Accessed 20 8 2023].

[44] Intel Corp., "Intel® Arria® 10 FPGA and SoC FPGA Family Overview Product Table," 1 March 2019. [Online]. Available:

https://www.intel.com/content/www/us/en/content-details/714167/intel-arria-10-fpga-and-soc-fpga-family-overview-product-table.html?wapkw=FPGA%20soc%20family%20comparison. [Accessed 20 October 2023].

[45] Intel Corp., “Intel® Stratix® 10 GX FPGA and Intel® Stratix® 10 SX SoC FPGA Family Overview Product Table,” 1 October 2020. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/652478/intel-stratix-10-gx-fpga-and-intel-stratix-10-sx-soc-fpga-family-overview-product-table.html?wapkw=FPGA%20soc%20family%20comparison. [Accessed 20 October 2023].

[46] P. J. Quinn, “FPGA based silicon innovation exploiting “More than Moore” technology,” in *Proceedings of the 2013 9th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, Villach, 2013.

[47] S. Randel, P. Matalla and M. A. Hossain, *Upcoming Challenges in Signal Processing for Optical Fiber Communications,* Glasgow, 2023, p. @1:25:23.

[48] J. Hruska, “As Chip Design Costs Skyrocket, 3nm Process Node Is in Jeopardy,” 22 June 2018. [Online]. Available: https://www.extremetech.com/computing/272096-3nm-process-node. [Accessed 15 October 2023].

[49] Intel Corp., “Intel® Arria® 10 FPGA and SoC FPGA,” [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/arria/10/article.html. [Accessed 25 October 2023].

[50] Intel Corportion, “Agilex™ 9 SoC FPGA Direct RF-Series,” [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/agilex/9/direct-rf-series.html. [Accessed 9 May 2024].

[51] AMD Inc., “Zynq UltraScale+ RFSoC,” [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/rfsoc.html. [Accessed 21 September 2023].

[52] MicroChip Technology Inc, “SmartFusion® FPGAs,” [Online]. Available: https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/smartfusion-fpgas#overview. [Accessed 29 June 2023].

[53] Intel Corp., “Intel® Stratix® 10 SX SoC FPGA,” [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10/sx.html. [Accessed 9 October 2023].

[54] Altera Corp., “What is an SoC FPGA,” [Online]. Available: https://www.altera.com/en_US/pdfs/literature/ab/ab1_soc_fpga.pdf. [Accessed October 2018].

[55] A. Hassan, R. Ahmed, H. Mostafa, H. A. H. Fahmy and A. Hussien, “Performance Evaluation of Dynamic Partial Reconfiguration Techniques for Software Defined Radio on an FPGA,” in *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Cairo, 2015.

[56] J. Huang, M. Parris, J. Lee and R. F. Demara, “Scalable FPGA-based architecture for DCT computation using dynamic partial reconfiguration,” *ACM Transactions on Embedded Computing Systems Vol.9 Iss.1,* p. Article No.9, October 2009.

[57] S. Yan, Y. Yan, B. R. Rofoee, Y. Shu, E. Hugues-Salas, G. Zervas and D. Simeonidou, “Demonstration of Real-Time Ethernet to Reconfigurable Superchannel Data Transport over Elastic Optical Network,” in *ECOC, Mo 4.2.3*, Cannes, 2014.

[58] S. Yan, Y. Yan, B. R. Rofoee, Y. Shu, E. Hugues-Salas, G. Zervas and D. Simeonidou, “Real-Time Ethernet to Software-Defined Sliceable Superchannel Transponder,” *Journal of Lightwave Technology,* vol. 33, no. 8, pp. 1571-1577, 2015.

[59] International Telecommunications Union (ITU-T), *G.709/Y.1331,* 2016.

[60] Kish, F. A., Lal, V., Evans, P., Corzine, S., Ziari, M. et al, “System-on-Chip Photonic Integrated Circuits,” *IEEE Journal of Selected Topics in Quantum Electronics,* vol. 24, no. 1, 2018.

[61] A. Cicuttin, M. L. Crespo, A. Shapiro and N. Abdallah, “A Block-Based Open Source Approach for a Reconfigurable Virtual Instrumentation Platform Using FPGA Technology,” in *IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006)*, San Luis Potosi, Mexico, 2006.

[62] S. Blazo, “Tunable laser calibration system”. US Patent US20030132375A1, 17 July 2002.

[63] Amin, R., Maiti, R., George, J. K., Ma, X. et al., "A Lateral MOS-Capacitor-Enabled ITO Mach–Zehnder Modulator for Beam Steering," *Journal of Lightwave Technology,* vol. 38, no. 2, pp. 282-290, 2020.

[64] C.-H. Park, M.-K. Woo, B.-K. Park, S.-W. Jeon, H. Jung, S. Kim and S.-W. Han, "Experimental Demonstration of an Efficient Mach–Zehnder Modulator Bias Control for Quantum Key Distribution Systems," *Electronics,* vol. 11, p. Article 2207, 2022.

[65] E. Ackerman, S. Wanuga, D. Kasemset, A. S. Daryoush and N. R. Samant, "Maximum dynamic range operation of a microwave external modulation fiber-optic link," *IEEE Transactions on Microwave Theory and Techniques,* vol. 41, no. 8, pp. 1299-1306, 1993.

[66] J. Svarny, "Bias driver of the Mach-Zehnder intensity electro-optic modulator, based on harmonic analysis," in *Advances in Robotics, Mechatronics and Circuits*, Wseas LLC. ISBN: 978-1-61804-242-2, 2014, pp. 184-189.

[67] A. Hilt, "Microwave harmonic generation in fiber-optical links," in *13th International Conference on Microwaves, Radar and Wireless Communications. MIKON - 2000*, Wroclaw, Poland, 2000.

[68] M. Sotoodeh, Y. Beaulieu, J. Harley and D. L. McGhan, "Modulator Bias and Optical Power Control of Optical Complex E-Field Modulators," *Journal of Lightwave Technology,* vol. 29, no. 15, pp. 2235-2248, 2011.

[69] The SciPy Community, "scipy.special.jv," [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.jv.html#scipy.special.jv. [Accessed 2 June 2023].

[70] E. R. H. C. C. Ackerman, "Bias Controllers for External Modulators in Fibre Optic Systems," Lightwave, 2001.

[71] P. S. Cho and M. Nazarathy, "Bias Control for Optical OFDM Transmitters," *IEEE Photonics Technology Letters,* vol. 22, no. 14, pp. 1030-1032, 2010.

[72] M. H. Kim, B. M. Yu and W. Y. Choi, "A Mach-Zehnder Modulator Bias Controller Based on OMA and Average Power Monitoring," *IEEE Photonics Technology Letters,* vol. 29, no. 23, 2017.

[73] G. Goertzel, "An Algorithm for the Evaluation of Finite Trigonometric Series," *American Mathematical Monthly,* vol. 65, no. 1, pp. 34-35, 1958.

[74] K. Banks, "The Goertzel Algorithm," *Embedded Systems Programming Magazine,* September 2002.

[75] C. H. Cox and E. I. Ackerman, "Effect of pilot tone-based modulator bias control on external modulation link performance," in *International Topical Meeting on Microwave Photonics MWP 2000*, Oxford, UK, 2000.

[76] Intel Corp., "Intel® MAX® 10 FPGA Development Kit," 26 June 2015. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/development-kits/max/10m50.html. [Accessed 14 September 2023].

[77] RocketBoards.org, "Altera MAX10 10M50 Rev C Development Kit Linux Setup (ACDS version 16.0)," 17 March 2017. [Online]. Available: https://www.rocketboards.org/foswiki/Documentation/AlteraMAX1010M50RevCDevelopmentKitLinuxSetupV160. [Accessed 20 September 2017].

[78] A. B. Nassif, M. A. Talib, Q. Nasir and F. M. Dakalbab, "Machine Learning for Anomaly Detection: A Systematic Review," *IEEE Access,* vol. 9, pp. 78658-78700, 2021.

[79] A. Panjević, T. Uzunović and B. C. Ustundag, "Development of Correction Models for Three-Electrode NO2 Electrochemical Sensor," in *XXVIII International Conference on Information, Communication and Automation Technologies (ICAT)*, Sarajevo, Bosnia and Herzegovina, 2022.

[80] D. Zibar, A. Ferrari, V. Curri and A. Carena, "Machine Learning-Based Raman Amplifier Design," in *Optical Fiber Communications Conference and Exhibition (OFC)*, San Diego, CA, USA, 2019.

[81] S. Sharma, "Activation Functions in Neural Networks," Towards Data Science, 6 September 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed 2 December 2022].

[82] D. Zibar, M. Piels, R. Jones and C. Schaeffer, "Machine Learning Techniques in Optical Communication," *Journal of Lightwave Technology,* vol. 34, 2016.

[83] J. Mata, I. d. Miguel, R. J. Durán, N. Merayo, S. K. Singh, A. Jukan and M. Chamania, “Artificial intelligence (AI) methods in optical networks: A comprehensive survey,” *Optical Switching and Networking,* vol. 28, pp. 43-57, 2018.

[84] Y. Xie, Y. Wang, S. Kandeepan and K. Wang, “Machine Learning Applications for Short Reach Optical Communication,” *Photonics,* vol. 9, no. 1, p. 30, 2022.

[85] Zibar, D., de Carvalho, L. H. H., Piels, M., Doberstein, A. et al, “Application of Machine Learning Techniques for Amplitude and Phase Noise Characterization,” *Journal of Lightwave Technology,* vol. 33, no. 7, pp. 1333-1343, 2015.

[86] M. Schaedler, M. Kuschnerov, S. Calabrò, F. Pittalà, C. Bluemm and S. Pachnicke, “AI-Based Digital Predistortion for IQ Mach-Zehnder Modulators,” in *Asia Communications and Photonics Conference (ACP)*, Chengdu, China, 2019.

[87] J. Lee, T. Song, J. He, S. Kandeepan and K. Wang, “Recurrent neural network FPGA hardware accelerator for delay-tolerant indoor optical wireless communications,” *Optics Express,* vol. 29, no. 16, pp. 26165-26182, 2021.

[88] Freire, P. J., Srivallapanondh, S., Anderson, M., Spinnler B. et al, “Implementing Neural Network-Based Equalizers in a Coherent Optical Transmission System Using Field-Programmable Gate Arrays,” *Journal of Lightwave Technology,* vol. 41, no. 12, pp. 3797-3815, 2023.

[89] T. Yang, Y. Wei, Z. Tu, H. Zeng, M. A. Kinsy, N. Zheng and P. Ren, “Design space exploration of neural network activation function,” *IEEE Transactions on Computer-Aided Design of Integrated,* vol. 38, no. 10, p. 1974–1978, 2018.

[90] Intel Corp., “Intel® High Level Synthesis Compiler,” 3 December 2023. [Online]. Available: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html. [Accessed 11 December 2023].

[91] AMD Inc., “AMD Vivado High Level Design,” [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/high-level-design.html. [Accessed 11 December 2023].

[92] Freire, P. J., Napoli, A., Spinnler, B., Anderson, M. et al.;, “Reducing Computational Complexity of Neural Networks in Optical Channel Equalization: From Concepts to Implementation,” *Journal of Lightwave Technology,* vol. 44, no. 14, pp. 4557-4581, 2023.

[93] M. Kuschnerov, F. Hauske, K. Piyawanno, B. Spinnler, M. S. Alfiad, A. Napoli and B. Lankl, “DSP for Coherent Single-Carrier Receivers,” *Journal of Lightwave Technology,* vol. 27, no. 16, pp. 3614-3622, 2009.

[94] P. J. Freire, M. Anderson, B. Spinnler, T. Bex, J. E. Prilepsky, T. A. Eriksson, N. Costa, W. Schairer, M. Blott, A. Napoli and S. K. Turitsyn, “Towards FPGA Implementation of Neural Network-Based Nonlinearity Mitigation Equalizers in Coherent Optical Transmission Systems,” in *2022 European Conference on Optical Communication (ECOC)*, Basel, Switzerland, 2022.

[95] Abadi, M., Agarwal, A., Barham, P., Brevdo, E. et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015.

[96] A. Gulli and S. Pal, Deep learning with Keras., Packt Publishing Ltd, 2017.

[97] AMD Inc., “Versal AI Core Series VCK190 Evaluation Kit,” AMD Inc, [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vck190.html. [Accessed 17 October 2023].

[98] J. Caba, F. Rinc´on, J. Barba, A. Jos´e, J. Dondo and J. C. L´opez, “Towards driven development for fpga-based modules across abstraction levels,” *IEEE Access,* vol. 9, p. 31581–31594, 2021.

[99] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers and Z. Zhang, “High-Level Synthesis for FPGAs: From Prototyping to Deployment,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 30, no. 4, pp. 473-491, 2011.

[100] AMD Inc., “Vitis High-Level Synthesis User Guide (UG1399),” 7 June 2022. [Online]. Available: https://docs.xilinx.com/r/2022.1-English/ug1399-vitis-hls/Getting-Started-with-Vitis-HLS. [Accessed 20 February 2022].

[101] AMD Inc., “Vivado Design Suite,” 2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html. [Accessed 9 March 2022].

[102] AMD Inc., “ug904 Vivado Implememtation strategy,” 2022. [Online]. Available: https://docs.xilinx.com/r/2022.1-English/ug904-vivado-implementation/Defining-Implementation-Strategies. [Accessed 20 March 2022].

[103] A. Lopez-Gasso, “What are the best vivado synthesis and implementation strategies???,” April 2021. [Online]. Available: https://miscircuitos.com/vivado-synthesis-and-implementation-strategies/. [Accessed April 2022].

[104] AMD Inc., “Versal ACAP Memory Resources Architecture Manual (AM007),” 24 11 2020. [Online]. Available: https://docs.xilinx.com/r/en-US/am007-versal-memory/Block-RAM-Summary. [Accessed 28 06 2023].

[105] G. Huang, Q. Zhu and C. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing,* vol. 70, no. 1-3, pp. 489-501, 2006.

[106] M. Li, S. Wibowo and W. Guo, “Nonlinear Curve Fitting Using Extreme Learning Machines and Radial Basis Function Networks,” *Computing in Science & Engineering,* vol. 21, no. 5, pp. 6-15, 2019.

[107] G. Chen, C. Parada and G. Heigold, “Small-footprint keyword spotting using deep neural networks,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, 2014.