THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

GPF: GIGAHERTZ PULSE FITTER

Master's Thesis

ALİ BAŞARAN

İSTANBUL, 2012

THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
ELECTRICAL AND ELECTRONICS ENGINEERING

GPF: Gigahertz Pulse Fitter

Master's Thesis

Ali Başaran

Supervisor: ASST. PROF. H. FATİH UĞURDAĞ

İSTANBUL, 2012

BAHÇEŞEHİR UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
ELECTRICAL AND ELECTRONICS ENGINEERING

Name of the thesis: Gigahertz Pulse Fitter
Name/Last Name of the Student: Ali Başaran
Date of the Defense of the Thesis: January 26, 2012

This thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof F. Tunç BOZBURA
Graduate School Director
Signature

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Assoc. Prof Ufuk TÜRELİ
Program Coordinator
Signature

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members Signature

Thesis Supervisor
Asst. Prof. H. Fatih UĞURDAĞ: -------------------

Thesis Co-Supervisor
Assoc. Prof. Taylan AKDOĞAN: -------------------

Member
Asst. Prof. Yalçın ÇEKİÇ: -------------------

Member
Assoc. Prof. Sezer GÖREN UĞURDAĞ ---------------------

Member
Asst. Prof. Levent Eren ---------------------

ACKNOWLEDGEMENTS

This thesis is the result of a project funded by TUBITAK. (No: 107T538) I would like to thank the following people who contributed to this project:

- Special thanks to Asst. Prof. H. Fatih UĞURDAĞ who is not only my supervisor but my indispensable mentor. He is the architect of the hardware design presented in this thesis. I owe my title and career as a design engineer mostly to this phenomenal character.
- Assoc. Prof. Taylan AKDOĞAN is the chief founder of this project. He helped me understand the algorithm and the theory behind this thesis. His humanitarian approach combined with his genius is a true inspiration to many students including myself.
- Veli Uğur GÜNEY also worked in this project as a Master's student. Along with his theoretical explanations regarding this project, he helped me cope with life as a dear friend.
- Assoc. Prof. Sezer GÖREN UĞURDAĞ carefully reviewed this thesis and provided insightful feedback. She was also my teacher who taught essential concepts regarding Digital Design.
- Ericsson Microelectronics Design Center (EMDC) provided tools necessary for this design work. Their supportive approach, especially Mehmet EKMEN's, was indeed vital.
- Asst. Prof. Yalçın ÇEKİÇ took the time and examined this thesis as well as being there as a committee member.

All my work included in this thesis, is dedicated solely to my mother. She is the symbol of true strength and determination. I would like to end my acknowledgements with an innocent wish to join her spiritual entity, "Deniz ve Mehtap".

Ali BAŞARAN

ABSTRACT

GPF: GIGAHERTZ PULSE FITTER

Başaran, Ali

Electrical and Electronics Engineering
Thesis Advisor: Asst. Prof. H. Fatih Uğurdağ

January 2012, 45 Pages

High energy particle physics experiments require the processing of a superposition of signals from many particle detectors. Such signal contains many high frequency pulses, each of which belongs to a particle. The mathematical characteristics of a pulse, such as rise/fall times and amplitude, indicate the particle type. Processing of these signals on the fly, as they are received from detectors, is critical. Sending them to an array of hard disks to be processed later by a farm of computers would have multiple drawbacks. It would require too much bandwidth between the data acquisition cards and the storage array, too many disks, and too many computers so that they can keep up with the incoming data. Our solution to this problem is Gigahertz Pulse Fitter (GPF). GPF is a Data Acquisition System (DAQ) with a Field Programmable Gate Array (FPGA) next to Analog-to-Digital Converter (ADC). The FPGA processes the pulses as they occur and send only the pulse parameters to the storage/computer farm, thus enormously reducing bandwidth, storage, and compute requirements of the farm. This thesis outlines the design of GPF from concept to C code, from C code to SystemC code, from SystemC to HW architecture, from HW architecture to FPGA implementation. During this process, this thesis contributes in the following departments. It outlines a flow so that design verification stops being a moving target and the design works the first time it is programmed on the FPGA. It presents a novel architecture that combines pipelining and parallelism. The parallel part of the architecture is based on our concept of Optimized Performance Per Unit Block (OP-PUB). OP-PUB architecture is flexible and can be adapted to any pulse rate by calculating the necessary number of Identical Parallel Processors (IPPs) and FIFO sizes based on a formula. OP-PUB features a priority encoder based dispatcher at the top level and "Loop Pipelining" inside the IPPs. The IPP is a specialized CPU executing a fixed iteration body with an indeterminate number of iterations. On the FPGA implementation side, we use code generation techniques as well as smart pipelining and resource utilization. The architecture and design flow proposed are generic enough to withstand changes in the specifics of the curve fitting algorithms employed.

Keywords: FPGA design, Loop Pipelining, Particle Experiment, IPP, OP-PUB.

ÖZET

GDB: GİGAHERTZ DARBE BETİMLEYİCİ

Başaran, Ali

Elektrik-Elektronik Mühendisliği
Tez Danışmanı: Yrd. Doç. Dr. H. Fatih Uğurdağ

Ocak 2012, 45 Sayfa

Yüksek enerji parçacık fiziği deneylerinde parçacık dedektörlerinden gelen üstüste koyulmuş sinyallerin işlenmesi gerekir. Söz konusu sinyaller, her biri bir parçacığa karşılık gelen yüksek frekanslı darbeler içerir. Darbeye ait genlik, yükselme ve alçalma zamanı gibi matematiksel öğeler parçacığın cinsini belirtir. Dedektörlerden gelen sinyallerin hiçbir gecikme olmaksızın işlenmesi önemlidir. Sinyalleri sabit disklerde saklayıp sonradan işlenmek üzere bilgisayarlara gönderemezdik. *Data Acquisiton (DAQ* kartları ile sabit diskler arasında çok fazla bant genişliği, çok sayıda sabit disk ve gelen veriye yetişebilmek için çok sayıda bilgisayar gerekirdi. Bizim bu probleme bulduğumuz çözümün adı Gigahertz Darbe Betimleyici (GDB). GDB, Analog-Dijital-Çevirici'nin yanındaki *Field Programmable Gate Array*'den (*FPGA*) oluşan bir *DAQ* olarak tanımlanabilir. Darbeler geldikçe sözkonusu *FPGA* bu darbeleri işler ve bilgisayarlara darbeye ait örneklerin kendisinden ziyade darbeye ait parametreleri gönderir. Böylece bant genişliğini ve sabit disk ihtiyacını önemli ölçüde azaltırız. Bu tezde GDB dizaynı; konseptten C koduna, C kodundan *SystemC* koduna, *SystemC* kodundan donanım mimarisine, donanım mimarisinden *FPGA* uygulamasına ana hatlarıyla anlatılmaktadır. Dizaynın doğrulanması *FPGA* aktarımından bağımsız gerçekleştirilmiştir; dizayn *FPGA* ile programlandığı ilk seferde çalışır durumdadır. *Pipelining* ve paralelleştirme tekniklerinin birleştirilmesiyle yeni bir mimari sunmaktadır. Mimarinin paralel kısmı bizim *OP-PUB (Optimized Performance Per Unit Block*) konseptine dayanır. *OP-PUB* mimarisi esnektir ve yeterli *IPP (Identical Parallel Processor)* sayısı ve *FIFO* derinlikleri hesaplanarak her darbe sıklığına göre ayarlanabilir. *OP-PUB* üstünde öncelik temelli bir dağıtıcı ve *IPP*'lerin içinde "*Loop Pipelining*" barındırır. *IPP*, sabit bir iterasyonu önceden belirlenmemiş sayıda gerçekleyen özelleştirilmiş bir *CPU* olarak düşünülebilir. *FPGA* uygulaması kısmında, kod üretimi tekniklerinin yanı sıra verimli kaynak kullanımı ve akıllı pipelining kullanmaktayız. Tezde yer alan mimari ve dizayn akışı gerçeklediğimiz algoritmaya özgü değişikliklere uyum sağlayabilecek ölçüde kapsamlıdır.

Anahtar Kelimeler: FPGA dizayn, Loop Pipelining, Parçacık Deneyleri, IPP, OP-PUB.

TABLE OF CONTENTS

LIST OF TABLES vi
LIST OF FIGURES vii
LIST OF ABBREVIATIONS viii
1. INTRODUCTION 1
    1.1 PROBLEM DEFINITION 1
    1.2 CONTRIBUTIONS OF THE THESIS 2
    1.3 HISTORY OF THIS RESEARCH WORK 3
2. PREVIOUS WORK AND BACKGROUND 5
    2.1 DAQ SYSTEMS 5
    2.2 LOOP PIPELINING 7
3. SYSTEM ARCHITECTURE AND DESIGN METHODOLOGY 8
    3.1 SYSTEM ARCHITECTURE 8
    3.2 GPF ARCHITECTURE 11
    3.3 OP-PUB CONCEPT 14
    3.4 DESIGN METHODOLOGY 16
4. SUBMODULE LEVEL MICRO ARCHITECTURE 20
    4.1 TR (TRIGGER) 20
    4.2 PD (PULSE DETECT) 27
    4.3 GP (GUESS PARAMETERS) 30
    4.4 DF (DISPATCHER/FITTER) 34
5. RESULTS AND CONCLUSION 40
    5.1 RESULTS 40
    5.2 CONCLUSION 41
REFERENCES 42
CURRICULUM VITAE 45

LIST OF TABLES

Table 4.1: TR interface……………………………………………………………………….20
Table 4.2: PD interface……………………………………………………………………….27
Table 4.3: PD tag information.……………………………………………………………….28
Table 4.4: GP interface……………………………………………………………………......30
Table 4.5: DF interface……………………………………………………………………….34
Table 5.1: Resource usage report in GPF………………………………………………….....40

LIST OF FIGURES

Figure 1.1 Traditional setup versus GPF ……………...……………………………….......2

Figure 3.1: System level view of experiment setup ……………………………….……......8

Figure 3.2: Particle detector view ……………...……………………………….……......9

Figure 3.3: Data acquisition unit ……………….…………………………………… ……......9

Figure 3.4: FPGA layout ……………………….………………………………………......10

Figure 3.5: GPF macro architecture ………………………...…………………………......11

Figure 3.6: Pulse shape with threshold ……………………………………………….....12

Figure 3.7: Loop pipelining flow ……………………………………………………......16

Figure 3.8: Implementation flow ………………………………………………………......17

Figure 3.9: Floating versus fixed point example ………………………………………......17

Figure 3.10: Fixed point validation …………………………………………………......18

Figure 4.1: TR macro architecture……………………………….………………........21

Figure 4.2: Threshold comparison high-to-low standard...……..……………………….......22

Figure 4.3: Reduction of comparators ……………...……..…………………………....23

Figure 4.4: TR state machine ……………………….……..………………………….........24

Figure 4.5: TR circular buffer ……………………….…..………………………….......25

Figure 4.6: Removal of multiplier ……………………..……………………………......26

Figure 4.7: PD macro architecture ……………………..………………………………......28

Figure 4.8: Max binary tree ………………………..……..……………………………......29

Figure 4.9: GP macro architecture ……………………..……………………………........31

Figure 4.10: Pulse store micro architecture ………………..…………………………........31

Figure 4.11: Pulse process state machine ……………...……..…………………………........32

Figure 4.12: Postlude inner modules ....……………..……..……………………………........33

Figure 4.13: DF macro architecture ……………………..…………………………......34

Figure 4.14: Fitter macro architecture…………………..……..…………………………......36

Figure 4.15: Fitter global FSM…………………………..…………………………........37

LIST OF ABBREVIATIONS

Msps:	Megasample per Second
VME:	Virtual Machine Environment
DAQ:	Data Acquisition
ADC:	Analog-to-Digital Converter
FPGA:	Field Programmable Gate Array
OP-PUB:	Optimized Performance per Unit Block
IPP:	Identical Parallel Processor

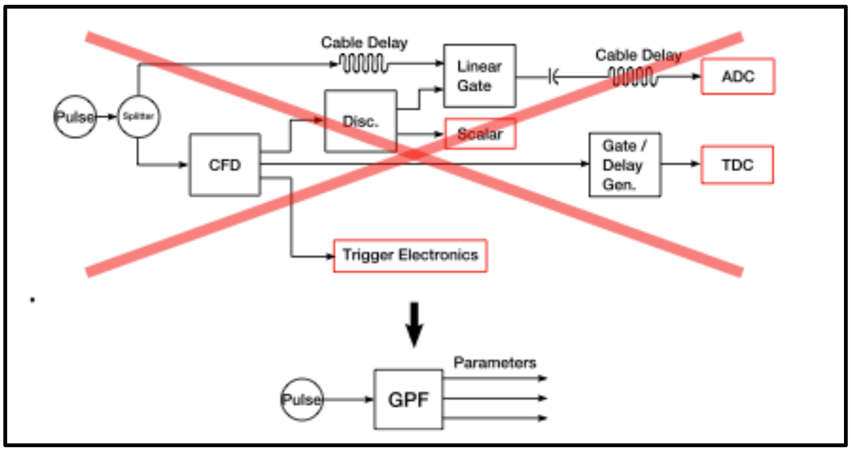
1. INTRODUCTION

In this chapter, the main problem is defined and contributions of this thesis are listed in addition to brief explanation of the history of our work and the outline of the whole thesis.

1.1 PROBLEM DEFINITION

High-energy physics experiments deal with high frequency pulses. They require DAQ systems in order to detect and process the incoming signals from particle detectors. Whenever there is a particle, the detectors output a series of signals, which we call “pulse”. This pulse is unique for each particle type and has certain mathematical characteristics such as amplitude, t_0, and tau. Therefore, the curve defined with the mentioned parameters actually indicates the type of particle. In the usual experiment setup, unique electronic devices are used to measure each parameter. Firstly, there is a trigger mechanism to decide whether or not the pulse data will be recorded. Then a huge variety of different hardware is used for every parameter, which makes the whole system noisy, costly and even risky. Our goal is to combine the functionality of all electronic devices, each devoted to a unique parameter. Thus our project deals with the problem of building a single compact DAQ system to detect the pulse and apply an algorithm to define the parameters of the curve that best fit the pulse data. This DAQ system uses a programmable hardware called FPGA. In this thesis, we focus on the implementation of the hardware block called, “GPF”. Figure 1-1 shows the simplicity of our proposal versus the traditional experiment setup.

Figure 1.1: Traditional setup versus GPF



Reference: Veli Uğur Güney, (2010), *Triggerless Particle Identification System*, June 2010

It is also essential to maintain the system in dead-timeless manner. The signals coming from the particle detectors are analyzed and processed on the fly. This ensures that we do not lose any pulse during computation time. In digital designer point of view, our challenge is to implement a pipelined architecture, which will also have Identical Parallel Processors that execute the curve-fitting algorithm. The rate of pulse coming from the particle detectors can vary and our system has to have sufficient flexibility to handle it.

Lastly, our problem can be broken down to 3 phases: Detection of the pulse, guessing an initial set of parameters, and computing the best set of parameters.

1.2 CONTRIBUTIONS OF THE THESIS

- The most important contribution of this thesis is the efficiency offered to High Energy Physics Experiments. FPGA implementation of the curve-fitting algorithm proposes a much faster and less complex experiment flow.

- Electronic modules dedicated to calculate a unique parameter will be replaced by a dead-timeless FPGA design. In traditional setup, each electronic module is locked during its computation time, i.e., it is unable to accept a new input while it is busy. Unlike the current devices, our design outputs a set of valid parameters on the fly, which accelerates the whole process by approximately 10 percent. This speed factor will in turn drastically reduce power consumption and cost.

- The area reduction and simplicity should also be taken into account since all parameters will be computed using a single compact board instead of a collection of electronic devices.

- We use FPGA instead of CPU to fit our curve. Our trials show that using a C code to process the incoming samples would be able to handle a maximum pulse rate of 1 kHz. However, our FPGA design, GPF, can handle 1MHz pulse rate and even more. In terms of performance, GPF is as powerful as 1000 computers.

- GPF is composed of a series of pipelined modules and a variable amount of identical parallel processors. Each IPP has a unique architecture that consists of Loop Pipelining. The concept of Loop Pipelining allows us to share resource and maximize throughput at the same time.

- At this point, we also propose the concept of Optimized Performance Per Unit Block. The unit blocks are the IPPs executing an algorithm whose runtime is indeterminate. The number of IPPs and the amount of resource used in each IPP is such that the throughput of GPF is maximized, i.e., we output a valid set of parameters as frequently as possible.

- It is always essential for programmers to maintain flexibility. One of the advantages of FPGA design is the fact that the design itself is reprogrammable. Our design work consists of a parameter definition interface, as a result of which, hardware definition code with a sufficient number of IPPs can be generated based on a specification of pulse rate and clock frequency.

1.3 HISTORY OF THIS RESEARCH WORK

This thesis is based on the TUBITAK project proposed by Taylan Akdogan in 2007. Main goal was to establish a system with significant computational power that serves the purpose of many special electronic modules used for physics experiments in CERN. Taylan Akdogan and Veli Ugur Guney provided theoretical fundamentals and the core algorithm whereas H. Fatih Ugurdag and Ali Basaran were responsible for the implementation on FPGA. Alongside these four people, Onur Baskirt also contributed to the project in early phases. Although the design was verified on testbench from TR to GP, we could not upload the bitfile on FPGA since there were some issues with the Virtex5 board.

2. PREVIOUS WORK AND BACKGROUND

In this chapter, earlier projects and articles that inspired our system are listed. Then a brief section will follow, giving background information about the concept of loop pipelining.

2.1 DAQ SYSTEMS

The DAQ that resembles our system the most was used in a mid-scale experiment in MIT (Akdogan T., 1999). Compton electron beam polarimeter system was introduced in 1999 and is still present. MIT/Bates electron beam polarimeter was used to measure the polarization of an electron beam that was polarized with the highest current. (Akdogan T., 2005) High-energy photons were produced after Compton interaction and detected with CsI crystal detectors. In this structure, every measurement had to be completed within 15 minutes and the measurement system had to deal with 100 kHz event rate. Due to the fact that handling this event rate with traditional methods was impossible, Taylan Akdogan and his colleagues developed an ADC-based system with VME-based sampling. The events were sampled at 100 Msps and recorded using a basic threshold mechanism. This dead-timeless measurement setup was more than satisfactory. (Franklin W.A., 2000) Unlike our design, all signal processing is done on CPU. Nevertheless, this project is the basis of our thesis since data is processed on the fly in dead-timeless manner.

Hien and Senzaki from Japan (2001) added FPGA to their system created for nuclear spectroscopy. The sampling rate in this study has reached only up to 40 Msps and dead time for each trigger is about 1 μs. Although FPGA is not used for signal processing and the system has dead-time, it is an important step forward since an FPGA is used in a DAQ.

Bolic and Drndarevic developed an FPGA-based photon spectroscopy system (2002).

This system measured pulse length and intensity with FPGA. This study minimized the usage of analogous circuits by bringing the concept of realizing digital pulse processing on FPGA. Although the 8-bit 60 MHz digitizer unit used in this experiment is modest, it is conceptually essential since CPU was replaced with FPGA.

Nicolau from Italy (2006) also designed an FPGA-based ADC sampling DAQ system. The most important contribution of this study is that the time resolution is set below 1 ns with the usage of 200 Msps 8-bit ADC. However, FPGA is only used as control unit for triggering the pulse instead of processing the samples. The system is triggered using a basic threshold mechanism.

Similar concepts are applied to a Positron Emission Tomography (PET) based medical imaging system using a non-stop sampling ADC. (Streun M., 2002) The FPGA in this work processes 12-bit samples at 40MSps and computes electron-positron annihilation time with a precision better than the sampling period (2 ns versus 25 ns).

In 2007, Giachero from Italy developed a dead-timeless DAQ system. This system was also VME-based using 18-bit ADC which ensured high precision. However, FPGA was still not used for signal processing. In this system, the sampling rate was as low as 5 kHz.

Gua J.R. and his colleagues made a research (2005) on how FPGA can be utilized in DAQ systems. Main goal was to establish a programmable system with high-performance data processing. In this study, an FPGA with 10 GHz input was presented which shows us that FPGA-based DAQ systems can handle all kinds of pulse-based signals in experimental physics.

Arcidiacano designed a trigger supervisor for NA48 experiment at CERN. (1999) It was an FPGA-based 40 MHz pipelined hardware system which is almost dead-timeless. The system processed trigger information from local trigger sources. Unlike our design, the frequency is low and no intelligent algorithm is applied on the input. However, considering the fact that this system was developed at 1999, it is a vital step forward.

Khomich and his colleagues investigated the advantageous role of FPGA versus CPU by implementing a tracking algorithm for ATLAS experiment (2006). They used a hybrid FPGA-CPU implementation and compared the performance to CPU-only implementation. It was shown that the parallel processing nature of hardware, FPGA, gave up to 3 times speed.

The work that is most similar to ours was presented by Haselman M. and Hauck S. (2009) Their setup is used for PET and they fit a model to the pulses. However, the model they fit to pulses is not the same as ours. The parameters they derive from the pulse are pretty similar to ours though, indicating pulse timing, length, energy, and DC offset. They do not use a high-precision compute- intensive iterative algorithm like ours to converge on to the curve parameters, because they are interested in a specific type of pulse shape, for which they use a reference shape. They target 100 MHz sampling, 220 kHz pulse rate, and a pulse timing precision of 60 ps, whereas our targets are 1 GHz or higher sampling, several million pulses per second, and 20 ps precision in pulse timing.

2.2 LOOP PIPELINING

The concept of loop pipelining is crucial in our design since it enables us to maintain an optimal balance between throughput and resource usage. Rodrigues from Portugal (2002) used this concept for fast DCT algorithm. Like our design, the subsequent loops were made to follow one another by a global FSM.

Jin Qu conducted a research on the scheduling for loop pipelining. (2010) Since loop pipelining can be scheduled in various ways and the scheduling affects both resource usage and maximum clock frequency, finding the most optimal scheduling is essential. For this purpose, a generic method is presented in his paper.

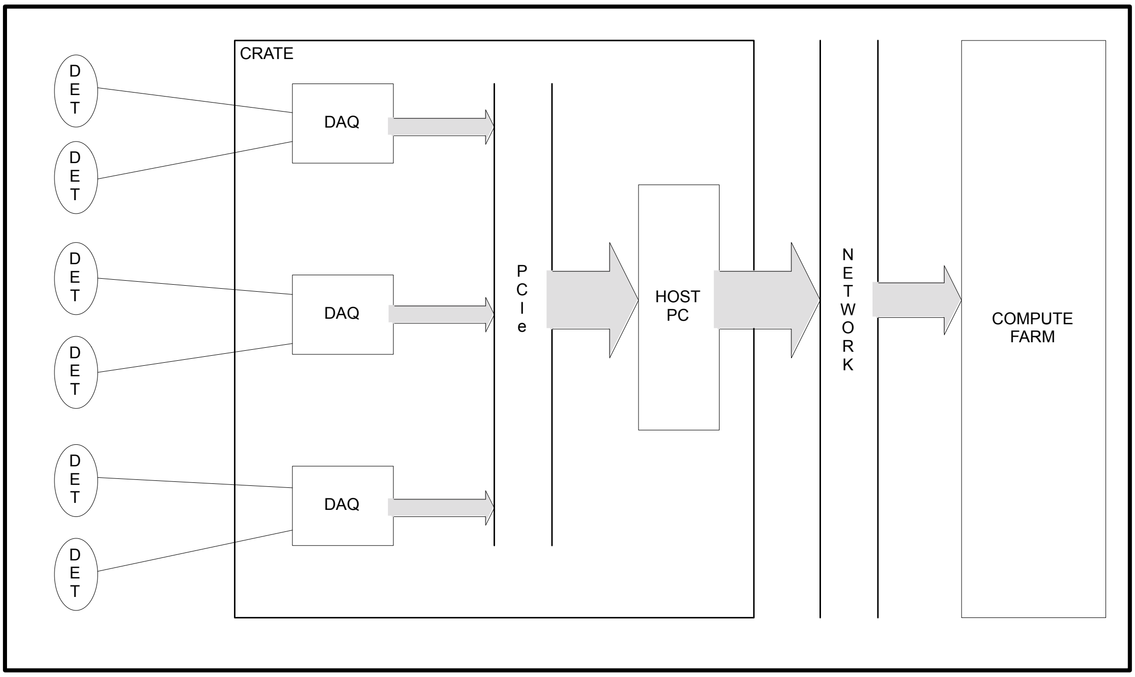
3. SYSTEM ARCHITECTURE AND DESIGN METHODOLOGY

In this chapter, system architecture, on which GPF will be made use of, will be described. Then the GPF overview will be given along with the equation and graphical view of our curve model. OP-PUB concept that is introduced by this thesis will be explained in the following section and then our implementation flow including why we chose SystemC and fixed point will be told.

3.1 SYSTEM ARCHITECTURE

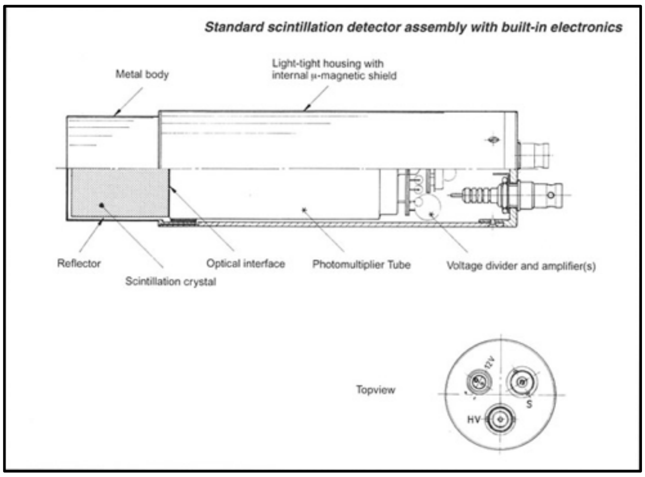
As mentioned before, our system is designed to be used in high energy particle physics experiments. It consists of a series of particle detectors, data acquisition units, PCIe interface, a host computer and a compute farm, as shown in Figure 3.1.

Figure 3.1: System level view of experiment setup



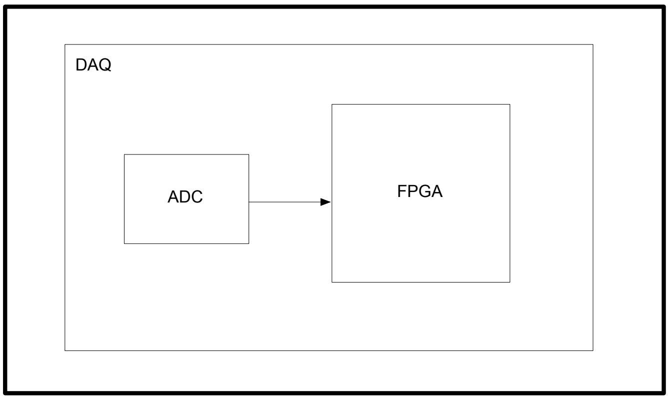
Particle detectors are made up of a scintillator coupled to photomultiplier tube. (PMT) PMTs absorb the light emitted by the scintillator and reemit it in the form of electrons via the photoelectric effect. The subsequent multiplication of those electrons results in an electrical pulse. This is the kind of electrical pulse that is pushed toward the data acquisition units. Particle detectors output analogous signal which means that it needs to be digitized so that it can be processed in hardware. Figure 3.2 shows the inner regions of a compact scintillation detector. Details of its structure are beyond the scope of our thesis.

Figure 3.2: Particle detector view


Standard scintillation detector assembly with built-in electronics
Metal body
Light-tight housing with internal μ-magnetic shield
Reflector
Scintillation crystal
Optical interface
Photomultiplier Tube
Voltage divider and amplifier(s)
Topview
12V
S
HV

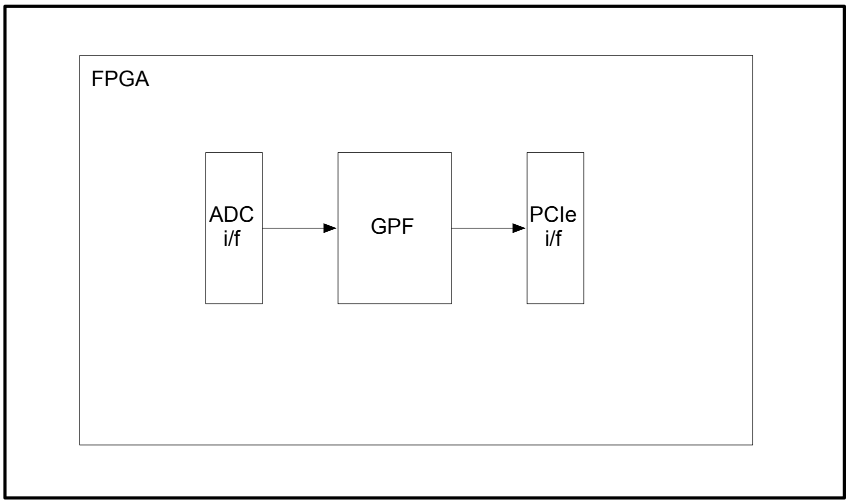
Data acquisition units should have an Analog-to-Digital Converter. (ADC) The ADC converts the continuous signal to quantized digital data. The samples regarding the electrical pulse, are basically 8-bit integers. The frequency of ADC is 1.5 GHz, which means we have 12 Gbps data burst from ADC towards the rest of DAQ.

Figure 3.3: Data acquisition unit



Our FPGA is placed at the output of ADC, as shown in Figure 3.3, thus it has an ADC interface. The ADC interface receives 8-bit samples with 1.5GHz frequency. However, the clock frequency of GPF is 200 MHz, almost 8 times slower than the incoming 8-bit samples. For that purpose, ADC interface collects 8-bit samples in an asynchronous FIFO outputs 64-bit 8-sample blocks to GPF. This means that every clock cycle GPF receives 8 samples instead of 1. In some cases, GPF does not receive valid sample blocks due to the fact that our clock frequency is not exactly one eighth of the sample frequency of ADC. GPF has to discard its input when the valid signal from ADC is low. Interfaces are shown in the FPGA layout below.

Figure 3.4: FPGA layout



GPF, our design, takes this 64-bit 8-sample blocks and the valid signal from ADC and outputs the pulse parameters when the curve-fitting is done. GPF communicates with host computer via PCIe interface. It takes some parameters regarding trigger condition from the host computer as input and operates accordingly. This feature enables us to maintain a hardware-software co-design. The hardware description language written for GPF takes into account the information driven by the host computer. It provides flexibility and eases debugging. Thus when FPGA is powered on, GPF “talks” to the computer and the experimenter can alter the parameters on-the-fly.

PCIe is an improved computer expansion card standard of PCI. It is widely used in high-performance systems. (Koop M.J., 2008) Briefly, PCI is a computer bus that enables us to attach hardware devices to computers, such as FPGA. The output of GPF is sent to the computer using this protocol. The IP that is synthesized to interface this protocol was provided to us by VMETRO company.

3.2 GPF ARCHITECTURE

The architecture of GPF is divided into 6 parts. The order of data path is Trigger, Pulse Detect, Guess Parameters, Dispatch Logic, Fitter Agents and Report Logic respectively.

Figure 3.5: GPF macro architecture

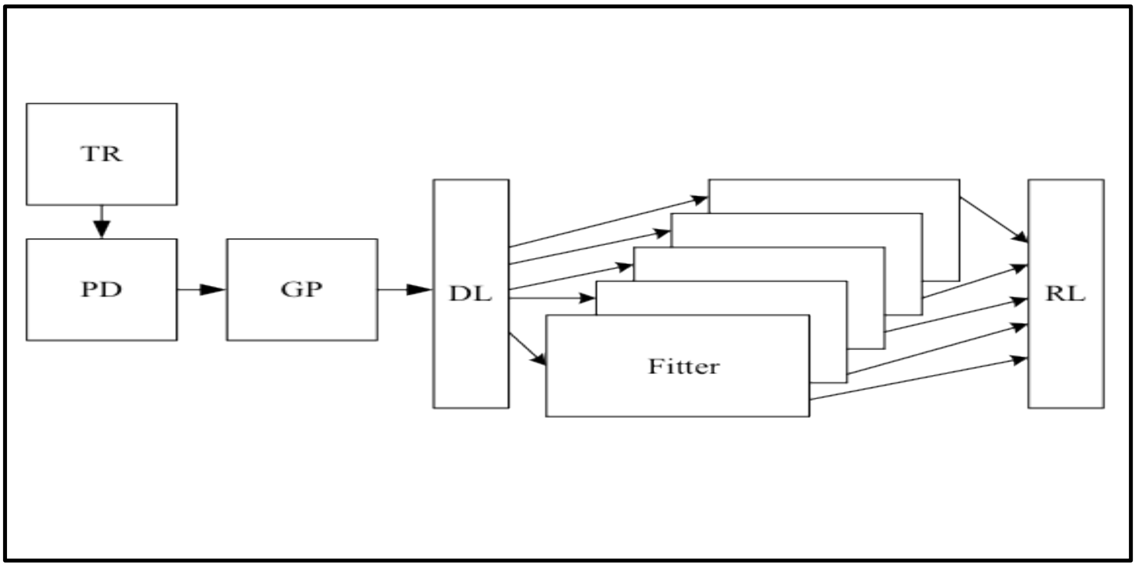


Figure 3.5 shows the submodules that belong to GPF. The data flow from TR to DL is pipelined. However, the output of DL is dispatched to a variable amount of parallel fitters which we call Identical Parallel Processors. Thus the architecture of GPF is mixed; we have series of pipelined modules and parallel processors. IPPs also have a pipelined architecture to maximize throughput.

As it can be seen from Figure 3.5, the output of TR is connected to the Pulse Detect module. The task of PD is to compute the parameters that can be computed on-the-fly. The amplitude, offset and vHalf are calculated as data is pushed towards GP; there is no need to store the samples. GP makes the initial guess of final parameters and for that purpose, storage of the parameters from PD and sample data are required.

When GP is done with the parameters, its output is dispatched to one of the available processor which we call “Fitter”. The dispatch logic is embedded in a separate module called “DL”. DL also stores the output of GP when no fitter is available. The report logic receives the availability information from each fitter and sends it back to DL. Runtime of a fitter is indeterminate due to the nature of the algorithm called “Downhill Simplex Method”.

Figure 3.6: Pulse Shape with threshold

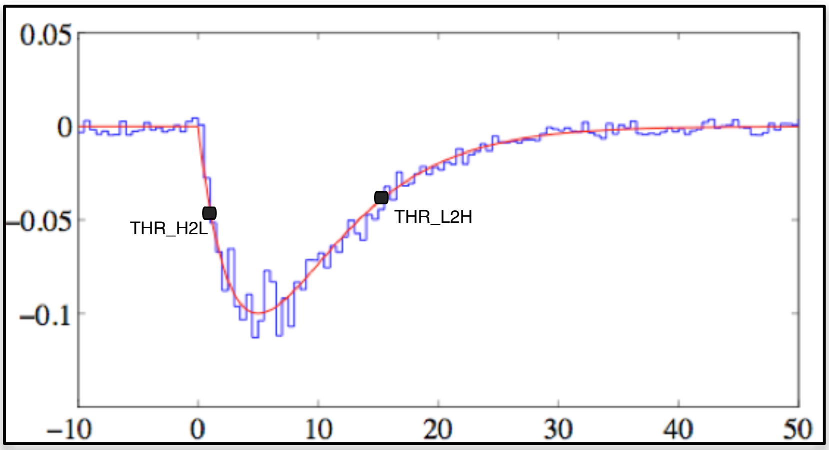


Figure 3.6 actually shows the input and output of GPF. The digital data input from ADC is shown in blue and the continuous red line is the curve that will eventually be defined by the output of GPF; the set of parameters. The pulse can be defined by the following parameters: amplitude, tau, offset, t_0. The curve model is given below:

$$V(t) = V_0 + u(t - t_0) \cdot A \cdot \frac{t - t_0}{\tau} \cdot e^{1 - \frac{(t - t0)}{\tau}} \tag{3.1}$$

Naturally, every sub module is assigned a unique task. TR takes in 5 parameters from the host computer via PCIe as well as the valid signal and 64-bit data (8 samples) from ADC. The 5 parameters that TR takes from ADC are: prelength, postfactor, maximum pulse length, high-to-low threshold and low-to-high threshold. As mentioned before, these parameters are defined by the user and can be modified during runtime.

Figure 3.6 has two threshold points randomly placed as an example. If the curve is going downwards, i.e. tau is negative, the pulse can be called a “negative pulse” as in Figure 3.6. In this case, samples first cross high-to-low threshold, thr_h2l, and then thr_l2h. When the pulse is positive, the order is obviously vice versa.

3.3 OP-PUB CONCEPT

This thesis introduces a concept called, “OP-PUB”, which stands for Optimized Performance Per Unit Block. It is actually a solid solution to the problem of finding the most optimal design architecture to implement our curve-fitting algorithm.

In our verification environment, the average pulse rate is 1MHz and our FPGA operates under single, synchronous clock domain with 200MHz frequency. This means that we have valid pulse data and guess parameters output by GP in every 200 cycles. There has to be a set of modules that take these samples and 3 initial guess parameters as input and apply our curve-fitting algorithm. In order to express the generic nature of our concept, detailed mathematical explanation of the algorithm will not be discussed here. However, we have to state that it executes a loop with indeterminate number of iterations. It can be as few as 4 iterations and as much as the highest possible number specified by the user. In our case, this limit is set to 500. Each iteration calls a function whose runtime is dependent on pulse length. Therefore, the runtime of our algorithm can be described by the following equation:

$$T = k_1 + N \bullet (k_2 \cdot L + k_3) \tag{3.2}$$

In the equation above, T is the number of instructions fetched by the processor, or in hardware designer's perspective, the number of clock cycles. In either case, it is basically a quantitative description of the runtime of our algorithm. The symbol, N, stands for the number of iterations and L is the length of pulse. Note that both N and L can not be pre-determined but they obey a certain range and a probability distribution. The range for N is [4,500] and the average is approximately 22. The pulse length varies from 8 to 128 and the average is nearly 36. These variables, especially N, are in such a probability distribution that the standard deviation is very low. Given these facts, we had 3 architectural options: (1) No pipelining and No parallelism (2) Pipelining without parallelism (3) Combination of pipelining and parallelism: OP-PUB.

Let us suppose that we chose option (1). In that case, Fitter would consist of cascaded instruction stages where each stage would correspond to an arithmetic equation. Theoretically, the area of Fitter, except for the enormous line of multiplexers, would be: one memory as big as $L_{max} \cdot 8$ (note that each sample in pulse is in 8-bit integer format) for the pulse data, a multiplier and an adder that can handle arithmetic operands with maximum bit width, owing to the fact that we can use the same hardware module in every stage. This architecture, alone, is highly utilized in terms of resources but is very poor in terms of runtime and causes the necessity of implementing enormous memory at the output of GP, the pulse data has to be stored in every 200 cycles because Fitter would be busy. When the Fitter hits a long pulse with N_{max}, for instance, it would not be able to accept a new input from GP for as long as $T_{max} = k_1 + 500 \bullet (k_2 \cdot 128 + k_3)$. Given that k_2 is 12, this would mean 768000 cycles (ignoring k_1 and k_3), in other words, 3840 valid pulse data would have to be stored since there is a pulse in every 200 cycles. Considering a series of pulses whose N and L are slightly above average, the need for storage would grow exponentially with unknown rate, thus there would definitely be a memory overflow.

Option (2) offers a greedy approach in terms of runtime. Suppose we divide all the set of instructions to pipeline stages. In every 200 cycles, the new pulse data and parameters are pushed towards this new Fitter that is always available. However, this Fitter must have huge built-in hardware to handle N_{max} and L_{max}. In other words, 3840 pipeline stages have to be implemented. This means there would not be any resource sharing, leading to enormous non-utilized area. The designer would not make use of the indeterminate run time of the algorithm, assuming a fixed maximum for each variable.

OP-PUB, however, combines pipelining and parallelism such that the total area is relatively small and runtime is as low as Option (2). We have a variable amount of Fitters and a dispatcher logic that sends the pulse data and parameters from GP to the available Fitter with the highest priority. (see Figure 3.6) Unlike Option (1), when GPF hits a pulse with high N and L values, only one of the Fitters are busy, the next pulse data is sent to an available one. Also, these Fitters do not accept a new input while they are busy, enabling us to share resource. We call them, “Identical Parallel Processors”.

Identical Parallel Processors can now be said to operate under average values for N and L due to the equation above. Thus the runtime for each IPP is $T_{avg} = k_1 + 22 \bullet (12 \cdot 36 + k_3)$, which is 9504. If we want to push pulse data and parameters whenever there is a valid (200 cycles in average), we must have 48 Fitters. Note that this amount can be reduced if we can eliminate k_2, which is the number of constant operations performed on each sample whenever the function, namely FindChi2, is called. If we choose not to share any resource, we can divide FindChi2 to 12 pipeline stages and eliminate k_2 out of the equation. However, two of these operations require multiplication. Since a multiplier is “expensive” in terms of area, we want to pipeline such that only one multiplier is used per each IPP.

Figure 3.7: Loop pipelining flow

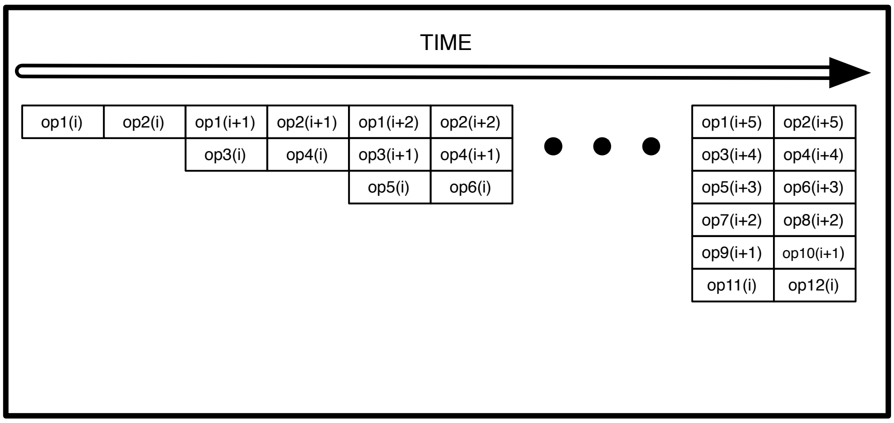


Figure 3.10 is an abstract illustration of our loop pipelining flow. Our challenge was to provide a way to reduce the run time of this function along with keeping the single multiplier constraint. The standard pipelining would consist of 12 distinct hardware units for the 12 different operations since they have to be executed simultaneously, on the same cycle. However, in our loop pipelining flow, the operations are divided to two stages because two of the 12 operations require multiplication and we only have one multiplier in each Fitter. The resource, multiplier, is shared between two stages and the data is still piped. Thus k_2 is reduced to 2 instead of 12.

Note that the number of fitters can be calculated by the following equation:

$$N_{\mathrm{Fitter}} = T_{\mathrm{avg}} \cdot F_{\mathrm{pulse}} / F_{\mathrm{clk}} \qquad \textbf{(3.3)}$$

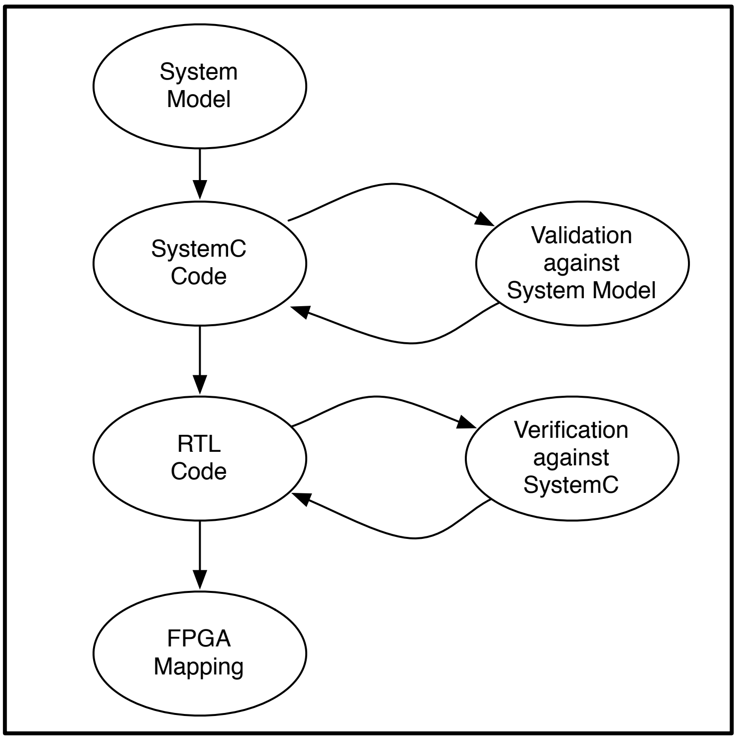
In order to support flexibility, we added a perl script that takes N and L as input arguments and according to the given equations, generates RTL such that the sufficient amount of fitters are instantiated.

3.3 DESIGN METHODOLOGY

Our design methodology resembles the industry-standard FPGA design flow. Firstly, the system model for the algorithm is written, usually in MATLAB.

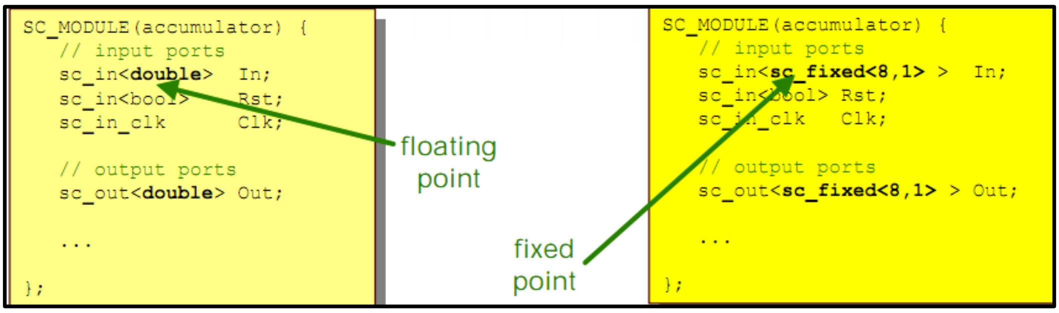
Then this system model is validated using software, SystemC. After the system model validation is completed, the SystemC code is translated to RTL. The RTL behavior also has to be verified. Eventually, the RTL is synthesized and programmed on FPGA. Like industry, these implementation steps can not be seperated with sharp borders. SystemC code might be modified during RTL development due to a bug or to ease verification, for example, and this modification causes the validation of SystemC code against System Model to be repeated.

Figure 3.8: Implementation flow



There are several reasons as to why we chose SystemC. SystemC is not actually a different software language, in fact it is a superset of C++ that supports multiple concurrent processes. We can emulate our hardware modules using the threads in SystemC that works parallel. These threads can be interpreted as different hardware modules and SystemC allows communication between those modules. This makes RTL translation a lot easier than any other “language” such as C.

Figure 3.9: Floating versus fixed point example

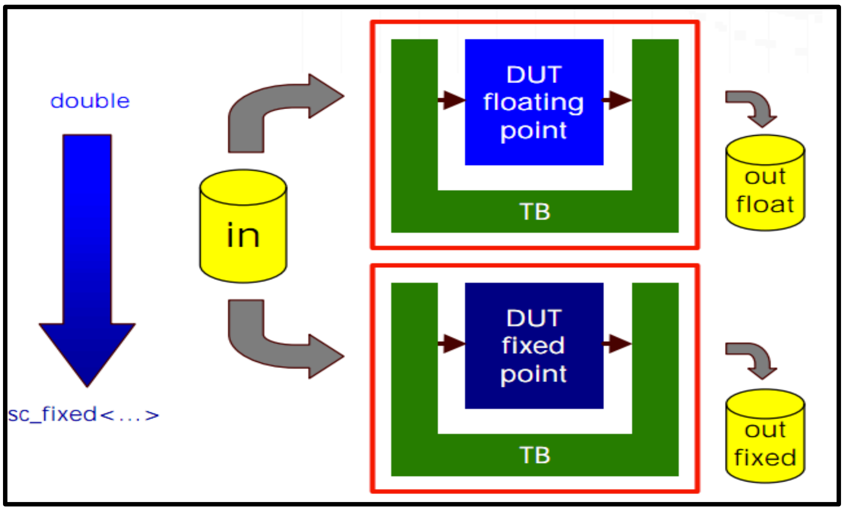


```
SC_MODULE(accumulator) {
   // input ports
   sc_in<double>  In;
   sc_in<bool>    Rst;
   sc_in_clk      Clk;

   // output ports
   sc_out<double> Out;

   ...

};
```

```
SC_MODULE(accumulator) {
   // input ports
   sc_in<sc_fixed<8,1> >  In;
   sc_in<bool> Rst;
   sc_in_clk   Clk;

   // output ports
   sc_out<sc_fixed<8,1> > Out;

   ...

};
```

Due to the fact that there are lots of arithmetic computations used in our algorithm, we had to choose between two data types: floating point and fixed point. Figure 3.12 shows two SystemC code samples (note that SystemC also has a built-in fixed point library) showing the difference between these two data types. Floating point has the advantage of "unlimited" precision but floating point DSPs are more expensive and generally not preferrable if the designer can afford the loss of precision introduced by fixed point. We chose to use fixed point after validating our SystemC code against our System model and concluding that the precision is more than satisfactory. This validation process was to input the same set of files that represent the likely output of ADC towards both models and compare their output, as shown in the figure below. This comparison was done via plotting the final curve using MATLAB.

Figure 3.10: Fixed point validation



Although SystemC resembles hardware behavior, direct translation of the SystemC code to RTL is usually impossible. In a SystemC module, all computations are executed in a single cycle; however in RTL, we have to meet a high frequency constraint (200 MHz), thus each computation is divided into stages. We have to synthesize our RTL to see whether we can meet our constraint and if we cannot do so, we must modify our RTL which is usually to divide the combinational architecture into more pipeline stages. Thus we have to verify our RTL against SystemC iteratively, each flop insertion requires another long simulation run.

Our verification methodology consists of automated self-checking testbenches written for each submodule that belong to GPF. The RTL behavior is automatically compared to the behavior of SystemC. Basically, the same set of inputs at each clock cycle is driven to the SystemC module and RTL, and the output of RTL is compared to the output of SystemC via the testbench written in Verilog. It can be considered as a direct comparison of two output files. Verification environment is in Linux and the tool we used was Icarus Verilog and GTKwave for viewing waveforms.

4. SUBMODULE LEVEL MICRO ARCHITECTURE

In this section, the detailed micro architecture is given for each module in GPF.

4.1. TR (Trigger)

Table 4.1: TR interface

Name	Direction	Bitwidth	Source/Destination
clk	input	1	ADC
rst	input	1	ADC
dIn	input	64	ADC
wEn	input	1	ADC
preLength	input	3	CPU
postFactor	input	5	CPU
param_threshold_h2l	input	8	CPU
param_threshold_l2h	input	8	CPU
param_width_max	input	8	CPU
dOut	output	64	PD
dOutTag	output	2	PD

Clock is 200MHz internal FPGA clock and shared between all GPF submodules as well as reset. They are both synchronous. TR receives 64-bit sample block from ADC. This block consists of 8 consecutive samples in ascending order - the first sample in [7:0] range, second in [15:8] and so on.

ADC cannot output valid sample blocks in approximately every 8 cycles. Thus there is a signal called wEn to disable TR from storing the incoming sample block when the output of ADC is not valid. In other words, TR stores the sample block in memory whenever wEn is high.

TR receives the user-defined parameters from CPU. The preLength is the amount of 8-sample blocks that are going to be included in the pulse once one of the received samples is below or above the specified thresholds. The maximum number of samples that can be included in the pulse data is 56.

There is also another signal, postFactor, that determines the amount of samples that are going to be written to and read from FIFO after one of the incoming samples passes the second threshold line. Samples come in multiples of 8, thus 3 fractional bits are reserved for this signal as well as 2 bits for integer.

TR outputs the sample blocks, dOut, as well as a handshaking signal called dOutTag.

Figure 4.1: TR macro architecture

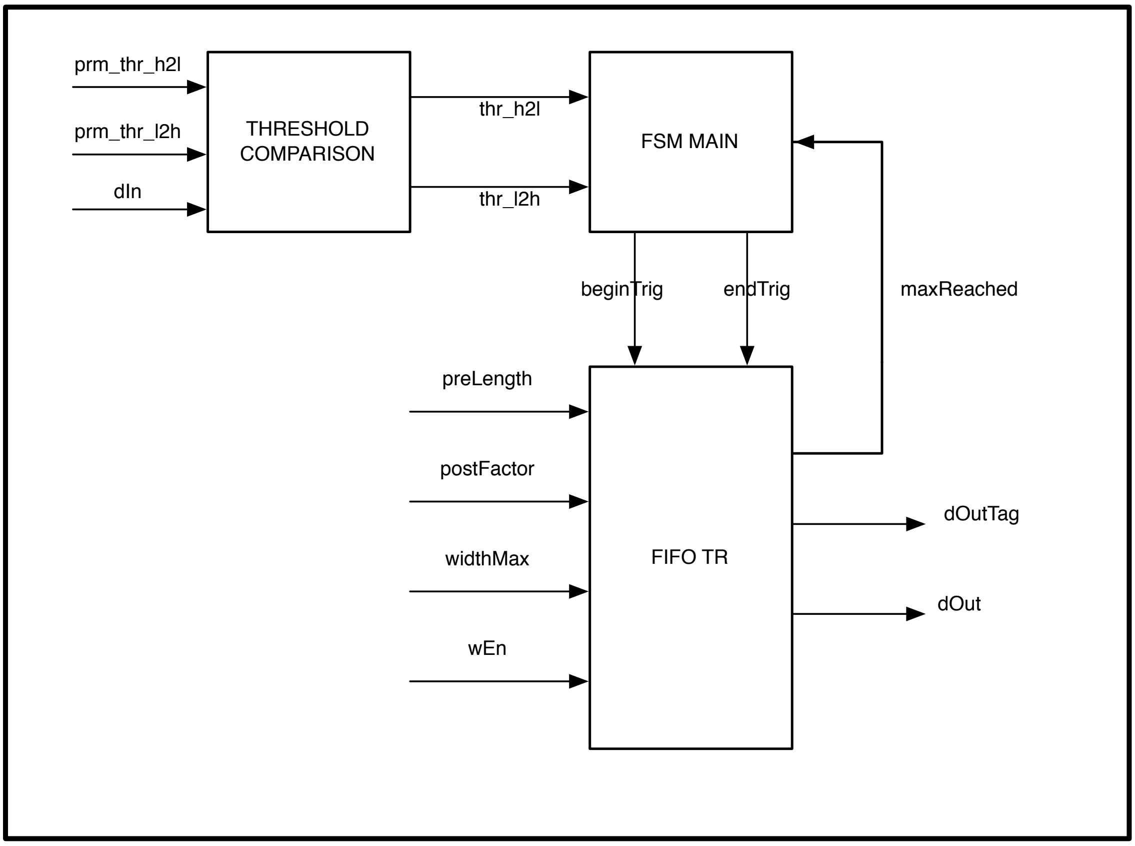


Figure 4.1 shows the macro architecture for TR that is composed of three main units. The task of threshold comparison is to find whether there is any pair of samples that cross prm_thr_h2l or prm_thr_l2h. We have to compare each sample with these values and tell whether S(i-1) is larger than high-to-low threshold and S(i) is smaller. Obviously, vice versa is true for low-to-high threshold. Suppose the output of each comparator is a single bit signal that is logic high when the sample is larger than the given threshold and logic low when it is smaller. We have to find whether the output of S(i) is logic high and the output of S(i-1) is logic low.

Figure 4.2: Threshold comparison high-to-low standard

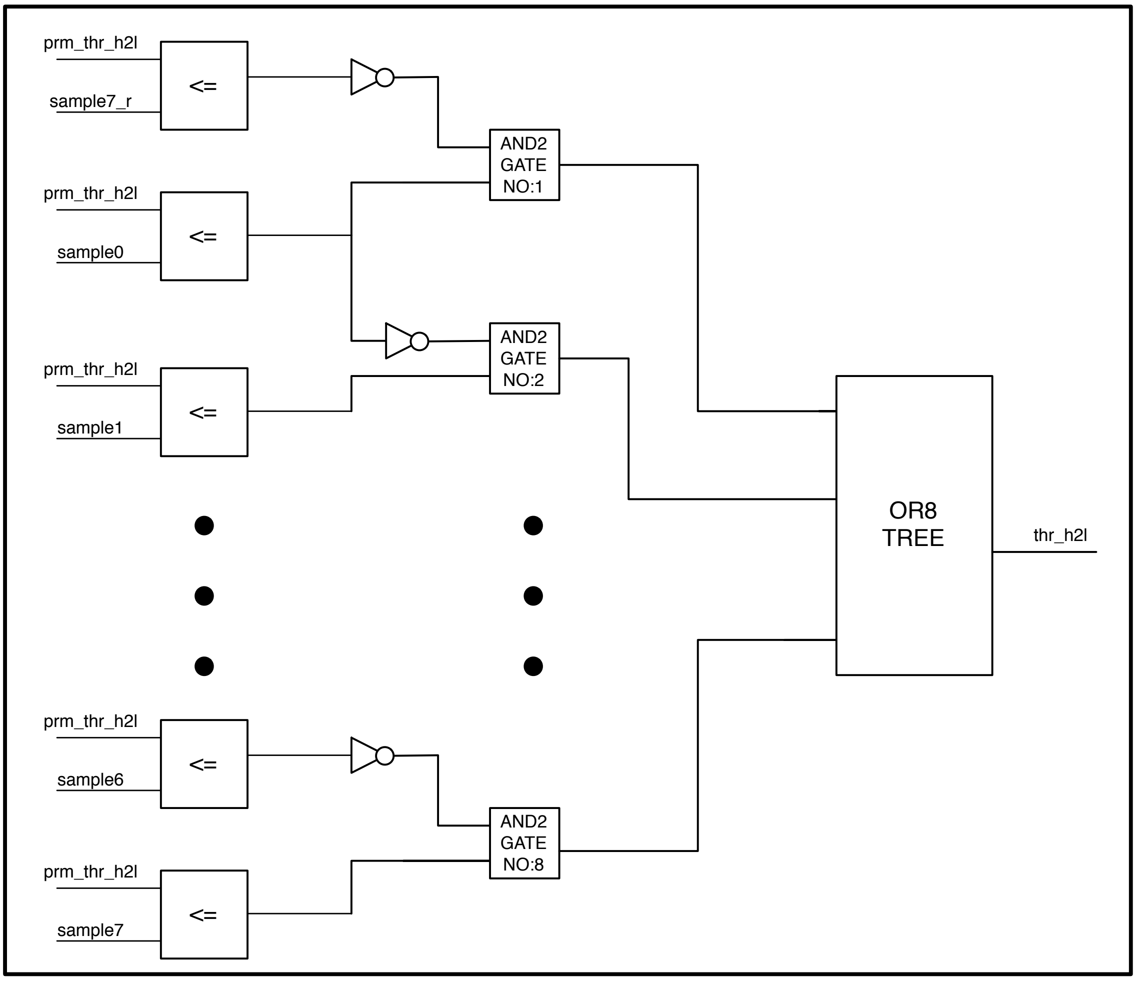
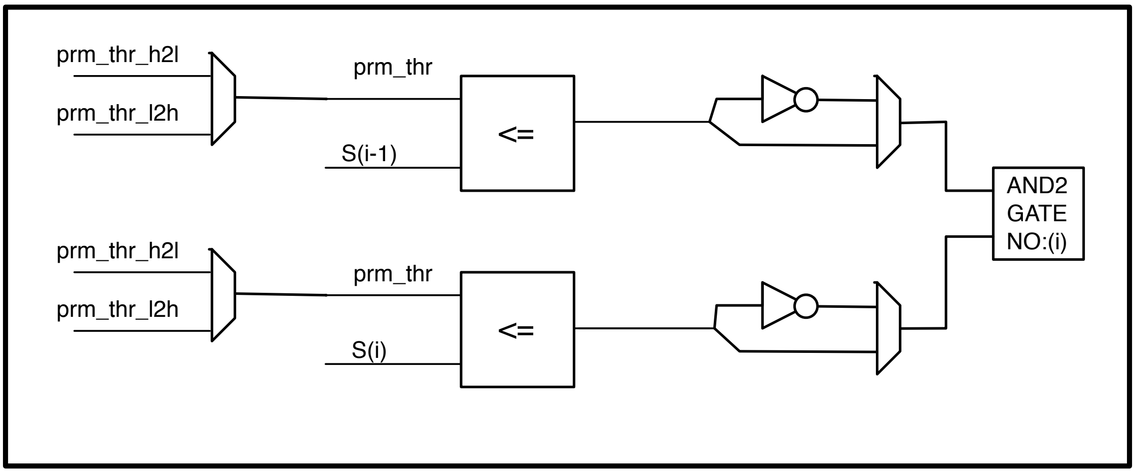


Figure 4.2 is a low-level architectural description of the standard threshold comparison approach. The three dots can be visualized as the same pair of comparators tied to an AND2 gate between samples 2-5. Since the output of a comparator is 1 when S(i) is smaller than or equal to threshold and 0 when it is larger, there is an inverter in front of the output of each comparator. Thus the output of AND2 gates will be high when there is a crossing between S(i-1) and S(i) and the OR8 tree is high when any of these crossings occur. Notice the first comparison is done between the last sample from the sample block in the previous cycle, sample7_r. This is basically the flopped output of sample7.

This architecture contains 8 comparators, 8 inverters, 8 AND2 gates and 1 OR8 tree that is composed of 7 OR2 gates. Considering the fact that this is duplicated for threshold comparison low-to-high except for the location of inverters, we have 16 comparators, 16 inverters, 16 AND2 gates and 14 OR2 gates. If the pulse is decaying (tau is negative), S(i) will be greater than or equal to S(i). The question is whether we can use this information to optimize this architecture in terms of area.

If we were sure that the pulses were always long enough that both crossings can not occur in the same 8-sample window, the solution would be to put a multiplexer in front of each comparator and compare the samples to either prm_thr_h2l or prm_thr_l2h depending on TR state. (see Figure 3.8) This would easily be our choice since multiplexers demand much less area than comparators.

Figure 4.3: Reduction of comparators



As we can see from Figure 4.3, each parameter input of comparator and the output is multiplexed. This makes a total of 16 multiplexers instead of 8 more comparators. The problem with this architecture is the select signal of these multiplexers. If both crossings in a single cycle were impossible, as mentioned above, the select signal would be the TR state and the output of the comparison between any two samples. This approach would not introduce latency while decreasing the total area. However, since both crossings are expected in a single cycle, the select signal tied to the multiplexers on S(i) would need the output of the AND2 gate on S(i-1). This would increase the critical path up to 8 times longer. Thus we did not choose this architecture. However, we offer the user the option to ignore short pulses and synthesize circuit with the architecture shown in Figure 4.3.

Figure 4.4: TR state machine

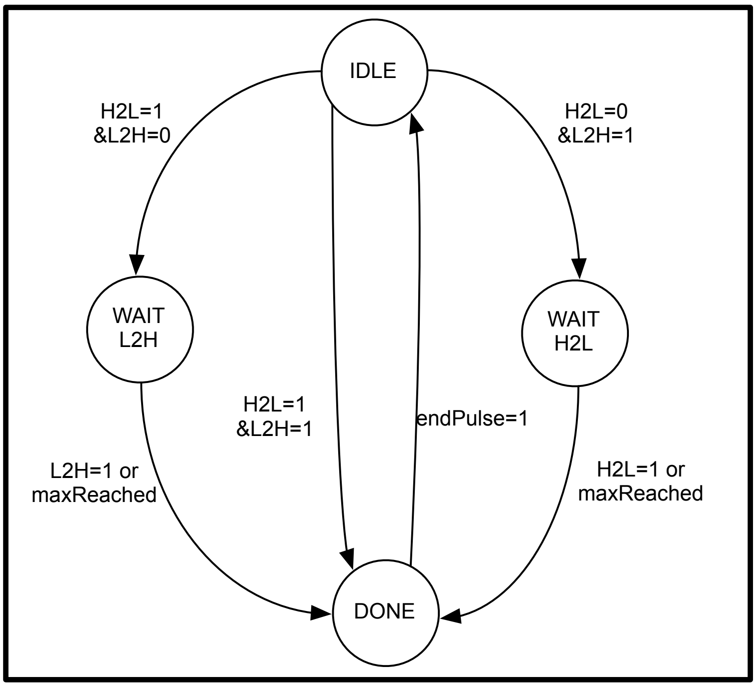
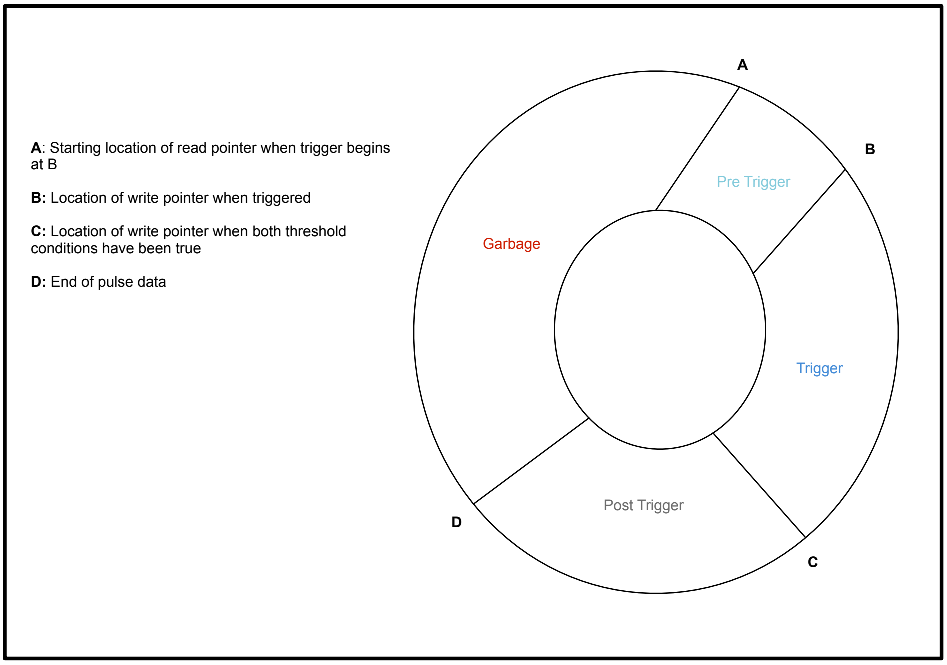


Figure 4.4 shows the state machine for TR. The initial state (or reset) of TR is IDLE which means that there is no trigger whatsoever. This is the most likely situation since the clock frequency is 200 MHz and the average pulse frequency is 1 MHz. Given the fact that dynamic power is consumed when there are transitions in the circuit, this control is essential.

When TR is in IDLE state, it waits for samples to cross either thr_h2l or thr_l2h or both. If the samples cross thr_h2l first, it means that the pulse is negative. After one of the thresholds is crossed, TR waits for the other threshold. When the other threshold is crossed, TR outputs the remaining samples until end of pulse is reached. One corner case is that the other threshold may not be crossed for a long time and in that case, TR waits until the maximum number of samples for the pulse is received and then goes to the DONE state. This rare situation indicates the maximum pulse length determined by the user is insufficient or there is no pulse at all. GPF treats this corner case as if there is a pulse.

Another rare occurrence is when the pulse is too short and both thresholds are crossed in the same 8-sample window.

Figure 4.5: TR circular buffer


A
B
C
D
Pre Trigger
Garbage
Trigger
Post Trigger
A: Starting location of read pointer when trigger begins at B
B: Location of write pointer when triggered
C: Location of write pointer when both threshold conditions have been true
D: End of pulse data

TR stores every valid sample block in a circular buffer shown in Figure 4.5. The sample blocks stored in between A and D are output as valid TR sample blocks. The samples from A to B, B to C, C to D are the valid samples before the first threshold, after the first threshold and before the second, after the second threshold, respectively. The amount of samples before the first threshold is determined by the user as pre length and the amount of samples after the second threshold is the product of the length between B to C and another parameter defined by the user as post factor.

As it can be seen from Figure 4.5, the valid pulse data does not actually start when the first threshold is crossed; i.e. there is valid data before TR transits from IDLE to WAIT_H2L or WAIT_L2H. The amount of valid data before trigger is determined by the user. If pre length is 16, the read pointer starts from the previous two locations since each memory slot stores 8 samples. From that point on, we write to and read from the memory simultaneously at every clock cycle except when the valid signal from ADC is low. Here the ADC valid frequency introduces an intriguing problem if we choose to continue reading from the circular buffer even if the data from ADC is not valid.

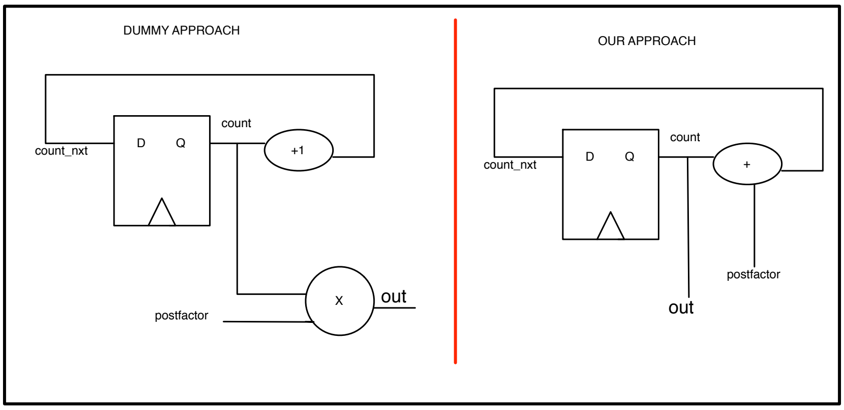
Regardless of the memory size, the read pointer might “catch” the write pointer if the pulse length exceeds a certain limit that depends on the valid frequency and the maximum allowed pre length parameter set by the user.

$$PulseLen_{max} = PreLen_{max} \bullet F_{clk} / (F_{clk} - F_{vld}) \qquad \textbf{(4.1)}$$

The equation above is actually derived from a simple “pool” problem. The write pointer starts from the point B and the read pointer starts from the point B. Suppose the valid pulse data frequency is 7/8 of the clock frequency. Then the distance between the read pointer and the write pointer will be decremented in every 8 cycles, leading to the constraint above. Given the prelength and the frequencies, the size of the circular buffer is modified via code generation using a Perl script.

TR also includes another area optimization trick. After the second threshold is crossed, we are at point C of the circular buffer shown in Figure 4.5. At that point, the amount of samples we need to read from the circular buffer is postfactor multiplied by the amount of samples between B and C. There are two approaches to this problem. The first one is the software approach: (1) count the number of samples between B and C, (2) multiply this count by the postfactor when point C is reached. However, this would require a multiplier which is costly in terms of area and speed. Thus we chose the second approach: add postfactor to itself each cycle within the range B and C. The figure below is a comparison of these two approaches.

Figure 4.6: Removal of multiplier



As it can be seen from Figure 4.6, a huge multiplier and an incrementer is replaced by a single adder. These are the kinds of optimizations that EDA tools can not make.

4.2. PD (Pulse Detect)

Table 4.2: PD interface

Name	Direction	Bitwidth	Source/Destination
clk	input	1	ADC
rst	input	1	ADC
PDdIn	input	64	TR
PDdInTag	input	2	TR
PDdataVld	output	1	GP
PDdOut	output	64	GP
PDparamVld	output	1	GP
PDparamOut	output	112	GP

Clock and reset are shown to be coming from ADC since the clock and reset are not modified within GPF by any means.

PDdIn and PDdInTag are inputs received from TR as dOut and dOutTag. (see Table 4-1) This is actually the 64-bit data, i.e. the 8-sample block and the 2-bit handshaking signal.

Table 4.3: Tag information

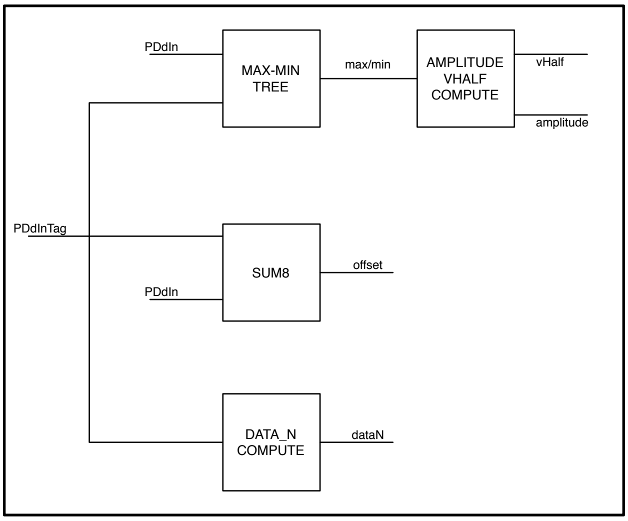
PDdInTag[1:0]	Definition	Action
2‘b00	TR is inactive. No valid data for PD	No action
2‘b01	Signals the first cycle trigger has begun	timestamp is captured
2‘b10	TR is sending valid pulse data	pulse data is analyzed
2‘b11	Signals the last valid pulse data	outputs parameters regarding pulse

PD module needs to know the beginning and end of the pulse as well as the sample blocks in between. That is why we used a 2-bit handshaking system instead of implementing the same control inside PD.

PDdOut is the 64-bit flopped output consisting of the 8 samples received from TR. PDparamOut, however, is the collection of parameters that is computed on-the-fly. It consists of 112 bits: timestamp(64), dataN(12), offset(12), amplitude(12), Vhalf(12). These parameters are required by GP to make the initial guess.

The mathematical explanations behind the operations performed on the sample data are beyond the scope of our thesis. From a hardware designer point of view, as it will be shown more clearly in the following chapters, the only goal is to implement the given algorithm in the most optimal and flexible way.

Figure 4.7: PD macro architecture

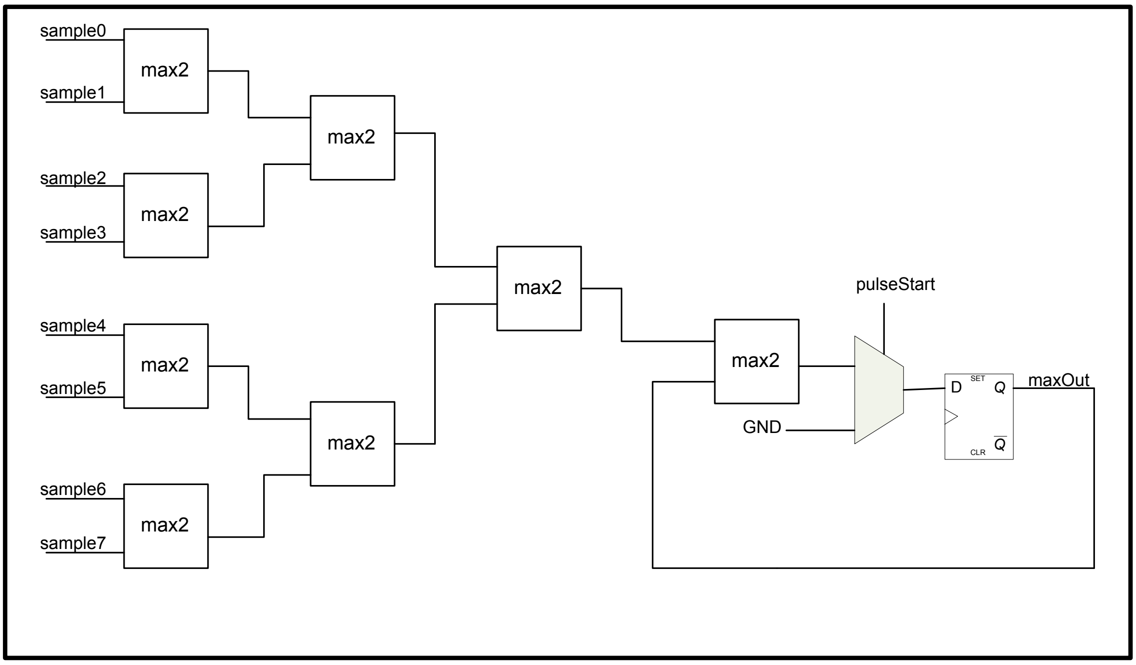


The task of PD is to compute the parameters that can be computed on-the-fly. Thus PD requires no storage of pulse data whereas TR had to have storage of all valid data from ADC and push the data that is included in the pulse towards PD.

The computation of the number of samples is solely based on the tag information received from TR. When the tag is equal to 2 or 3 (see Table 4-3), dataN is incremented by 8. Adding by 8 is basically the same as adding by 1 in terms of hardware complexity since the number 8 is a constant whose only one bit is high.

Offset calculation also depends on the tag information as well as the pulse data because offset is defined as the average value of the first 8 samples. Thus when the tag information in the previous cycle is 1 and in the current cycle it is 2, the output of the summation tree has to be captured as the offset value.

Figure 4.8: Max binary tree



The figure above is a low-level description of how PD computes the maximum sample of the pulse data on-the-fly. The signal, pulseStart, is actually high only when the tag is equal to 1, meaning the 64-bit data is actually equal to the timestamp. This control enables us to reset the maximum to zero (GND) when the new pulse data is introduced. Note that the maximum of the 8-sample input in a given cycle is compared to the current maximum on the fourth level.

Although not shown in the figures, these kinds of combinatorial paths have been pipelined in our RTL. The latency of PD is determined by the user and the number of pipeline levels are arranged accordingly. This flexibility is required since the user may want to synthesize GPF for lower frequencies to test functionality.

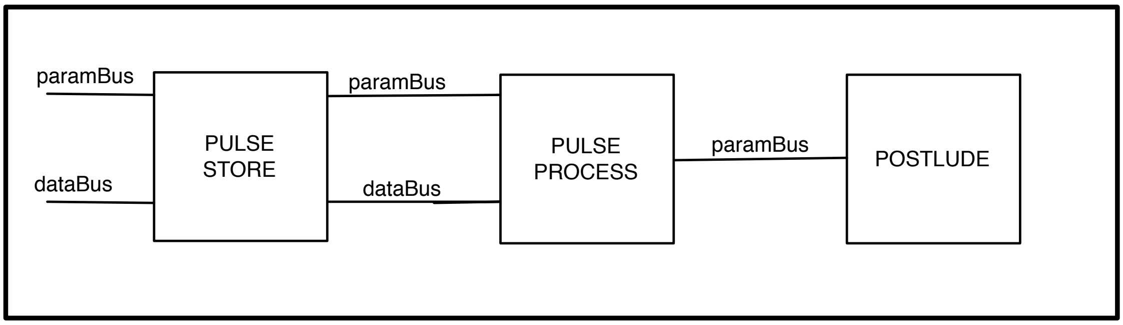
4.3. GP (Guess Parameters)

Table 4.4: GP interface

Name	Direction	Bitwidth	Source/Destination
clk	input	1	ADC
rst	input	1	ADC
i_GPdataVld	input	1	PD
i_GPdata	input	64	PD
i_GPparamVld	input	1	PD
i_GPparam	input	112	PD
o_GPdataVld	output	1	DF
o_GPdata	output	64	DF
o_GPparamVld	output	1	DF
o_GPparam	output	144	DF

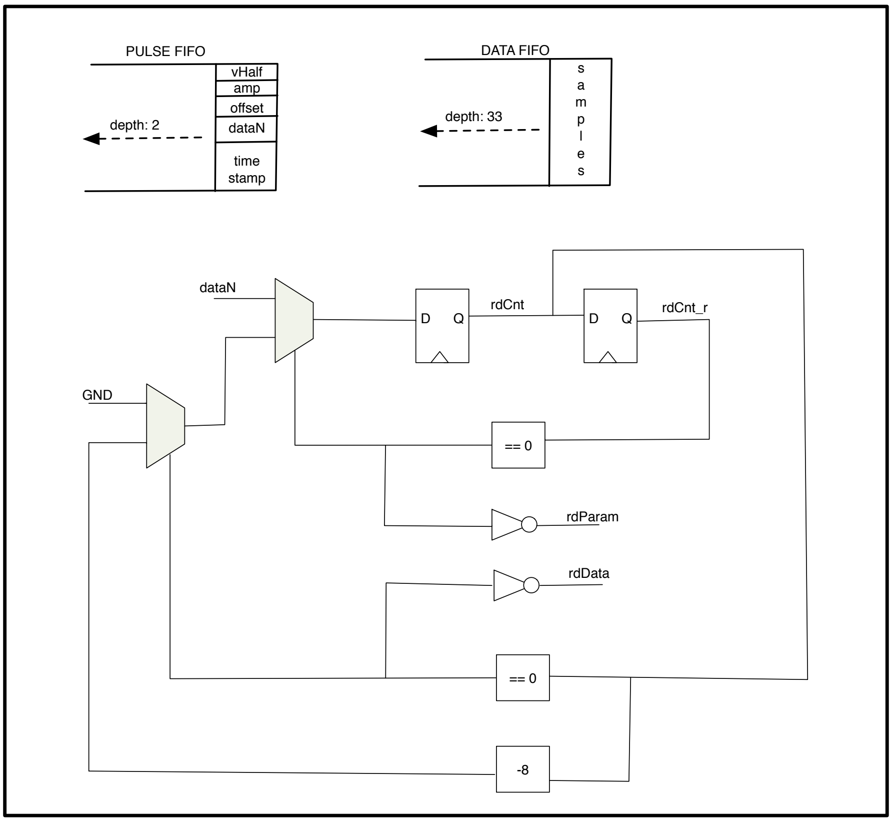
GP receives its inputs from PD. All parameters are passed through the same data bus. The handshaking mechanism is the same as in PD. Whenever the parameters that are output by GP are valid, the output o_GPparamVld is high. The parameters of GP include: timestamp, dataN, offset, t_0, aDivTau, invTau. The data output of GP is the untouched version of the pulse’s sample content.

Figure 4.9: GP macro architecture



GP consists of three main units: Pulse Store, Pulse Process, PostLude. Figure 3.9 is a simplified illustration of the relationship between these units. A bus is defined as the set of data and its handshaking signal, i.e. paramBus is a set of parameters and their valid signal. All parameters share these same handshaking signal. PostLude does not directly receive the sample data since only high-complex arithmetic implementations are included in this unit.

Figure 4.10: Pulse store micro architecture

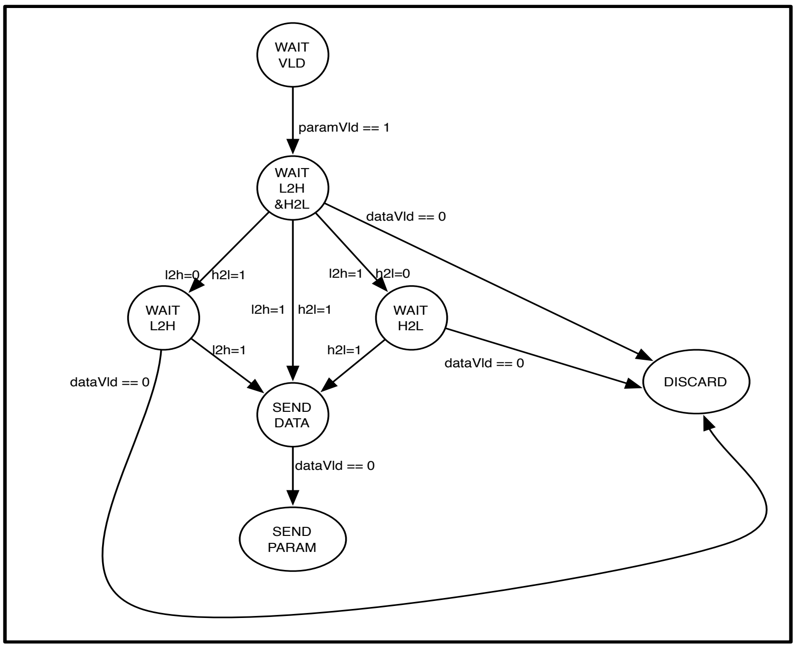


Pulse Store unit has two memory modules : parameter FIFO and pulse FIFO. The parameter FIFO stores the parameters that are received from PD: amplitude, offset, vHalf, number of samples included in the pulse (dataN) and timestamp. The pulse FIFO, on the other hand, stores the 8-sample blocks coming from PD. Why did we have to use memory inside GP? The reason is that GP, unlike PD, can not make the guessing of the final parameters on-the-fly. In order to find t_0, for example, we have to determine where the samples cross vHalf which is done in Pulse Process unit. The width for pulse FIFO is of course 64 (8-samples each of 8 bits) and for parameter FIFO it is 112. (see PD interface for the parameter output width)

How did we determine the depth for these memory modules? Firstly, the pulse FIFO is filled with sample values and after one cycle of delay, the parameters regarding the pulse is written to parameter FIFO. This one cycle is the reason why the depth of parameter FIFO is 2 instead of 1. During that one-cycle, before we start reading from the parameter FIFO, another sample-block regarding the pulse can come from PD. Knowing that the maximum pulse length is 256 which makes 32 slots in data FIFO, we add another slot for the above mentioned cycle and the depth is chosen to be 33. For all practical purposes, these two parameters are also left to the user.

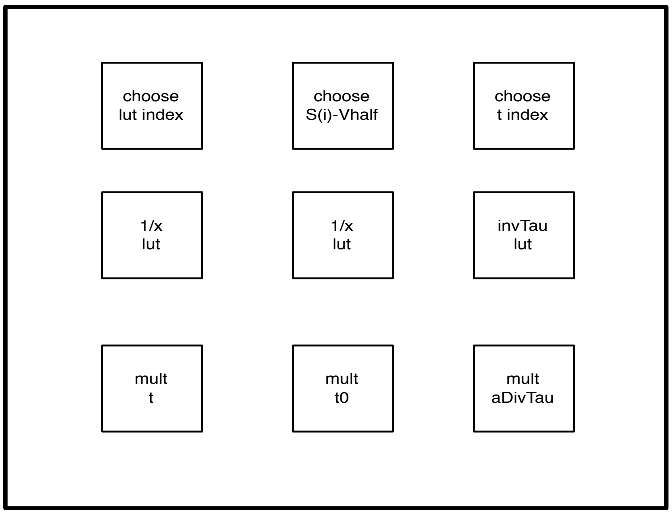
The read logic for these memory modules are also shown in Figure 4.10. If the parameter FIFO is not empty and the previous value of rdCnt is zero, the bit range on which dataN is held is assigned to rdCnt. After one cycle, we read from the parameter FIFO, that is, rdParam is high. The signal rdParam can stay high because the depth of the parameter FIFO is maximum 2. Also, the signal rdCnt is decremented by 8 until it reaches zero and the read enable for data FIFO is high if it is non-zero. This ensures that we read from the data FIFO as many as the number of samples included in the pulse, not more. Note that the value of rdCnt can not be assigned to the next pulse's dataN during the cycles where we read from the data FIFO since the condition depends on the previous value of rdCnt being zero.

Figure 4.11: Pulse process state machine



The purpose of Pulse Process is to find where the crossings of vHalf occur and send appropriate values towards PostLude. Thus a state machine is required as shown in Figure 4.11. Since we do not know whether the pulse is positive or negative, the first crossing of vHalf can be either low-to-high or high-to-low. The state machine is usually idle, waiting for a valid set of parameters from Pulse Store. When this set of parameters is valid, it transits to waiting for low-to-high and high-to-low. At this state, t_h2l and t_l2h values are updated as well as checking for both high-low and low-to-high crossings. After any of the crossings occur, the state is transferred to either wait_h2l or wait_l2h. At these states, only the values (t_l2h or t_h2l) for the crossing that has not yet occurred are updated. In all waiting states, the crossings may not occur until all data is processed. This corner case is handled using a discard state. After the crossings occur, the remaining data is sent during the send data state. When the receiving of all data is finished, the parameters required by Postlude (see parambus in Figure 4.9) is sent and the Pulse Process completes its task.

Figure 4.12: Postlude inner modules



PostLude can be considered a hardcore arithmetic module. All these arithmetic units that are shown in Figure 4.12 is put inside a different module because these computations do not need the sample content (no databus) and the latency can be changed depending on the operating frequency. These units depend on each other and they are basically the implementation of arithmetic equations taken from SystemC code. The RTL for lookup tables are generated using SystemC software which ensures that the design is aligned with system model.

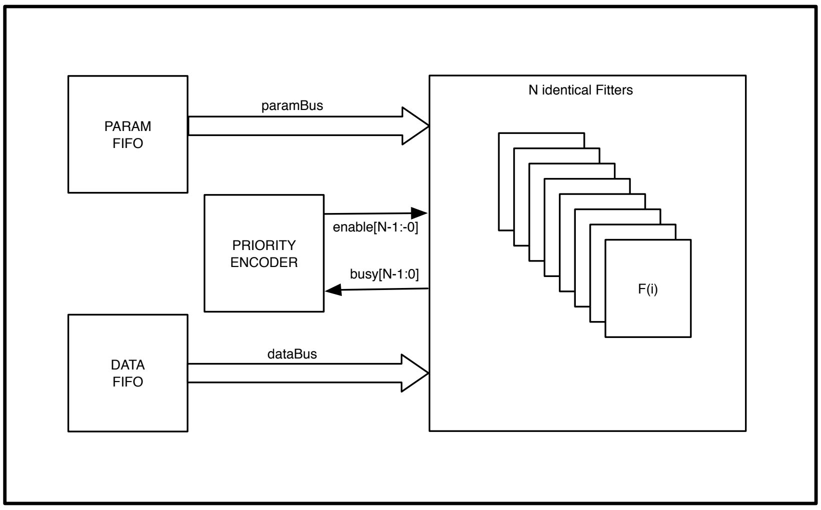
4.4. DF (Dispatcher/Fitter)

Table 4.5: DF interface

Name	Direction	Bitwidth	Source/Destination
clk	input	1	ADC
rst	input	1	ADC
i_DataVld	input	1	GP
i_Data	input	64	GP
i_ParamVld	input	1	GP
i_Param	input	144	GP
o_Full	output	1	RL
o_Vld	output	64	RL
o_Param	output	144	RL

Dispatcher/Fitter has the highest area and complexity amongst all other modules. It receives data and parameters from GP and outputs the valid signal and final parameters of the curve whose fitting process is finished. There is also another signal to report that all fitters are full and DF is unable to accept a new pulse from GP.

Figure 4.13: DF macro architecture

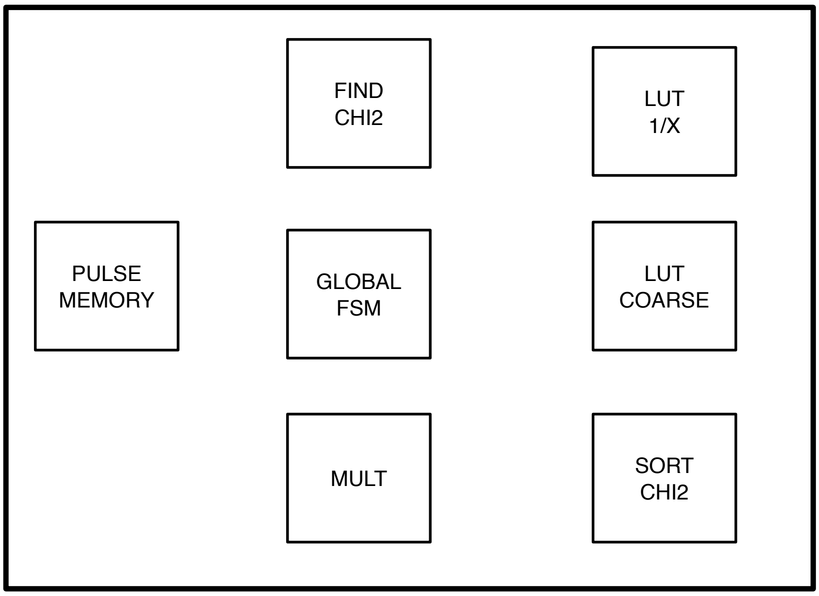


The write logic for both parameter FIFO and data FIFO is simple. Whenever the data and parameters from GP are valid, they are written to these FIFOs. We first read from the paramFIFO because like in GP, we need to know how many sample blocks are going to be read from dataFIFO. As the read count is set to the approprite value stored in the paramBus, we read from the dataFIFO decrementing read count by 8 until it is zero.

The corner case here occurs when the pulse is discarded. (see Figure 4.11) GP sends the data and valid signals while it is trying to find the crossings of vHalf. However, after all the data is sent and still there is no crossing, GP does not send any parameter. While GP is sending the data, they are stored in the dataFIFO that belongs to DF. Thus we have sample-blocks that are useless. How do we handle this situation? One solution would be to modify GP such that it sends an extra signal inside its paramBus indicating that the data that has been written to the dataFIFO is garbage. In order to make GPF efficient from TR to GP, we chose to let DF understand whether the pulse has been discarded or not. Since GP does not send any valid parameter if the pulse data is to be discarded, DF waits for as many cycles as the latency of GP and concludes that discard has occurred if there is no valid parameter data. When DF reaches that conclusion, the redundant data from dataFIFO is read but none of the fitters are enabled.

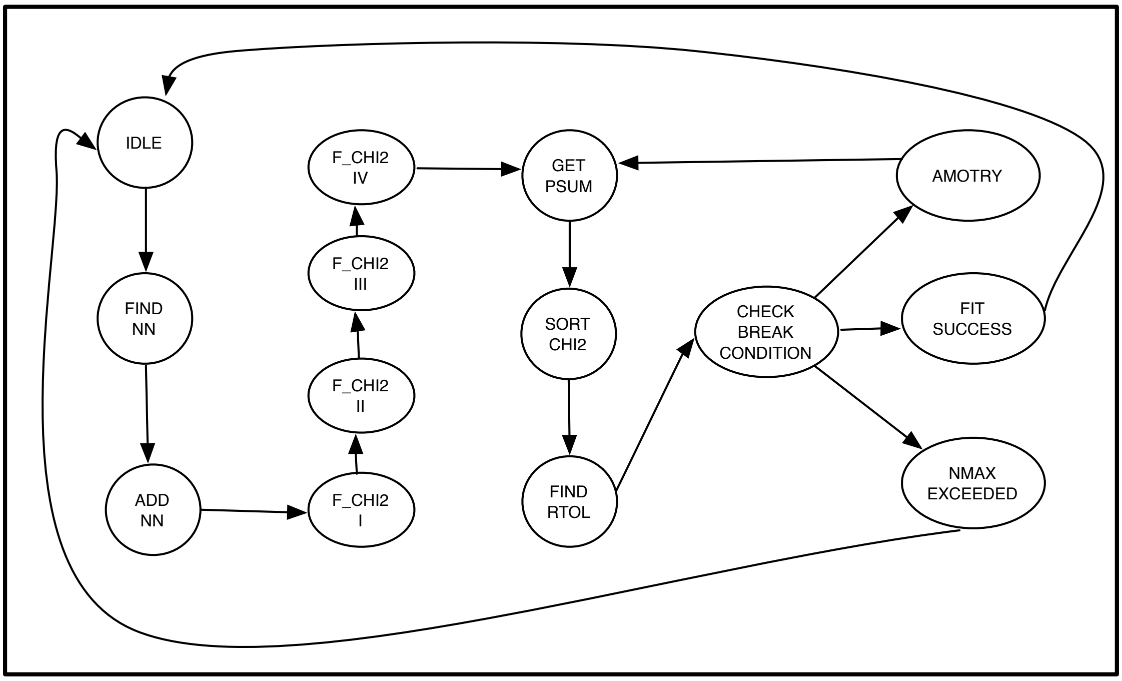
There is also another essential unit called, "priority encoder". The runtime for fitters is indeterminate; thus we need to have a signal from each fitter stating whether it is busy or not. Priority encoder is a pure combinational circuit that receives these signals from fitters and outputs the index of the next fitter to be enabled. At this point, we address another contribution of our thesis : flexibility. The number of fitters can be changed by the user and as the number of fitters vary, the complexity and input/output bit widths of the priority encoder also has to vary. This forces us to make the priority encoder generated by a Perl script. Our script takes N, the number of fitters, as input and generates RTL accordingly.

Figure 4.14: Fitter macro architecture



Fitter is ruled by a global finite state machine. This FSM consists of necessary steps to apply the given curve-fitting algorithm that will further be explained. The pulse data has to be stored in a certain memory whose size is as much as it can hold the longest pulse possible. The reason is that the pulse data is used every time find chi2 is activated. Sort Chi2 is a module needed to sort the chi2 values of the given points. LUT coarse is there to be used during FindChi2 operation. However, LUT 1/x is used both for FindChi2 and other operations. It is actually the same generated RTL that is used in PostLude for GP. (see Figure 4.12) Using only one multiplier per fitter was the core challenge of our design. The inputs to the multiplier change in every multiplication operation, thus there is a long line of cascaded multiplexers and a flip-flop in front of it.

Figure 4.15: Fitter global FSM



Fitter is usually in the IDLE state. It awaits the valid signal coming from the dispatcher's priority encoder so that it can be activated. The first 5 states after activation can be considered as the prelude stage because once these states complete their tasks, there is no return.

Firstly, the leading DC part from the pulse data has to be gotten rid of. The amount of samples that are going to be discarded is named "nn" that is equal to t_0 minus 1. This value requires modification of some of the initial guess parameters (t_0, timestamp, dataN) received from GP. Thus there is another state for that modification. Now we have 3 parameters that define our curve which is only a guess, not as good as the final parameters we aim to find after the whole fitter process. We start applying the curve fitting algorithm due to Nelder and Mead (1965). Our aim is to find the set of parameters that gives us a curve whose chi2 is minimum. Since we have 3 parameters (t_0, invTau, aDivTau), we first find the chi2 for the initial guess and then for 3 others made by adding a constant value to each parameter. That is why we need 4 different FIND CHI2 states. We can think of the chi2 values as one point on the centre (initial guess) and 3 other points around it. This can actually be considered as the initial shape of our amoeba.

Each point of our amoeba actually represents a curve. We compute the sum of each parameter in the GET PSUM state. Thus we have 3 different summations. Note that initially they are p(i)*3+p(i)+step. These summations are required in the latter stage of our algorithm: AMOTRY. After computing these sums, we sort the chi2 values that are initially provided by the prelude stage. This is required by the COMPUTE RTOL state. The value, “rtol”, can be considered as a quantitative measurement of how well we have fit our curve. For this computation, we require both division and multiplication. Since division in hardware is too expensive, we use our (1/x) look-up table and do two multiplications. Note that at this stage, we can not simultaneously find the chi2 because we have only one multiplier per fitter.

Once we have our value to check our curve parameters, we transit to CHECK BREAK CONDITION state. At this point, we can either conclude that amongs the four points we have, the one with the minimum chi2 (note that we have already sorted them) is our final output or we can continue “moving our amoeba” or if we have tried long enough we can conclude that we can not fit this curve. This is what makes the runtime of fitter indeterminate. Obviously, if the value “rtol” is small enough we transit to FIT SUCCESS stage and fitter is done and goes to IDLE. Otherwise we transit to AMOTRY stage. The unfortunate situation would be to reach a certain maximum number of find chi2 calls and discard this set of pulses. At the NMAX EXCEEDED stage, fitter is again done and reverts back to IDLE. The maximum limit is determined by the user.

The stage called, “AMOTRY” includes inner states because there are a lot of if-else conditions. Briefly, we modify our parameters by scaling down or up and compute the new chi2 values. Then we check them against the sorted chi2 values and decide the direction of our amoeba. Note that the less we visit this stage the better it is, because this is where we call find chi2 at least two times.

The runtime for find chi2 is essential it is executed 22 times in average. We had the challenge of implementing a process that uses single multiplier as well as optimizing the latency. This is where loop pipelining occurs. It actually consists of 4 prelude stages that lasts one cycle each. Then for each sample from the pulse, there are 14 cycles of operation. However, we do not have 14 different states for that. If we list these operations such as: op0, op1, .. op14; we pipeline such that while we do op0 for s(4), for example, at the same cycle we do op8 for s(0). The throughput is 0.5, meaning that we finish operating a sample from the pulse in every two cycles. Note that the throughput would normally be improved if multiplier were not our bottleneck. We have two multiplication operations to be performed for each sample and we can not perform two multiplications within a single cycle.

Finally, when fitter is done with the pulse it asserts a valid signal as well as the final set of parameters. This is actually the valid output of the whole DF. After this valid signal, the busy output is deasserted. In the unfortunate cases where the maximum number of find chi2 calls have been reached, the busy output is still deasserted but the valid signal is not asserted.

5. RESULTS AND CONCLUSION

In this section, verification and synthesis results will be provided along with the revisit of our contributions as the conclusion.

5.1. RESULTS

We were able to verify and synthesize our design, namely GPF, starting from TR to GP. The output of TR and PD were 100 percent match with the output of SystemC for both positive and negative pulses. Our sample size includes corner cases such as back-to-back pulses and very short pulses whose length is as small as 8. GP was more than 99.9 percent match with the output of SystemC, the shown mismatch occurred on some pulses that needed to be discarded any way. Thus we consider GPF to be fully verified from TR to GP. Our simulation runs were 8ms with an average pulse rate approximately 1MHz.

We also synthesized our design using Synplify Premier. Xilinx ISE was used to place and route. The type of FPGA we used for mapping was Virtex5 SX95T. The worst slack was 0.015 with a target period of 5.0 ns since our target frequency was 200MHz.

Table 5.1 shows the resource usage in GPF according to our synthesis report. Only 4 out of 244 block rams, 3 out 640 DSP slices and 2165 LUTs are used in TR to GP which consists of less than 2 percent of the available resources in Virtex5 SX95T. Note that PCI-e interface and ADC interface is not included in this synthesis report.

Table 5.1: Resource usage report in GPF

Resource	Usage
DSP48E	3
FD	600
FDE	278
FDR	514
FDRE	620
FDRS	8
FDS	8
FDSE	8
GND	39
MUXCY	31
MUXCY_L	854
RAM32M	29
RAM32X1D	1
RAMB18	1
RAMB36	2
RAMB36SDP	1
VCC	39
XORCY	772
LUT1	355
LUT2	629
LUT3	229
LUT4	232
LUT5	112
LUT6	304
LUT6_E	3
IBUF	1
IBUFG	1
OBUF	210
BUFG	1

5.2. CONCLUSION

In this thesis, we proposed an FPGA design for High Energy Physics Experiments. GPF allows the experiment makers a very efficient platform. Instead of using specialized electronic modules dedicated to calculate a certain parameter, an FPGA card can be used along with a computer.

The OP-PUB concept introduced by this thesis is essential. By combining the functionality of loop pipelining and parallelism, we offer a hardware implementation method to optimize iterative algorithms in terms of resource usage and timing. As described in Section 3.3, our method is generic enough to be applicable in many other implementations. Our design work had the challenge to meet a high frequency, 200 MHz, as well as utilizing the limited resources in FPGA.

Flexibility provided to the user is another contribution of our work. The user can alter the parameters regarding the algorithm such as threshold values and maximum pulse length allowed for each pulse on-the-fly.

Using FPGA instead of CPU was another novel approach proposed by this thesis. The implementation of the curve-fitting algorithm on FPGA is almost 1000 times faster than a computer, which makes our experiment setup cost less as well as providing significant simplicity.

Finally, GPF offers dead-timeless experiment setup. Unlike the electronic modules that are currently used in traditional setups, the system does not get locked up while processing the pulse.

REFERENCES

Güney V.U., 2010. *Triggerless Particle Identification System*. Bogazici University Physics Dept. Master's Thesis, pp. 2.

Akdogan T., 1999. *Flash ADC Based DAQ System Design for the MIT/Bates Compton Polarimeter*. Technical Design Report, MIT/Bates - Blast 9907.

Akdogan T., 2005. *Performance Report and Polarization Results of the MIT/Bates Compton Polarimeter.* Project Report, MIT/Bates - Blast 0505.

Franklin W.A., Akdogan T., Marfuta P., 2000. *A Compton polarimeter for the MIT/Bates South Hall Ring*. Prog. in Particle and Nuclear Physics, 44, 6.

Crawford C.B., Sindile A., Akdogan T., 2007. *Measurement of the proton's electric to magnetic form factor ratio from H(e,e'p)*. Phys. Rev. Lett. 98, 052301.

Hien D.S., Senzaki T., 2001. *Development of a fast 12-bit ADC for a nuclear spectroscopy system*. Nucl. Inst. and Meth. in Physics A, 457, 356.

Bolic M., Drndarevic V., 2002. *Digital gamma-ray spectroscopy based on FPGA technology*. Nucl. Inst. and Meth. in Physics A, 482, 761.

Nicolau C.A., 2006. *An FPGA-based readout electronics for neutrino telescope.* Nucl. Inst. and Meth. in Physics A, 567, 552

Streun M., Brandenburg G., Larue H., Zimmermann E., Zoemons K., Halling H., 2002. *A PET system with free running ADCs*. Nucl. Inst. and Meth. in Physics A, 486, 18.

Giachero A., Guardincerri E., Musico P., Pallavicini M., Ottonello P., 2007. *Design and performances of a multichannel high resolution simultaneous sampling ADC card with on-board data elaboration capabilities*. Nucl. Inst. and Meth. in Physics A, 572, 365.

Gua J.R., You C., Zhou K., 2005. *A 10 GHz 4:1 MUX and 1:4 DEMUX implementted by a Gigahertz SiGe FPGA for fast ADC*. Integration, the VLSI Journal, 38, 525.

Arcidiacano R., Barberis P.L., Benotto F., Bertolino F., Govi G., Menichetti E., 2000. *The trigger supervisor of the NA48 experiment at CERN*. Nuclear Instruments and Methods in Physics Research Section A, vol. 443, pp. 20-26.

Khomich A., Hinkelbein C., Kugel A., Manner R., Muller M., 2006. *Using FPGA coprocessor for ATLAS level 2 trigger application.* Nucl. Inst. and Meth. in Physics A, 566, 80

Haselman M., DeWitt D., McDougald W., Lewellen T.K., Miyaoka R., S. Hauck S., 2009. *FPGA-Based Front-End Electronics for Positron Emission Tomography*. ACM/SIGDA Symp. on Field-Programmable Gate Arrays, pp. 93-102.

Haselman M., Hauck S., Lewellen T.K., Miyaoka R.S., 2009. FPGA- based pulse parameter discovery for positron emission tomography. *IEEE Nuclear Science Symp. and Medical Imaging Conf.,* pp. 2956- 2961.

Koop M.J., Huang W., Panda D.K., 2008. *Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand.* High-Performance Interconnects.

Cuveland J., Rettig F., Angelov V., Lindenstruth V., 2008. An FPGA-based high-speed, low-latency trigger processor for high- energy physics. Proc. *International Conf. on Field Programmable Logic and Applications*, pp. 293-298.

Muller H., Pimenta R., Yin Z., 2006. *Configurable electronics with low noise and 14-bit dynamic range forphotodiode-based photon detectors.* Nucl. Inst. and Meth. in Physics A, 565, 768.

Shimazoe K., Yeol Y.J., Minamikawa Y., 2007. *Development of 40 channel waveform sampling CMOS ASIC board for Proton Emission Tomography.* Nucl. Inst. and Meth. in Physics A, 573, 99.

Liu M., 2008. ATCA-based computation platform for data acquisition and triggering in particle physics experiments. *Proc. International Conf. on Field Programmable Logic and Applications*, pp. 287-292.

Liu M., Lu Z., Kuhn W., Jantsch A., 2011. FPGA-based particle recognition in the HADES experiment. *IEEE Design & Test of Computers*, vol. 28, pp. 48-57.

Nelder J.A., Mead R., 1965. A simplex method for function minimization. *The Computer Journal*, vol. 7, no. 4, pp. 308-313.

Chao L.F., LaPaugh A.S., Sha M., 1997. Rotation scheduling: a loop pipelining algorithm. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 229-239.

Qu J., Zhao R., Liu T., Zhang, 2010. The research of FPGA-based loop optimization pipeline scheduling technology. *Computer and Communication Technology in Agriculture Engineering, International Conference*.

CURRICULUM VITAE

Full Name: Ali BAŞARAN

Address: Hüsrev Gerede Cad. No:18 D:23 Beşiktaş

Birth Place/Year: Istanbul/1986

Foreign Language: English (advanced)

Elementary School: Reşat Nuri Güntekin İlköğretim Okulu (1992-1997)

High School: VKV Koç Özel Lisesi (1997-2004)

BS: Istanbul Technical University (2004-2008)

MS: Bahçeşehir University (2008-2012)

Institute: Natural and Applied Sciences

Programme: Embedded Video Systems – Chip Track

Work Experience: November 2010 – ongoing
Ericsson Microelectronics Design Centre
September 2008 – June 2010
Bahçeşehir University