Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-MS-2017-07

M.Sc. Thesis

Embedded Real Time Partial Discharge Pulse Feature Extraction

Ayush Joshi

Abstract

Partial Discharges(PD) are commonly produced in defects within the insulation systems of high voltage equipment. These discharges are typically nanosecond current pulses in the amplitude range of milli-amperes. A long term exposure of the insulation system to these partial discharges accelerate the aging mechanisms that eventually lead to the final breakdown of the insulation system. Such insulation breakdowns in High Voltage (HV) / Medium Voltage (MV) equipment typically involve arc-flash/fire hazards, posing safety threats. Moreover probable undelivered power and huge financial losses are also associated.

Early detection of PD activity can provide warnings about pending insulation/device failures and hence, maintenance or repair activities can be scheduled before breakdown occurs. Moreover, clustering of PD due to different types of sources is of practical importance as it indicates the severity of defect and provides an insight into the time available for repair activities before complete breakdown. State of the art tools for electrical PD monitoring are expensive and cannot be economically deployed over a large network of HV/MV assets. Moreover, they employ classification schemes based on less robust PD features. This thesis marks the completion of the first stage in the process of building an open source, cost-effective, automated embedded online partial discharge detection tool for feature extraction and PD classification based on new, advanced robust *features* of partial discharges. As an outcome of this thesis, an embedded solution for real time PD detection and feature extraction was developed to facilitate future PD classification.

Embedded Real Time Partial Discharge Pulse Feature Extraction

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Embedded Systems

by

Ayush Joshi
born in Srinagar(Garhwal), India

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Embedded Real Time Partial Discharge Pulse Feature Extraction"** by **Ayush Joshi** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 28-11-2017

Chairman: _____
Prof. Dr. Ir. Rene Van Leuken

Advisor: _____
Prof. Dr. A. Rodrigo Mor

Committee Members: _____
Prof. Dr. Ir. G. J. M. Janssen

_____

Abstract

Partial Discharges(PD) are commonly produced in defects within the insulation systems of high voltage equipment. These discharges are typically nanosecond current pulses in the amplitude range of milli-amperes. A long term exposure of the insulation system to these partial discharges accelerate the aging mechanisms that eventually lead to the final breakdown of the insulation system. Such insulation breakdowns in High Voltage (HV) / Medium Voltage (MV) equipment typically involve arc-flash/fire hazards, posing safety threats. Moreover probable undelivered power and huge financial losses are also associated.
Early detection of PD activity can provide warnings about pending insulation/device failures and hence, maintenance or repair activities can be scheduled before breakdown occurs. Moreover, clustering of PD due to different types of sources is of practical importance as it indicates the severity of defect and provides an insight into the time available for repair activities before complete breakdown. State of the art tools for electrical PD monitoring are expensive and cannot be economically deployed over a large network of HV/MV assets. Moreover, they employ classification schemes based on less robust PD features. This thesis marks the completion of the first stage in the process of building an open source, cost-effective, automated embedded online partial discharge detection tool for feature extraction and PD classification based on new, advanced robust *features* of partial discharges. As an outcome of this thesis, an embedded solution for real time PD detection and feature extraction was developed to facilitate future PD classification.

Acknowledgments

Firstly, I would like to thank Prof. Dr. Ir. Rene Van Leuken and acknowledge his method of providing subtle directions throughout my thesis and not disclosing the *correct way* (as there is none). As the project had to be designed from scratch, Rene's fine sense of an abstracted, high level understanding and deep technological know-how helped me a lot in achieving my thesis goals. Secondly, I extend my gratitude to Prof. Dr. Armando Rodrigo Mor for always being friendly and supportive and showing great willingness for imparting knowledge from his domain. Moreover, I am grateful to Dr. L.C. Castro Heredia, for clearing my doubts from time to time. This thesis work would not have been possible without his considerate efforts.
I would also like to thank my friends Milan, Divyam, Shashwat, Hemanth, Parag , Parul, Uttam and Vishnu for being a constant source of entertainment in this high-pressure Masters life.
Lastly, but most importantly I would express my love and gratitude towards my mother, Dr. Alpana Joshi, father Dr. Pradeep Kumar Joshi and loving sister Lt. Cdr Vartika Joshi for being the three pillars of love and inspiration all throughout this journey. Without their support, none of this would have been possible.

Ayush Joshi
Delft, The Netherlands
28-11-2017

Contents

List of Figures

List of Tables

1 Introduction

"*Electricity is really just organized lightning*"
- George Carlin

Although the late American comic did have a point, one can only imagine the immense amount of scientific research that mankind has put in over the past few centuries, just to solve this not so trivial task of *organizing lightning*.
The electrical insulation sits at the very heart of this *organization*. Along with being a shock prevention measure, it also prevents the formation of electrical contact between parts of electrical equipment that are at different electric potential levels, thus preventing short circuits, fire hazards and subsequent equipment and property damages. Hence, needless to say that their integrity is of utmost importance.
This chapter provides an introduction to the Partial Discharge (PD) phenomena and substantiates why PD detection and monitoring is important. Further, the chapter helps the reader in realizing the motivation behind this thesis. This is followed by defining thesis goals.

1.1 Context

In High Voltage (HV) and Medium Voltage (MV) systems, electrical insulation of equipment are carefully designed in such a way that they can withstand the colossal electrical stress levels that they are exposed to. However, factors such as aging, improper installation, manufacturing defects, environmental damage and third party damage are, if not inevitable, very likely to occur and lead to faults in the insulation. As a result, faults such as gas voids in solid epoxy insulation or bubbles in liquid transformer oil develop. These faults within the insulation have a couple of critical properties.

- Firstly, they have a low electrical permittivity as compared to the surrounding insulation. This implies that they offer much less resistance to the applied electric field and thus, the electric field inside the void would be much higher than the surrounding insulation.
- Secondly, they possess much lower dielectric strength than the surrounding insulation itself.

These two factors together trigger the phenomena known as *partial discharge* (PD). *Partial Discharge*, by definition[3], is a localized electrical discharge that only partially bridges the insulation between conductors. In layman's terms, PD activity is a localized electrical sparking, which can occur at any point in the insulation system where the electric field strength exceeds the breakdown strength of that portion of the insulating

material. Due to the two properties mentioned earlier, one can imagine that the faults in electrical insulation of HV equipment serve as ideal sites for PD activity.
PD is associated with the dissipation of energy in the form of heat, sound, and light. Localized heating from PD may cause thermal degradation of the insulation. This in turn aggravates the fault(s) which leads to even higher voltage stress concentrations developing within the void. It implies that the vicious insulation degradation cycle continues until complete dielectric failure of the insulation, typically accompanied by an electrical explosion and failure of HV equipment. This insulation degradation cycle is shown in Figure 1.1. The colors in the diagram demonstrate the levels of severity of the insulation fault, with green denoting the no fault zone, orange denoting the successive degradation loop and red denoting complete dielectric breakdown and eventual equipment failure.

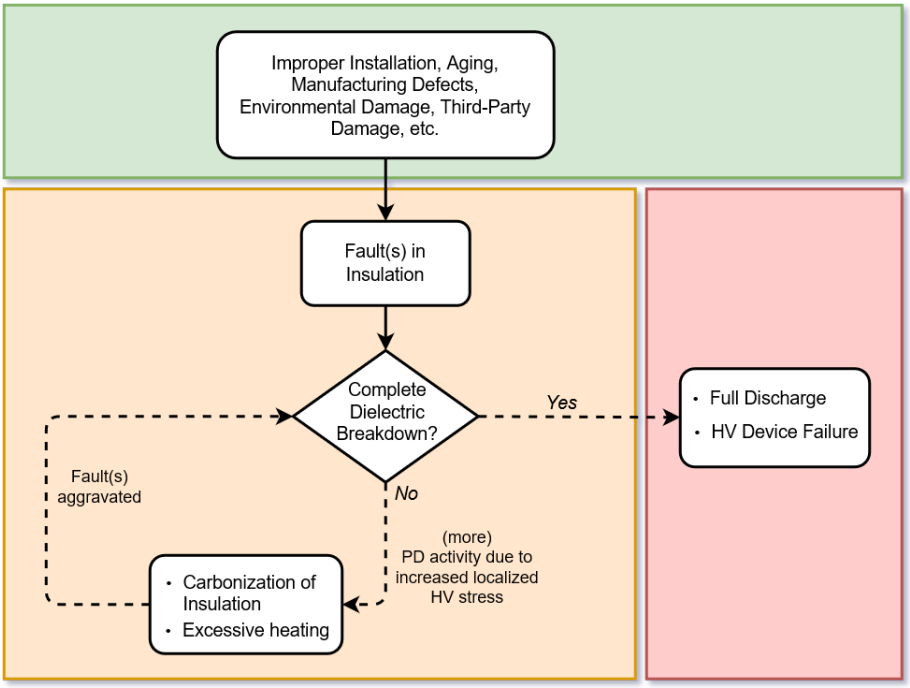

Figure 1.1: HV equipment insulation degradation

It is important to note here that once the PD phenomena starts within the insulation, the insulation only degrades over time (due to more and more PD) and it is just a matter of time when the insulation completely breaks down and the HV equipment fails. In fact, PD is one of the biggest causes of HV equipment failure. Figure 1.2 shows effects of PD on HV/MV equipment.

Figure 1.2: Deteriorating/Failing HV assets due to prolonged PD activity

On the positive note, PD activity can be seen as a clear indicator of asset deterioration and early detection by monitoring PD activity can be vital in preventing pending device failures. Following points further substantiate the need for PD monitoring:

- Due to the fact that complete dielectric breakdown of insulation in HV equipment is typically associated with arc-flash/fire hazards, early detection of PD ensures safety of workers in HV/MV power stations.
- Unexpected failures are associated with huge financial losses along with undelivered power for a long time (if the failing equipment is not redundant in the HV system).
- Because these HV/MV assets are typically extremely expensive, it provides huge savings for the plant owners as unnecessary asset replacement can be avoided.
- Useful for quality assurance at the time of commissioning of a new plant.
- To provide a direction to the maintenance team of a HV/MV plant in efficiently targeting pending failures.

From the discussion above, it becomes needless to stress that monitoring of PD is extremely important for timely remedial, repair or replacement actions to be taken. Along with PD detection, it is also of huge practical importance that the source of the PD can be classified. Classification of sources into possible PD sources like *Internal Discharges*, *Surface Discharges*,*Positive Corona Discharges*, *Negative Corona Discharges*, *Floating Particle Discharges* etc. facilitates the PD source identification process. As PD activity has direct correlation with the dielectric insulation's aging process, PD source

identification becomes extremely important for insulation condition monitoring. The presence of PD activity in a particular equipment due to a particular source indicates the substantial risk (severity level) of a pending insulation failure [4]. For instance, a *corona* discharge may imply that there is actually no defect in the insulation itself. However presence of discharges like *internal discharge* is more severe and may imply the need of urgent remedial actions to be taken [5]. Further, such a classification is integral in eliminating noise from actual PD pulses which can subsequently be analyzed for defect recognition [6]. Moreover, while commissioning of a new power plant, it is important to check for possible PD activity due to all these sources to make sure that the equipment are in a healthy state.
Thus, PD detection and its *classification* into different types of *PD* sources to facilitate defect recognition is crucial for ensuring *safety, stability and reliability* in HV/MV electrical power systems.

1.1.1 The classification process

The PD classification process is as shown in Figure 1.3. As can be observed, the entire

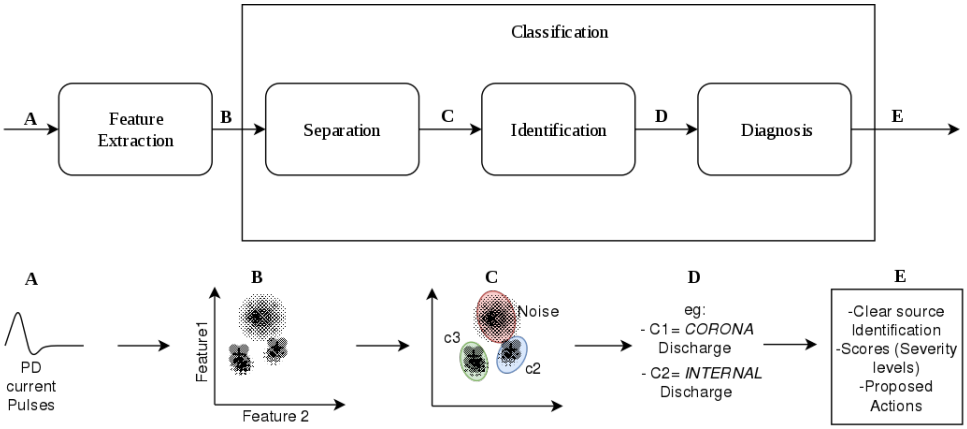

Figure 1.3: The classification process

PD classification process depends on *feature extraction*. Further, classification accuracy is only as good as the features employed for classification.
Once the features are computed, PD separation (clustering) can be performed, for instance using density based clustering algorithms. The output of separation stage are clusters where legit PD sources and noise are separated. Further, for each of these clusters, identification source *identification* is performed based on techniques like Phase Resolved PD pattern recognition [7, 8]. Finally, knowledge based diagnosis is done which results in clear source identification, along with the severity levels of insulation fault. Also actions (remedial,repair) are proposed at the output of this stage.

1.1.2 PD detection and monitoring techniques

PD events are complex mechanisms involving rapid electron avalanches and as such, it is unfeasible to measure them directly. What can be measured though, are the changes

in certain variables excited by PD. These variables can then serve as *features* for PD classification. After the PDs are separated into distinct classes/clusters, source identification can be done, for instance by using the phase resolved PD patterns (PRPD) [7, 8]. The classification can be based on electrical, chemical, mechanical, acoustic or optical features of PD[9].
Conventional electrical detection methods described in the standards. IEC 60270 [3] are widely used for conducting industrial and research oriented PD testing. However, these methods suffer from a major drawback that they offer limited bandwidth which is not wide enough to resolve the shape of PD pulse in time. But the shape of the PD pattern is one of the integral features for PD classification [10]. To overcome these limitations, unconventional methods with bandwidths lying in several MHz are frequently used for PD testing in cables, transformers, etc [11, 12]. Unlike conventional measurements, since there are no recommendations given by any standard for acquisition of PD signals using unconventional measurements, acquisition parameters like sampling frequency, vertical resolution, acquisition time, etc. are arbitrarily set by the user [13]. The classification map [14] is a widely used, state of the art tool in the domain of unconventional PD measurements for PD classification. It involves calculations of equivalent time ($\mathbf{T_{eq}}$) and equivalent Bandwidth ($\mathbf{W_{eq}}$) which serve as the PD discriminatory features of interest.

1.2 Motivation

The commercially available PD monitoring solutions suffer from a number of issues as described under:

- Expensive : Commercially available PD monitoring solutions are extremely expensive. To understand the dynamics of this problem, lets highlight some facts in discussed in [15, 16].

  - Almost 100 % of the distribution of power in the Netherlands is realized by means of underground MV cables, cable joints and transformers. Here, the total MV cables' length is approximately 100,000 km and a significant part of the investment cost of the distribution network is spent on this MV cable network.
    When the total power grid is considered, a major part of power-delivery outages can be attributed to the faults in distribution power grid. Also, a huge majority of the distribution grid outage times is due to failures in MV cables. Further, a plethora of references show that more than 70% of the breakdowns in MV cable network are caused by internal defects in the insulation system of the cable[17, 18].
    Most parts of The Netherlands' distribution grid infrastructure were constructed more than 30 years ago. *Due to regulations of the energy market, asset managers are forced to reduce costs and postpone investments, while maintaining the reliability of power delivery.* Because of the ever increasing demand for electricity along with aging infrastructure, the asset managers

have to employ various maintenance and replacement strategies by using tools to predict pending asset failure.

Given the fact that the state of the art tools are extremely expensive, they cannot be *economically* deployed over different assets of a power plant or a distribution grid. Thus, this becomes a serious concern for grid *reliability*.

- Lack Robustness : Widely used unconventional PD source clustering tools like that of the *classification map* are shown to be significantly influenced by signal to noise ratio and the user specified acquisition parameters like sampling frequency, number of samples, acquisition time and vertical resolution of the `PD` acquisition system [19]. Thus, although features like T_{eq} and W_{eq} are supposed to show significant differences for different `PD` sources and should form a single cluster for a single source, they might end up forming a single cluster for distinct sources due to their non-robust nature under the influence of changing user acquisition parameters in noisy conditions. Since the *acquisition parameters are set by the user* the chances of clustering errors are considerable.

- Lack Automation : Most available solutions present today lack automation. Experts (human intervention) in the field of PD are required for diagnosis. Considering the case of monitoring of the 100,000 km length of MV cable network in The Netherlands, the desirability of a fully automated inexpensive solution can be perceived.

- Closed Source : Commercially available PD measuring and source recognition tools are closed-source. This limits widespread industrial application and research of partial discharge as a diagnostic tool.

New, more reliable *features*: Current peak ($\mathbf{I_{peak}}$), Estimated charge($\mathbf{Q}$) and energy($\mathbf{E}$) for classification have been identified and a bench-marking test platform has been designed (section 2.1) by researchers at TU Delft. However, the platform, as shall be discussed in Chapter 2, involves oscilloscope based PD detection and acquisition and bulky circuitry. After acquisition, feature extraction and classification are performed *offline* using `MATLAB`.

1.3 Thesis Objective

- This thesis is an important step towards the goal of an open source, cost-effective, automated embedded online partial discharge detection tool for feature extraction based on new, advanced reliable *features* of partial discharges.

1.4 Contributions

- Developed an *elegant* FPGA based *streaming* architecture and realization which can *d*etect partial discharges and extract reliable *features* in *real time*. This *real time* property opens multiple avenues for reliable future PD classification.

- Integrated provision in hardware for estimating PD *feature* extraction using computations in two domains i.e, *frequency domain* and *time domain*.
- Developed and integrated a basic standalone software application (proof of concept) to configure the hardware sub-system (user inputs) and view results.
- The features extracted in hardware have been validated against MATLAB based reference outputs.

1.5 Methodology

The first step was the identification of the requirements for efficient PD detection and feature extraction, based on related literature. Further, the underlying hardware (`Red-Pitaya`) was analyzed and architectural constraints and comprehensive specifications were identified. Based on the specifications, an elegant *packet processing* based *streaming, real-time* hardware architecture was devised using High Level Synthesis (`HLS`). The design and verification process was intertwined for all functional blocks employed in the architecture. To complement designed hardware, a standalone `C` based application was developed which can be used to configure the hardware with user inputs. The hardware software co-design was also tested on an FPGA based platform (`ZYBO`) as a proof of concept.

1.6 Report Organization

The remainder of the thesis is structured as follows:
Chapter 2: The chapter begins with an immediate background of this thesis work. Further, the chapter provides an in depth analysis of requirements for the designed embedded solution, followed by comprehensive discussions about architectural alternatives.
Chapter 3: A bird's eye view of the designed system is provided in this chapter and packet processing approach is introduced.
Chapter 4: This chapter provides detailed implementation details. Further, by taking an example test case, all functional blocks are verified and functionality of each block is demonstrated.
Chapter 5: This chapter provides detailed testing schemes employed. The chapter goes on to evaluating the designed system based on latency, throughput and utilization.
Chapter 6: This chapter covers limitations of the devised system and provides a view of future works required. The chapter goes on to summarize conclusions from the thesis work.

Related Literature and Pre-Implementation Analysis

2

Before any design implementation, detailed *specifications* have to be drafted. These specifications are effected by the design *requirements* and target *hardware specifications*. The requirements are realized by thorough understanding of related literature and realizing the gaps that have to be filled with the new design.
In previous section, it was emphasized that PD *detection* and further *classification* to facilitate *source recognition* is crucial for ensuring safety, stability and reliability in HV/MV electrical power systems. Also, the lack of robustness to changing acquisition parameters and noise influences in classification features $\mathbf{T_{eq}}$ and $\mathbf{W_{eq}}$ used in state of the art unconventional PD measuring tools was recognized. Thus, there is an evident need for new, more robust features for PD separation. Finally, the need for an automated, affordable, open-source online PD monitoring solution based on more robust classification features was identified.
This chapter presents a careful amalgamation of related literature and the process of *realizing* requirements based on this literature. The outcome of this chapter is a set of *Design Specifications* required to for an automated system for PD detection and feature computation for facilitating PD classification.

2.1 PD Test Platform

A PD test platform [6] (Figure 2.1 and Figure 2.2) facilitating unconventional PD testing and measurements was introduced in the High Voltage Laboratory of Delft University of Technology.
This platform, is an excellent setup for bench-marking the performances of experimental PD signal processing algorithms for their classification or PD source recognition proficiency. The platform as shown in Figure 2.1 supports six electrode samples carefully designed to emulate six most common sources of partial discharge pulses i.e *positive corona*, *negative corona*, *surface discharges*, *internal discharges*, floating electrode and a *free moving particle* discharges. As any unconventional PD measuring system typically require a combination of **wide-band sensors**, **fast acquisition systems** and digital signal processing amalgamated with **Classification tools** for carrying out PD measurements, the test platform as shown in Figure 2.1 also provides all these features as described.
The discussion following feature description is critical in making the design decisions that were subsequently taken, as will be discussed further in this chapter.

- **Wide-Band Sensor**: The platform involves a high frequency current transformer (HFCT) as the sensor of choice for sensing the current of PD pulses. The sensor offers a bandwidth from $34.4kHz - 60MHz$ when terminated into a 50 Ω resistor

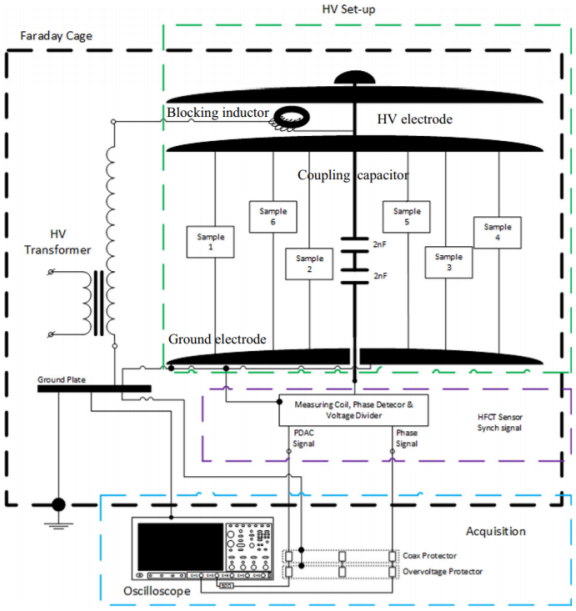

Figure 2.1: Scheme of partial discharge test platform

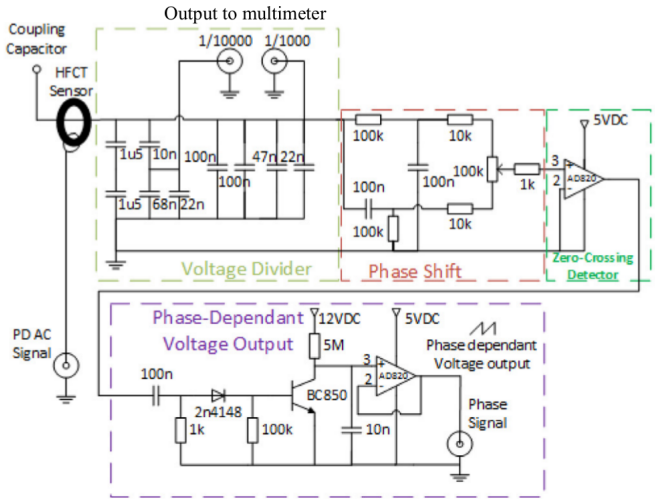

Figure 2.2: Scheme of partial discharge test platform (Elaborated)

at the oscilloscope. The frequency response of the sensor thus obtained is as shown in Figure 2.3.

– PD occurs when electric field is applied to the faulty insulation. Therefore, the energizing electric field and PD pulse get superimposed. Thus, it is required by the measuring platform to have a frequency response as a high pass filter with cutoff frequency high enough to filter out the energizing signal and low enough not to distort the PD signal is needed[20]. As can be seen, the sensor's frequency response limits the upper cutoff frequency of the complete

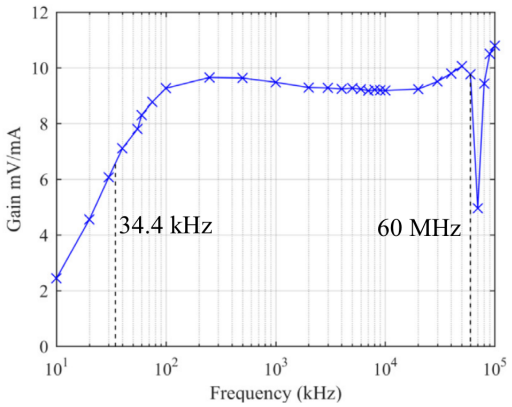

Figure 2.3: Frequency response of HFCT sensor

measuring system to $60MHz$, rendering a band-pass filter behavior.

- **Fast Acquisition System**: The setup features a high performance oscilloscope (`Tektronix DPO7354C`) with 8 bits of vertical resolution and a maximum sampling frequency of 40GS/s. Moreover, the oscilloscope features a 'Fast Frame Acquisition Mode' to allow PD acquisitions with trigger rearming below 1us. This oscilloscope is capable of storing a maximum of up to 50,000 PD pulses at 250MS/s. Along with PD pulse acquisition, the oscilloscope also records a synchronization signal to calculate the phase of a PD trigger event with respect to the electricity mains to facilitate PD recognition using phase resolved PD patterns (PRPD)[7, 8].
  - In [21], it is suggested that a uniform step quantization would not represent adequately an analog signal with non-uniform amplitude distribution (as in the case of PD pulse) and white noise gets added in the digitization process of ADC. Thus, a vertical resolution of more than 8 bits will be desirable.
  - The Fast Frame Acquisition mode implies that if two PD pulses are narrowly spaced in time, the oscilloscope can avoid missing of the second pulse, thus, improving the *pulse resolution*[3] in time. For instance, pulses coming from a *corona* source can be narrowly spaced and then, this utility comes in handy [22]. Moreover, this ability of the oscilloscope gives and additional advantage that the pulse *repetition rate* can also be derived with a high degree of precision. The repetition rate can be a parameter of interest because of two reasons. Firstly, the repetition rate can be correlated to the severity of the insulation defect. Secondly, the repetition rate can be an excellent parameter to segregate actual PD pulses and noise signals[23]. Since, the main technical limitation of online PD monitoring systems is the high probability of false indications of harmful PD detection due to noise [23], repetition rate can be a a vital in PD clustering and hence, having a high pulse resolution is desirable.
  - Apart from the PD pulses, the oscilloscope receives a Synchronization signal (Figure 2.4) to facilitate further PD identification using PRPD. This signal is a phase dependant voltage output in the form of a ramp signal which increases from one zero-crossing of the sinusoidal wave (mains) to the other.

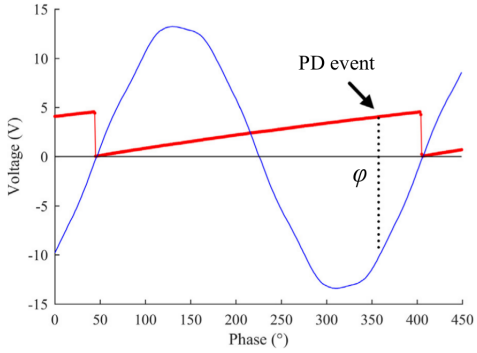

Figure 2.4: Phase dependent voltage output derived from test setup

However, to produce this output, a group of analog circuit blocks are required, namely, the Phase shift block (to correct shift in phase due to voltage divider block), a zero crossing detector and a phase dependent voltage output generator, as shown in Figure 2.2.

- The limitations of acquisition unit can potentially limit the upper-cutoff frequency of the measuring system. However, in the case of this test setup, the limiting factor (for upper cutoff frequency) is the PD sensor (as the oscilloscope offers sufficient sampling frequency support up to $40Gs/s$).
- This high performance oscilloscope costs $40,500 [24].

- **Classification Tool** : The digitally stored PD pulses are available for feature extraction and PD classification. Separation parameters I_{peak}, Q and E are extracted using algorithms in MATLAB. Further, clustering is done using I_{peak}QE clusters [22]. Finally, the source of the PD is recognized using PRPD recognition technique.
  - **In [22], fundamental PD features as that of peak current ($\mathbf{I_{peak}}$), the apparent charge (Q) and energy (E) were explored for their ability to differentiate between different sources of PD. The results of this exploration proved that these basic features are suitable for separation of sources if the PD pulse shapes are different. Moreover, clustering based on $\mathbf{I_{peak}}$QE clusters was proved to be independent of changing acquisition parameters, in contrast to state of the art unconventional measuring systems, for which these changes are relevant. Thus, in this context, $\mathbf{I_{peak}}$QE have proved to be more robust.**
  - The parameter extraction and classification related digital signal processing operations are performed offline in `MATLAB` on the PD pulse data logged by the oscilloscope. Our final aim, however, is a PD monitoring tool which completes the parameter extraction and classification task online, i.e while the high voltage device is working and is powered on.

Thus, we now have a complete platform for PD detection and acquisition facilitating feature extraction and further classification. Our `MATLAB` algorithms can extract fea-

tures Q,E and I_{peak} which are more robust features for PD separation than the ones used in the state of the art(T_{eq}, W_{eq}). Further, another set of `MATLAB` algorithms can classify PD based on its source (one of the six samples in the test setup). However, still, from the discussions above, few shortcomings are evident in the test setup shown in Figure 2.1.

- The setup lacks automation. The oscilloscope saves PD samples which are subsequently utilized for feature extraction and further classification performed offline using `MATLAB` on the stored samples.

- The acquisition unit consists of an extremely expensive oscilloscope, which is certainly not in harmony with our vision of an affordable PD detection and classification solution.

- The setup has analog circuitry for phase shift phase-shifting, zero-crossing detection and phase dependent voltage output generation and removing it is desirable for cost effectiveness of the final system.

Thus, it is required to have an **affordable** *embedded solution* which can act as a **replacement of the oscilloscope** from the Figure 2.1. Further, it is also desirable for this magic embedded solution to have **feature extraction** and classification possibilities built in, which will bring about the much needed automation.The classification possibilities, however, are **NOT** in the scope of this thesis.
In the forage for the best embedded solution to get the job done, one readily available FPGA based platform stands out - The *Red Pitaya*.

2.2 About Red Pitaya

Red Pitaya is a popular hardware platform which is developed to be alternative for expensive laboratory measurement instruments. The main attraction for choosing Red Pitaya are its 2x fast analog inputs *on-board* which provide a sampling rate of 125Msps each, which is desirable for our project. Further, the Red Pitaya is way less expensive (*259 euros starter kit*) as compared to the *40,500 euro* oscilloscope discussed earlier. Thus, if the required functionality is successfully realized using Red Pitaya, the cost to benefits ratio would be extremely high.

2.2.1 Zynq 7010 SoC

The platform offers immense compute capabilities as it possesses a ZYNQ 7010 SoC on board. As shown in Figure 2.6, the SoC consists of two main parts. The Processing System (`PS`) and the Programmable logic (`PL`). The PS is centered around Dual Core ARM Cortex A9 processors. The PL on the other hand is an FPGA fabric and there exists numerous high performance interface for PS-PL communication.

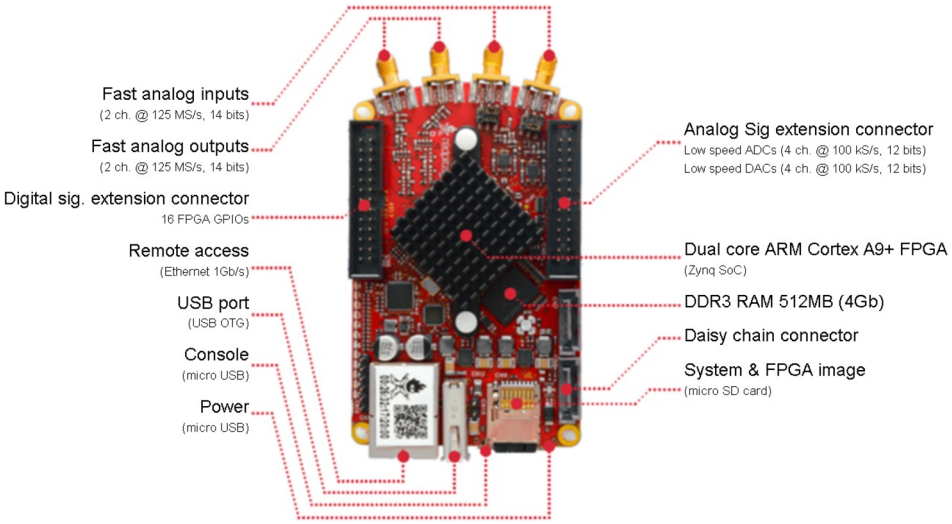

Figure 2.5: Red Pitaya Overview

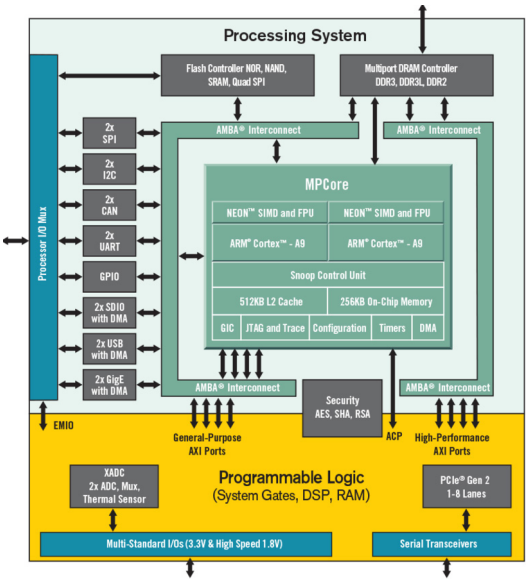

Figure 2.6: ZYNQ 7010 SoC Overview

2.3 Scope of thesis

The final required system for PD feature detection (excluding classification) should look like the one shown in Figure 2.7
However, this thesis marks the completion of the first stage of the project which involves:

- Creating a hardware framework of a group of functional blocks (IPs) within the PL of ZYNQ 7010 SoC which can handle ADC samples at required sampling rates, compute required output parameters (using frequency and time domain calculations) in real time and store the parameters in memory (DRAM).

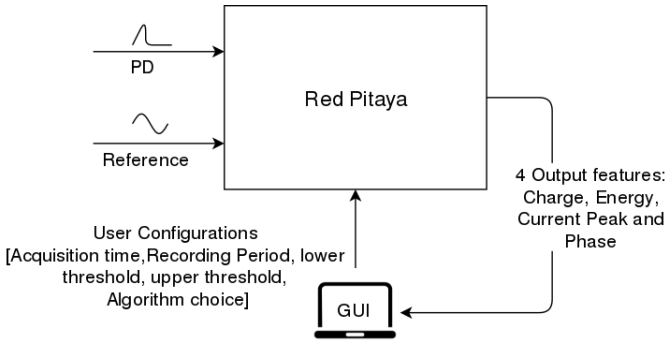

Figure 2.7: Scheme of user requirements from the finished feature detection system.

- Creating a bare-metal software application (`PS`) to facilitate user in providing input configurations along with the control signals to the `IP`-subsystem for transferring data from the `PL` to the `PS` side of the SoC.

NOTE: In the entire course of this thesis, there is no notion of data transfer over ethernet between measuring platform and the remote user using a dedicated GUI and retrieving the required output parameters back over ethernet to the user's GUI as shown in Figure 2.7. Moreover, no actual ADC inputs are employed. Instead ADC emulators are used for final on-board testing, as a proof of concept. Thus, all the data is inside the embedded solution at all times and there is no connection with inputs from outside the embedded solution. The thesis is concentrated on how computations are done, data is transferred (within the `IP`-subsystem and between PS and PL) and stored **within** the platform.
From here onwards, design requirements will be identified one by one and the thorough analysis done in order to make design decisions and retrieve design specifics based on these requirements will be presented.

2.4 High level platform Requirements

1. **Input Impedance**: The Red pitaya features 2 Fast ADC input channels. Two ADC inputs are required, one for acquiring the PD pulses (`ADC0`) and the other to acquire the synchronization signal(`ADC1`) to extract present phase information of any PD event with respect with the 50Hz electricity mains. Both ADCs of Red Pitaya offer an input impedance of $1M\Omega$.

   (a) As `ADC0` is responsible for acquiring nano-second PD pulses, it is required to connect a 50Ω impedance in parallel to this port (implying an effective impedance $=50\Omega$). If the default 1M impedance is used , the analog input would not be able to resolve the PD in time (due to high value of RC time constant).

   SPECIFICATION: An input impedance of 50Ω should be connected

in parallel to `ADC0`.

(b) `ADC1` is responsible for gathering phase information from synchronization signal. As the voltage divider in Figure 2.2 is capacitive, it is required that `ADC1` offers a high input impedance so that there is no phase shift. Thus, Red Pitaya's $1M\Omega$ impedance is perfect for phase signal acquisition.

2. **Synchronization signal:** Red pitaya, unlike the oscilloscope, has immense compute capabilities. Thus, the following specification was drafted.

   SPECIFICATION : The output sinusoidal signal from voltage divider stage Figure 2.2 should be directly fed to `ADC1`. Phase detection using positive zero crossing detection and maintaining internal counts should be done locally on the FPGA.

   This will get rid of the bulky ramp generation analog circuit as shown in Figure 2.2 and will be an important step towards *affordability* of the final system.

3. **Analog to digital Converter(ADC) Sampling Frequency** :

   (a) The PD sensor (HFCT) offers a bandwidth between 34.4KHz and 60MHz when terminated at 50Ω input impedance of the measuring device. The PD pulses can have frequency components anywhere in this bandwidth (and even outside). Hence, for acceptable reconstruction of analog PD pulses (for our algorithms), an ADC sampling rate should be a minimum of 120Msps according to *Nyquist Criterion.*The *Red Pitaya* offers a sampling rate of 125Msps which is sufficient to resolve PD pulses in time.

   SPECIFICATION: `ADC0` Sampling Rate of 125Msps should be chosen for PD acquisition.

   (b) As will be pointed out later (section 2.5), the phase resolution required is 0.5 degrees. This implies that 720 points have to be detected in every 50Hz (20ms) sinusoidal reference cycle. This corresponds to a sampling rate requirement of $\geq 36Ksps$. Red Pitaya's ADC default is 125Msps. For the sake uniformity, this sampling rate was chosen (and down-sampling was done within FPGA functional blocks (`IP`s) internally).

   SPECIFICATION: `ADC1` Sampling Rate of 125Msps should be chosen for reference sinusoidal signal acquisition.

4. **ADC Range**: Naturally, one of the requirements is that the white noise introduced in the process of PD acquisition due to digitization from ADC should not

lead to a high degree of errors in feature computation that are outside the allowable error limits (described in section 2.5).
Red Pitaya can be configured with one of the two voltage available ranges i.e. *High Voltage Range* (±20V) and *Low Voltage Range* (±1V) using jumpers as shown in Figure 2.8.

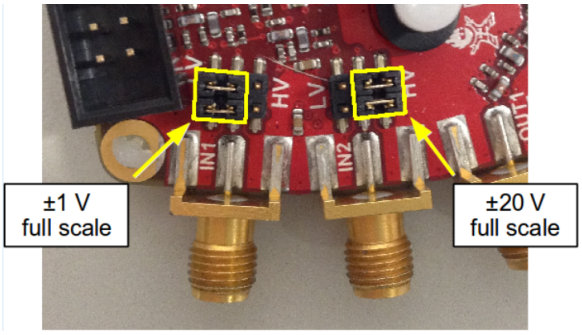

Figure 2.8: Jumper settings for fast ADC inputs of Red Pitaya

PD pulse Acquisition (`ADC0`) :
A test set of pulses recorded by oscilloscope at the High Voltage Laboraory (TU Delft) at 200Msps is shown below.

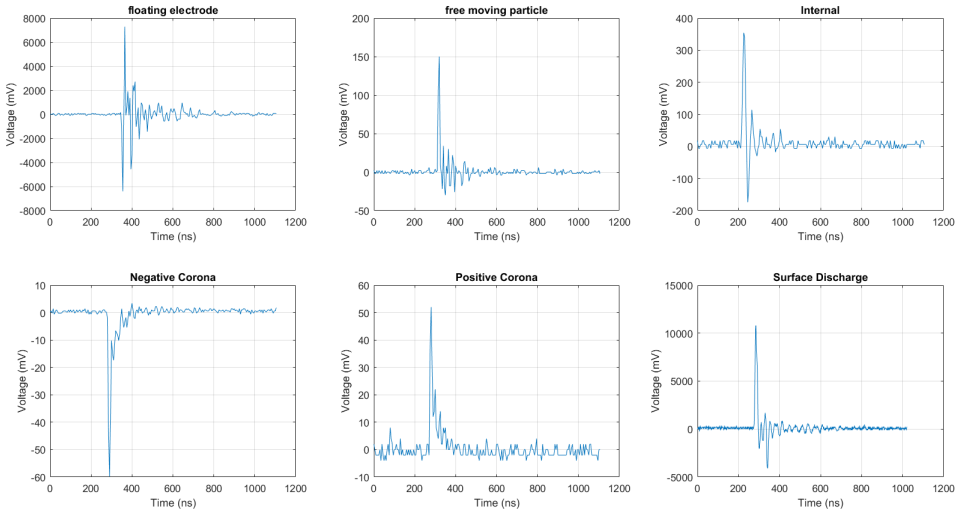

Figure 2.9: A PD pulse test-set recorded at at 200Msps

As can be noticed in Figure 2.9, the voltage ranges may vary to a significant extent. Further, the voltage ranges desirable to be recorded may be well beyond the ±1V (eg, *Surface Discharge*, *Floating Electrode* in the test set shown in Figure 2.9). Thus, as far as ranges are concerned, HV setting of Red Pitaya is a better option. However, a couple of points should be considered before selecting a jumper alternative.

(a) The data-set of PD pulses demonstrated in Figure 2.9 is not a typical representation of PD pulses. In fact, the notion of 'typical' PD pulses itself is invalid. Due to the stochastic nature of PD, significant variability in pulse properties like shape and amplitude can be expected in every measurement. For instance, the shape of PD depends on the distance between PD source and sensor. Thus, in case of PD measurements in cables, the PD at the source (defect) will experience attenuation, distortion and elongation as it travels across the length of the cable. Therefore, depending on where the sensor is installed, it will detect variable voltage ranges (for the same defect) and so will the ADC of the measuring instrument (*Red Pitaya* in this case). As a note, all the pulses shown in Figure 2.9 can also fall in ±1V for a different test set.

(b) Apart from voltage range, the other important factor to consider is precision of the ADC. *Red Pitaya* offers a 14-bit ADC which implies that its chosen (HV/LV jumper setting) input voltage range can be represented in 2^{14} (i.e. 16384) distinct levels. Thus, in LV($\pm 1V$) jumper setting, the resolution of ADC is $0.12207\,mV$. Further, HV($\pm 20V$) setting offers a resolution of $2.44mV$. A `MATLAB` based simulation to calculate the effect of ADC resolution on charge (one of the output features) of positive corona PD shown in Figure 2.9 was done. The results are as follows:

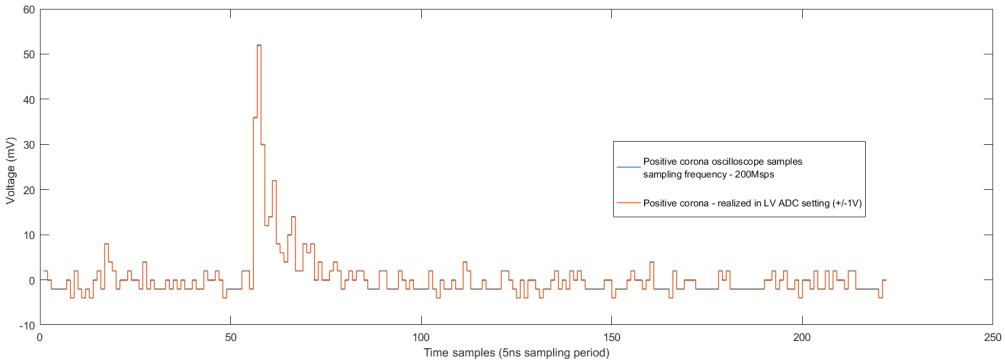

Figure 2.10: ADC interpretation for LV mode

Due to limited resolution of the ADC in HV mode, quantization errors in Figure 2.11 are clearly visible. In contrast, the LV mode (Figure 2.10) is able to fairly reconstruct the minute changes of the positive corona. Finally, a feature of interest, apparent charge (Q) was calculated for the oscilloscope's logged data, and the ADC interpretations in HV and LV settings (time domain calculations : section 2.5 Integral between zero crossings (on either side of peak) of filtered current pulse). With the charge estimation from oscilloscope's logged pulse taken as reference, the percentage error for LV setting was 2.2%. On the other hand, the estimated charge of pulse in HV setting was **35.1%**, which is unacceptable, rendering the feature(Q) useless.

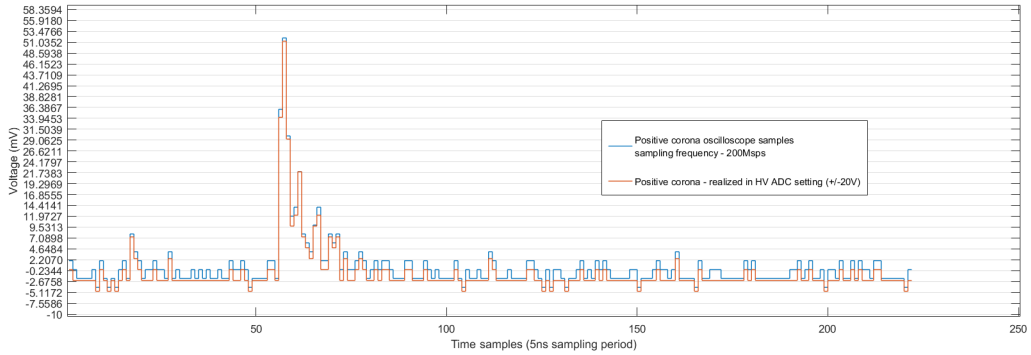

Figure 2.11: ADC interpretation for HV mode

Thus, based on the discussion and experiment above, an important design decision was taken. The Low voltage ADC setting ($\pm 1V$) was chosen for `ADC0` (PD acquisition). Thus, although, some PD pulses (exceeding $\pm 1V$) will be clipped off due to limits put on `ADC0` range, those pulses can be **discarded** before the classification stage. This discarding can be based on peak current value which is another feature to be computed. If the computed peak current for a pulse is **exactly $\mathbf{V_{peak}}$/PD sensor gain** ($\pm 1V/9.1 = \pm 109.8mA$), there is an extremely high probability that the pulse was clipped. Hence, it can be chosen to discard the output features corresponding to the pulse. Hence, some pulses from the input data-set may be lost but it will be made sure that the computed features have errors within allowable limits. In contrast, if HV setting is chosen, although a bigger range of input PD voltages can be covered, there is a significant probability for computed parameters being wrong (for PD pulses with small current values) due to lack in precision. Moreover, there will be no way to detect if computed parameters are erroneous or not before classification.

SPECIFICATION : Low Voltage jumper setting ($\pm 1V$) should be chosen for `ADC0` (PD acquisition)

Reference synchronization signal Acquisition (`ADC1`) :
From the test setup shown in Figure 2.1 and Figure 2.2, it is easy to retrieve the sinusoidal signal for PD synchronization in a $\pm 1V$ range. Moreover, as we are only interested in the (positive) zero crossing points of the sinusoidal reference and not the actual amplitude of the reference signal, the Low voltage setting of ADC can be safely selected for reference signal acquisition.

SPECIFICATION : Low Voltage jumper setting ($\pm 1V$) should be chosen for `ADC1` (reference acquisition)

2.5 Functional Requirements

1. **Oscilloscope utilities**: As the entire thesis is centered around the need for completely replacing the oscilloscope from the PD acquisition and measurement Equation with our autonomous FPGA based solution, it is required that the FPGA should possesses utilities like *threshold based acquisition triggering*, *recording period*, a *pre-trigger* mechanism and *acquisition time*, which are integral to oscilloscopes. In the following discussion, requirements are realized and acquisition related terminologies are clarified.

   (a) **Threshold based triggering:** It is required that as soon as an incoming PD sample from `ADC0` (acquiring PD pulses) exceed the user specified upper or lower thresholds (between -1V to 1V), the programmable logic(`PL`) of `ZYNQ 7010 SoC` should start accepting samples (for feature computation) (see Figure 2.12).

   (b) **Acquisition Time:** As discussed in section 2.1, the oscilloscope can record 50,000 PD pulses (at 250Msps). The features corresponding to this amount of PD pulses are considered to be sufficient for effective PD classification. In our context, acquisition time is defined as the total time duration for which PD pulse samples acquired by `ADC0` are accepted by the `ZYNQ PL`. The user should be able to tweak this parameter to get the required amount features for PD classification.

   SPECIFICATION: The user should be able to *start* and *stop* the PD acquisition based on *software controlled* acquisition time.

   (c) **Recording Period and Pre-trigger:** A trigger event (ideally) implies an arrival of a PD pulse at the ADC input. Hence, the entire length of this pulse should be completely 'recorded' in order to compute required classification features for this pulse. Thus, pulse acquisition takes place in terms of *frames* of discrete (ADC) samples. The length of each frame constitutes the *recording period*. The term *frame* will be used to represent the analog *recording period* in digital domain throughout this report. In long cables, partial discharge pulses suffer from shape distortions and elongations. Further, pulse width for each source of PD differs. Hence, it is desirable to have a range of *frame lengths/recording periods* that the FPGA can cover. A Recording period from *1us to 10us* is sufficient for covering these variable pulses. With the PD acquisition rate specified at 125 Msps, these *recording periods* boil down to the frame lengths as shown in Table 2.1.

   Figure 2.12 demonstrates a frame in the *acquisition time*. For this frame, the acquisition of PD samples start at discrete time N = n where the rising pulse exceeds the user specified Trigger High. Now, it can be seen that the Post-Trigger part of frame will not be able to provide a complete picture of the pulse shape. This is because of the fact that the shape of pulse before

Recording Period (us)	Number of samples(125Msps)	Pre-trigger samples
1	125	12
2	250	25
3	375	37
4	500	50
5	625	62
6	750	75
7	875	87
8	1000	100
9	1125	112
10	1250	125

Table 2.1: Required *Recording Periods/Frame Lengths* and required *pre-trigger* history

triggering is unknown and could have been A,B,C or an anything else. Thus, the to be calculated parameters like charge and energy will loose this information which can further lead to classification errors. Hence, it is required to have a *pre-trigger* mechanism which can cater for remembering the history of these pulses before the actual trigger event.
This history should be typically around 10% of the frame length as can be observed in Table 2.1

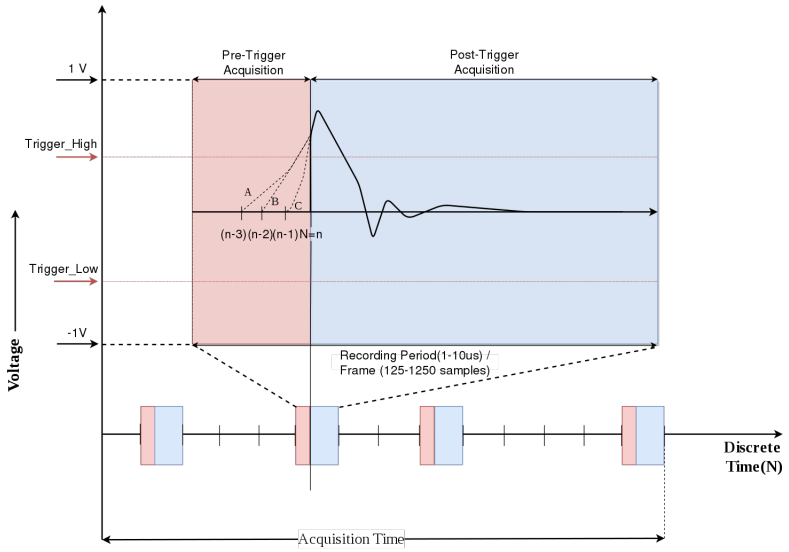

Figure 2.12: Scheme of PD acquisition

2. **Algorithms (Q, E and I_{peak} computation):** As discussed earlier, PD pulse's fundamental features - **Apparent charge(Q)**, **Energy(E)**, and **Current peak(I_{peak})** have proven to be an excellent choice to separate different sources of PD. In this thesis, two methods, one based on time domain calculations and the other based on frequency domain calculations are explored.
The (simplified) PD measurement circuit is shown in Figure 2.13.

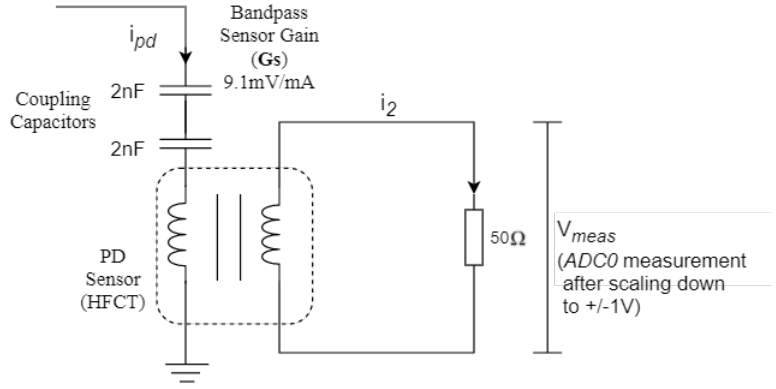

Figure 2.13: Simplified PD Detection circuit terminated at *Red Pitaya's* input `ADC0` (50Ω effective input impedance)

Here, `ADC0` of Red Pitaya is connected to the secondary of HFCT sensor. The 14 bit ADC will measure PD voltage in 2^{14} discrete quantized levels i.e, 0-16383. This range has to be scaled down back to $\pm 1V$ to represent PD voltage. The output of this scaling is V_{meas}, as shown in Figure 2.13.
In the following discussion, each feature calculation is described, followed by design perspectives, in order to give the reader a decent idea of how the algorithms would be mapped onto the FPGA (on a data-flow level), without going into design details. Also, **NOTE** that in the design perspectives only present an analysis of the required resources. However, the exact number and type of the resources utilized *cannot* be provided before designing. For instance, a 32 bit multiplication and accumulation operation can be performed using different resources present on the PL of ZYNQ 7010 (LUTs, dedicated adders and multipliers, DSP48 (25*18 multiplier,48-bit accumulator)). However, depending on the *data types* of variables, *timing constraints* and user specified synthesis *directives*, the synthesis tool (Vivado HLS in our case) can bind to any type and number of resources as required. Thus, having exact numbers beforehand is not possible (and not required). However, having a *decent* idea of the type of resources, their amounts and identifying the computationally expensive resources is extremely important.

As the stream of PD voltage data from the ADC is arriving at every sampling period to the PL side, an important question needs to be answered, which will have direct implications on the architectural choice.

Do the feature computation formulae inherently require all ADC samples in a *frame* to be *cached* beforehand, or do they allow computation of features sequentially? (i.e the results can evolve as the ADC samples arrive in a sequence and the required features of interest can (mathematically) be made available as soon as the last sample arrives)

The key for implementing high performance (throughput) architectures is to maintain constant flow of data samples through the FPGA. If the formulae allow sequential computations as described above, the formulae will inherently provide a base for employing a free flowing high performance *streaming* architecture for handling sequentially arriving ADC samples. A *streaming* architecture is desirable in order to achieve a high throughput (= sampling rate (125Msps) in our case) with controlled resource utilization. Moreover, if computations are sequential (in terms of data accesses), realizing a high throughput design will be less complex than if they involve non-sequential accesses.

(a) **Time Domain Calculations:**

-Current Peak ($\mathbf{I_{peak}}$):

This is the peak current (i_{pd}) of PD pulse at the primary of HFCT sensor. Thus it can be given by Equation 2.1

$$I_{peak} = peak[|V_{meas}(n)|]/G_s \tag{2.1}$$

where n = 1 to N (Frame length) and G_s is the sensor gain of 9.1mV/mA.

-Design Perspective:

The flow for current peak calculation is *sequential* and thus, inherently supports *streaming* architectures. Every time a new ADC sample arrives at PL side, a comparison can be done between the present sample and the previous maximum V_{meas} value. This way, in the end of every frame, the maximum in that frame is available. As *Gs* is constant, no actual division (high compute and time complexity) has to be performed and the division can be seen as a multiply with ($1/G_s$). Further, single cycle *shift*

Multiply (*)	1
Greater than (>)	1

Table 2.2: Operations - I_{peak}

operations and add operations can be performed on this maximum of the frame to replace a multiplication with (1/9.1 = 0.1099).
Thus, only one *comparator*, and a *few* shift registers and addition operators will be utilized in this scheme and they will be *reused* for all the samples.

-Estimated Charge(Q):

In time domain, charge is defined as the integral of current over the

duration of sampled PD current pulse.

$$Q = \int_0^T i_{pd}\, dt \tag{2.2}$$

where T is the *recording period*.
Ideally, the measured PD pulse should be unipolar and there should be no oscillations. However, due to the limited bandwidth of the PD sensor, the characteristics of PD pulse at the source (defect), the distance of the PD source from sensor and the location of PD source within the equipment, factors like attenuation, distortion and oscillation are introduced in the shape of the pulse [13]. In order to mitigate the effect of oscillations in the estimation of charge, the pulse is first passed through a second order butterworth low pass filter. Further, the charge can now be approximated as the peak value of the integral evolution of current pulse over its time duration (if the lower cutoff frequency of measuring system tends to zero) [20].
Finally, a better approximation of charge can be calculated by integrating the current pulse between the zero crossing points on either side of the maximum peak value [13]. Thus, the final scheme of charge calculation in time domain is shown in 2.18 followed by the charge estimation Equation 2.3.

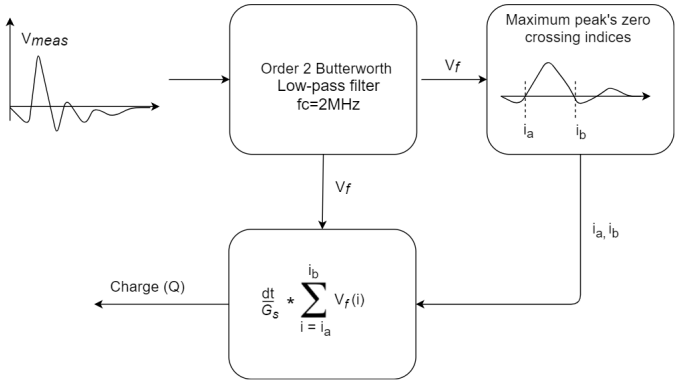

Figure 2.14: Time domain calculations flowchart

$$Q = \frac{dt}{G_s} * \sum_{n=i_a}^{i_b} V_f(n) \tag{2.3}$$

where, dt the sampling period (8ns) and Gs is the sensor gain (9.1 mV/mA).

-Design Perspective:

On careful inspection, it was recognized that the flow for charge calculation can be serialized to support streaming architectures.
Following scheme can be followed for computing charge in time domain in streaming mode on the FPGA.

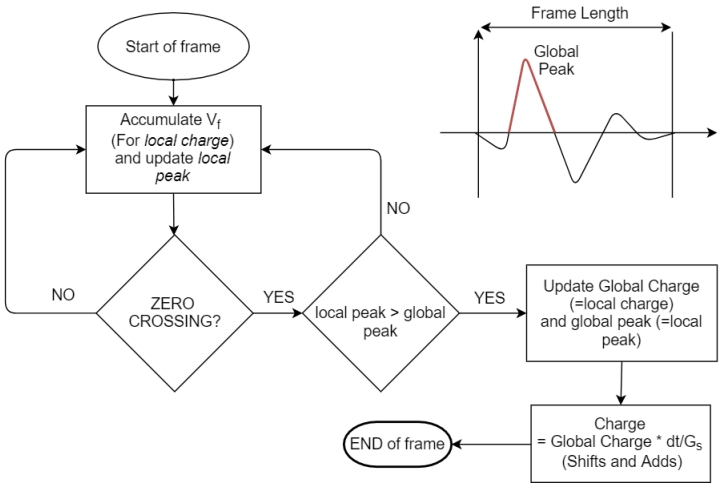

Figure 2.15: Streaming Scheme for Charge Calculation

As can be seen from the self explanatory Figure 2.18, a streaming scheme can be followed where the input voltages can be accumulated (addition operator) between all zero crossings (camparator(*present sample's sign != previous sample's sign*)) within a frame and global peak (another comparator to compare local peak and global peak **magnitude-wise**) and its area under the curve (one adder) can be computed as the ADC sample stream arrives (serially). After the last sample in a frame is accumulated, required scaling(*dt/G_s) can be performed.

Multiply (*)	1
comparators(!=,>)	2
Add(+)	1

Table 2.3: Operations - Q (Time domain)

Again, multiplication can be avoided using *shifts and adds*.

-Energy(E):

For our test setup, energy can be defined as the amount of power dissipated over time by the 50Ω effective impedance of *Red Pitaya's* `ADC0`(PD Detection) input.

$$E = \frac{V_{meas}^2}{R} * T \ = \frac{dt}{R} \sum_{n=1}^{N} V_{meas}^2(n) \tag{2.4}$$

where T is the recording period and dT is the sampling period.

-Design Perspective:

Energy computation also supports streaming architectures. Each incoming sample can be multiplied with itself and added to the previous result. The process continues till the end of the frame. As soon as the last sample in the frame arrives, required scaling (dt/R) can be performed using shifts and adds.

Multiply (*)	2
Add(+)	1

Table 2.4: Operations - E (Time domain)

From the design perspectives, it can be concluded that time domain calculations are sequential and inherently **support streaming architectures**. Moreover, these algorithms are **not compute intensive** as they require relatively simple operators like **adders, multipliers, comparators and shift registers**. Thus, because of simple calculations, the resulting circuit would be both *area* and *power efficient*. Further, all these operations can be divided into different stages of a **pipeline**, offering potentially high throughput. Further, the incoming data from the ADC can potentially be handled in the most **natural** manner.
As the computational complexity is less and no concurrent memory (Block RAM) accesses are evident, this pipeline (having operations for Time Domain feature computations) can potentially consume input ADC samples (filtered and unfiltered) continuously at the sampling rate of ADC (125Msps) and thus, offer no **bandwidth bottleneck** to the incoming samples (implying a **high throughput of 125Msps**).

(b) **Frequency Domain Calculations:**

-Current Peak ($\mathbf{I_{peak}}$):

The Equation 2.1 for current peak calculation in the Time Domain calculations holds.

-Charge(Q):

These calculations are based on the fact that the Fourier Transform of a function inherently carries information about its integral over time. Also, the low frequency components of the current pulse render a good estimate of the current pulse's charge(provided pulse duration are short and lower cutoff frequency of the measuring system tends to zero)[20].
The discussion for charge calculation in time domain applies here in frequency domain also. The only change is that now, instead of integrating the filtered PD pulse voltage, the integration itself is estimated using the **second component** of Fourier transform (of filtered voltage pulse). Thus,

in frequency domain, charge can be estimated as:

$$Q = \frac{dt}{G_s} * |FFT(V_f)[2]| \tag{2.5}$$

where V_f is the filtered PD measured voltage, dt is the *sampling period*(8ns) and G_s is the sensor gain (9.1mV/mA).

-Design Perspective:

The charge calculation in frequency domain differs from that of the time domain as here, instead of accumulating voltages, a FFT function is used and the magnitude of second point of FFT is required.
An obvious choice is to employ `Xilinx FFT IP LogiCore` to compute FFT. The initial tests with this core utilizes more than 50% of LUTs available on the PL side of `ZYNQ 7010` for only a 256-point FFT.
However, it was realized that in Equation 2.5, all points in the FFT are not required. Instead, only the 1st frequency component (to estimate DC component) is required. Thus Equation 2.5 can be re-written as :

$$Q = \frac{dt}{Gs} * \sqrt{[\sum_{n=0}^{N-1} V_f(n) cos(\frac{2\pi n}{N})]^2 + [\sum_{n=0}^{N-1} V_f(n) sin(\frac{2\pi n}{N})]^2} \tag{2.6}$$

where N is the *recording period*.
The required *recording periods* are known and can vary from 1 to 10 us. `ZYNQ 7010 SoC's Artix-7` FPGA provides a block RAM support of *2.1Mb*. This storage can be utilized to store all sines and cosines corresponding to all possible recording periods to be used as *look-ups* ([6875 sines + 6875 cosines* 16bits each]=*0.21Mb*). They can further be utilized to successively multiply the incoming filtered PD samples. Further, multiplications with sines and cosines can be accumulated independently till the end of *frame*. In the end however, as the magnitude is required, two multiplications and an addition have to be performed, followed by the computationally expensive square root operation.

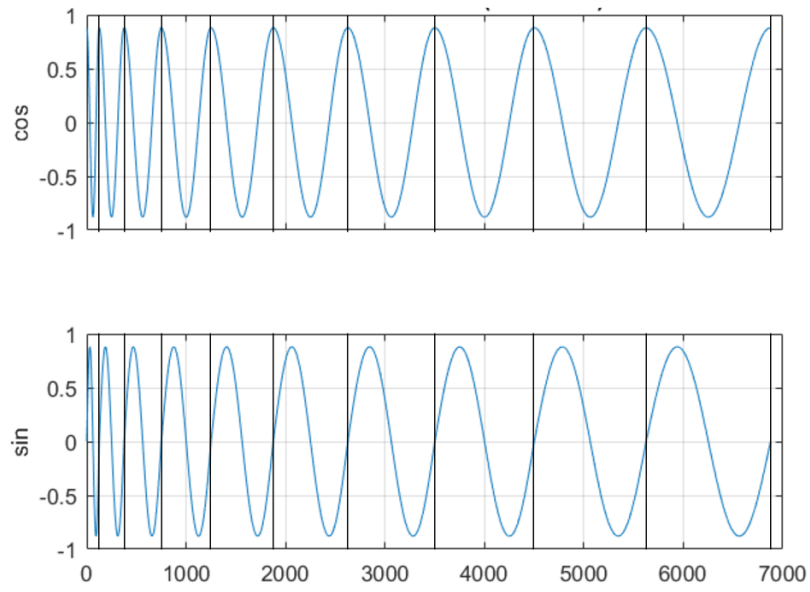

Figure 2.16: sines and cosines corresponding to recording periods from 1 to 10us stored as *look-ups* in FPGA's BRAM

As can be observed from Figure 2.16, the scaling information (dt/G_s) is also stored in these *look-ups*, to avoid scaling in the end. In such a scheme, the utilized operators would be as shown in Table 2.5

Multiply (*)	4
Add(+)	3
sqrt	1

Table 2.5: Operations - Q (Frequency domain)

-Energy(E):

Similar to the charge calculations, for energy calculation, square of PD's measured voltage is estimated using the Fourier transform as shown.

$$E = \frac{V_{meas}^2}{R} * T \ = \frac{dt}{R} * |FFT(V_{meas}^2)[2])| \tag{2.7}$$

where T is the recording period and dT is the sampling period.

-Design Perspective:

As described in charge calculations, the FFT here can also be replaced by the magnitude of multiplications with respective sin and cosines and independent accumulations for both, rendering the new energy calculation

Equation as shown.

$$E = \frac{dt}{R} * \sqrt{[\sum_{n=0}^{N-1}[V_{meas}(n)]^2 cos(\frac{2\pi n}{N})]^2 + [\sum_{n=0}^{N-1}[V_{meas}(n)]^2 sin(\frac{2\pi n}{N})]^2} \quad (2.8)$$

However, the only difference here is that unfiltered V_{meas} samples have to be squared each time, rendering an additional multiplication operation. Moreover, the lookups were scaled by a factor of (dt/G_s) to facilitate charge calculations but the factor required here is (dt/R). Thus, the new scaling factor to be multiplied to the results at the end of each *frame* should be (Gs/R) (which can further be replaced by shifts and adds).

Multiply (*)	6
Add(+)	3
sqrt	1

Table 2.6: Operations - E (Frequency domain)

Thus feature computation algorithms in frequency domain can also be realized in a sequential manner, which will inherently help creating **high performance streaming architectures** to handle the sequential ADC input. However, frequency domain algorithms are bound to utilize **more hardware resources** as compared to the time domain ones. Moreover, the **square root** operation is an operation with inherently high time complexity and is resource intensive. Rest operations utilized for these computations are fairly simple, the formulae do not pose a possible bottleneck for creating a high throughput design for our system.

3. **Filtering:** As discussed in previous section, there is a need to implement a butterworth Low Pass filter with 2MHz cutoff frequency for charge(Q) calculations (in both domains). A low pass filter (Direct form 1) as shown has to be implemented

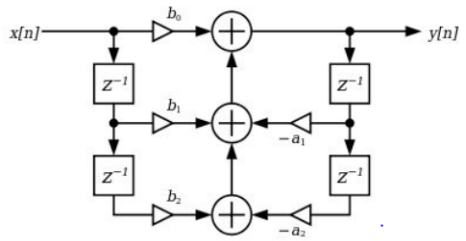

Figure 2.17: Second order Butterworth filter (Direct form 1)

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] - a_1y[n-1] - a_2y[n-2] \quad (2.9)$$

This is a compute intensive block as in every iteration, 5 multiplies and 4 additions/subtractions need to be performed. However, these operations ideal to be

Multiply (*)	5
Add/sub (+/-)	4

Table 2.7: Operations - Low pass butterworth filtering

bound to the low latency `DSP48` blocks present on the FPGA (Total 80 available) which are optimized for multiplication and accumulation operations.

4. **Phase calculations:** It is an important requirement to calculate the phase of the pulse with reference signal. The phase of the pulse is an important feature as it will facilitate pulse identification using `PRPD` pattern recognition.

$$PD\,phase(\%) = (current\,time/current\,period) * 100 \tag{2.10}$$

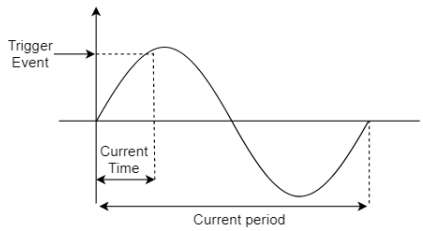

Figure 2.18: Phase calculation

5. **Allowable Error limits:** The allowable error limits for feature computation are shown in Table 2.8

Charge (Q)	**10%**
Energy (E)	**5%**

Table 2.8: Error constraints

For phase calculation, the resolution should be at least **0.5 degrees**.
Please note here that these constraints are not hard and fast. These numbers are just an estimate based on heuristics and literature.

6. **User control Options:** The user should have options to choose the recording period, trigger levels, acquisition time and algorithm of choice.

SPECIFICATION ARM core 0 of the PS should be programmed to send user configuration to the PL compute block *via* M_AXI_GP (PS slave) port present on the boundary of PS and PL.

2.6 Architectural Constraints

1. **Memory Constraint:** In order to do facilitate the classification process, it is required that the features (Q,E,I_{peak} and phase) corresponding to **50000 pulses**

(*test-set*) should be saved. Thus if each feature is allotted, say 4bytes, the total memory requirement boils down **0.76 MB**. Red pitaya's DRAM offers storage upto 512MB, which is sufficient for theoretically holding 673 such test sets (features corresponding to 33.65M (50000*673) *frames*). The On chip Memory (OCM) available on the SoC can only store 0.25MB of data and is not capable of storing all results from even 1 test set.

SPECIFICATION: Off-Chip DRAM should be used to store output features.

2. **Near real time response (*latency*) constraint**: There is a high level near real time requirement that as soon as the user enters the input configuration of choice (Figure 2.7), the outputs corresponding to the test set (50,000 *frames*) should be available with visually less *latency*.
   This real time constraint is *soft* and is only concerned with the user experience. As an educated approximation, it was decided that a *latency* of 3 seconds (visually perfect) between user configuration *input token* and the *output token* (features corresponding to 50,000 input *frames* to be visible on user's GUI).
   For finding a worst case latency estimate, it is assumed that there is a trigger event in every recording period of 10us. Thus, 50 thousand pulses correspond to a total *recording time (T_{in})* of 0.5 seconds.
   Further, the SoC features a Gigabit ethernet (maximum transfer rate from red pitaya to the remote user GUI = 1Gbps). Thus, theoretically, outputs(features) can stream out from the box (via ethernet) at a rate of 128MBps. As the features corresponding to one test set are 0.76 MB, the time to transfer data from the box to the remote GUI (T_{out}) is theoretically 5.93ms(Other factors like printing delay on the user GUI are not accounted)
   Thus, the available window or *maximum latency* of datapath (compute + transfer from PL to DRAM) should be around (3 - T_{in} - T_{out}) = 2.49 seconds, which should be comfortable to achieve.

3. **Real time *frame* (PD) Acquisition constraint**: In the discussions in section 2.1, it was realized that having a system facilitating high *frame rate* (for high *pulse resolution*) is desirable as this implies that narrowly spaced pulses are not missed. Also, the *repetition rate* information can be derived with a higher degree of accuracy if the number of pulse misses are less.
   Thus an additional *hard real time constraint* for acquisition of PD from `ADC0` was drafted for facilitating a better system (best possible *pulse resolution*).
   For each test-set, all input `ADC0` (PD) samples should be acquired at 125Msps (worst case maximum frame acquisition rate =1MFPS). Thus, there should be no *wait state* between the acquisition of two successive frames (PD pulses). This way, no intermediate PD pulse is lost in between.

2.7 Architectural alternatives

In [10], it is recognized that the most suitable features for PD classification are *phase of occurrence*, the *shape* of the PD pulse and the *repetition rate*. The features of interest, Q, E and I_{peak} give an estimate about the shape of the pulse to facilitate separation process. On the other hand the phase of occurrence is critical for PRPD pattern recognition.
The *repetition rate*, as pointed earlier, can be derived with highest precision if the FPGA does not miss any input samples in between two frames (thus, the real time frame acquisition requirement).
To calculate features Q,E and I_{peak} with the highest precision, *floating point arithmetic* hardware should be employed on the FPGA.
A simplified example is shown to demonstrate how an acquire and accumulate operation using *floating point arithmetic* would be performed. Such an accumulation is a part of time domain calculations (Equation 2.3).

```
void funct(dT ADC_in, dT* out) /*dt->float*/
{
#pragma HLS PIPELINE II=1
    static dT sum=0;
    sum  = sum + ADC_in;
    *out = sum;
}
```

The hardware block corresponding to the code above is intended to perform accumulations of the input variable ADC_in (ADC sample). As it is required to handle a new sample at the sampling rate of 125Msps, a *directive* for pipelining the block was put, i.e an II=1 to specify that the block should accept a new input in every cycle (clock = sampling rate = 125MHz). However, when the block was synthesized using *Vivado HLS*, it was realized that the floating point adder that vivado binds to has an inherent latency of 6 cycles and thus, cannot be pipelined with II=1. Thus, it cannot accept a new input every cycle.
Now, there can be two valid workarounds to handle this issue:

- **Cached *frames*:** The incoming ADC samples(125Msps) can be *cached* into a buffer of *frame length*. In worse case , the frame length is 1250. BRAM can be allocated for implementing this buffering. After, 1250 cycles (125MHz), the cache will be filled. Thus, if the compute block can compute outputs(floating point accumulation) corresponding to the *frame* within next 1250 cycles (II $\leq$ 1250), a *dataflow* is maintained between acquiring, caching and computations and thus, no inter-frame samples are lost (desirable). As discussed earlier, the latency of compute block was 6 cycles. Thus, if only one unit is used, total addition will take 7494 cycles (6*1249) which implies a violation of inter-frame requirement. However, to fulfill the *Initiation interval* requirement ($\leq$=1250), *parallel* adder units and parallel accesses have to be employed. This implies that the BRAM can now not be utilized, instead the array has to be partitioned into LUTs rendering long combinational, sub-optimal multiplexing tree. Finally, instead of one adder, which should mathematically complete accumulations as soon as the last (1250th)

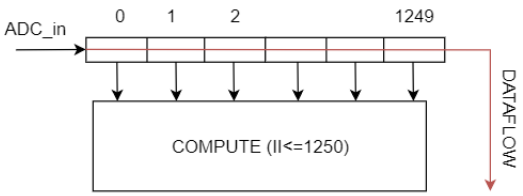

Figure 2.19: Cached Frame

value is available, multiple Floating Point adders have to be employed. In this discussion, there was only an adder employed. But can be observed from our discussion of algorithms, operations like adders, multipliers and square root are required for every algorithmic iteration. In such cases, the utilization of the FPGA will explode. The area of fabric is of importance because this is just the first stage of the project. Another set of functional blocks to perform actual classification process has to be put on the FPGA in future to bring about complete automation.

- **Cached Sub-frames:** This method is a subset of the previous method and reduces the resource requirement. The method is demonstrated in Figure 2.20. In

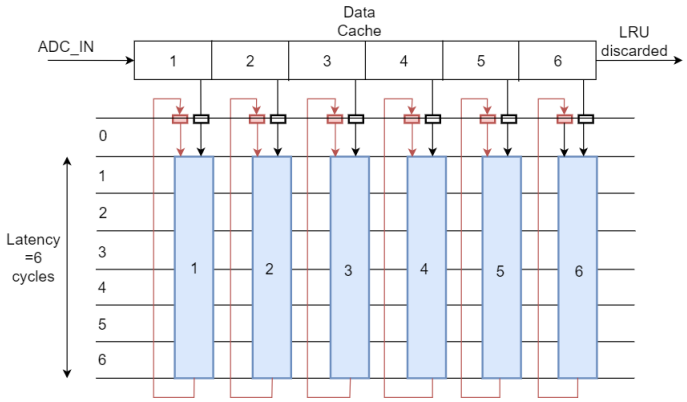

Figure 2.20: Cached Sub-Frame

this case, instead of caching the entire frame, only N samples are cached, where N is the latency of the functional block (6 in our case). Thus, it is essentially an attempt to hide the latency of 6 cycles by employing 6 Floating point additions instead of one and the whole unit is pipelined at *II=6*. In the end of a frame, 5 more units are required to add the results of successive additions (red-registers). This method is elegant and requires way less resources than the previous method, still maintaining the required dataflow. However, it should be noted that the control in this trivial scenario was negligible. However, we require control capabilities like decisions based on zero-crossing detection in PD pulse. In such a case, *feeding data* to all these parallel resources *may* be unfeasible and II of 6 *may* not be fulfilled. Moreover, the coding complexity for realizing such parallel architectures using high level language (C++ in our case) is inherently more than serial ones.

2.7.1 Fixed Point based Architecture

For the same code of accumulation, the data-type for all variables is now changed to fixed points. A highly desirable property is observed after synthesis. The fixed point adder unit that `Vivado HLS` binds to can be pipelined at *II=1*. This implies that it can readily accept a stream of new ADC samples (`ADC_in`) every cycle (125MHz). Thus, no parallel units are required for operation. Only one fixed point adder unit is sufficient and the required *dataflow* is also maintained. Thus, the stream of input samples is handled in the most **natural** manner as shown in Figure 2.21. Moreover, since less number of operations are employed to do the same work, fixed point arithmetic schemes facilitate *power efficient* designs.

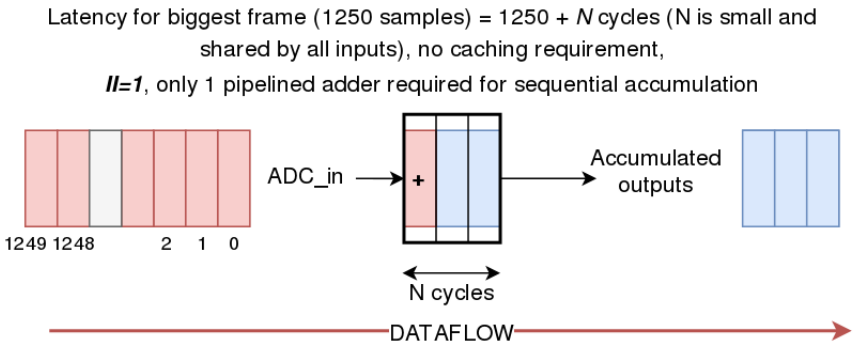

Figure 2.21: Scheme of Fixed Point Compute block facilitating real time acquisition and compute capabilities with controlled utilization

However, this *magic* does not come for free. Potential errors related to overflow/lack of precision are associated with employing this arithmetic. However, if thorough analysis of datapath and input ranges is done, fixed point arithmetic can practically result in similar results as that of floating point arithmetic.
In section 2.5, the error limits for charge and energy were shown. The fact that there are *allowable error limits* provides a window of opportunity for employing fixed point instead of floating point arithmetic.

2.7.2 An alternative architecture

A valid argument can be that although there is a real time PD *frame acquisition constraint*, there is **no constraint** to do feature computation itself in **real time**. Thus, the entire *test set* can first be acquired from ADC (*after triggering and history recording on FPGA*), streamed from PL to PS and saved on the off chip DRAM(512MB) at 125Msps. Such a streaming is possible as there is no theoretical bandwidth bottleneck in the datapath for samples (14 bits) arriving at 125Msps (=250MBps). The *PL Direct Memory Access (DMA)* and high performance (*HP*) *PS slave port* can both support this bandwidth requirement[25]. Here, the total data corresponding to only one *test set* would be (119.2 MB, each ADC sample occupying 2bytes in DRAM [512MB capacity]). Thus the transfer is feasible. Once the transfer is done, the data will have to be be streamed back from PS to PL side for further computations and the results

will again have to be streamed back to the DRAM.
This thesis however, is an attempt to find the *best*(speed, area, throughput, data reliability and power consumption) solution to tackle the problem at hand. With that in mind, it is left to the reader to decide whether this solution or the chosen solution (employing fixed point arithmetic) which facilitates the most natural flow of data within the system is *best*.

Here, it can be appreciated that *real time* acquisition is a requirement but *real time processing* is just an intentional constraint put for realizing the *best* hardware architecture (Figure 2.21).

2.8 Throughput and Bandwidth Requirements

In the ideal fixed point scheme of operations as desired (2.21), for ADC inputs arriving at 125Msps(PD), the compute blocks should provide no *resistance* to the flow of samples through them.
In the worst case (for throughput), the *frame length* is 1us. Thus, the maximum throughput required from the data-path from **ADC to DRAM** would be 1 PD pulse per micro-second or **1M PD pulses/s**.
As can be seen from Figure 2.22, the precise **throughput requirement** for the compute block to handle the incoming samples in real time is rate is **208.61 MBps**. Also, in worst case, a recording period of 1us (125 samples) will correspond to 4 output features (4 byte each). Thus, the **bandwidth availability** to to support real time transfer of output samples from input PL to PS should be **15.25MBps** (16bytes/us). For large datasets (as we have), the high performance **PL DMA** is a suitable choice [25]. The PL DMA supports a maximum bandwidth of up to 500MBps (at 125MHz) [25], which is sufficient for fulfilling our bandwidth requirement.

SPECIFICATION: The **PL DMA** should be used for transferring data from PL to DRAM *via* the PL-PS boundary.

`ZYNQ 7010 SoC` offers multiple interface alternatives to facilitate PL-PS transfers. For output feature transfer to the DRAM from PL through PS, the **high performance HP port (PS slave)** present on the PL-PS boundary offers more than sufficient (theoretical) bandwidth (125MHz* 4 [half bus-width (feature width)] = 500MBps) to support our bandwidth requirements [25]. There are other alternatives which can also provide sufficient bandwidths (GP,ACP ports). However, since there is only one stream (stream of *features*) to transfer to DRAM, HP port was chosen.

SPECIFICATION: **HP** (PS master) port should be used for data transfer from PL to DRAM through PS.

The PS memory interconnect (2,840 MB/s[25]) is also a part of PL to PS datapath. It can easily support the our 15.25MBps bandwidth requirement. Thus, the datapath from PL to PS will not provide any hindrance to the flow of our only output stream.

SPECIFICATION: The compute block should have a throughput of 208.75MBps.

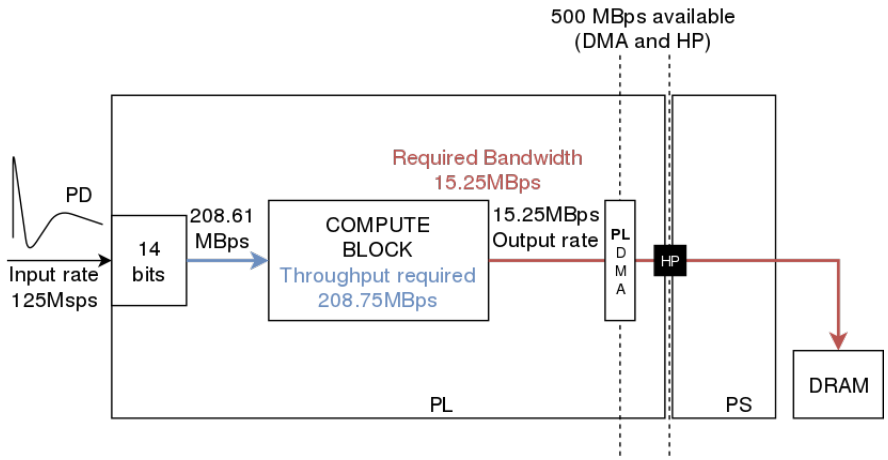

Figure 2.22: Throughput and bandwidth requirement to facilitate real time constraint

Moreover, as the computations involved are not extremely compute intensive, utilizing a faster clock (more power consumption) is not the ideal choice. Thus, a clock frequency of 125MHz (corresponding to sampling frequency) should suffice computations.

SPECIFICATION: The clock frequency of 125 MHz should be used for the compute block and DMA.

2.9 Fixed point analysis

By this point, it has been established that utilizing fixed point arithmetic would be integral in facilitating the implementation of the identified *best hardware design* choice. Fixed point data type representation is shown in Figure 2.23.

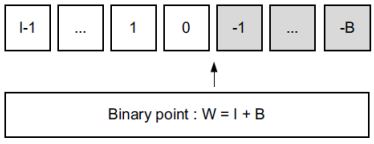

Figure 2.23: Arbitrary precision fixed point data type [1]

Here, W represents the word size, I represents the decimal part and B represents the number of decimal places in the word.
Employing fixed point data-types imply a significant increase in designer effort (compared to floating point arithmetic) as the designer has to *fix* the decimal point for all variables (registers for storing inputs, outputs and intermediate values) that will be a part of the input to output data-path. Thus, the *optimum mix* of W, I and B for every variable has to be calculated, such that the *range* and *precision* required from these

variables is achieved. However, as discussed earlier, there is no notion of a *typical* PD pulse input due to its stochastic nature. Thus, in order to calculate optimal positions for fixed point variables in the design, a worst case *range and precision analysis* has to performed.
A test set consisting of 4 voltage pulses were designed to simulate variable PD pulses detected at `ADC0`. This set is shown in Figure 2.24

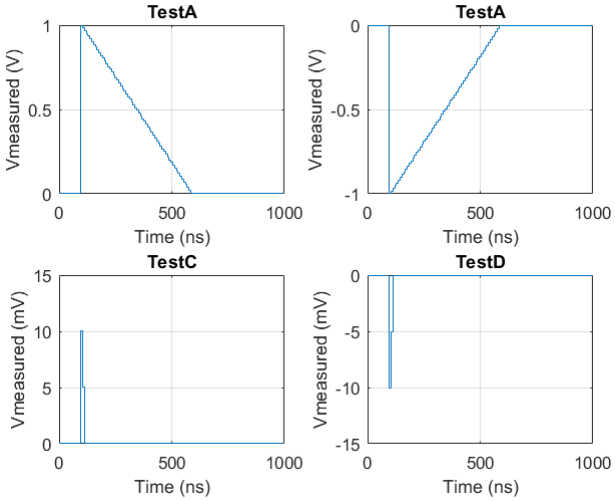

Figure 2.24: Fixed point analysis test set (voltage pulses)

Tests A and B were demonstrate two extremely wide (496ns each : rise time - 8ns), positive and negative hypothetical PD pulses. The aim with these tests is to exercise the *range*, i.e the I part in fixed point representations of the variables in the data-path to their maximums. These tests cover the more than maximum of current peak, charge and energy that the design is likely to encounter (in LV ADC settings). Thus, over-designing is done for *overflow* safety.
Further, tests C and D, in contrast, exercise the *precision*, i.e B part in fixed point representations of all the variables in the data-path. The scheme of Fixed point analysis done using `MATLAB's Fixed Point Converter` application is shown in Figure 2.25. As can be seen, the process is recursive and continues till the errors are within allowable limits.
From previous discussions, we know that there are three essential ingredients for functionality of the *compute block*: A filter, calculations in time domain and calculations in frequency domain. All the three were first realized in `MATLAB` using the default *double precision floating point arithmetic*. Applying tests A, B, C and D to this chain of functions rendered the *golden* reference outputs. These functions were then analyzed using `MATLAB Fixed point converter` application one by one and W,I,B were tweaked for each variable in the *compute* code according to the proposed values and corresponding fixed point codes for each function were *generated*. Finally the output from this chain of fixed point blocks was compared to the *golden* reference and successive tweaking was done until the errors were under limit.
The proposals (Figure 2.26) are based on the histogram (bit weights displayed along

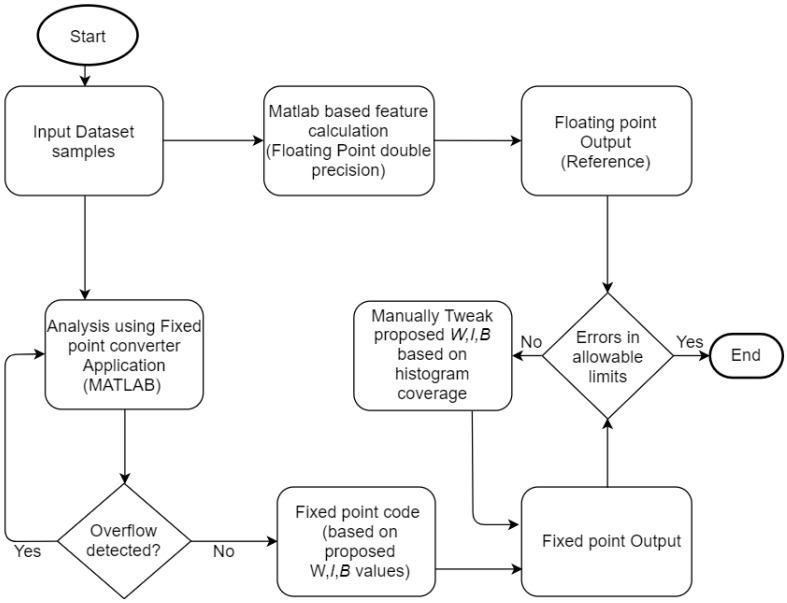

Figure 2.25: Flowchart representation of Fixed Point Analysis

the X-axis, and the percentage of occurrences in that bit position along the Y-axis) coverage of test inputs. Thus, the test sets should be exhaustive.

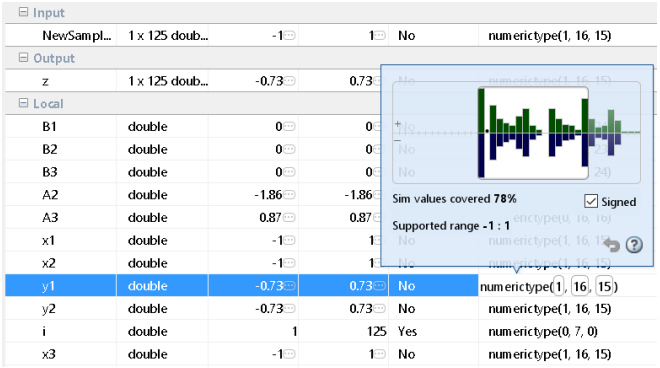

Figure 2.26: Fixed point proposals based on histogram coverage

The widths of these proposed variables can significantly effect the resource utilization and timing of hardware blocks. Thus, for some variables, the precision (B) in final hardware are not the same as proposed. Instead, the words were shortened (by reducing B) to adapt to less complex hardware and rounding schemes were applied to get the similar (in some cases, better) precision.

2.10 System Specifications

The derived system specifications are shown in Figure 2.27. Moreover, fixed point specifications were also derived for all variables in the design (all not shown in Figure).

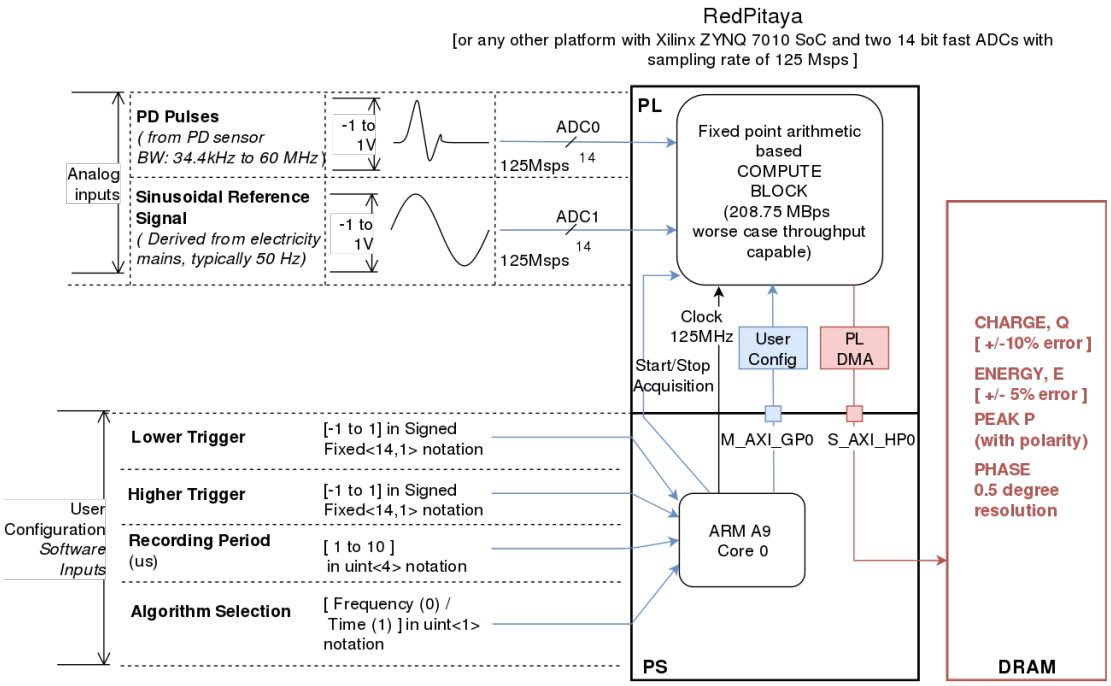

Figure 2.27: Specifications

2.11 High Level Synthesis

Programming hardware (using HDLs) has traditionally been a time-consuming and resource-intensive process. Moreover integrating the design into hardware/software environment (SoC) is difficult for a majority of general-purpose programmers. To address this challenge, there are techniques available to allow programmers develop hardware using higher-level languages (HLLs like C, C++, etc.). The process of generating RTL from HLLs is termed as High-level synthesis (HLS) in common literature. The active development and adoption of HLS by major vendors like *Xilinx* and *Altera* has significantly raised the abstraction levels of programming hardware resulting in rapid development and faster design space exploration. The generation of RTL from a higher level language basically starts like a software compilation process which involves the generation of control and data flow graphs. This is followed by three important tasks (scheduling, allocation and binding, and controller synthesis) that the tool must solve to generate the hardware description model. Scheduling as the name suggests, involves the generation of a schedule such that data and control dependencies are not violated in the final design. During Allocation and binding, the type and number of hardware resources for the design are determined after which the operations are mapped to individual cores from technology libraries. Finally, the controller that sequences the design and controls the functional and storage units in the datapath is derived. After this, the RTL is converted to a bit-stream that can be uploaded to the FPGA.
Note that at each stage the programmer can guide the compiler to obtain a design

that best fits the users needs. Even though HLS provides higher abstraction levels, the programmer still needs to understand low-level hardware details and change the algorithm to obtain the most optimal design. In this thesis, the functional blocks for FPGA (PL) were designed using `Xilinx Vivado HLS`.

3 System Overview

This chapter provides a brief overview of the functional blocks in the system and how these blocks are interconnected to efficiently tackle the task of PD detection and feature extraction.

3.1 Data Acquisition from ADCs

The very first task is to get the data from ADC to the `FPGA` side of our `ZYNQ SoC`. When `Red Pitaya's` board definition files are loaded in Vivado environment, both `ADC0` and `ADC1` (14 wires each) input channels are available at the `PL` side of the `SoC` as adc_dat_a[13:0] and adc_dat_b[13:0] in Vivado's *IP integrator*. Along with these 14-bit data inputs, the `red pitaya`'s differential clock inputs *adc_clk_p* and *adc_clk_n* are also available. Here, an open source IP[2], *DataAcquisition*(Figure 3.1) can be utilized as shown in Figure 3.2 as shown.

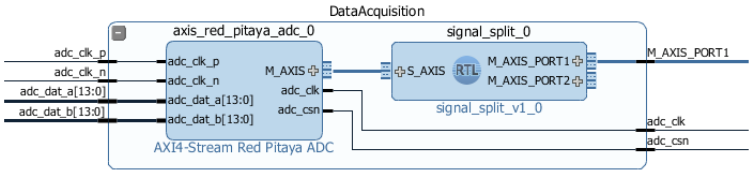

Figure 3.1: Data Acquisition IP [2]

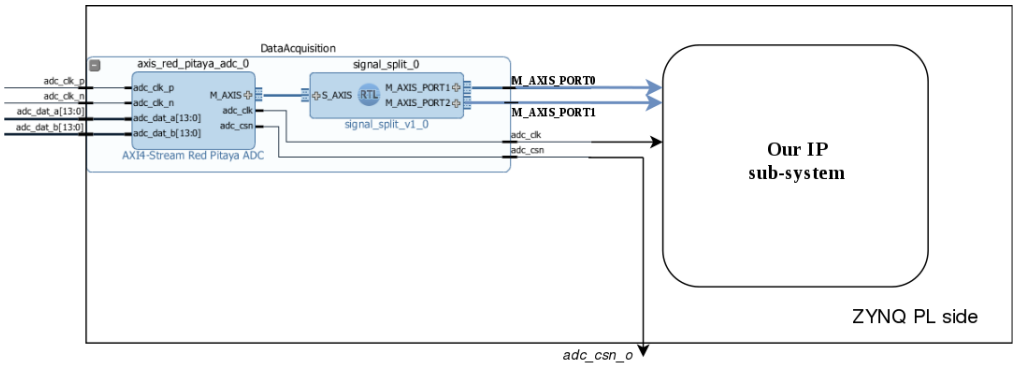

Figure 3.2: Data Acquisition IP connected to our system

This `IP` has two main functions:

1. It converts the external `ADC` clock (125 MHz) from *adc_clk_a* and *adc_clk_b* differential external ports into our programmable logic as a *adc_clk* clock. Thus, this

ADC clock itself can be used as the main clock for synchronizing our IP subsystem. Using the same ADC clock(125MHz) for FPGA's IPs has a clear advantage that **no input buffers** will have to be used between ADC inputs and our IP subsystem for synchronization.

2. It reads the ADC data from two input channels which becomes available on each adc_clk clock cycle (8ns) and makes it available over the AXI Stream (AXIS) interfaces *M_AXIS_OUT0* (ADC0) and *M_AXIS_OUT1* (ADC1). It is interesting to note here that because the output ports have an associated axis interface (*t_data* with *t_valid*, and *t_ready* signalling) and they are directly fed to our IP sub-system (*AXIS* compatible), if the IP sub-system is not ready to accept new samples, it will apply **back-pressure** on the DataAcquisition IP (through *t_ready*=0 signalling). Thus the IP-subsystem will not consume samples for which it is not ready for, which implies data sanity guarantied within the IP-subsystem at all times.

   It should be noted that Red Pitayas ADC core (see Figure 3.1) has an additional output port (adc_csn) connected to the external port adc_csn_o for a clock duty cycle stabilization.

NOTE1: This IP is **not** in the implementation scope of the thesis work. The discussion above is only presented for the reader to get a complete picture of actual Data acquisition from Red Pitaya's ADCs. From this point onwards, the term '**IP-subsystem**' will imply only the IPs designed in this thesis work. Further, the term **Data Acquisition** in general will imply data that can be acquired from *AXIS* output ports of DataAcquisition IP (Figure 3.1) and **not** from the actual ADCs of red pitaya

NOTE2: The ADC emulator IPs discussed in section 4.2 which are integral in IP verification will emulate the data coming out of ports **M_AXIS_OUT0** and **M_AXIS_OUT1** of DataAcquisition IP (Figure 3.1) and **not** the actual ADC ports which have no associated interface. Thus, these emulators will also have *AXIS* compatible master output ports.

NOTE3: Discussions henceforth are applicable to any ZYNQ 7010 based platform and **not only** Red Pitaya **specifically** (unless explicitly mentioned). In fact, the platform used for system design and testing was ZYBO[26] (for reasons mentioned in section 4.2) which possesses the same SoC specifications as that of Red Pitaya. So the IP subsystem developed in this thesis work can directly be put on Red Pitaya's FPGA without any compatibility issues for final implementation. Compatibility is also guaranteed as the ADC specifications are laid down keeping Red Pitaya in mind.

3.2 The Main System

The main system developed under this thesis work can be broadly segregated into two important segments:

1. **Hardware design** - This part involves the design and verification of an IP-subsystem specifically targeted towards tackling the task of high speed PD acquisition and real-time PD feature (*Charge*, *energy*, *Phase*, *Peak*, *Polarity*) extraction

based on time domain and frequency domain algorithms [] in accordance with user specified input configurations (*recording period*, *algorithm selection*, *trigger values*). This part was targeted for the Programmable Logic (PL) side (FPGA) of ZYNQ 7010 SoC. *Fluidity* is the essence of the designed IP-subsystem. A *stream and compute* approach is amalgamated with packet processing to render a high throughput, robust and elegant IP subsystem. This segment was the major part of the thesis work.

2. **Software design** -This part involves designing a standalone application for one of the ARM cortex A9 processor of ZYNQ 7010. The application is responsible for triggering the ADC emulators present on ZYNQ FPGA(PL) side which further exercises the IPs with pre-stored PD pulse and reference signal patterns. The application also controls the IP subsystem by providing user configurations and is responsible for initiating transfers of output samples from FPGA(PL) to off-chip main memory. This segment is the minor part of thesis work which provides a sense of completion and a conceptual proof of working of IP subsystem.

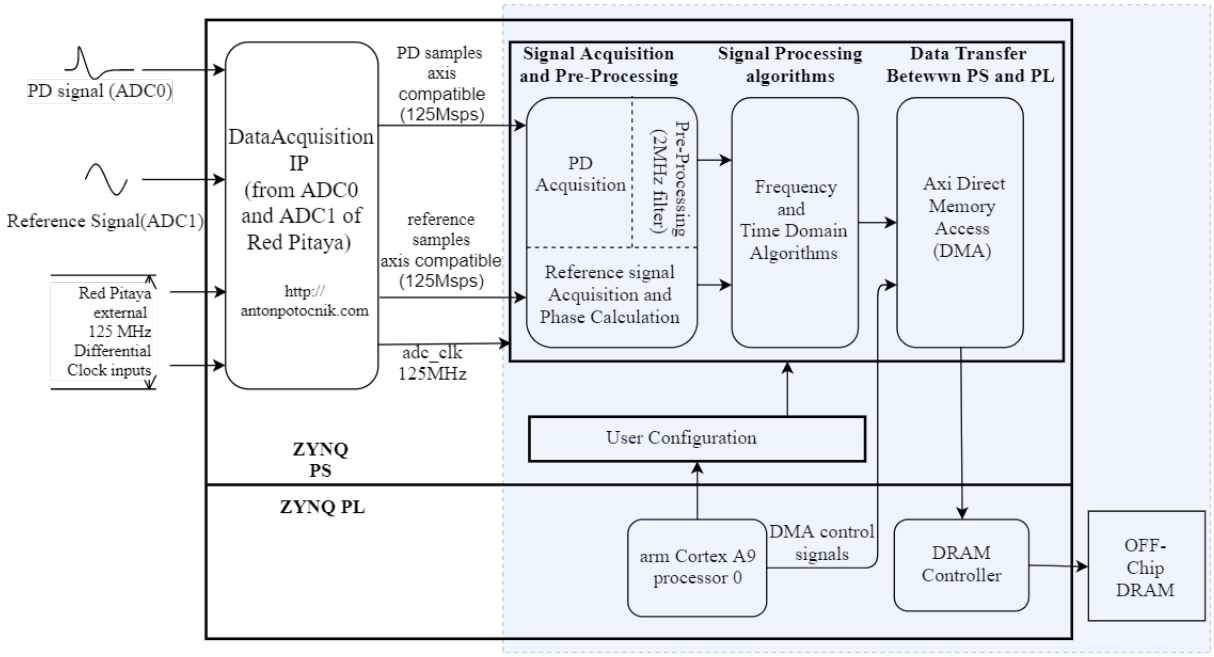

Figure 3.3: System level implementation overview and scope of implementation in this thesis work (colored region)

For the Hardware design, leveraging the knowledge gained from detailed discussions in section 2, three distinct sets of workloads were identified and groups of IPs were designed and chained together to efficiently target these workloads using a deeply *pipelined* architecture. These workloads are demonstrated at a block level in Figure 3.3 followed by a discussion of workloads and IPs targeting them.

1. **Signal Acquisition and Pre-Processing (Filtering):** The very first task for

the IP sub-system is that of accepting the pd pulses and sinusoidal reference data into the system every 8 ns. A set of three IPs form the pillars for tackling this workload. These IPs work together in coordination as shown in Figure 3.4 and are discussed thereafter.

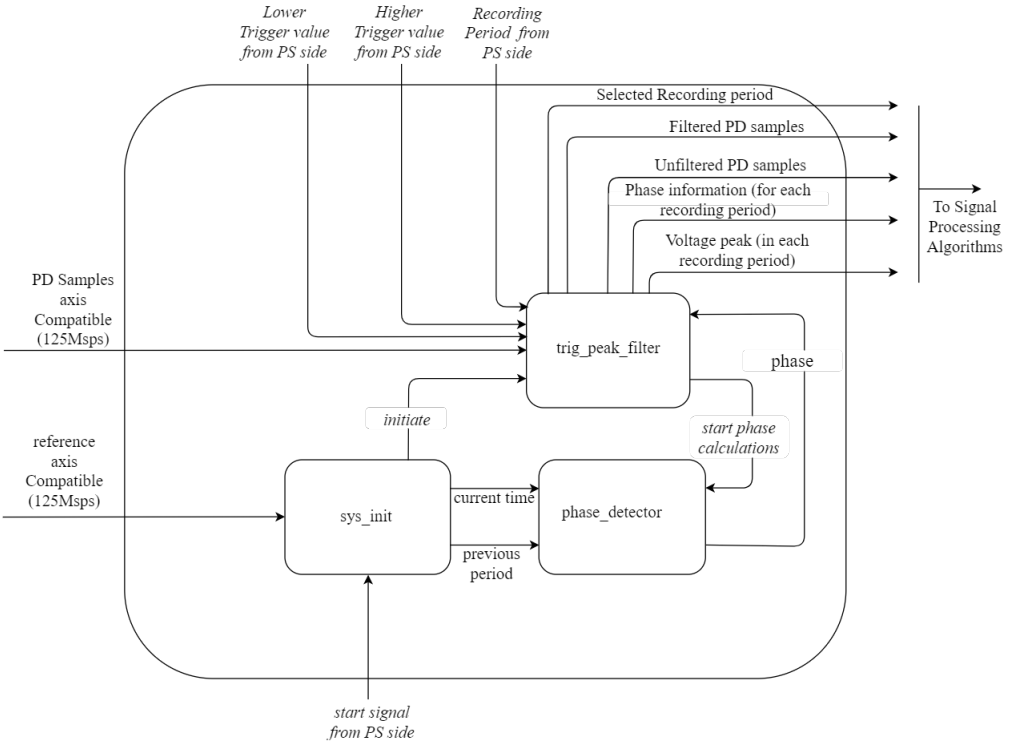

Figure 3.4: Block Diagram demonstrating data movement and control signals (in *italics*) between the IP's handling *Signal Acquisition and pre-processing* workload

(a) `trig_peak_un_filter IP` (***see section 4.3.2 for details***): The requirements for threshold based trigger mechanism, storing history of PD samples and an acceptance of PD samples for a duration of user specified recording period are handled by this IP. Further, the peak current value of PD has to be detected by the IP subsystem. This IP contributes to the task by detecting peak voltage for each recording period and sends this detected peak to IPs sitting downstream in the pipeline for the final current calculation which is extremely simple (V/R); hence, dividing labor between IPs at different stages of the deep pipeline.

Although, filtering is a part of the MATLAB algorithms which are to be implemented within the IP subsystem, it was recognized that it is better that filtering is incorporated within the PD acquisition (`trig_peak_un_filter`) IP and not with the algorithm IPs. A merit of the IP subsystem is its workload distribution within different stages of the pipeline. Thus, the relatively more compute intensive algorithms were assigned separate IPs which are provided with already pre-filtered data samples along with unfiltered data samples

simultaneously (because energy calculations in both algorithms require unfiltered data).

Moreover, it was decided that this `IP` also streams out the recording period information along with filtered and unfiltered data at the beginning of each recording period. This information can be used by other `IP`s to get information about the beginning of a new set of `PD` samples (*recording period*).

(b) `sys_init` (***section 4.3.1***) **and** `phase_detector` (***section 4.3.3***): From specifications (2.10), it is evident that along with PD pulse acquisition, a reference signal also has to be acquired and further, phase information for each PD pulse occurrence with respect to this reference has to be calculated. This implies calculating which point (or degree) of the 360 degree sinusoidal reference (`ADC1`) has the PD(`ADC0`) pulse triggered the `trig_peak_un_filter IP`.

One can imagine that it is essentially a divide operation, where the phase of `PD` pulse is given by:

$$PD\,phase = current\,time / current\,period \tag{3.1}$$

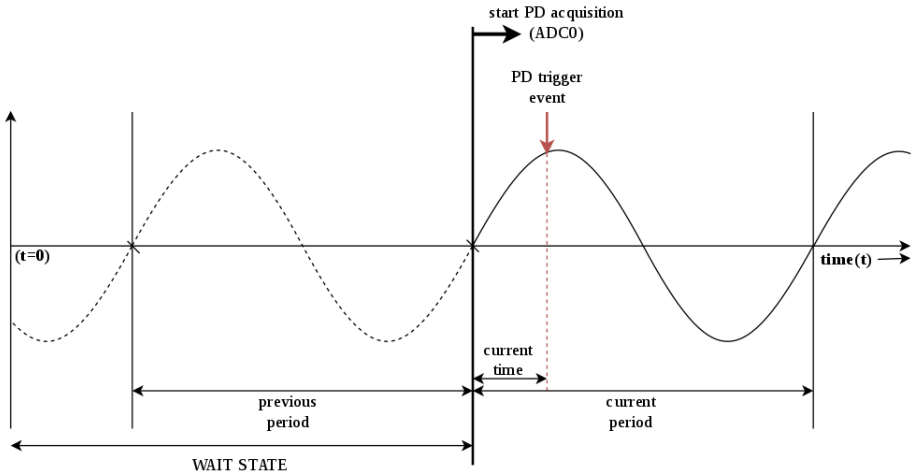

Figure 3.5: Phase Calculations

For a $50Hz$ electricity mains signal, *current period* (Equation 3.1) should ideally be $20ms$. However, due to the continuously changing load of the power grid and the generator's reaction to these load variations, frequency is not constant and may vary by a little margin. For instance, All India electricity grid [27] is operating in a band 49.90-50.05 Hz for nearly 75% the time. However, the the minimum and maximum frequencies can touch 49.70 Hz and 50.30 Hz respectively. This implies a variation of $\pm 0.6\%$ in electricity mains frequencies and hence a variation by the same factor in the time period. Now, although this variation looks small at the first glance, a $\pm 0.6\%$ variation implies that if *current period* (3.1) is fixed at 20ms (50Hz), the phase calculation can be off by (± 2.16 **degrees**). Further not all countries use a 50

Hz supply. For instance U.S uses a 120V/60Hz supply. Due to these reasons, it was decided **not to fix the *current period*** in Equation 3.1 as a constant but the `IP` subsystem should dynamically track the period information instead. However, this implies other issues that now have to be tackled.
Firstly, the division (Equation 3.1) now becomes a variable/variable division (actual division step required) instead of a variable/constant division(multiply/shifts and adds by constants) which is less in computational complexity and thus requires less time than actual division. Further, in section 2.7 it can be realized that a *stream and compute* approach was adopted instead of a *store and compute* one. Thus, the samples are not recorded for a particular recording period, but rather, streamed through a chain of pipelined `IP`s which perform computations on the samples as they arrive and no samples are stored in the process. Moreover, in section 2.5 it has been established that the nature of algorithms is essentially sequential.

Now, due to this sequential, streaming nature of designed architecture the *current time* information is easy to extract. However, ***current period* information cannot be retrieved** as the IP subsystem cannot look ahead in time and know the current period at the time of PD trigger event (see Figure 3.5). Although looking ahead in time is impossible, the IP subsystem can definitely keep a track of just **previous period** as shown in Figure 3.5. Moreover, the *previous period* would be equal to the *present period* (for all practical purposes) because the electricity mains frequency will not vary suddenly for instance, from 49.7 Hz to 50.3 Hz (the Indian power grid context). Thus, the way phase is actually calculated by the IP subsystem is by using *previous period* instead of present as shown:

$$PD\,phase = current\,time/previous\,period \tag{3.2}$$

In Figure 3.5, acquisition of sinusoidal reference signal starts at time $t = 0$. Now, it can be realized that if `trig_peak_un_filter IP` starts acquiring PD pulse samples before one complete time period (**period between two positive zero crossings of sinusoidal reference**) has been elapsed, there can be no corresponding phase information for that pd pulse because there is no *previous period* data available within the `IP` subsystem by that time (see Equation 3.2). Moreover, there is also a need that user(software) should be able to control when to start accepting PD samples.
The `sys_init` and `phase_detector IPs` are specifically designed to target these issues. `sys_init` is dedicated for reference signal acquisition (`ADC1` samples) and initiating the `trig_peak_un_filter IP` (to be able to start accepting new `PD` samples `ADC0`) when at least one complete time period has elapsed (`WAIT STATE` in Figure 3.4) **and** the user(software side) is ready for PD acquisition. To visualize, this *initiate* signal (see Figure 3.4) acts like a *floodgate* to control the stream of *water* (`PD` samples) entering the `IP` subsystem *via* `trig_peak_un_filter IP`, with the user having complete control over the 'floodgate' (after first complete period has elapsed). From Figure 3.4 it can be seen that `sys_init` also provides *current time* and *previous*

period data to the *phase detector IP* which is essentially a *divider* which waits for (*start phase calculation* in Figure 3.4) from `trig_peak_un_filter IP` after which it starts phase calculation (division) with *current time* and *previous period* data.

2. **Signal Processing Algorithms :** This part consists of discussions about of IPs handling the user selection of algorithms (signal from `PS`) and the time, frequency domain algorithms. A block diagram of these IPs is shown in Figure 3.6.

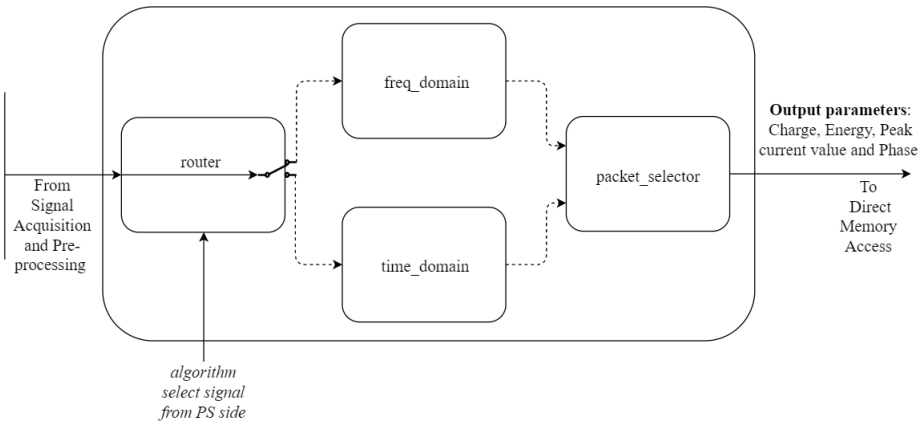

Figure 3.6: Block Diagram demonstrating inter-connections between IPs handling *signal processing algorithms* workload

(a) `router`(see 4.4.1 for details): This `IP` accepts data from *Signal Acquisition an Pre-Processing* workload handling unit and *routes* the data-sets (sets of PD samples) of *recording period* length to the `IP` handling algorithm selected by user (from `PS`). If the user changes *algorithm select* signal (see Figure 3.6) in the middle of a recording period (which represent one PD pulse), the `IP` waits for the recording period to finish before handling the user's request (for data sanity purposes).

(b) `algo_freq`(see 4.4.2 for details): This IP implements frequency domain algorithm for PD parameter extraction. The outputs of this IP are the required charge, energy, peak (polarity is simply the polarity of peak) and phase for acquired PD samples of length *recording period*.

(c) `algo_time`(see 4.4.3 for details): This IP implements time domain algorithm for PD parameter extraction. The outputs of this IP are also the required charge, energy, peak (polarity is simply the polarity of peak) and phase for acquired PD samples of length *recording period*.

(d) `packet_selector`(see 4.4.4 for details): This IP is responsible for selection of a valid set of output samples from one of the Algorithm handling `IP`s (*freq_domain/time_domain*). Further, DMA requires a signal(T_LAST) which indicates the end of each packet to be transferred to PS side. The packet_selector accepts this information from the user and plugs it (in the

form of a *last* signal) with the output samples, thus indicating the boundaries of a DMA packet. The user, however, enters this information in the form of *number of packets*.

NOTE that this *number of packets* is the number of output packets from the IP subsystem. Thus, each such packet will have 4 samples (4 output features). The packet length of a DMA transfer, is different, being (number of packets(user input) * 4). The ability to change the size of packet for DMA transfer has been provided, as there is an input port in this IP where designer can tie relevant constants. This is important because the size of DMA packet effects the DMA transfer bandwidth. As this is just the first stage of the bigger project, the length of the DMA transfer can be experimented with to find an optimum length to achieve desired transfer bandwidths.

3. **Data transfer from FPGA to Processing System of Zynq :** For the data transfer from PL to DRAM, as indicated in section 2.10, PL AXI DMA is utilized.

Finally, an interrupt based C based software application to configure DMA and IP subsystem is also designed.

3.2.1 Packet Processing

One of the requirements is that the user can choose between a recording periods of 1 to 10 us. With the `ADC` sampling period of $8ns$, this implies the recording period can alter between 125 samples to 1250 samples. Each set of samples (whether it contains 125 or 1250 samples) is intended to represent a particular `PD` pulse. Thus, each set, after being processed in the `FPGA` by a chain of `IP`s, should quantify the behavior of that PD pulse in terms of the four output features, i.e *Pulse Charge*, *Energy*, *Peak* and *phase*. Moreover, the user also has the liberty to choose from one of the two algorithms. Thus, each set of incoming samples can only be associated with one algorithm.
In order to efficiently process these 'sets' of samples of varying lengths, the notion of a **packet processing system** is introduced in this thesis work. The advantages of having such a system are summarized below:

- In addition to having samples of the user specified lengths, these packets also carry control information. Thus, instead of individual samples, each `IP` in the system now reads successive packet *words* (samples in a *frame*) arriving at their input ports at every clock cycle and take decisions about what to do with that sample (and the upcoming packet *word*) based on some *field* of the current packet *word*. Thus explicit control signalling can be avoided and a **natural flow of samples** downstream the `IP` chain can be ensured.

- Because of this packet processing approach, in the chain of pipelined IPs involved within the system, each IP can contribute something to the final output, tag the packet with its contribution and the following IP can start working from that point onwards. Thus, the outputs *evolve* over time and packetizing helps in systematic **workload distribution** among `IP`s sitting at different stages in the pipeline.

- The packets are created by the initial sender IP and reassembled by subsequent receiver IPs which are bound to adhere to the protocol specifications. Hence, any outside interference (user interference in this case) is avoided by all IPs at all time instants, thus ensuring **data sanity**, leading to a **robust** architecture.
- The high level constructs like structures offered in C++ can be leveraged to produce such packets. Vivado HLS offers the capability to convert these C++ structures into *buses* of equal widths as that of the members of the structure summed up together. One can now imagine that the C++ programmer can have the capability to handle each set of wires in the bus (fields in the packet *word*) *via* these systematic structures, which offers a great **ease of design** to the programmer.

Thus, in Figure 3.3 the set of IP(s) handling workload 1 prepare and pre-process packets of data. Further, all transactions ,to and from the set of IPs handling workload two are handled in sets of PD samples i.e packets, rather that individual samples. Further, the DMA also transfers data from PS to PL in the form of packets of data. The **notion of a packet** however, in this case is different and depends upon when the DMA receives T_LAST signal. In the upcoming chapter, these packets can be understood in greater detail and the design choice of opting for such a system can be appreciated.

Implementation and IP verification

4

This section takes the reader through the process of converting the block-level design discussed in the previous chapter to an elegant FPGA architecture. The chapter provides comprehensive implementation details about the hardware-software co-design. The employed design and testing methodology is discussed followed by description of a test case. Using this test-case, the functionality of each sub-block in the system is verified.

4.1 FPGA Design and Verification Methodology

In the process of realizing FPGA hardware, *design* and *verification* are intertwined. The Design and verification approach is summarized in the following points.

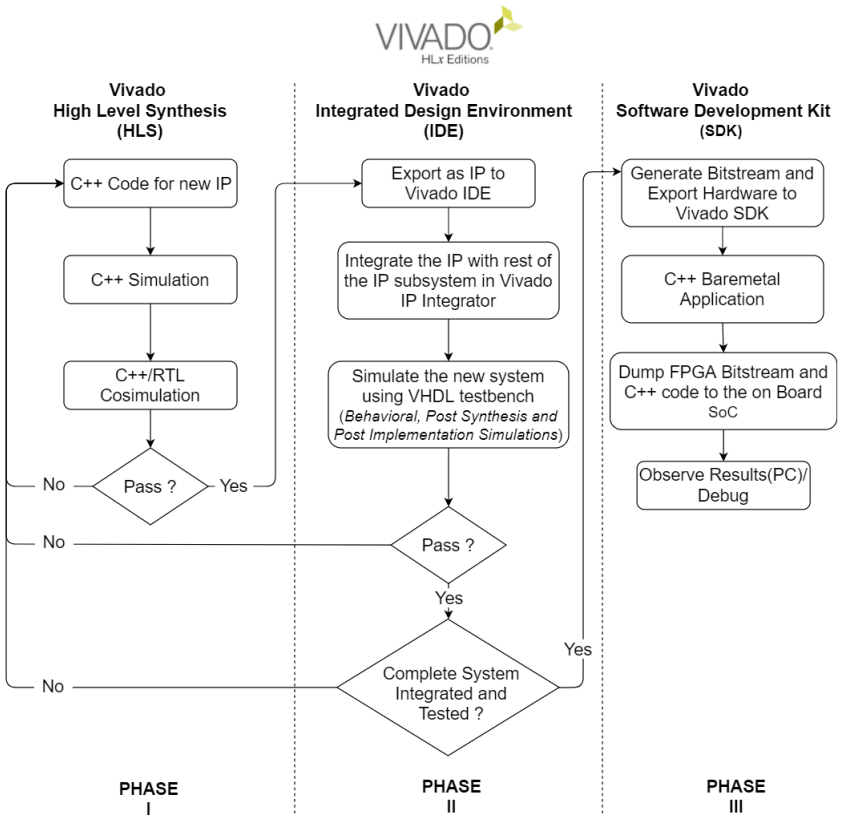

Figure 4.1: Three phase Design and Verification Approach

1. A **Bottom Up** design approach was adopted during the course of this thesis. The complex task of high speed PD pulse acquisition and parameter detection was divided into a number of much simpler tasks/functional blocks.

2. Each functional block is realized as an individual *Intellectual Property* (*IP*) in the design and performs a dedicated function. IPs are made using High Level Synthesis and tested rigorously.

3. `Vivado`'s three phase verification approach was utilized (Figure 4.1) where basic verification is done using C++ based simulations and co-simulations (HLS environment) for each individual IP (phase 1). This is followed by verification in `Vivado` IDE (Phase 2) using VHDL test-benches which gives more control over test inputs, hence facilitates thorough testing.

4. Phase 3 is dedicated to final testing of hardware software function correctness and will be elaborated towards the end of this chapter.

Further, two `ADC` inputs are required for getting the external analog `PD` pulses and sinusoidal reference signal into the `FPGA` side of Zynq 7010 `SoC`. However, for testing the `IP` subsystem that will eventually cater for accomplishing the task at hand, real `ADC` hardware cannot be used (for final board testing). One way to tackle this problem is to drive the IP subsystem's `ADC` inputs (*axis* compatible) directly from the VHDL testbench's outputs. However, there are a couple of shortcomings with this approach in our scenario.

- Although one can verify the entire IP subsystem using such a testbench, it would be impossible to integrate the ADC behaviour also into the actual design in hardware without using an actual ADC. Thus, in the final testing, one has to put the entire setup of analog signal generators connected to the `FPGA` platform which is further connected to a computer (*via* `UART`) in order to observe results as shown in Figure 4.2. This directly implies having a lot less control on the actual inputs that the `IP` subsystem is receiving. Thus, there is a high probability that the *golden reference* will not match after system integration even when the `IP`s are functioning as intended.

Figure 4.2: A testing approach (after Software-Hardware integration)

- Out of personal experience with the *Red Pitaya* platform, it was observed that although prototyping is possible, it does not comfort programmers with a hassle free, rapid prototyping experience. This makes sense because the platform itself is not really meant for `FPGA` prototyping. Rather, it is meant to be used as an *off the shelf* testing and measurement tool. For this reason, a switch of platforms was made to a much user-friendly `ZYBO` board [26] which has exactly the same `SoC` on board and hence, same resources as that of the *Red Pitaya*. However, `ZYBO` lacks the fast ADC support. Thus, if this approach was used, there would be no way to test the `IP` subsystem on `ZYBO` after integration in the real hardware.

Due to these shortcomings, it was decided not to simulate ADC behaviour within the main testbench. Instead, both the ADCs were modelled as dedicated IP's (ADC emulators). These IPs can now act like *testbenches* and be used for verification (both functional and timing), during the entire project implementation phase. Further, after software hardware integration in VIVADO environment, these emulators can also be put into the FPGA (along with the rest of the IP sub-system) in the form of bitstream. Thus, in the final testing phase, outputs (observed on PC) can directly be matched to the *golden reference*. Also, there is no need to use Red Pitaya, instead, the much programmer friendly ZYBO board can be utilized. Hence, instead of Figure 4.2, the final testing approach looks like the one as shown in Figure 4.3.

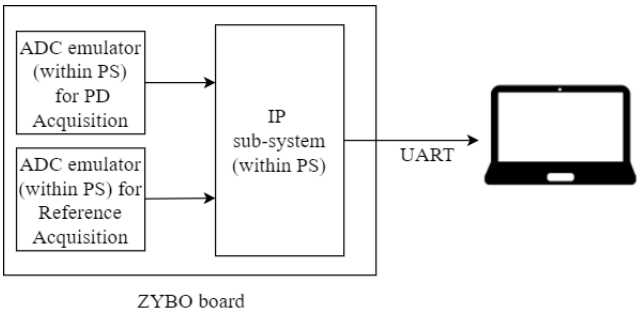

Figure 4.3: Final test setup after software hardware integration

These ADC emulators (4.3) will emulate samples coming out of ports *M_AXIS_PORT1*(ADC0-PD) and *M_AXIS_PORT2*(ADC1-Reference) shown in Figure 3.1.

4.2 IP Verification Environment Setup

Before discussing the actual FPGA implementation, a glimpse of the setup used for verifying the functional and timing correctness of the IP subsystem is provided. For testing the functional correctness of the designed IP subsystem, it is required to have an input data-set for which the correct outputs (*golden reference*) are known beforehand. Hence, input data-sets representing PD pulses was produced using MATLAB. As discussed previously, the measuring setup shows a band-pass filter behaviour. To retrieve the golden reference, current pulses are passed through this filter in MATLAB as can be observed from Figure 4.4. This gives the V_{meas} pulse which will be observed at *red pitaya's* input ADC0. For a reference, this V_{meas} signal is passed to MATLAB algorithms (double precision floating point computations) and *Golden Reference Outputs* (Q, E, I_{peak}) are retrieved. However, the voltage range of -1 to 1V will be perceived as 0 to 16383 quantized levels by Red Pitaya's ADC0. This quantization was simulated in MATLAB by allocating a specific level to each voltage (V_{meas}) sample among 16384 total levels (based on resolution based scaling, offset addition and rounding down to nearest level). The retrived values are stored in arrays (BRAM in FPGA) and can act as ADC0 emulator. The output (stream of PD pulses in 16384 quantized ADC levels) is then passed at every clock (8ns-corresponding to sampling frequency) to the Vivado HLS

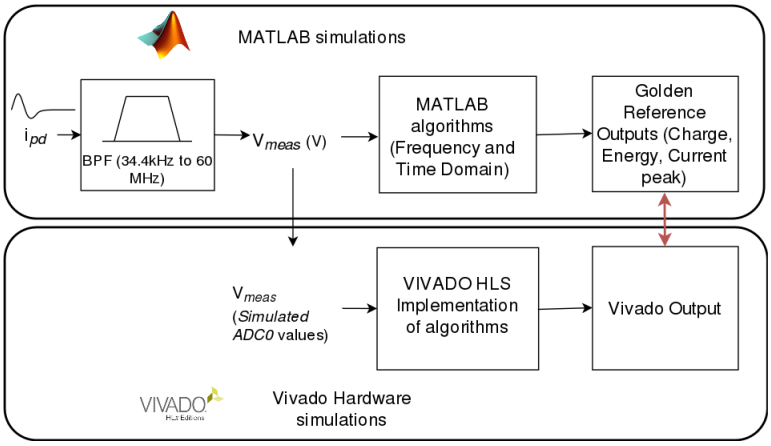

Figure 4.4: Functional Verification Process

algorithms (IP subsystem) in both Phase 1 and Phase 2 environments and outputs are verified against *golden reference*.
Note here that this thesis report, only covers the outputs of comprehensive testing from phase 2 (Figure 4.1) and Figure 3(proof of concept).

A single test-set is presented in this section give the reader a clear understanding of how the IPs are reacting to the input stimulus (Emulated ADC outputs). This stimulus set is shown in Figure 4.5.

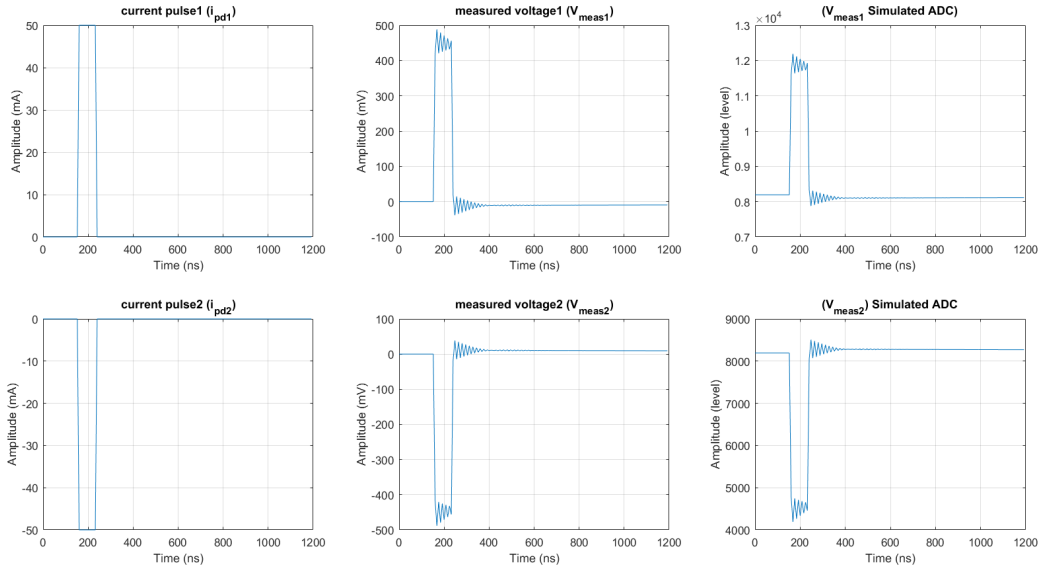

Figure 4.5: Test set 0 - Simulated ADC values for two pd pulses (50mA and -50mA), pulse width = 80ns

A pair of consecutive incoming pulses (50mA and -50mA) (ADC simulated - Figure 4.5) stored in an ADC emulating IP is utilized for discussions about IP verification in

this chapter.
`MATLAB` based outputs (Q, E,I_{peak}) for these inputs are tabulated:

		Time Domain	**Frequency Domain**	**I_{peak}(mA) [after BPF]**
50mA	Charge Q(nC)	3.9009	4.0123	53.6824
	Energy E (nJ)	0.3247	0.3192	
-50mA	Charge Q(nC)	-3.9009	4.0123	-53.6824
	Energy E (nJ)	0.3247	0.3192	

Table 4.1: Golden Reference (*test set 0*)

Further, for sinusoidal reference generation, another emulator(IP) is utilized which generates a 20ms sinusoidal wave having a phase offset of $\pi/3$ as shown in Figure 4.6.

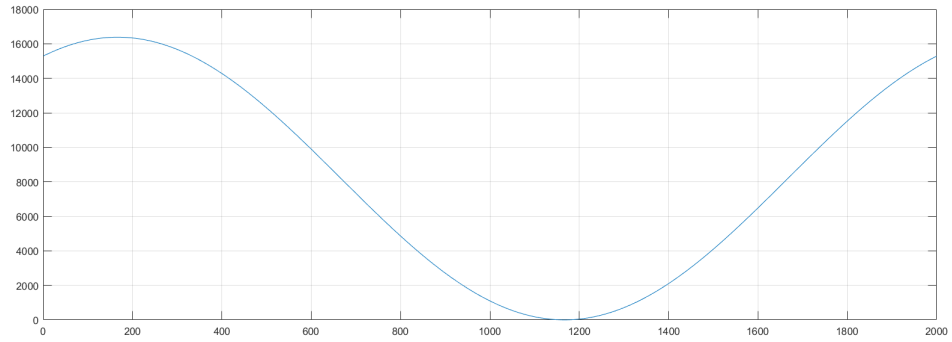

Figure 4.6: Test phase - Simulated ADC values for sine wave with $\pi/3$ offset

A total of 2000 samples constitute the waveform shown in Figure 4.6. Although the ADC specification is 125Msps, this IP spits out new samples at a rate of 100 Ksps (the acceptance rate of `sys_init` IP). This is done due to the fact that if actual sampling rate is chosen, the samples corresponding to 20 ms that have to be stored in the emulator would be 2500000 corresponding to a storage requirement of 4.7MB which is unavailable in FPGA BLOCK RAM. As these emulators were intended to be employed both for verification as well as FPGA testing (proof of working), the sample outgoung rate for this IP was decided to be 100Ksps. This is as good as the real scenario because even then, the consumer IP would anyways consume at 100 Ksps, regardless of the input sampling rate for `ADC1`.

4.3 Signal Acquisition and Pre-Processing Workload

The `trig_peak_un_filter`, `Sys_init` and `phase_detector` IPs constitute the Acquisition and Pre-processing stage discussed in chapter 3.2.

4.3.1 System Initiator (`Sys_init`) IP

4.3.1.1 Functional Description

- **Acquires** sinusoidal reference samples from its *axis* compatible input port(*sin_ref_V_V*) at a rate of **125Msps**. However, it **accepts** the samples at a much slower rate (down-sampling) of **100Ksps** (2.7 times greater than 36Ksps requirement - section 2.4).

- Grants permission (*start_sys_out_V*=1) to `trig_peak_un_filter` for acquisition of `PD` pulses if user(`PS`) is ready (i.e *user_start_V=1*) and at least one complete period (two consecutive positive zero-crossings) of sinusoidal reference signal has elapsed. Dismisses the permission (*start_sys_out_V*=0) when any of the two is not true. This feature can be used for controlling ***acquisition time*** from the software side.

- Provides current time (*current_cnt_out_V* updating at a rate of 100 KHz) and previous period (*prev_period_out_V*) information to *phase_detector* IP, hence facilitating phase calculations based on Equation 3.2.

4.3.1.2 Implementation

The `IP` tracks the required *previous period* and *current time* (Equation 3.2) information by means of a counter which starts at each positive zero-crossing of input reference signal and counts until the next positive zero-crossing. Further, there is another counter to maintain the count of positive zero-crossing events and renders rest of the IP-subsystem in *wait state* until two such events have elapsed, thus, maintaining phase computation *reliability* (3.2). Here, it is important to understand the design choice of having a **sample acceptance rate of 100Ksps (*down-sampling*) and not 125Msps**. Consider a case when up-counting rate is chosen the same as the sampling frequency i.e 125Msps(8ns). In this scenario, for each varying time period, say exactly 20ms(50Hz) of sinusoidal reference signal, the IP will count from 1 to 2500000. Now these counts have to be fed to the `phase_detector IP` which will will employ a divide operation to calculate phase. The number 2500000 requires at least 22 bits to be represented in binary. However, if we lower the acceptance rate to 100Ksps (which is >36Ksps requirement), for the same 20ms period, the maximum count can now only be up to 2000, which requires only 11 bits to be represented. This directly implies that the divide operator (which is inherently computationally complex and has relatively higher area and latency requirements than other basic arithmetic operators) used in the consuming IP (`phase_detector`) will be much simpler in case of a 100Ksps acceptance rate. At the same time the required *resolution* for phase calculation (0.5 degrees) is intact as for every 360 degree (20ms), there are *2000 distinct counts (current_cnt)*, which implies a resolution achieved of **0.18 degrees** (360/2000) for phase calculations.
The IP offers a *latency* and *Initiation Interval(II)* of 1 cycle each.

4.3.2 Trigger|Peak|Filtering (trig_peak_un_filter)

4.3.2.1 Functional Description

- Handles the acquisition of the incoming stream of samples arriving from the ADC every 8ns and scales the input samples to the actual analog voltage interval of -1V to 1V (*Red Ptaya's* Low Voltage ADC jumper setting).
- Provides a systematic solution to the requirement for having a **threshold based trigger** mechanism and a **pre-trigger** provision for recording the history of samples.
- Packetizes the input sample stream into sets of samples, with each set having its own control information (according to user specified settings) for controlling the other IPs downstream in the IP chain.
- Contributes towards the final **peak current** output. Finds peak voltage value (magnitude-wise) within a packet and tags the last *word* of the packet with that value.
- Indicates the phase_detector IP when to start phase calculations corresponding to each packet.

4.3.2.2 Implementation

The IP operates at a clock frequency of 125MHz in order to be able to match the rate at which ADC0 is sampling. Thus, it accepts a stream of ADC data samples representing the PD pulse every clock cycle *via* its input AXI stream (axis) slave port *input_stream_V_V*. The *pragma* HLS PIPELINE II=1 was employed in the code to enforce a pipelined RTL design which can accept new ADC samples every 8 ns.
The IP only starts accepting new PD samples when it receives *permission* from sys_init IP at its ap_start port (block level interface), which controls when the block can start processing data. Thus, this *permission* acts like a *flood gate* to allow input PD stream into the IP subsystem and can be tweaked for providing the *acquisition time* utility (from software).
The IP is essentially a state machine with two states.

1. State 0: The IP accepts an ADC data sample (from *input_stream*) and scales it back to its original voltage level ($\pm 1V$). Further, the sample is checked for trigger conditions, if it is greater than either positive or negative trigger values specified by user. If the trigger condition meets, the state of IP changes to 2 for the next incoming sample. Also, at such an event, the IP issues a request to phase_detector IP (*start_ph_calc* =1 for two consecutive cycles as minimum *II* for phase_detector is 2) to begin phase calculations.
   The IP saves a history of samples to provide the required *pre-trigger* functionality. From table 2.1, it can be observed that the maximum required pre-trigger is of 125 samples (corresponding to 10us recording period). As can be seen from Figure 4.7, the maximum storage for pre trigger is allotted to be 126 samples (1 more

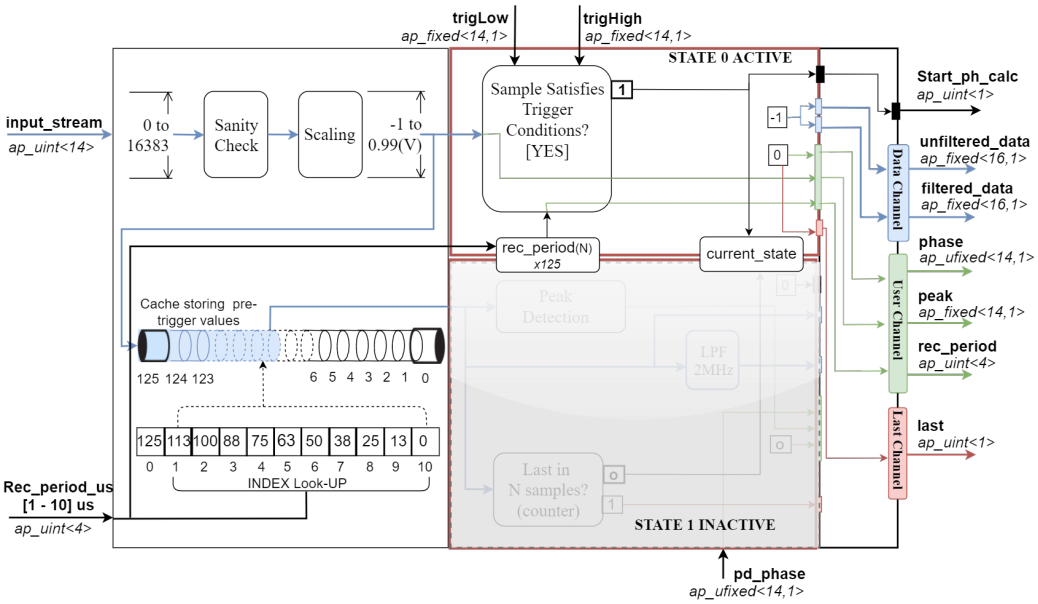

Figure 4.7: `Trig_Peak_un_filter` in STATE 0

than required for each recording period).
The size of this cache for storing history is altered on the fly based on recording period value entered by the user (software) as there is an index value (to determine cache length) corresponding to each recording period (1-10). If any othe value is entered, the system will not make a transition to state 1. The cache (length 126 as shown in Figure 4.7) employs a least recently used (LRU) policy where samples older than required history are *shifted out*. Each cycle, shift operations are preformed to achieve this utility. For for loop for shifting is unrolled using pragma `HLS UNROLL` in order to support parallel accesses for shifting. Further, due to the fact that all elements (126) have to be accessed every cycle, a BLOCK RAM cannot be employed it would not be able to allow these many accesses per cycle (due to insufficient ports). Thus, another *pragma* `HLS ARRAY_PARTITION` is employed to partition the array into LUTs. Moreover, if the recording period is changed, *wait states* are introduced in between for garbage value prevention.

2. `State 1:` In this state, as can be seen from Figure 4.8, the cached (pre-trigger) samples along with the incoming samples are accepted.
The AXI4-Stream Interfaces with *Side-Channels* [1] were employed for encapsulation of samples into packets for IPs downstream to consume. These side channels can be realized as *structures*(convenient) in C++ where each field in the structure corresponds to a particular channel (should have valid names- *data, keep, strb, user,last, id* or *dest*). In this design, along with the data channel (32 shared bits for filtered and unfiltered data), side channels *user* (32 shared bits for phase, peak and recording period) and *last* (1 bit) were utilized and relevant information (control and data) was packed into different channels of AXI stream interface every clock cycle.

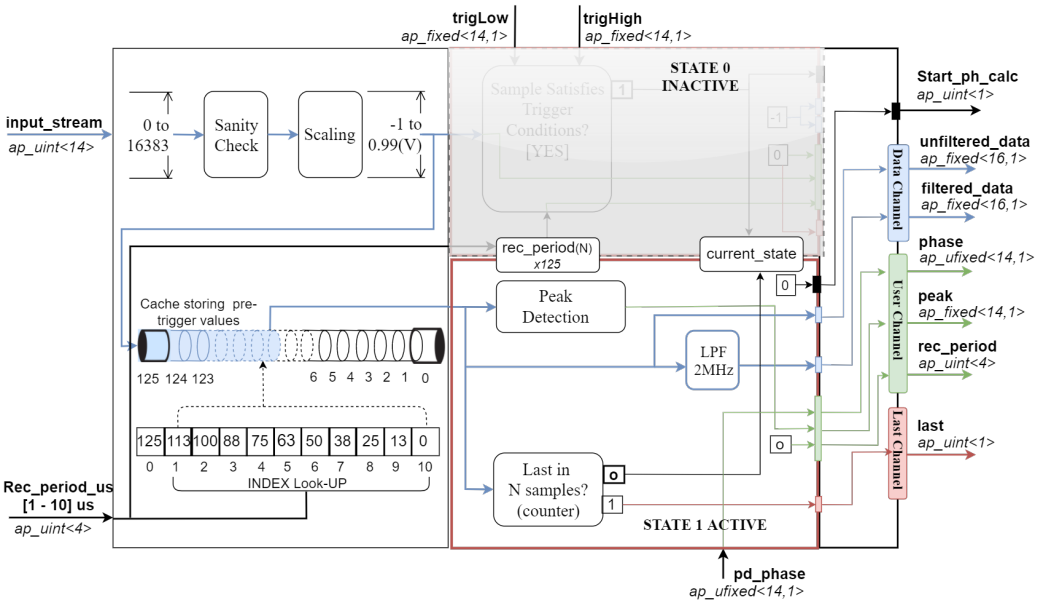

Figure 4.8: `Trig_Peak_un_filter` in STATE 1

The incoming samples are passed through a butterworth low pass filter (2MHz) and sent to the IPs downstream via *data* channel. Further, unfiltered PD (required for energy calculations) is sent through the same *data* port.
The recording period (utilized by `algo_freq`) is streamed out *via user* channel at the occurrence of PD trigger event and marks the beginning of a packet (Figure 4.9) . Successive peak (voltage) detection is performed on each incoming sample and sent to IPs downstream via the peak (sub-field) of *user* channel. In this way, the voltage peak for the recording period is available at the end of the packet (4.9) which can further be utilized by other IPs (down-stream) for current peak calculation. In state 0, phase calculation request was sent to `phase_detector`. Because the `phase_detector` IP (divider) has a latency of 58 cycles, relevant phase information is available at the *pd_phase* (4.8) port only after 58 cycles. Since the smallest packet size to handle is 1us (125 samples or 125 cycles), the valid phase information will always be available by the end of all packet *frames* (***dataflow***). This phase result is sent to the IPs downstream *via user* port. Finally, the *last* field is plugged in with a *1*, indicating the boundary of the packet (along with *recording period*).

From Figure 4.9, it is noticeable that two (*-1*s) are sent in the beginning of every packet. This is because the first *word* of every packet will eventually be *discarded* by the algorithm (frequency and time domain) handling IPs. The extra first *word* is just to provide an indication to other IPs that valid data is arriving from next cycle onwards until the *last=1*. This explains why an extra sample is cached in `state 0` of this IP.
Here, it can be noted that the notion of *recording period* deviates from its literal sense. This is because apart from the pre-trigger samples (history), *no samples*

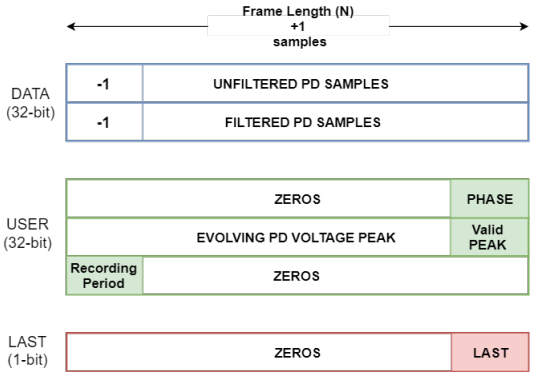

Figure 4.9: Encapsulated Packet

are being recorded. Instead, samples are *streamed* through the IP subsystem in the form of easy to handle *packets* (variable *packet frame lengths-recording periods*) which essentially renders an essence of a recording period.

- Butterworth Low Pass Filtering (2MHz): Equation 2.9 governs the design of this filter. The multiply and add operations are absorbed into `DSP48s` (HLS resource allocation). These low latency DSPs are allocated in successive cycles of the pipeline, ensuring that pipeline *II* of 1 is respected and the the operations fit within each clock cycle (8ns) at the same time.
  Moreover, with a clock cycle of 8ns, it was observed that fixed point rounding schemes cannot be applied for each filtered value (corresponding to each *sample* in a *frame*) due to rounding latency overhead. Instead, the output (*y[n]* in Equation 2.9) is allocated a big register (25 bit) with no rounding schemes. This register value is further pushed outside the block *via axi stream* port, where 25 bit register is finally rounded to a 16 bit output. Here, filtering function is in-lined (*pragma* `HLS INLINE`) with the calling function to reducing function call overhead, improving latency of each pipeline stage within the block, hence adhering to the required 8 ns clock constraint.

The IP offers a *latency* of 4 and *Initiation Interval(II)* of 1 cycle.

4.3.3 Phase Detector (`phase_detector`)

4.3.3.1 Functional Description

- Waits for *start_ph_calc(=1)* signal from trig_peak_un_filter IP and returns trig_peak_un_filter the computed phase value (corresponding to the trigger event) after a latency of 58 cycles.

4.3.3.2 Implementation

The IP is a state machine with two states. In the first state (`state0`), the IP checks if it received the *start_ph_calc* request from *trig_peak_unfilter* IP. As the IP has an II of 2, it can only accept new inputs in 2 cycles. Thus, *start_ph_calc* signal is pulled up

to *1* for two cycles (by trig_peak_un_filter). Further, If no request is received, the IP remains in `state0`.
As soon as it receives a request, it latches the *current time* and *previous period* information available at its *current_cnt* and *prev_period* ports respectively (coming from sys_init IP). At the same clock, the state variable makes a transition from 0 to 1 state. In `state1`, a divider instance is employed which divides *current time* by *previous period* to retrieve phase information. No optimization directive is required for this IP as there is an available window of 125 cycles (*smallest frame length*) which is sufficient for the phase output to be generated.

4.3.4 Verification

The self explanatory verification scheme is shown in Figure 4.10.

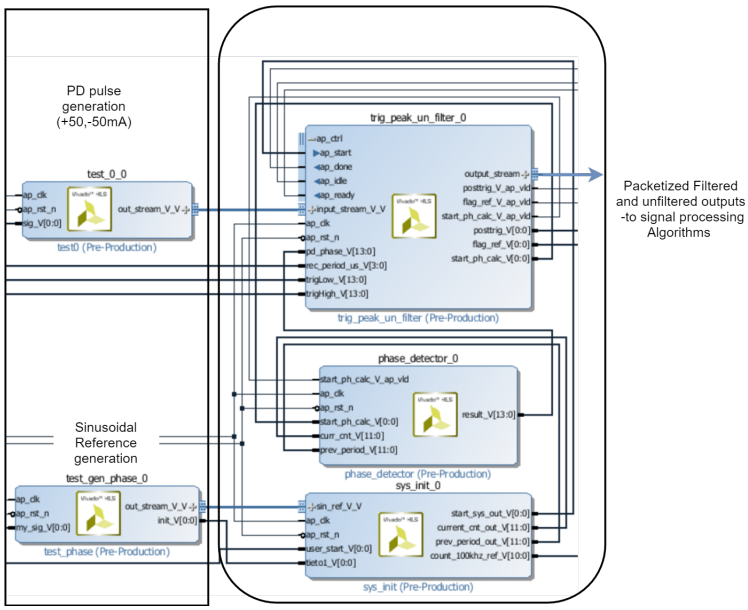

Figure 4.10: Verification scheme for Acquisition and pre-processing

The `sys_init` IP starts accepting the sinusoidal reference stream at an acceptance rate of 100Ksps. The response of IP to the input sinusoidal stream. The sinusoidal refer-

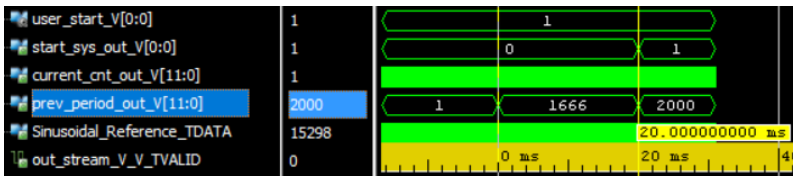

Figure 4.11: Response of sys_init

ence signal has a phase offset as shown in Figure 4.6. The IP does not issue the start signal (start_sys_out) until two positive zero crossing events have not been detected.

As the *user_start* is *1* and two consecutive zero crossings are detected, the IP grants permission to start PD acquisition. The IP also updates *current count* information at its output port *current_cnt_out* at 100Ksps (every 10 us). Further, it also updates *previous period* information after each period elapse (between consecutive positive zero crossing events) at its output port *prev_period_out*.
After the start signal is received by `trig_peak_un_filter` IP, it is allowed to receive input PD stream from ADC emulator as shown in Figure 4.12. If the input PD exceed

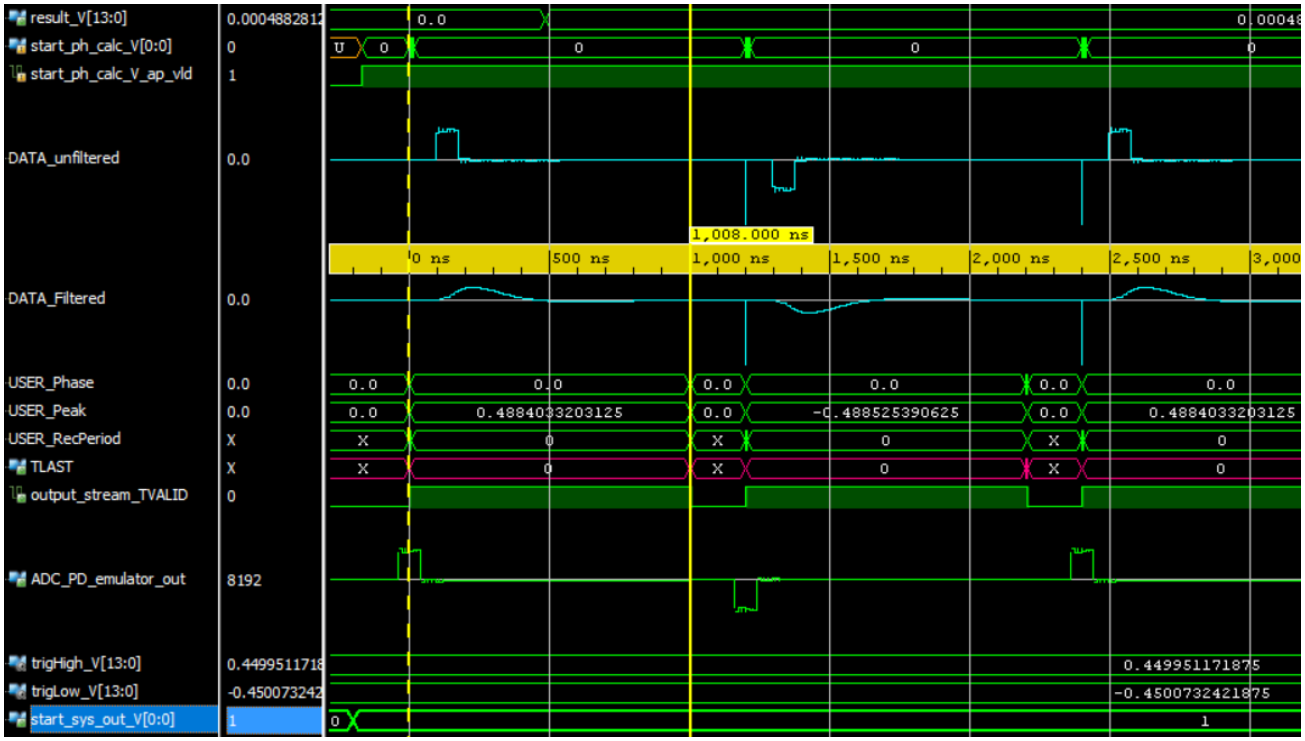

Figure 4.12: Response of `trig_peak_un_filter` and `phase_detector`

the employed trigger (voltage) values (here 0.45, -0.449V), the IP switches to its `state 1` and starts producing packets as shown in Figure 4.9 (after its own latency of 4 cycles). As can be seen from Figure 4.12, the required output filtered and unfiltered data are produced. Notice the delay between *ADC_PD_emulator_out* and the output valid packet's filtered and unfiltered data. This delay is a consequence of fulfillment of pre-trigger requirement. Thus, the data channel outputs the required history along with data. Also, notice that as soon as a trigger event is detected by trig_peak_un_filter IP (negative *-1* spike, beginning of the packet), the IP issues the start_ph_calc request for phase calculation to `phase_detector` IP. However, `phase_detector` IP returns the *result_V* (phase information) after its latency of 58 cycles and thus, the result is available at the 59th cycle (less than 126, so *dataflow* is maintained), which is further plugged in the last sample (*user* channel) of the packet. Here, recording period of 1(us) was selected. As the packet length is N+1 samples, here it is 1.008us (corresponding to 125+1 samples). This packet along with data, encapsulates the voltage peak, phase

and Recording period information and *last* signal at positions according to Figure 4.9 and forwards these packets to the *Signal Processing Algorithm* workload. (3.2).

4.4 Signal Processing Algorithm Workload

The IPs `router`, `algo_freq`, `algo_time` and `packet_selector` constitute this workload as discussed in section 3.2. These IPs are briefly discussed.

4.4.1 Router(`router`)

4.4.1.1 Functional Description

- The IP is responsible for reliable redirection of the incoming packets to one of the two available feature computation choices (`algo_freq` or `algo_time`) based on the user *algo_sel* input.

4.4.1.2 Implementation

This IP is a simple state machine with two states (0 and 1). In state 0, the IP waits for the beginning of a packet (arriving at its input port *via* axis interface with side-channels). The beginning of the packet is detected if recording period (in the *user* channel) is anywhere between 1 to 10. As soon as a packet arrival is detected, the sample is redirected to one of the output axis ports (with side-channels)based on the user choice (*algo_sel*). The user choice is saved and the IP changes state to 1. Further, for the entire length of the packet (detected by *last* field/channel) the IP redirects samples to the saved user's choice of algorithm. Thus, a packet cannot be corrupted if the user choice changes in between of the packet.
The IP offers a *latency* and *Initiation Interval(II)* of 1 cycle.

4.4.2 Frequency domain Algorithm(`algo_freq`)

4.4.2.1 Functional Description

- Performs feature (**Q, E, $\mathbf{I_{peak}}$**) computations using frequency domain Equations (2.1,2.6,2.8) and streams them out along with **phase** successively *via* its *axis* output port (*s_out_freq*) to the `packet_selector` IP downstream.

4.4.2.2 Implementation

The implementation for this IP is along the lines of *design perspectives* discussed in section 2.5.
The IP is a state machine. In the first state, the IP waits for the detection of arrival of a new packet is detected (recording period between 1 to 10). As soon as a new packet arrives, the IP initializes all the accumulation registers to 0 (i.e accumulators for successively accumulating multiplications of incoming samples with sin and cosine required for charge and energy). As discussed, the sines and cosines corresponding to all *recording periods* from 1 to 10 us are stored in BRAMs (Figure 2.16). The recording

period information is extracted from the packet to retrieve initial indices for look-ups of sin and cosine. Then the state is switched.
In the second state, the IP accepts filtered and unfiltered PD samples arriving at its *axis* input *data* channel and multiply them to the look-ups according to equations 2.6 and 2.8. For smaller multiplications like *filtered_data*(16-bit) * respective sine/cosine (16-bit), LUT based instances are used. As the multiplication data-path grows bigger, one or more dedicated `DSP48` units are utilized for multiplication. For instance, for energy calculations, unfiltered_data (16bit) * unfiltered_data (16bit) (=32 bit output) is performed using 16*16 LUT based multiplier instance. However, when this 32 bit output is multiplied to a 16 bit look-up (sin/cosine), the 32*16 multiplication is absorbed into two dedicated `DSP48` units as each DSP possess only one 25*18 multiplier. Further, the IP waits for *last=1* (arriving at last channel of axis input), indicating packet boundary. When such an instance is detected, the IP finds the magnitude ($\sqrt{a(32bit)^2 + b(32bit)^2}$) using accumulations corresponding to both charge and energy, which corresponds to 16 more `DSP48` instances. At this point, the square root has to be performed to retrieve features **Q** and **E**. This operation is computationally expensive. The square root core which `Vivado` binds to requires high latency and utilization.
Moreover, the last sample in every packet has Voltage peak and phase information corresponding to the packet. Thus I_{peak} is calculated by scaling Voltage peak and phase data is stored. Lastly, the 4 required outputs **Q, E, I_{peak} and phase** are saved and streamed out in the next 4 cycles from its output axis port (no side channel). The *latency* of the IP is 60 cycles out of which 44 can be attributed to the *sqrt* operation. Further, as desirable, the IP has an II of 1, enabling it to accept new PD samples every cycle.

4.4.3 Time domain Algorithm(`algo_time`)

4.4.3.1 Functional Description

- Performs feature (**Q, E, I_{peak}**) computations using Time Domain Equations (2.1,2.3,2.4) and streams them out along with **phase** successively *via* its *axis* output port (*s_out_freq*) to the `packet_selector` IP downstream.

4.4.3.2 Implementation

The implementation of this IP is exactly along the lines of design perspectives discussed in section 2.5. The only implementation detail is that now, the algorithm is modelled as a state machine similar to the one described for Frequency Domain calculations. The IP is resource friendly. Four features corresponding to each input packet are produced by this pipelined IP after its *latency* of 2 cycles. The IP also offers an *II* of 1.

4.4.4 Packet Selector(`packet_selector`)

4.4.4.1 Functional Description

- Selects valid stream (of 4 features) among the two streams arriving at its two input axis compatible ports (no side channel).

- Streams out the valid stream to the DMA along with *tlast* side channel information based on the constant tied to its *num_packets* input port.

4.4.4.2 Implementation

This IP is again a state machine with two states. In the first state, the IP checks for a valid input arriving at either of its input ports. At one instance in time, only one stream will be valid because the `router` routes PD packets to only one algorithm. As soon a valid input is detected, the feature is pushed through the IPs *str_out* axis port. Now, the state is switched and three remaining consecutive samples are sent successively (one after the other) to the DMA. In this state however, the IP keeps a track of the number of these output packets (consisting of only 4 features each) and plugs in the *tlast* =1 signal according to the value at *num_packet* input port. Thus, if num_packets is 20(N), the *tlast* bit will correspond to the fourth feature of 20th (Nth) packet i.e, the 20*4=80th (N*4th) feature. These 80(N*4) features will then constitute 1 DMA packet for transactions to DRAM.

4.4.5 Verification

The verification scheme applied for checking the functionality and timing of the IPs handling signal processing workload is as shown in Figure 4.13. As can be seen, the

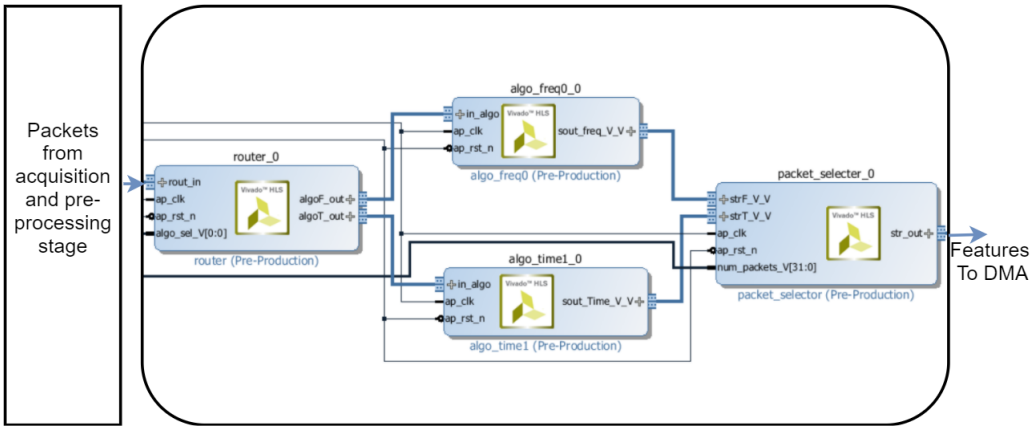

Figure 4.13: Verification scheme for signal processing algorithms stage

outputs from the previous stage (discussed in section 4.3.4) are applied to IPs on this stage and results are cross-checked with `MATLAB` implementation (Figure 4.4, table 4.2). In the simulation shown in Figure 4.14, algorithm selected is 0 (frequency domain). Thus, `router`'s *algoF_out_TVALID* port is 1, and `algoT_out_TVALID` is 0 for the entire length of packet(s). Further, after the data, user and last information is streamed out of `router` to `algo_freq`, features are computed and a valid stream of four features is generated by the IP (`algo_freq`) after its latency of 60 cycles (480ns). The `packet_selector` retrieves this stream and forwards it to DMA after its own latency of 1 cycle (total 488 ns from the last output packet sample leaving `router`) along with *str_out_tlast* signal as required by DMA (Figure 4.15).

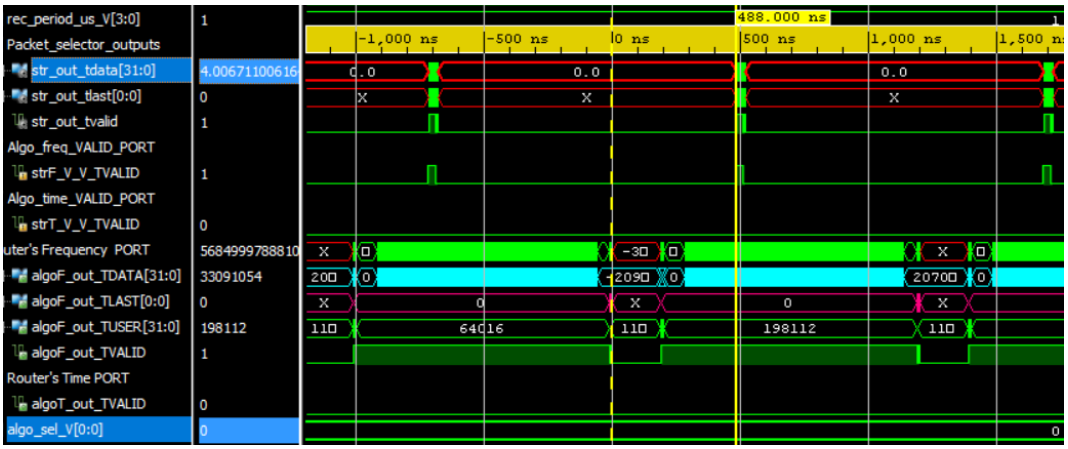

Figure 4.14: Response of signal processing algorithm stage (algorithm selected =0 [Frequency Domain])

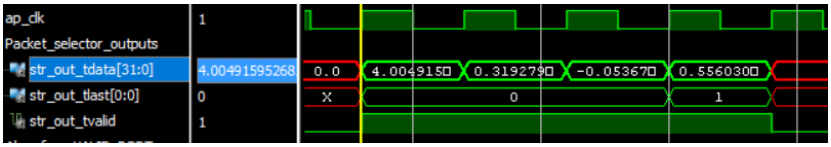

Figure 4.15: Features extracted (frequency domain)

The features extracted in time and frequency domain along with errors with respect to golden reference (4.2) are shown.

		Time Domain	Frequency Domain	Error%(Time)	Error%(Freq)
50mA	Charge Q(nC)	3.8922	4.0067	-0.22	-0.13
	Energy E (nJ)	0.32469	0.319	$-3.07*10^{-3}$	-0.06
-50mA	Charge Q(nC)	-3.8956	4.0049	-0.136	-0.18
	Energy E (nJ)	0.3248	0.3192	-0.03	0.02

Table 4.2: Extracted Feature verification

As can be observed, for this ideal case of 50mA and -50mA current pulses, both the implementations deviated from `MATLAB`'s golden reference by less than 0.3% even when these errors also involve errors due to quantization (inputs to implementations are quantized ADC levels).

4.5 Hardware software co-design and integration

The entire IP-subsystem (hardware) discussed above is wrapped into a single IP core (`P_DETECT`). However, the required input configuration has to be fed to this IP core using software running on the PS side. For this purpose, another IP (`user_config`) was designed (using `Vivado`'s create and package IP - new AXI peripheral utility). As can be seen from specifications (section 2.10), the AXI GP0 master port is utilized to set these configurations. In order to *talk* (accept configurations) to this port, this new

IP possesses an AXI lite slave port.
Five memory-map registers of the ARM processor are allotted for setting the required user configuration. Thus, before each acquisition, five configuration memory mapped registers have to be set using software. These configurations would then be reflected at the five output ports of `user_config` which are connected to `PDETECT` as shown in Figure 4.16. Also, another specification is to use PL DMA facilitate data (computed

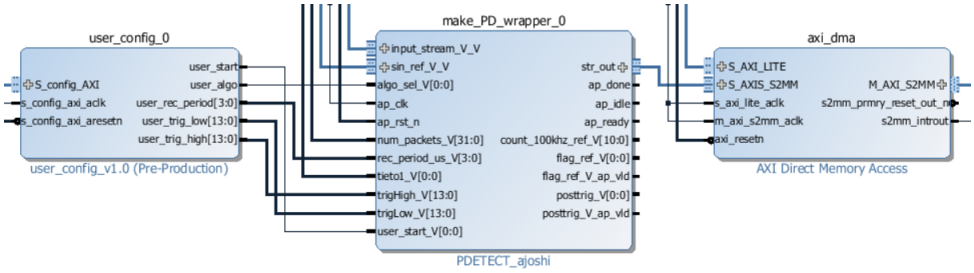

Figure 4.16: Configuration IP connected to IP subsystem

features) transactions from PL to DRAM *via* PS. This DMA is utilized in its default configuration (2^{14} byte internal buffer and a maximum burst size of 256). 256 burst size is chosen because there is a single stream of DATA to be transferred from PS to PL and there are no other PL AXI memory map masters. Hence, the highest throughput configuration can be safely chosen. Further, DMA's slave light port is connected to the same Master AXI GP port for its initialization and transaction configuration.

4.5.1 Software application

A standalone software application (setup - 4.3) was developed for on board (ZYBO board) testing of the implementation, the scheme of which is shown in Figure 4.16.

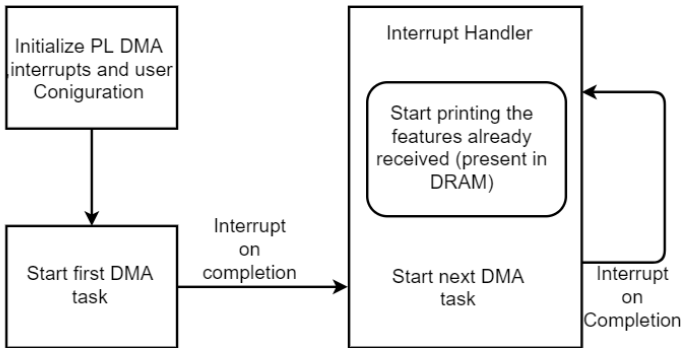

Figure 4.17: Software Application Scheme(`ARM 0`)

The DMA is configured in Direct Register mode. As can be observed, the AXI DMA and IP subsystem is first initialized with (user configuration). Also, the interrupt system is initialized. Further, the first transaction from DMA to DRAM is initiated by writing the destination DRAM address (in memory mapped register S2MM_DA) and the length of the packet required from DMA `in memory-mapped register`

S2MM_LENGTH register). As the interrupt system is initialized for generating interrupt on completion (IOC) of a data transfer task (PL to DRAM) for a packet (determined by the *TLAST* that *packet_selector* plugged in), the control goes to the interrupt handler. Within the interrupt handler, the values (extracted *features*) received in the previous transaction are simply printed on standard output. After completion of this printing task, the DMA is again configured for the next transaction (packet) with an increased DRAM address and the loop continues forever as shown in Figure 4.17.

4.5.2 Verification

This verification is in the third phase shown in Figure 4.1. In the previous discussions, the hardware outputs were found to be almost error free (Vivado). The same fixed point outputs in *signed decimal* format are shown in Figure 4.18

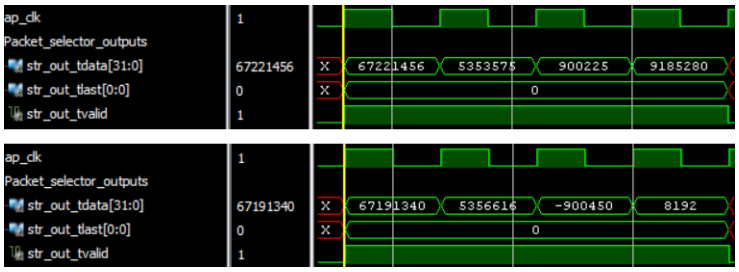

Figure 4.18: Extracted *features* for 50 (up) and -50 mA(down) PD - signed decimal notation - Frequency Domain - Vivado IDE

The printed outputs (*signed decimal notation*) of Vivado the software application observed in Vivado SDK are shown in Figure 4.19.

```
Rx data[1144] of interrupt cnt(1) = 67221456  C4011E0
Rx data[1145] of interrupt cnt(1) = 5353575   C4011E4
Rx data[1146] of interrupt cnt(1) = 900225  C4011E8
Rx data[1147] of interrupt cnt(1) = 40960   C4011EC
Rx data[1148] of interrupt cnt(1) = 67191340  C4011F0
Rx data[1149] of interrupt cnt(1) = 5356616   C4011F4
Rx data[1150] of interrupt cnt(1) = -900450   C4011F8
Rx data[1151] of interrupt cnt(1) = 40960   C4011FC
Rx data[1152] of interrupt cnt(1) = 67221456  C401200
```

Figure 4.19: Extracted *features* for 50 (up) and -50 mA(down) PD - signed decimal notation - Frequency Domain - Vivado SDK - Numbers on extreme right being DRAM addresses

As can be observed from Figure 4.18 and 4.19, the Q, E and I_{peak} values exactly match. Because in this case there are only two pulses, the application prints the same outputs corresponding to 50 and -50mA pulses over and over again.

Thus, it can be concluded that **correct features Q, E and I_{peak} are reaching DRAM** as expected. Moreover, as can be noticed, the phase information does not have a one to one correspondence. This however does not imply that phase values are incorrect. As can be seen from Figure 4.17, there are software delays involved between every time the DMA is programmed. The phase generation emulator IP on the other hand continuously produces the sinusoidal reference signal, without experiencing any *back-pressure* from the software (DMA *ready* signal). This implies that the notion of taking a reference for phase would be theoretically incorrect and hence, the values of phase cannot be verified without using actual ADC. What can be verified however is the resolution we are achieving for phase calculations.
The DMA for this test was configured for a packet length of 8192 bytes. This implies we can expect total 2048 *features* (4 bytes each) corresponding to 512 PD pulses (4 *features* each). Phase values received in DRAM for each of these 512 PD pulses show a ramp-like trend (as desired) after printing, demonstrating a constantly increasing phase as shown in Figure 4.20.

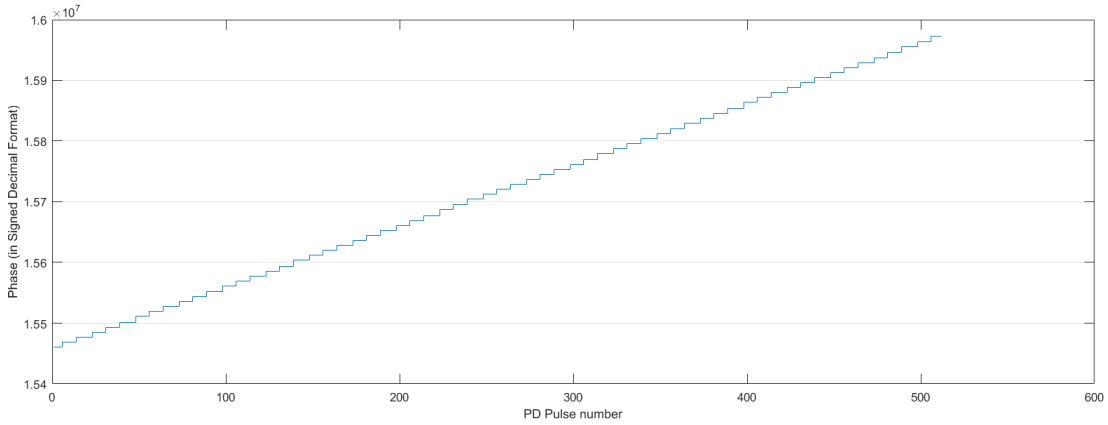

Figure 4.20: Constantly increasing phase corresponding to 1 DMA packet (512 PD in this case)

In section 4.3.1, it was claimed that 2000 distinct points (phase values) for every 360 degree of a 20Hz input signal is achievable.
The recording period in this test case was 1us with inter frame delay of 0.192us (Figure 4.21).

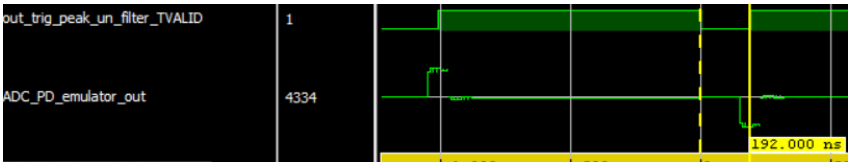

Figure 4.21: Inter (valid)frame delay

This implies that for the 512 pulses in a DMA packet correspond to a total time

of 610.304 us. Thus, if the claim of achieving 2000 distinct points in every 20 ms is correct, the number of phase values in a period of 610.304us should be 61, which indeed is the number of distinct phase values in Figure 4.20. Thus, the **achieved phase resolution of 0.18 degrees** is verified on FPGA.

5 Evaluation

This chapter evaluates the IP subsystem based on functional correctness, latency, throughput and FPGA resource utilization achieved.

5.1 Functional Verification

In the verification scheme showed earlier,individual IPs were verified and their responses to stimulus (ADC emulator samples) applied are observed. Now, for a variety test inputs are applied to check the input-output response of the IP-subsystem:

5.1.1 Tests A and B

For test A and test B, the testing scheme can be understood from Figure 5.1.

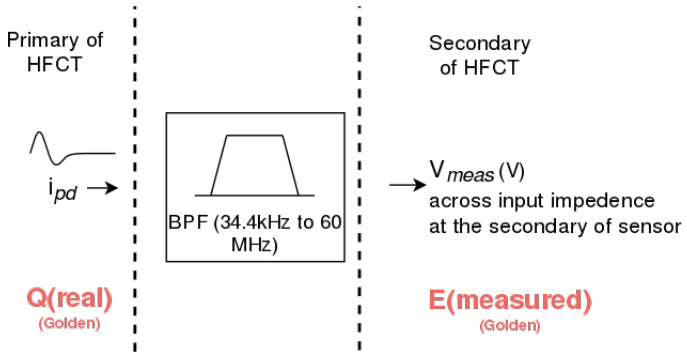

Figure 5.1: Verification scheme - testA and test B

The real charge is the charge of the PD pulse at the **primary** of the sensor. Hence, its true value (integrating current over time) is derived and is the Golden reference (shown in red [Q(real)] - Figure 5.1) for charge calculations. Further, the band-pass filter behaviour is simulated in `MATLAB` and the output voltage retrieved is fed to the IP subsystem (after quantization to simulate ADC levels). The output from IP subsystem after hardware simulations can then be verified against the golden reference. If the error is under 10% (constraints), it is a proof that the IP subsystem is working as intended. However, such an ideal reference cannot be derived for energy. This is because the notion of *real/true* energy would not make sense here as energy is measured at the secondary (energy dissipated at the input impedance). Thus, verification process is done taking E(measured) (Figure 5.1) as Golden reference for energy. E(measured) here is the energy calculated in `MATLAB` (simulation) in charge and frequency domains. Thus, instead of comparing to *true* energy, the correctness of hardware implementation with respect to `MATLAB` algorithm outputs (golden references in this case) is verified.

1. Test A: This is a stream of ADC samples (at secondary) corresponding to PD current pulses (at primary). The current pulses are shown in Figure 5.2.

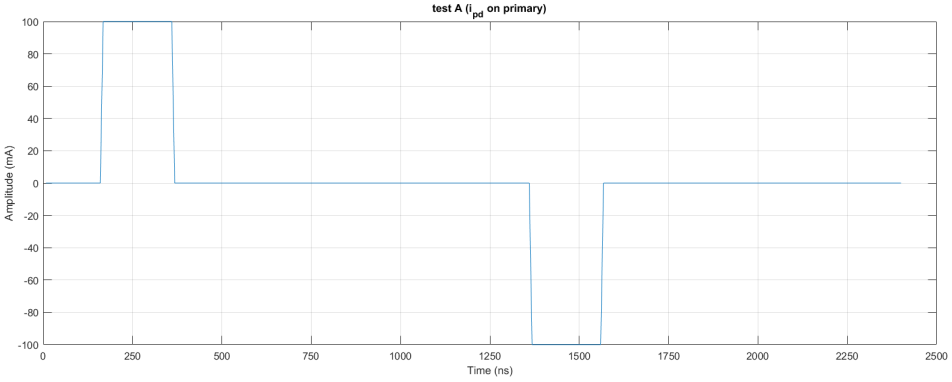

Figure 5.2: Simulated current pulse at primary of HFCT testA

As can be observed, these are wide pulses (200ns) with high amplitudes ± 100mA (around ± 910mV at secondary). These are intended to exercise and overflow the fixed point *ranges* employed in the IP subsystem.

2. Test B: This is another stream of ADC samples (at secondary) corresponding to PD current pulses (at primary). The current pulses are shown in Figure 5.3. As

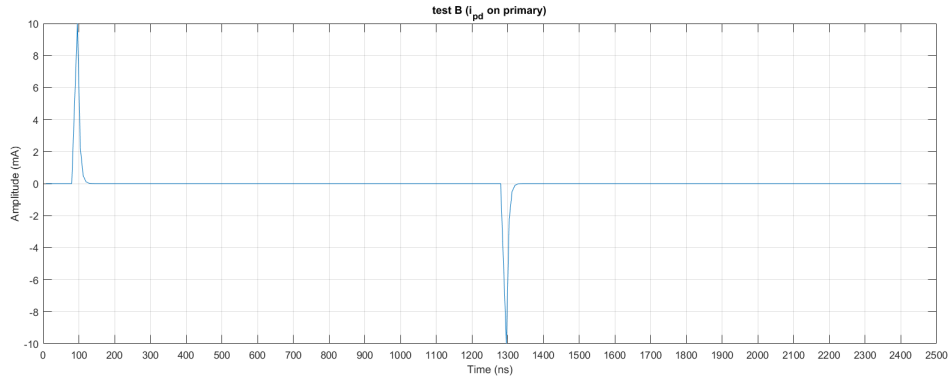

Figure 5.3: Simulated current pulse at primary of HFCT testB

can be observed, these exponential pulses are narrow with small amplitudes and are aimed at testing the precision of the IP-subsystem.

RESULTS:

Note, *recording period* employed for testing these pulses is 1us.

1. **Charge Estimations**:

PD current (peak)	Algorithm	Algorithm Outputs [MATLAB]	Hardware Outputs [Vivado]	Algorithm Error(golden) [%]	Hardware Error(golden) [%]
100mA	Frequency	18.9579	18.9285	-5.21	-5.3572
	Time	19.2193	19.1817	-3.9032	-4.0913
-100mA	Frequency	18.9579	18.9285	-5.21	-5.3572
	Time	-19.2193	-19.1864	-3.9032	-4.0678

Table 5.1: Charge(nC) estimation errors (test A) - Golden = Q_{real}(=±**20nC**) on primary

PD current (peak)	Algorithm	Algorithm Outputs [MATLAB]	Hardware Outputs [Vivado]	Algorithm Error(golden) [%]	Hardware Error(golden) [%]
10mA	Frequency	0.1434	0.1451	0.3205	1.5340
	Time	0.1398	0.1380	-2.1991	-3.4251
-10mA	Frequency	0.1434	0.1434	0.3205	0.3339
	Time	-0.1398	-0.1415	-2.1991	-1.005

Table 5.2: Charge(nC) estimation errors (test B) - Golden = Q_{real}(=±**0.143nC**) on primary

From Table 5.1 and 5.2, it is clear that hardware outputs simulated in `Vivado` (and emulated on FPGA) are well within the error constraint of ±10%. Further, because of the fact that algorithms (formulae) themselves are charge (and energy) *estimation* algorithms in frequency and time domains, they naturally differ from the real(true) value of charge. Thus, most of the deviation (from true charge) in the final hardware output can be attributed to the nature of algorithm. The error in hardware is because of simulated *quantization errors* and the *fixed point computations*.

2. **Energy Estimations**:

PD current (peak)	Algorithm	Algorithm Outputs [MATLAB] Golden Reference	Algorithm Output [Vivado]	Hardware Error (golden) [%]	ADC Error [%]
100mA	Frequency	2.9003	2.8997	-0.0206	-0.0161
	Time	3.1507	3.1510	0.0090	-0.0122
-100mA	Frequency	2.9003	2.9005	0.0092	0.0137
	Time	3.1507	3.1517	0.0307	0.0094

Table 5.3: Energy(nJ) estimation errors (test A) - Golden = E(MATLAB) (derived from voltage at secondary)

As no *true reference* is taken for energy calculations, the hardware outputs (`Vivado`) are compared against the software (`MATLAB`-Golden reference) ones to get a good estimation of deviation from intended behaviour. In Table 5.3, the maximum error observed (from `MATLAB` energy estimates) is *-0.0206%* which should to

PD current (peak)	Algorithm	Algorithm Outputs [MATLAB] Golden Reference	Algorithm Output [Vivado]	Hardware Error (golden) [%]	ADC Error [%]
10mA	Frequency	0.0017077	0.0017028	-0.2853	-0.2734
	Time	0.0017117	0.0017076	-0.2406	-0.2263
-10mA	Frequency	0.0017077	0.0017092	0.0915	0.1018
	Time	0.0017117	0.0017125	0.0448	0.0470

Table 5.4: Energy(nJ) estimation errors (test B) - Golden = E(MATLAB) (derived from voltage at secondary)

be acceptable for all practical purposes.
The analog to digital conversion errors (simulated in `MATLAB`) in estimation of energy are also shown in Tables 5.3 and 5.4. As a digitization scheme of assigning voltages to quantized levels followed by taking a *floor* was chosen while deriving ADC values for ADC emulators, an underestimation (overestimation in case of negative pulses) of features can be observed. As the pulses chosen for *test A* are fairly wide and possess high amplitudes, the quantization errors do not contribute much to energy computations. However, in the case of narrow PDs with small amplitudes, ADC errors contribute more to the estimated energy, leading to a higher error percentage in case of *test B*. Nevertheless, a *-0.28* % error in energy computations also seem practically acceptable. **Note** that this is the worst case scenario for energy calculations as both the causes of error i.e *fixed point computations* (in terms of sufficient precision) and *quantization* are tested to the extreme for this case.

5.1.2 Test C

The tests above cover estimations for recording period of 1. However, recording periods of 2 to 10 are not tested. This test sweeps *recording period* from 2 to 10 ns. The intention here is to check the functionality of algorithms specially frequency domain as it involves look-ups based on indices derived from recording period. Again, for this test, hardware outputs will be compared to software (`MATLAB`) ones (same as *test B*). The exponential pulse used in this section is shown in Figure 5.4. As can be observed, *testC* represents an elongated PD pulse. Further, white noise is added (SNR 10) to test IP subsystem to the extreme (for precision). It should be noted here that the idea behind this test set is to analyze the errors *achieved* on hardware with respect to `MATLAB` estimations and not to analyze the algorithms themselves.

Results:

Figure 5.5 and 5.6 demonstrate the errors in the results from sweeping recording period from 2 to 10 in both domains for charge and energy respectively.
As mentioned earlier, the errors for our embedded solution can be attributed to ADC quantization errors(unavoidable) and fixed point errors (true system errors). ADC

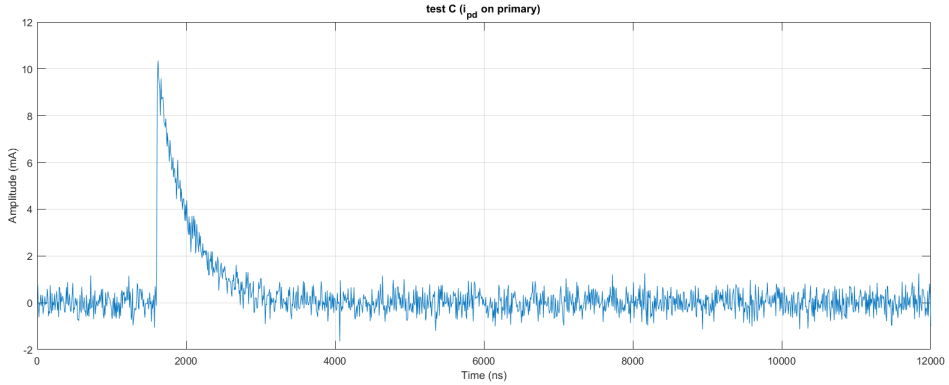

Figure 5.4: Simulated current pulse at primary of HFCT TestC

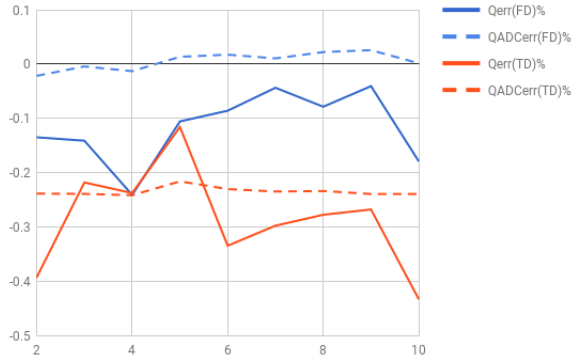

Figure 5.5: Percentage errors in charge estimation - golden reference - MATLAB charge simulations in frequency and time domain for recording periods between 2-10 us

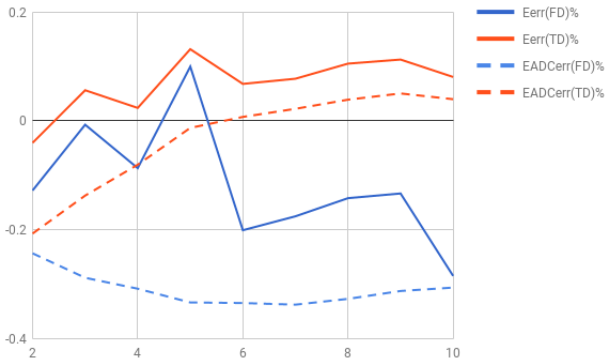

Figure 5.6: Percentage errors in energy estimation - golden reference - MATLAB energy simulations in frequency and time domain for recording periods between 2-10 us

behaviour was simulated in MATLAB as explained earlier. Thus, if there are no fixed point errors, the dotted lines (outputs when quantized inputs are considered) should be followed by bold lines (hardware outputs) in Figures 5.5 and 5.6. However, as can

be observed, there are differences between dotted and bold lines due to fixed point computation errors. First property that can be noticed is the tendency of the errors to shift in the negative direction. This negative shift is due to the ADC simulation scheme of taking the *floor* (assumption). However, there is still a *uniformity* as as all the samples (for ADC emulators) employ the same scheme.
Charge estimations in time domain demonstrate ADC errors perfectly (Figure 5.5). The exponential pulse in discussion is a positive pulse and charge calculations employ summing up input samples between zero crossings (corresponding to maximum peak). Thus, if each sample is digitized (and *floored* in our case), underestimations of charge are expected. In this sense, charge estimation in frequency domain appears to be more resilient to quantization errors. Adding up to these ADC errors in time domain for charge calculations, the fixed point errors for charge calculation further deviate the feature (charge), leading to a maximum negative error of 0.43% (recording period 10 in this case).
From Figure 5.6, it can be observed that maximum errors in energy estimations is under -0.3 % (0.28%). Also, in this case, system errors(fixed point) are less for Time domain as compared to frequency domain. This again makes sense because in frequency domain, the incoming sample (Fixed Point sample) gets multiplied to itself and then gets multiplied with sin and cosines (fixed points) followed by square root in the end. On the other hand, the time domain energy calculations only involve sample multiplication by itself, followed by successive additions.
As discussed in section 2.1, our measuring system is an *unconventional* one. Thus there are no guidelines for an error constraint as such. Much is left to actual testing in the lab to see if the extracted features are sufficient to separate PD sources or not. Making any further deductions from the outputs of this limited data-set will not be fair. However, what can still be appreciated is the fact that that even for this worst case test input featuring small and rapidly varying changes, the fixed points in employed in hardware seem to provide sufficient precision as the maximum errors achieved are on hardware **-0.28%** and **-0.3%**.

5.2 Latency and throughput

From the performance details in appendix A, Figure ref 5.7 is derived. Here, the similarities (in handling streams of PD samples) of the achieved IP-system compared to the one desired (2.21) can be appreciated.
The system offers a worst case *latency* of **66 cycles** (66*8 = 528 ns) per input PD sample (when frequency domain algorithm is opted). Further, in the best case, the latency achieved is **8 cycles** (8*8=64 ns). Also, each IP in the subsystem offers an Initiation Interval (II) of 1 which directly implies that each IP can accept a new input PD sample every cycle (8ns). This further implies that the entire IP sub-system can *accept new PD samples every cycle*, giving the system the desired *throughput* (2.10) for *real time processing*.
In Figure 2.21, it was shown that total latency of the compute block to handle x input samples would be $x + N$ (when II=1) where N is the latency of compute block.
From Figures 5.8 and 5.9 the response of IPs to stimulus (*testA*) can be observed

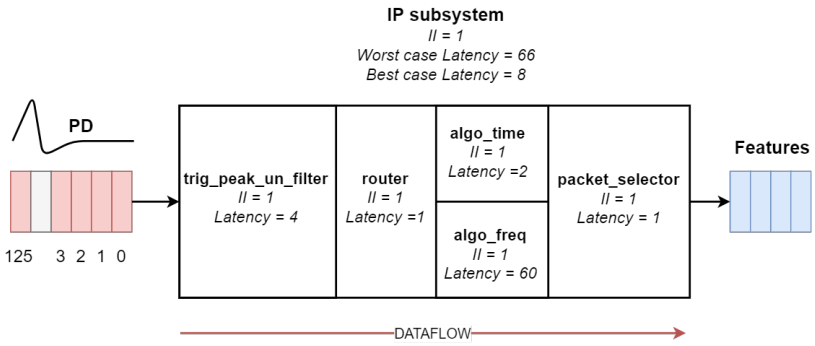

Figure 5.7: Achieved latency and throughput for IP sub-system

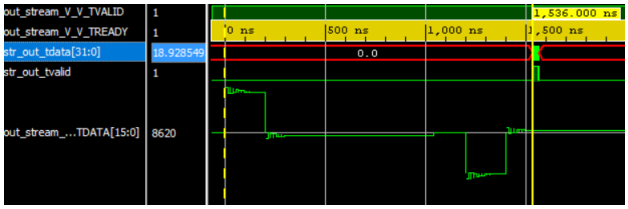

Figure 5.8: Response time - Frequency domain

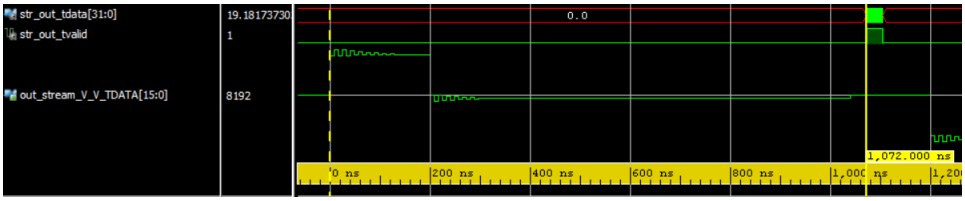

Figure 5.9: Response time - Time domain

(recording period = 1us [125 samples]). As we accept one additional sample in the system for each recording period (extra sample in history 4.3.2.2), x = 126. Further, in worse case (frequency domain), N = 66 implying a total latency of 192 cycles (x+N) i.e, 1536 ns (Figure 5.8). Similarly, for time domain, we get a best case latency of 134 cycles (126 + 8) i.e, 1072ns (Figure 5.9).

5.3 Utilization

The utilization estimates can be found at appendix A. Clearly, the frequency domain algorithms are more resource hungry than the time domain ones. Moreover, the overall utilization has not exploded (thanks to fixed points) and there is a lot of FPGA area (86.67% BRAM, 58.75% DSPs, 77.17% LUTs(Logic) and 50.55% still available) for more utilities to be put in future.

6 Conclusions and Future work

Before concluding the report, it is important to discuss the limitations in the achieved system. Following section summarize the limitations.

6.1 Limitations

1. **ADC :** As discussed earlier, the PD pulse voltages can extend well beyond the available range of $\pm 1V$.
2. **Sampling rate :** The sampling rate PD detection at every 8 ns. However, there are PD pulses with rise times smaller than 8ns. Thus, there is a probability that I_{pd} feature is under-estimated.
3. **Manual Trigger levels:** However, the system features a manual triggering. This implies human intervention. This implies triggering errors, mainly false triggers due to noise. Thus, it is extremely desirable for the final solution to be free of human interventions.
    - The *real time* acquisition and compute nature of our system can prove to be an excellent solution for the problem of false triggering due to noise. For instance, if 100 valid frames (each frame 1us) are *consecutively* (100us) recorded and 100 valid features are computed for these frames, there is a extremely high probability that noise is being triggered. Hence, a feedback routine to fix trigger values can be devised.
4. **Manual Recording period:** This is another source of human intervention. PD widths can extend beyond 1 us. However, in the present solution, there is no way to know if the acquired *frame* covered the length of PD or not. If it did not, the computed parameters charge and energy might not represent the PD.
5. **Basic application:** For this thesis, only a basic standalone application is developed as a proof of concept. However, for realizing the picture shown in Figure 2.7, it is essential to have communications to remote user and a Graphical User Interface to be developed.

6.2 Future work

The system has to be integrated to the ADC and tested with the real test setup to realize practical merits and demerits. Also, there should be an integration with remote GUI to facilitate testing. A complete software framework has to be developed to facilitate these interactions with remote user. Once rigorous testing is completed, additional

classification IPs can be designed for the FPGA to perform the entire process of PD detection and classification locally.

6.3 Conclusions

The achieved IP subsystem is an excellent starting point to facilitate further research using the test setup shown in Figure 4.5. All the goals defined for the thesis work were achieved. With the comprehensive testing performed in chapter 5 (and chapter 4), it is established that the features of interest I_{peak}, Q , E and phase are computed with high degree of accuracy, including errors due to fixed point computations.
It can be appreciated that as our `Red-Pitaya` based embedded system is capable of feature computations also, data compression (only 4 outputs corresponding to each PD pulse) is achieved as shown in table 6.1. Hence, 78.4% less amount of data has to be

	size per element	Per frame storage (10us)	Number of Frames	Required storage
Oscilloscope	1 byte(per sample)	1250 bytes	50000	59.6MB
Red Pitaya	4 bytes(per feature)	4*4=16bytes	50000	0.76MB

Table 6.1: Data compression(ratio-78.4%) facilitating classification process

analyzed for classification process as compared to the oscilloscope, which is an *in-built novelty factor of this thesis.*
The IP subsystem is based on a cost-effective platform (`Red Pitaya`), which will eventually make the benefits to cost ratio of deploying `Red Pitaya` on a large scale much higher than the state of the art. Further, the IP-subsystem will be open-source and made available on TU Delft's website, implying that instead of using expensive equipment like a high performance oscilloscope, PD researchers around the world can simply employ a cost-effective `Red Pitaya`, download our source code onto it and just like that, a ready to use PD detection and feature extraction tool based on more robust PD *features* is available. This will be an important step in the direction of promoting research in the field of PD detection and monitoring based on electrical methods.
The fixed point computation facilitated the desirable *real time* property to the IP-subsystem. Due to the real time acquisition and compute nature of devised solution, a number of avenues for a better future classification process have opened. For instance, repetition rates can now be derived with high accuracy, which implies better defect-severity indications to the user and can also be potentially employed as an additional *feature* for classification. Due to fixed point computations, the utilization and power consumption of the FPGA was (inherently) minimized. Even after realizing two algorithms, there is still plenty of area remaining in the fabric (`ZYNQ 7010 SoC`), which can be utilized for realizing (part of) PD classification in future. The designed solution facilitates user with selections of feature estimation algorithms based on time/frequency domains. Hence, it opens more possibilities of research and testing as both estimation methods have their merits and demerits.
Because an embedded platform with high compute capabilities is employed, a potential patient-doctor relationship can exist between the HV/MV insulation and user. Thus,

if the classification is done locally on the platform, a network of these cost-effective embedded devices can send their *health reports* to the user from time to time.
To conclude, I would like to emphasize that although the project is still in the very first stage i.e feature extraction and requires rigorous testing in practical scenarios, with the cost effective, real time solution that has been developed, *possibilities are limitless.*

Bibliography

[1] UG902 (v2017.1). Vivado design suite user guide - high-level synthesis, April 2017.

[2] Anton Potonik. Red pitaya fpga project 4. http://antonpotocnik.com/?p=519284.

[3] IEC 60270-2000. High-voltage test techniques - partial discharge measurements.

[4] Abu Bakar AH Jee Keen Raymond W, Illias HA. Classification of partial discharge measured under different levels of noise contamination plos one12(1): e0170111, 2017. https://doi.org/10.1371/journal.pone.0170111.

[5] A. Cavallini G. Montanari and F. Puletti. A new approach to partial discharge testing of hv cable systems, ieee electr. insul. mag., vol. 22, pp. 1423, 2006.

[6] AMor AR, Castro Heredia LC, Harmsen DA, and F.A.Muoz. A new design of a test platform for testing multiple partial discharge sources. *International Journal of Electrical Power Energy Systems*, 94:374–384, 2018.

[7] Rotating electrical machines-part 27off-line partial discharge measurements on the stator winding insulation of rotating electrical machines, iec standard 60034-27, 2006.

[8] C. Hudon and M. Belec. Partial discharge signal interpretation for generator diagnostics. *IEEE Transactions on Dielectrics and Electrical Insulation*, 12(2):297–319, April 2005.

[9] M. Wu, H. Cao, J. Cao, H. L. Nguyen, J. B. Gomes, and S. P. Krishnaswamy. An overview of state-of-the-art partial discharge analysis techniques for condition monitoring. *IEEE Electrical Insulation Magazine*, 31(6):22–35, November 2015.

[10] M Ghaffarian Niasar. Partial discharge signatures of defects in insulation systems consisting of oil and oil-impregnated paper. *(Licentiate dissertation). KTH, Stockholm*, 2012.

[11] Jrgen Fabian, Martin Neuwersch, Christof Sumereder, Hans Michael Muhr, and Robert Schwarz. State of the art and future trends of unconventional pd-measurement at power transformers. *Journal of energy power engineering*, (8):1093–1098, 2014.

[12] S. Meijer, P. D. Agoris, P. P. Seitz, and T. J. W. H. Hermans. Condition assessment of power cable accessories using advanced vhf/uhf pd detection. In *Conference Record of the 2006 IEEE International Symposium on Electrical Insulation*, pages 482–485, June 2006.

[13] A. R. Mor, L. C. C. Heredia, and F. A. Muoz. Estimation of charge, energy and polarity of noisy partial discharge pulses. *IEEE Transactions on Dielectrics and Electrical Insulation*, 24(4):2511–2521, 2017.

[14] A. Cavallini, A. Contin, G. C. Montanari, and F. Puletti. Advanced pd inference in on-field measurements. i. noise rejection. *IEEE Transactions on Dielectrics and Electrical Insulation*, 10(2):216–224, April 2003.

[15] Ravish Preshant Yashraj Mehairjan. Risk based maintenance in electricity network organisations, 2016.

[16] Ravish Preshant Yashraj Mehairjan. Application of statistical life data analysis for cable joints in mv distribution networks, 2010.

[17] Frank Wester. Condition assessment of power cables using partial discharge diagnosis at damped ac voltages.rotterdam, the netherlands. December 2004.

[18] X. Zhang, E. Gockenbach, V. Wasserberg, and H. Borsi. Estimation of the lifetime of the electrical components in distribution networks. *IEEE Transactions on Power Delivery*, 22(1):515–522, Jan 2007.

[19] A. Rodrigo Mor, L. C. Castro Heredia, and F. A. Muoz. Effect of acquisition parameters on equivalent time and equivalent bandwidth algorithms for partial discharge clustering. *International Journal of Electrical Power Energy Systems*, 88:141–149, 2017.

[20] A. R. Mor, P. H. F. Morshuis, and J. J. Smit. Comparison of charge estimation methods in partial discharge cable measurements. *IEEE Transactions on Dielectrics and Electrical Insulation*, 22(2):657–664, April 2015.

[21] I. Shim, J. J. Soraghan, and W. H. Siew. Digital signal processing applied to the detection of partial discharge: an overview. *IEEE Electrical Insulation Magazine*, 16(3):6–12, May 2000.

[22] A. R. Mor, L. C. Castro Heredia, and F. A. Muoz. New clustering techniques based on current peak value, charge and energy calculations for separation of partial discharge sources. *IEEE Transactions on Dielectrics and Electrical Insulation*, 24(1):340–348, Feb 2017.

[23] G. C. Stone. Partial discharge. vii. practical techniques for measuring pd in operating equipment. *IEEE Electrical Insulation Magazine*, 7(4):9–19, July 1991.

[24] Tektronix dpo7354c pricing. https://www.tek.com/oscilloscope/dpo7000-digital-phosphor-oscilloscope.

[25] Zynq-7000 All Programmable SoC. Technical reference manual measurements, September 27 2016.

[26] Zybo Zynq-7000 ARM/FPGA SoC. http://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/.

[27] India Central electricity regulatory commission, New Delhi. http://www.cercind.gov.in/2017/draft_reg/GC-copy/Power%20System%20Operation%20Corporation%20Limited%20%28POSOCO%29.pdf.

Appendix

A.1 Performance

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
60	60	1	1	function

Figure A.1: Performance - `trig_peak_un_filter`

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1	1	1	1	function

Figure A.2: Performance - `router`

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
60	60	1	1	function

Figure A.3: Performance - `algo_freq`

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2	2	1	1	function

Figure A.4: Performance - `algo_time`

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1	1	1	1	function

Figure A.5: Performance - `packet_selector`

A.2 Utilization

Name	Block RAM Tile (60)	DSPs (80)	LUT as Logic (17600)	LUT as Memory ...	LUT Flip Flop Pair...	Slice (4400)
test_PD_wrapper	15.83%	41.25%	23.16%	2.55%	11.28%	50.23%
test_PD_i (test_PD)	15.83%	41.25%	23.16%	2.55%	11.28%	50.23%
make_PD_wrapper_0 (test_PD_m...	13.33%	41.25%	22.87%	2.55%	11.16%	49.45%
U0 (make_PD_wrapper)	13.33%	41.25%	22.87%	2.55%	11.16%	49.45%
make_PD_i (make_PD)	13.33%	41.25%	22.87%	2.55%	11.16%	49.45%
algo_freq0_0 (make_PD...	13.33%	33.75%	12.99%	2.45%	8.74%	25.20%
algo_time1_0 (make_PD...	0.00%	1.25%	2.09%	0.00%	0.68%	2.98%
axis_register_slice_0 (m...	0.00%	0.00%	0.13%	0.00%	0.05%	0.57%
axis_register_slice_1 (m...	0.00%	0.00%	0.13%	0.00%	0.03%	0.43%
packet_selecter_0 (mak...	0.00%	0.00%	0.57%	0.00%	0.22%	1.11%
phase_detector_0 (mak...	0.00%	0.00%	1.04%	0.03%	0.44%	2.57%
router_0 (make_PD_rou...	0.00%	0.00%	0.10%	0.00%	0.05%	0.36%
sys_init_0 (make_PD_sy...	0.00%	0.00%	0.35%	0.00%	0.12%	0.75%
trig_peak_un_filter_0 (...	0.00%	6.25%	5.46%	0.07%	0.48%	17.25%

Figure A.6: IP-subsystem's post-implementation utilization