S-Transform-Based Fault Detection Algorithm for Distance Protection on FPGA

Quanhao Yu

S-Transform-Based Fault Detection Algorithm for Distance Protection on FPGA

by

Quanhao Yu

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 24, 2022 at 10:00 AM.

Student number:	5361397	
Project duration:	December 4, 2021 – August 24, 2022	
Thesis committee:	Prof. dr. ir. S. Wong,	TU Delft, supervisor
	Prof. dr. ir. M. Popov,	TU Delft, supervisor
	Prof . ir. A. Lekic,	TU Delft

Abstract

With the replacements of coal and nuclear plants with renewable energy sources such as Type-3 and Type-4 wind turbines and photovoltaic system, the fault current levels of the electrical power system become lower. This leads to a weakened ability of the commercial distance relays to detect fault currents. Therefore, new fault detection algorithms have been developed based on different approaches with the intention of raising the ability to detect faults under different circumstances. Among them, Stockwell Transform has been proved to be a useful tool to determine the occurrence of faults. This thesis presents an FPGA implementation of one of the newly developed S-Transform-based fault detection system on Zynq device. The conceptual methods of building an IEC communication interface to connect the system to an RTDS are investigated. The algorithm on hardware is implemented in fixed-point representation, fully parameterized, and fully pipelined with the operation frequency at 100MHz. The final implementation is able to provide the result within 570ns. The hardware implementation of the S-Transform-based fault detection system is tested with simulations proving it can perform identically with the MATLAB model of itself.

Contents

1 Introduction

It can be seen that renewable energy sources (RES) will take over the role of coal and nuclear power plants in the future, which is known for reduced fault current levels resulting from low system inertia [1]. The operation, control, and protection of such energy sources are complicated due to bi-directional fault currents and different composition of energy production. Distance protection for conventional power grids will also suffer in terms of performance. Typical examples of RESs are photovoltaic systems, Type-3 and Type-4 wind turbines which are interconnected to the power grid via partial or full-scale power electronics voltage source converters. The differences between the fault current contribution of the synchronous generator and the power electronics-based generator are fault duration, magnitude and the shape of voltage and current waveforms [1].

With different behaviours of fault current contribution, commercial distance relays that are currently available face difficulties in fault detection. In [1], it has been observed that the presently available commercial distance relays fail to detect phase-to-phase and three-phase faults in more than 50% of the cases. This leads to the necessity of the introduction of new fault detection algorithms. A new Stockwell Transform-based fault detection algorithm for distance protection is proposed in [1], in which the energy of the Stockwell Transform (S-energy) is the critical factor in determine the fault currents. The proposed fault detection algorithm presents accurate results in 99.88% of analysed cases [1].

1.1. Goals of This Work

This thesis focuses on how to efficiently implement of the S-energy calculation algorithm in the proposed fault detection algorithm on hardware and explore the possibilities of the IEC 61850 communication interface on hardware. Based on the S-Transform-based fault detection algorithm proposed in [1], the goals of this work are defined as:

1. Evaluate the possibilities of implementing the proposed algorithm on hardware
2. Propose a hardware implementation for the S-energy calculation algorithm used in the proposed S-Transform-based fault detection method
3. Implement the IEC 61850 communication interface on hardware and establish a data path between it and the S-energy calculation algorithm

The first two goals has higher priorities than the last on for this work. Therefore, this thesis mainly focuses on the feasibility and the implementation for the S-transform-based fault detection algorithm on hardware.

1.2. Methodology

To meet the goal as described in Section 1.1, first, the evaluation of the possibilities of implementing the algorithm on hardware is needed. Since the S-energy calculation algorithm is implemented in floating-point numbers in MATLAB, it is necessary to validate whether the algorithm is able to be converted into the fixed-point representation while maintaining acceptable accuracy. Second, it is essential to establish the design requirements and choose the hardware platform. The design requirements will be used to check if the design provides the expected results in time, and the hardware will be implemented on the chosen platform. Third, hardware implementation of the S-energy calculation algorithm and the IEC 61850 communication interface needs to be designed on the selected hardware platform. Forth, to justify the functionality of the hardware implementation of the S-energy algorithm, the output results of the hardware design need to be compared with the results from the MATLAB simulation.

1.3. Structure of Thesis

The rest of this thesis is organized as follows. Chapter 2 introduces the background of the work in this thesis and the related work that is done previously. The distance relays, the proposed S-energy calculation algorithm, S-Transform-based fault detection system, and the more information about IEC 61850 Sampled Values and GOOSE Messages are introduced in this chapter. Chapter 3 describes the high-level design choices, converting from floating-point to fixed-point representations, and system overview for implementing the proposed S-Transform-based fault detection system on hardware. In Chapter 4, the hardware implementation of S-Transform based fault detection system is presented in details. The conceptual investigation of implementing an IEC 61850 communication interface is also included in Chapter 4. Chapter 5 shows the experimental results for validating the hardware implementation of the proposed S-energy calculation algorithm are presented and discussed. Finally, the conclusion and possible future work are summarized in Chapter 5.

2 Background

In this Chapter, the background that is needed for this work is introduced. The previous work on the proposal of the enhanced distance relay module comprising the new Stockwell Transform-based fault detection algorithm is also briefly presented. The mechanism and the advantages of using the Fast Discrete Stockwell Transform are investigated and explained. The IEC 61850 communication protocol, which is widely used in the field of the electrical power system, is also introduced. The tools used in this project are presented as background information on the hardware platform.

2.1. Distance Relays

A distance relay is a device that detects the fault in the transmission line and initiates the activities after detecting a fault. The term "distance" indicates that the functionality of a distance relay is based on the distance between the fault point to the feed point on the transmission line. A distance relay compares voltage and current signals to provide impedance-plane and directional characteristics [2].

The motivation for introducing distance relays is the drawbacks of overcurrent and overvoltage relays. Since the overcurrent and overvoltage relays are designed to have only one pick-up value (the value where a relay reaches operated condition), it is impossible for them to provide protection to the same point from different types of faults.

Distance relays, however, usually implement algorithms based on the phasors at the fundamental frequency [3]. Phasors can be calculated in several ways: Fourier Transform-based algorithm [4, 5, 6], least error squares based methods [7, 8, 9, 10], Wavelet Transform-based algorithm [11, 12, 13, 14], etc. These methods are known as Phasor-domain-based relaying algorithms [3].

Another method for relays to evaluate the situations is to operate based on the time-domain instantaneous current and voltage values. Algorithms developed in this method are also known as Time-domain based relaying algorithms [3].

2.2. Related Work

The commercial distance relays work well in Traditional power grids. However, it is the trend in the electrical power and energy field that a large amount of coal and nuclear power plants will be replaced by renewable energy sources such as photovoltaic systems, Type-3 and Type-4 wind turbines[1]. Compared with the classical synchronous generators used for coal and nuclear power plants, renewable energy sources are generally connected to the power grid by full-scale or partial-scale power electronics voltage source converters. This contributes to the fast switching ability for renewable energy sources. However, this has an impact on the fault current level which results in a lower ability to detect the fault in the transmission line of renewable energy sources. The commercial distance relays also are affected by this impact, and therefore, new algorithms for fault detection are needed for renewable energy sources.

Several scenarios were studied in [1] to observe the cases where commercial relays are in trouble during fault detection. The results show in more than 50% of the studied cases, the commercial relays have difficulties in detecting phase-to-phase and three-phase faults. A new fault detection algorithm based on Fast Discrete Stockwell Transform (FDST) was therefore proposed in [1]. The testing for the proposed enhanced relay module comprising the new fault detection algorithm is illustrated in Figure 2.1 [1]. In Figure 2.1, four

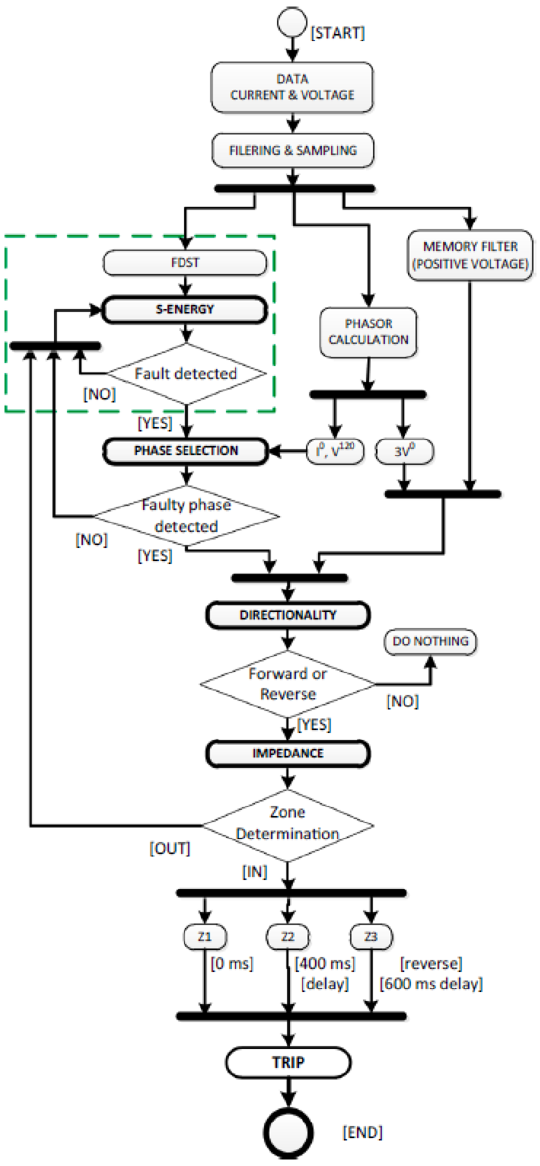


Figure 2.1: Testing for the enhanced distance relay module [1]

modules can be found to provide essential functionalities to the enhanced distance relay module, which are S-energy (FDST energy) module, phase selection module [15, 16], directionality module [17], and impedance module [18, 19]. The S-energy of each phase was chosen to be the indicator for detecting the faults since it changes quickly after the fault occurs. The advantages compared to other time-frequency domain analysing algorithms are introduced in Section 2.2.2. The term "Fast" shows it is a fast algorithm to perform Discrete Stockwell Transform. The term "Discrete" shows the transform is applied on discrete series, which are commonly used in digital signal processing and hardware implementation.

The analysis was carried out with this new S-transformed-based fault detection algorithm integrated into the RTDS in order to compare with the commercial relays. The results showed that the proposed fault detection algorithm outperforms different commercial relays and presents accurate results in 99.88% of analysed cases [1].

The work presented in [1] focuses on the S-energy module circled in green dot lines in Figure 2.1. Based on this S-energy module, this work proposes a hardware implementation for the FDST to calculate the Stockwell Energy. Therefore, it is essential to introduce the FDST and the S-energy calculation algorithm in [1].

2.2.1. Fast Discrete Stockwell Transform (FDST)

The FDST of a discrete time series $x[kT], k = 1...N$ can be expressed as equation 2.1:

$$S_{(jT,n/nT)} = \sum_{m=1}^{N} [\mathbf{H}^{\circ}\mathbf{W}]\, e^{i2\pi nj/N} \tag{2.1}$$

where:

- $n = 1...N$ represents the time point indices of a given cycle [20].
- $m = 1...N/2$ represents the frequency point indices of a given cycle [20].
- **H** is a two-dimensional matrix after rotating and concatenating the results from $FFT(x[kT])$.
- **W** is a modified two-dimensional Gaussian window.

The Gaussian window **W** is defined as equation 2.2, which is used to filter out information from unwanted frequencies [21]:

$$\mathbf{W}_{(m,n)} = e^{T1} + e^{T2} \tag{2.2}$$

where:

$$T_1 = -\frac{2\pi^2(n-1)^2F}{(a+bm^c)^2}, T_2 = -\frac{2\pi^2(N-n+1)^2F}{(a+bm^c)^2} \tag{2.3}$$

In equation 2.3, F is the window factor, b is the scaling factor controlling the number of oscillations in the window, a and c are the positive constants. The two-dimensional matrix **H** is the resulting matrix from the FFT and has the size of $M \times N$, and forms as follows, which is the concatenated and rotated:

$$\mathbf{H}_{M\times N} = \begin{bmatrix} x_2 & x_3 & \dots & x_N & x_1 \\ x_3 & x_4 & \dots & x_1 & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{M+1} & x_{M+2} & \dots & x_{M-1} & x_M \end{bmatrix} \tag{2.4}$$

The windowed time-frequency information is achieved by applying the Gaussian window in equation 2.2 to the concatenated and rotated matrix from FFT in equation 2.4. The matrix **H** is multiplied by the elements in the matrix **W**. And the resulting matrix of that operation **G** is the Hadamard product of **H** and **W**, described as follows:

$$\mathbf{G} = \mathbf{H} \circ \mathbf{W} \tag{2.5}$$

Therefore, the resulting matrix of the FDST **S** can be derived by the Inverse Fast Fourier Transform (IFFT) of the matrix **G**, described as:

$$\mathbf{S}_{(jT,n/nT)} = IFFT[\mathbf{G}] \tag{2.6}$$

As it can be seen, FFT plays a vital role in FDST. The methodology of FFT will be introduced in the next chapter, together with the hardware implementation.

2.2.2. Advantages of using S-transform

The application for determining time-local spectra is widely applied in analysing non-stationary time series, such as electrodiagrams [22], seismic signals [23], etc. There are some tools available for analysing such signals, which can be used in power frequency measurement for generator protection and protective relays [24] , signal filtering [25], the determination of power quality [26] , and fault detection and diagnosis of gird-connected power inverters [27]. Among these analysing tools, Short-Time Fourier Transform (STFT), Gabor Transform, Wigner Transform, and Stockwell Transform are the most popular ones.

Compared to STFT, the main difference is that the window width is constant in STFT, whereas it is a function of frequency in S-transform. This leads to the main advantage of S-transform over STFT, which is better performance in detecting and resolving low-frequency signals, as well as a better time resolution of high-frequency signals [28]. In [25, 29], it can be observed that the frequency-dependent resolution characteristic of S-transform makes it able to detect high-frequency bursts while STFT fails to detect. When there are more than two high-frequency bursts added, S-transform is still able to detect all of them. However, STFT can only detect some of them, missing two high-frequency bursts.

Compared to Gabor Transform, the main difference is the size of the Gaussian window is different in these two transforms. The window size of the Gabor Transform is a Gaussian function, whereas the window size of the S-transform is a function of frequency. This results in S-transform having better frequency-domain clarity in lower frequency and better time-domain clarity in higher frequency [28].

Compared to Wigner Transform, the main difference is that Wigner Transform contains an auto-correlation function, which causes cross terms that might introduce noise and distortion. On the other hand, S-transform avoids this noise and distortion by avoiding cross-term. This makes S-transform a better tool in filtering and modulation [28].

In short, the advantages of S-transform over other time-frequency analysing tools make it a better choice in the application of many fields other than fault detection and distance protection in power girds, such as geophysical signal analysis [29], EEG (Electroencephalogram) signal analysis [30], Optical 3D surface measurement [31], and Magnetic Resonance Imaging (MRI) [32], etc. Due to the advantages it brings by operating on a moving and scalable Gaussian window, S-transform provides a good frequency-dependent resolution while containing the information from both the time domain and frequency domain.

2.2.3. S-energy and S-energy threshold

After the calculation of the FDST, the S-energy can be calculated by:

$$S_E(t) = \sum_{m=1}^{M} \sum_{n=1}^{N} \left| S^2_{(m,n)} \right| \tag{2.7}$$

After the S-energy is calculated, a threshold of it is needed to be set in order to perform the fault detection function. To come to an S-energy threshold value, linear interpolation using the RMS values of the current signals and the corresponding S-energy values is carried out in [1]. The RMS values of the load current and their corresponding S-energy are shown in Table 2.1.

I_{RMS}(kA)	S_{TH}(dB)	S_{THsm}(dB)
0.027	-43.130	-40.523
0.062	-30.920	-28.313
0.100	-23.250	-20.643
0.137	-18.001	-15.394
0.171	-14.530	-11.923
0.211	-11.270	-8.663
0.245	-8.500	-5.893
0.285	-6.609	-3.462
0.329	-3.829	-1.222
0.366	-2.230	0.376
0.400	0.168	2.774

Table 2.1: RMS of load currents and S-energy [1]

In Table 2.1, I_{RMS}(kA) is the RMS value of the load currents, S_{TH} is the threshold S-energy value in dB, and S_{THsm} is the threshold S-energy value after applying a safety margin. After fitting the threshold with a safety margin, an empirical equation for calculating the S-energy threshold value is obtained from Table 2.1, which is defined as follows:

$$S_{THsm} = -2.6 \times 10^3 I_{rms}^4 + 3.5 \times 10^3 I_{rms}^3 - 1.7 \times 10^3 I_{rms}^2 + 4.2 \times 10^2 I_{rms} - 45 \tag{2.8}$$

The result of the FDST is a two-dimensional array containing information from both time and frequency domains. And the magnitude of the calculated S-energy changes in both time and frequency. This feature makes the FDST more sensitive to the changes of current signals and thus makes it have more potential for fault detection.

Comparison of S-Transform and the existing techniques like wavelet and STFT (cite here)

Therefore, the FDST algorithm defined in equation 2.1 and the fault detection can be achieved by the following steps:

- Apply the Fast Fourier Transform to the current signal.

- Concatenate and rotate the results of the FFT to get the matrix **H**.
- Apply the two-dimensional Gaussian window **W** to matrix **H** and get matrix **G**.
- Apply the Inverse Fast Fourier Transform to matrix **G** and get the resulting matrix **S** of FDST.
- Calculate the S-energy as defined in Equation 2.7 and compare the outcome with the threshold value derived from Table 2.1 or calculated by Equation 2.8.

2.3. IEC 61850 Standard

IEC 61850 standard defines the exchange of information and configuration in substations for the purpose of monitoring, protection, etc. With the application of IEC 61850 standard, the information in substations can be transferred digitally. In this section, an overview of the history of the IEC 61850 standard is introduced. The data formats of IEC 61850 Sampled Values and IEC 61850 GOOSE Messages which are related to this project are presented as a brief background of the communication interface.

2.3.1. Overview of IEC 61850

The motivation for introducing the IEC 61850 standard is the development of digital signal processing, which makes it possible for one Intelligent Electronic Device (IED) to produce thousands of data points. It is also essential for the IEDs to be able to communicate with substations in time for the real-time operations of the electrical power system. These bring new requirements for the communication protocol for the IEDs. Some of the requirements are [33]:

- Contain the information of both data and communication service
- High-speed and real-time communication
- Supports sampled voltage and current data
- Standard-based communications
- Security support
- General-purpose communication between devices from different vendors

Based on these requirements, the Utility Communication Architecture (UCA) was firstly developed with the definitions of protocols, data models, and abstract services, etc [33]. This work was thereafter further developed into the IEC 61850-Communication Networks and Systems in Substations by the IEC Technical Committee.

Since the IEC 61850 standard is designed specifically for communication in electrical power system applications, it fits uniquely with features such as object-oriented modelling, virtualized module, standardized object names and configuration language, etc. These features result in some major benefits of using IEC 61850 standard, which are [33]:

- Lowering the installation, transducer, commissioning, integration and migration costs
- Lowering the difficulties for design, configuration, setup, and maintenance
- Functional supports for substation communication

2.3.2. Sampled Values

The IEC 61850 SV messages are mapped to ISO 8802-3 Ethernet frame with a header and the APDU (Application Protocol Data Unit). A typical Ethernet frame is illustrated in Figure 2.2.

In an IEC 61850 SV message formatted as an Ethernet frame as illustrated in Figure 2.2:

- Destination address is within the range from `01:0c:cd:04:00:00` to `01:0c:cd:04:01:ff`. MAC (Media Access Controller) filtering and multicast are in use during the transmission of SV messages. The higher three bytes are assigned by IEEE and the fourth byte `04` is reserved for the use of SV messages.
- Source address contains the MAC address of the device which is sending the SV messages.

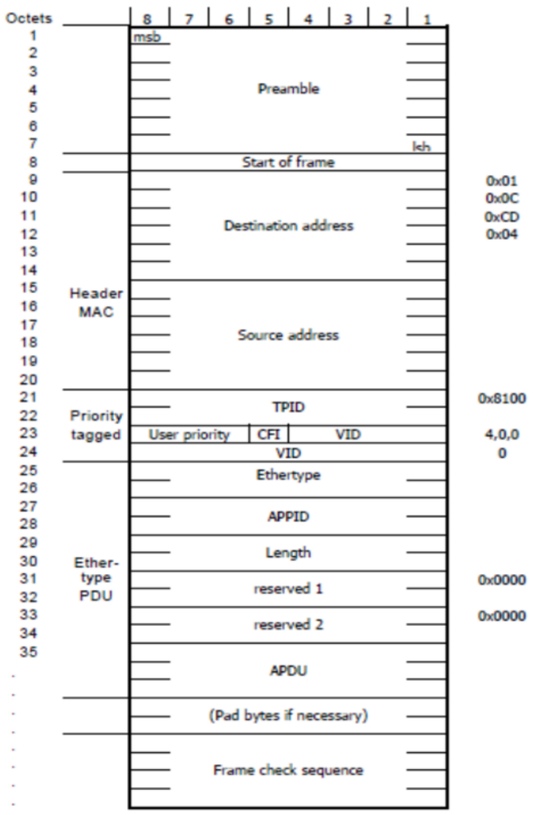


Figure 2.2: Typical format of an Ethernet frame

- ’Priority tagged’ is the VLAN tag in 802.1Q, where the TPID (Tag Protocol Identifier) is `0x8100`, the user priority is by default 4, and the VID (VLAN ID) is by default 0.
- The Ethertype reserved for SV messages is `0x88ba`.
- The APPID (Application ID) for SV messages ranges between `0x4000` and `0x7fff` .
- The APDU (Application Protocol Data Unit) contains the actual data that are transferred by SV messages.

The APDU of IEC 61850 SV messages are further defined as illustrated in Figure 2.3. The first thing which can be observed from Figure 2.3 is that multiple ASDUs (Application Service Data Units) can be included in one APDU. The measured values are contained inside each ASDU and are encoded in ASN.1 (Abstract Syntax Notation One) in `Data Set`, which is also defined in the IEC 61850 standard as Figure 2.4.

The current values and voltage values of three phases are included in the dataset. For example, in the first data `InnATCTR1.Amp.instMag.i`:

- `InnA` means it is a value of the current of Phase A
- `TCTR1` means it is from logic node TCTR1
- `Amp` means the unit of this value is Ampere
- `instMag.i` means it is an instant magnitude of the current

The data ending with a `q` is the quality of the measured values from each logic node. As can be observed from Figure 2.4, each dataset of the ASDU can be divided into eight instant measured values (currents in three phases and neutral, voltages in three phases and neutral), each taking four bytes and their corresponding qualities.

The measured values of the currents and voltages are transmitted in an integer type(32 bits) with a scaling factor (0.001 for currents and 0.01 for voltages) applied to the original value. Therefore, in order to restore the value when receiving the SV messages, this value needs to be re-scaled.

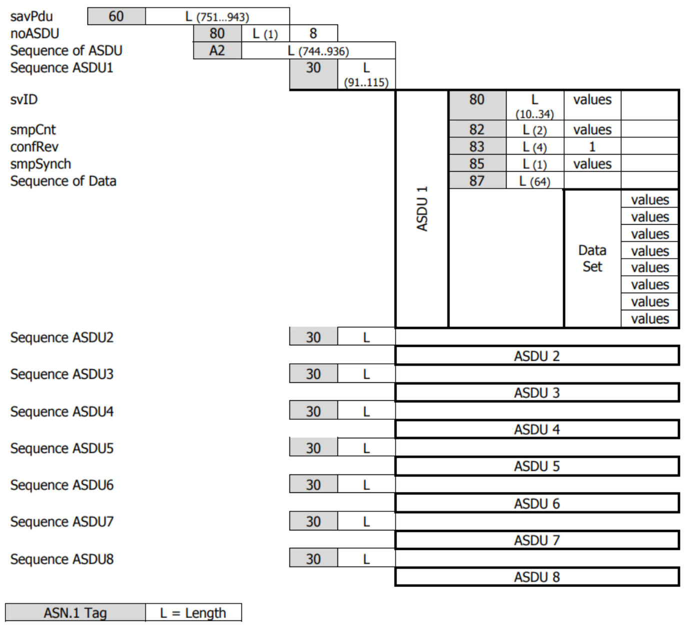


Figure 2.3: APDU definition of an SV message frame

2.3.3. GOOSE Messages

The IEC 61850 GOOSE messages are also mapped to ISO 8802-3 Ethernet frames with a header and APDU. In an IEC 61850 GOOSE message formatted as an Ethernet frame as illustrated in Figure 2.2:

- Destination address is within the range from `01:0c:cd:01:00:00` to `01:0c:cd:01:01:ff`. MAC (Media Access Controller) filtering and multicast are in use during the transmission of SV messages. The higher three bytes are assigned by IEEE and the fourth byte `01` is reserved for the use of GOOSE messages.
- Source address contains the MAC address of the device which is sending the GOOSE messages.
- ’Priority tagged’ is the VLAN tag in 802.1Q, where the TPID (Tag Protocol Identifier) is `0x8100`, the user priority is by default 4, and the VID (VLAN ID) is by default 0.
- The Ethertype reserved for SV messages is `0x0000`.
- The APPID (Application ID) for SV messages ranges between `0x0000` and `0x3fff`.
- The APDU (Application Protocol Data Unit) contains the actual data that are transferred by SV messages.

2.4. Tools used

In this section, the tools used in the development of this project are introduced. For the development of the hardware implementation of the S-energy calculation algorithm, VHDL language is used. The project is chosen to be implemented on a Xilinx Zynq MPSoC device, which is also introduced, and its tooling (Vivado Design Suite) provided by Xilinx.

2.4.1. VHDL 2008

VHDL (VHSIC Hardware Description Language) is the development language used for programming the FPGA in this project. It was initially released in the year of 1985 by a programme carried out by the United States Department of Defense [34]. Its standard was revised in 1993, 2000, 2002, 2008, and 2019 [35]. It is widely used in digital IC and FPGA hardware description and simulation.

VHDL-2008 includes a new package for the processing of fixed-point numbers in which new types are defined for a better and easier way to represent fixed-point numbers. This new package is named as `fixed_`

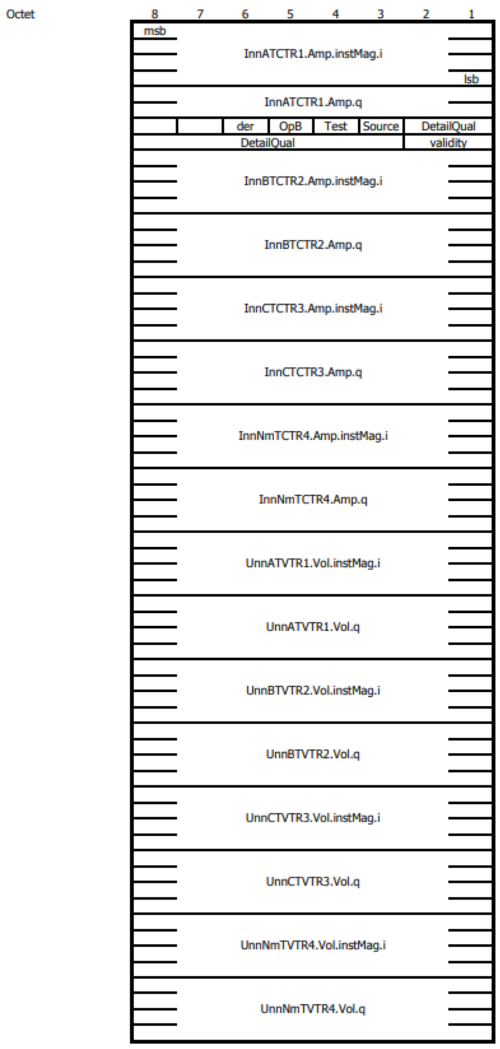


Figure 2.4: Encoding of the dataset in each ASDU

generic_pkg. It is developed based on the previous numeric_std package, which makes it available for most of commercial simulators and synthesizers without any further modifications.

It provides a better interface than the previous numeric_std package for digital signal processing, and it For example, fixed-point numbers can be defined as:

```
signal A = sfixed(3 downto -3)signal B = sfixed(2 downto -4)
```

which means there are 7 bits in total to represent the number A, where bit 3 to bit 0 are the whole part and bit -1 to bit -3 are the fractional part. Bit 2 to bit 0 are the whole part of the number B, and bit -1 to bit -4 are the fractional part. One of the major differences between the interface of mathematical operation in the new fixed_generic_pkg package and the previous numeric_std is that when the addition is performed, the numeric_std package deprecates the overflowing and underflowing bits, whereas the fixed_generic_pkg package automatically makes rooms for those bits. For example, the range of the result of operation A + B is max(A'left,B'left)+1 downto min(A'right,B'right) in the fixed_generic_pkg package. In this way, all the possible overflowing bits are reserved in operation. Therefore, the accuracy of the hardware implementation of the algorithm is easier to meet.

2.4.2. Xilinx Zynq MPSoC FPGA

FPGAs (Field Programmable Gate Arrays) are a kind of integrated circuits that can be reprogrammed to any kind of digital circuits and in any time after production [36]. This brings the FPGA a unique advantage when compared to ASICs (Application-Specific Integrated Circuits), who can achieve similar jobs in production (cite). Since the introduction of the FPGA, the application fields have kept expanding. Based on the feature of reconfigurability, the FPGAs are widely used in digital IC verification [37], image processing [38], communication [39], digital signal processing and control engineering [40], reconfigurable processor [41], deep learning [42],etc. FPGAs are capable in many other fields than those listed above.

As the requirements risen from different application fields, FPGAs have developed into many different types of efficient hardware architectures. Besides the traditional FPGA architecture, SoC [43] and MPSoC [44] (Multi-Processor SoC) architectures have become more and more popular since it combines the advantages of the fast development of the ARM core, as well as the high-speed computation of the FPGA. As will be mentioned in the later discussion, the FPGA fabric on a Xilinx Zynq MPSoC device is called Programmable Logic(PL), and the ARM core on the same device is called Processing System(PS). Due to these advantages, this project is chosen to be implemented on a Xilinx Zynq MPSoC device. A more detailed discussion on device choices can be found in the next chapter.

2.4.3. Vivado and Vitis

The hardware design part of this project is carried mostly on Vivado Design Suite. This is a development software and environment provided by Xilinx to develop and programme Xilinx FPGA and SoC FPGA devices. A usual design flow to develop on an FPGA is simulation, synthesis, and implementation. After the hardware design is programmed onto the FPGA in Vivado, the hardware file is then exported to the Vitis software for the development of the application on the ARM core.

In Vitis, first, a platform project is created and built for the later applications to run on. In this platform project, the core on which the application will be running is selected, and some basic configurations of the ARM core are defined, such as the operation modes of some IPs. Whether the application on this platform is running on bare-metal, or RTOS, or Linux is also configured here. This information is stored in a Board Support Package(BSP). Second, the application project is created after the platform is built. The real codes that are performing the functionalities are developed in this application project. After the development of the application project in Vitis, the Zynq device is supposed to perform the designed functionality. This project is developed on Vivado Design Suite HLx Edition (including Vitis) with version 2020.2.

2.5. Conclusion

In this chapter, the background that is necessary for understanding this work is introduced. The information about distance relays is provided. The previous work on developing the S-transform-based fault detection algorithm, the mechanism of the FDST and the S-energy calculation, as well as the performance of the algorithm, are explained. The reason for choosing S-transform is discussed. The widely-used IEC 61850 communication protocol is introduced, and the data format of the Sampled Values and GOOSE Messages are discussed. The data mapping to ISO 8802-3 Ethernet frames is shown to understand how the information in

IEC 61850 can be transmitted by an Ethernet interface. Finally, the tools used in this work (VHDL 2008, Xilinx Zynq SoC FPGA, Vivado and Vitis) are introduced.

3 High-Level Design Choices

In this chapter, the high-level design methodology is presented. First, the design requirements for the system are derived. Second, high-level design and device choices are derived based on the design requirements. Third, the high-level functional flow of the hardware design is presented. Fourth, an evaluation of converting the S-energy calculation algorithm in floating-point to fixed-point representation is described to validate the possibilities of such conversion.

3.1. Design Requirements

This design is supposed to fulfil the following functions:

- Receiving current value data (three phases) from the substation/RTDS in IEC 61850 Sampled Values format via the Ethernet port
- Computing the S-energy for three phases and the threshold energy value based on the proposed S-energy calculation algorithm
- Determining whether there is a fault in currents based on the comparison between the calculated S-energy and the threshold value
- Sending 'yes' nor 'no' fault back to the substation/RTDS in IEC 61850 GOOSE Messages format
- The Ethernet communication should be built with VLAN tag enabled to support IEC 61850 SV and GOOSE messages transmitting and receiving
- The speed of Ethernet communication should be 100Mbps
- The results of the comparison between the calculated S-energy and the threshold value should be available after no later than 71428 clock cycles.

Based on the functionality of the hardware implementation, the speed of the computation of the S-energy should be fast enough to be able to provide the 'yes' or 'no' fault determination within the window between the arrival of two incoming sets of current values receiving from the substation/RTDS. According to [1], with 50Hz of the power grid system, and the RTDS configuration of 80 samples per second, the sampling rate is $50Hz \times 80 = 4kHz$. With the algorithm This leads to the speed of Ethernet communication should be 100Mbps. And with the method in the proposed S-transform-based fault detection algorithm, the rate of data that are actually sent to the calculation system for S-transform (on PL) is $4kHz/3 \approx 1.4kHz$. With a possible operating speed of 100MHz of PL, this means that the results of the comparison between the calculated S-energy and the threshold value should be available after no later than $100MHz/1.4kHz \approx 71428cycles$.

3.2. Device Choices

Based on the functional design requirements in Section 3.1, and the fact that this whole design is planned to be implemented on an FPGA, the calculation of the S-energy and its corresponding threshold value is chosen to be implemented on the FPGA for maximum speed. And the IEC 61850 interface is chosen to be implemented by an Ethernet interface with VLAN tag enabled and a speed of 100 Mbps. For the simplicity of the design, this Ethernet interface is chosen to be implemented on an SOC with a baremetal application. This leads to the choice of the device to be a Xilinx Zynq SOC device which features Programmable Logic (PL) acting as a normal FPGA and Processing System (PS) containing double (Zynq 7000 devices) or multiple (Zynq Ultrascale+) devices. **The S-energy calculation and threshold calculation functions will be implemented on PL, and the IEC 61850 interface will be implemented on PS.**

With the discussion above, to validate the design on a development board featuring a Zynq 7000 or Zynq Ultrascale+ device, we first need to be sure to choose a development board with an Ethernet port embedded. Some potential choices of the development board that meet these requirements are listed below in Table 3.1.

Name	Series	CLB LUTs	CLB FFs	DSPs	BRAM(Mb)	Ethernet Port
ZCU102	Zynq Ultrascale+	274000	548000	2520	32.1	Yes
ZCU104	Zynq Ultrascale+	230000	461000	1728	11	Yes
ZCU106	Zynq Ultrascale+	230000	461000	1728	11	Yes
ZC706	Zynq 7000	218600	437200	900	10.1	Yes
ZCU702	Zynq 7000	53200	106500	220	4.9	Yes
Arty Z7-20	Zynq 7000	53200	106500	220	4.9	Yes
Zedboard	Zynq 7000	53200	106500	220	4.9	Yes

Table 3.1: Information of potential choices of the development board

In Table 3.1,

- CLB LUTs are for Configurable Logic Blocks working as Look Up Tables
- CLB FFs are for Configurable Logic Blocks working as Flip Flops
- DSPs are Digital Signal Processing slices for efficient implementation of mathematical operations
- BRAM is Block Random-Access Memory on chip

With the information listed in Table 3.1, several simple syntheses are carried out to evaluate if they are able to fit on these candidates. At this stage of design, this evaluation is formulated by synthesizing each sub-modules in the system and comparing the total resource utilization with the number of resources available on each device. Potential optimization for reducing the size of the design and fitting the design on a smaller FPGA is not considered at this stage. The resource utilization after synthesis are listed in Table 3.2.

Module	CLB LUTs	CLB FFs	DSPs
Input buffer	281	256	0
FFT	3331	1502	28
Gaussian window	333	140	8
IFFT	4713	1431	37
Post process	7414	3264	169

Table 3.2: Resource utilization of each sub-module after syntheses

With these information listed in Table 3.2, the comparison between the estimated resource utilization and the resources available can be compared. In this comparison, considering the potential system architecture on PL, and to make the evaluation more valuable, several different configurations are compared, which are:

- Configuration 1 contains eight FFT, Gaussian window, IFFT, and post process modules, and one input buffer module for a single phase
- Configuration 2 contains four FFT, Gaussian window, IFFT, and post process modules, and one input buffer module for a single phase
- Configuration 3 contains two FFT, Gaussian window, IFFT, and post process modules, and one input buffer module for a single phase

- Configuration 4 contains one FFT, Gaussian window, IFFT, and post process module, and one input buffer module for a single phase

The influences on the resource utilization caused by different control logic introduced by different configurations and possible optimization performed by the synthesis tool are not considered at this stage, since it is only a rough evaluation for whether the device is able to fit the design. The resource utilization of these four configurations for a single phase are listed in Table 3.3.

Configuration	CLB LUTs	CLB FFs	DSPs
1	126609	50952	1352
2	63445	25604	676
3	31863	12930	338
4	16072	6593	169

Table 3.3: Estimated resource utilization of different configurations for one phase

With the information listed in Table 3.1 and Table 3.3, the comparison can be made to decide on a development board to choose. The comparison is made based on whether the different configurations for a single phase is able to fit on the board, and the results are listed in Table 3.4.

Board	Config. 1	Config. 2	Config. 3	Config. 4
ZCU102	Yes	Yes	Yes	Yes
ZCU104	Yes	Yes	Yes	Yes
ZCU106	Yes	Yes	Yes	Yes
ZC706	No	Yes	Yes	Yes
ZC702	No	No	No	Yes
Arty Z7-20	No	No	No	Yes
Zedboard	No	No	No	Yes

Table 3.4: Comparison between estimated resource utilization and resources available on each board. 'Yes' means this configuration fits on this board. 'No' means this configuration does not fit on this board

Based on the information listed in Table 3.4, the choices of the development board can be firstly reduced within ZCU102, ZCU104, and ZCU106. Considering the price and factory lead time of each development board, the hardware platform of this design is finally chosen to be Xilinx ZCU104 development board. This development board features an XCZU7EV-2FFVC1156 MPSoC chip which belongs to Zynq Ultrascale+ family.

3.3. High-level Functional Flow of the Hardware Design

With the choice of a Zynq Ultrescale+ development board and the decision of S-energy and threshold value calculation being implemented on PL and communication interface on PS, the high-level functional flow of the hardware implementation is illustrated in Figure 3.1.

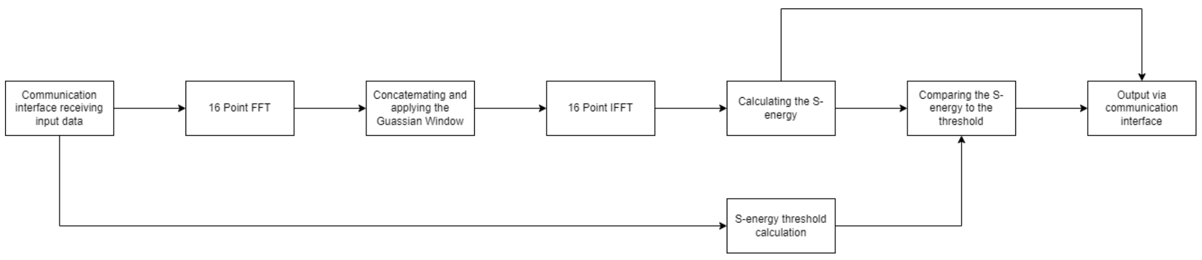

Communication interface receiving input data
16 Point FFT
Concatemating and applying the Guassian Window
16 Point IFFT
Calculating the S-energy
Comparing the S-energy to the threshold
Output via communication interface
S-energy threshold calculation

Figure 3.1: High-level functional flow diagram of the system

From Figure 3.1 it can be observed that the data are received via the IEC 61850 communication interface on PS and sent into PL for the S-energy and S-energy threshold calculation. The result of the comparison between the calculated S-energy and S-energy threshold values is sent from PL to PS and transmitted via the IEC 61850 communication interface on PS. Within the S-energy calculation, the operations applied to the received data on PL are listed as follows:

1. First, apply a 16-point FFT to the received data.
2. Apply the Gaussian window to the outputs of the 16-point FFT.
3. Apply a 16-point IFFT to the outputs of the Gaussian window. The outputs of this IFFT are the results of the FDST.
4. Calculate the S-energy based on the outputs of the IFFT (results of the FDST).
5. Calculate the S-energy threshold value every one or two seconds.
6. Compare the calculated S-energy value and the S-energy threshold value. If the calculated S-energy value is larger than the S-energy threshold value, there is a fault. If the calculated S-energy value is smaller than the S-energy threshold value, there is not a fault.

3.4. Evaluation from floating-point to fixed-point

For an FPGA implementation, a fixed-point representation of the numbers is more commonly chosen than a floating-point representation since the fixed-point representation usually results in fewer resource utilization and higher speed. However, a drawback of fixed-point representation of numbers is that the designer will need to consider the number of bits to represent the data so to meet the accuracy requirements.

In this section, the mechanism of the fixed-point numbers and floating-point numbers are introduced. Besides, some other simplifications to reduce the size and complexity of the design are also presented.

3.4.1. Fixed-point numbers

A fixed-point number is constructed by two parts, i.e., an integral part and a fractional part, and the (virtual) decimal point (in radix 10) or binary point (in radix 2) is in between. For the same set of fixed-point numbers, the decimal points remain in the same position. Therefore, it is generally not shown in the number. Within fixed-point numbers, signed fixed-point numbers and unsigned fixed-point numbers can be categorized based on whether the most significant bit (MSB) of the number represents the sign of the number or not.

The significant advantage of the fixed-point representation is that the arithmetic operations for fixed-point numbers require fewer resources and are normally faster.

3.4.2. Floating-point numbers

One motivation for the introduction of floating-point numbers is the limited dynamic range of the fixed-point representation. In other words, if a large number and a minimal number need to be represented simultaneously, the bits required for such representation may be large to distinguish the difference between them.

For most modern computers, the IEEE floating-point standard is used to represent real numbers. An IEEE-format floating point number is constructed by three parts, i.e., a sign part (±), an exponent part (e), and a significand part (s). As defined in IEEE-754 standard, floating-point numbers are represented with either 32 bits (single precision) or 64 bits (double precision) [45]. For example, an IEEE-format single-precision floating-point number can be represented as:

31	30-23	22-0
x	xxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxx
sign	exponent	significand

where:

- The most significant bit (MSB, 31st bit) represents the sign of the number. If this bit is 0, the number is positive. If this bit is 1, the number is negative.
- The exponent part (8 bits) indicates the power of the number. It is represented in a biased format. For a single-precision floating-point number, the real exponent is derived by subtracting 127 from the represented exponent. For a double-precision floating-point number, the real exponent is derived by subtracting 1023 from the represented exponent. The sign of the exponent part is omitted.
- The significand part (23 bits) represents the increasing negative powers of 2. The highest digit of the significant part is assumed to be 1 and thus omitted.

Therefore, the formula to calculate the decimal equivalence of a single-precision floating-point number can be derived as Equation 3.1.

$$n_{(ten)} = (-1)^{sign} 2^{e-127} f = (-1)^{s} 2^{e-127} (1 + \sum_{i=0}^{22} s_{23-i} 2^{-i}) \tag{3.1}$$

With the same method, the decimal equivalence of a double-precision floating-point number can be calculated using Equation 3.2.

$$n_{(ten)} = (-1)^{sign} 2^{e-127} f = (-1)^{s} 2^{e-1023} (1 + \sum_{i=0}^{51} s_{52-i} 2^{-i}) \tag{3.2}$$

3.4.3. Choice of fixed-point representation

The algorithm developed in MATLAB is represented in floating-point numbers, and the representation of numbers is normally chosen as fixed-point. Therefore, it is necessary to evaluate whether the transition from floating-point numbers to fixed-point numbers will affect the accuracy of the algorithm. Some key points to notice during the validation are:

- Whether the fixed-point representation of the algorithm can maintain a tolerable ability of fault detection
- At which part is the most significant difference

This evaluation is carried out on MATLAB and the results are discussed in Section 5.1. In short, 16-point fixed-point representation is chosen for the representing the data in the PL system for S-energy calculation.

3.4.4. Gaussian window optimization

As shown in Figure 3.1, a modified two-dimensional Gaussian window is applied after the FFT and before the IFFT. In a 16-point FFT configuration, this Gaussian window corresponds to a matrix with the size of 8×16. Each element in the Gaussian window can be stored in the registers as parameters. The elements in the Gaussian window are typically between 0 and 1, and many are minimal values. Based on the mechanism of fixed-point representation, it might require numerous data bits to represent these data precisely, resulting in much resource utilization due to the arithmetic operations.

Therefore, it is essential to think of a way to reduce the resource utilization caused by the elements in the Gaussian window. To start with, it is reasonable to guess that with the data width of 16 bits, values in the Gaussian window below 0.0001 can be treated as 0 while maintaining the tolerable accuracy of the entire algorithm. In this way, the resources that are required to store the minimal numbers and to calculate the outcome of the multiplication can be saved.

This optimization is examined alongside the 16-bit fixed-point representation of the algorithm itself, and the results are discussed in Section 5.1. As a result, elements in the Gaussian window with a value less than 0.0001 can be replaced with 0 while maintaining the general accuracy.

3.5. System overview

According to the design choices in Section 3.2, the whole system can be divided into two parts, i.e., the S-Energy calculation part on PL and the communication interface part on PS with a bare-metal application. Figure 3.2 illustrates the system architecture of the S-transform-based fault detection system.

In the system architecture shown in Figure 3.2, the intended completed system works as follows:

- The PS receives packets from the RTDS via Ethernet port in IEC61850 SV(sampled values) format
- The data received from the Ethernet port are stored in the DDR memory on chip controlled by the DMA controller
- The DMA controller send the data in DDR memory to the PL through AXI bus
- PL generates an interrupt when the outcome of the algorithm is 'fault' for PS to start the error processing
- PS sends 'yes' through Ethernet port in IEC61850 GOOSE messages format, indicating the fault's detection when receiving the interrupt. Otherwise, the PS keeps sending 'no'

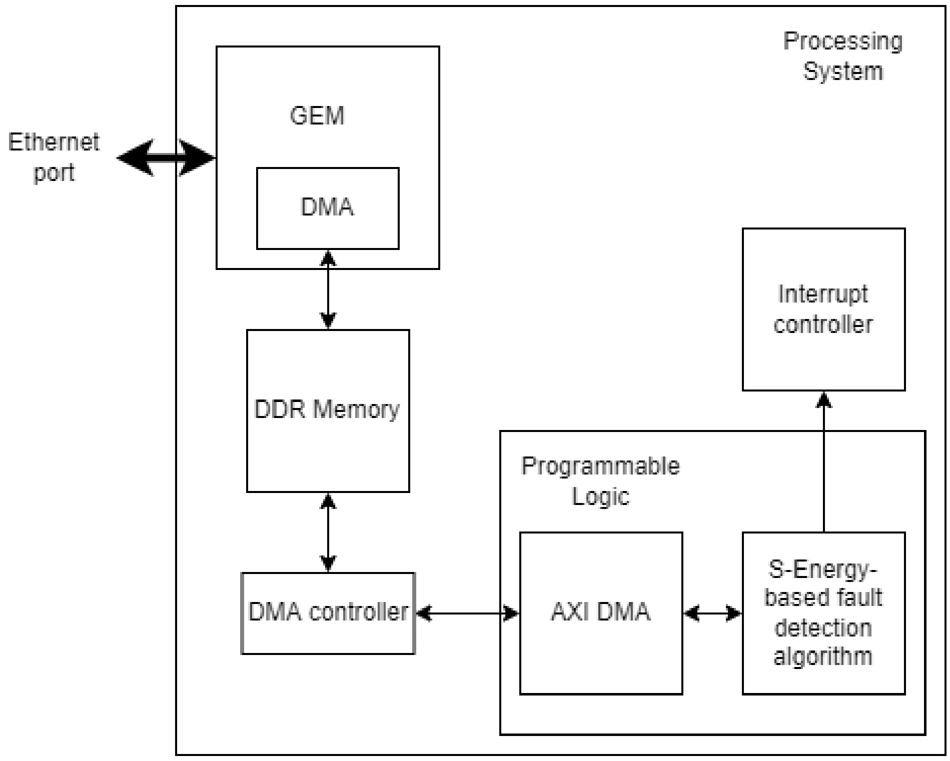


Figure 3.2: System architecture for S-energy-based fault detection algorithm

3.6. Conclusion

In this chapter, the functional requirement for the hardware design is described. High-level design choices are made based on the functional requirements, which are:

- The hardware platform of this design should be a Xilinx Zynq SoC device.
- The S-energy calculation and threshold calculation functions are implemented on PL
- IEC 61850 communication interface is implemented on PS

After the hardware platform is chosen, the specific development board for validating the design are determined based on the analysis of whether the candidate development boards contain enough resources to fit the design. The hardware platform is therefore further chosen as the Xilinx ZCU104 development board which features a Zynq Ultrascale+ XCZU7EV-2FFVC1156 MPSoC chip.

The hardware design's functional flow is illustrated to better understand the modules that need to be developed to fulfil the applicable requirements. It also provides guidelines for hardware implementation. The methodology of the floating-point and fixed-point representations are also introduced, and the evaluation on MATLAB is carried out. The results show it is valid to choose a 16-bit width of fixed-point data on hardware without losing too much accuracy. At the same time, the optimization of the Gaussian window is validated when replacing the elements in the Gaussian window whose values are smaller than 0.0001 to 0. Finally, a system architecture overview on the Zynq devices shows how PS and PL receives signals and interacting with each other.

4
Hardware Implementation of S-Transform-Based Fault Detection Algorithm

4.1. System architecture

With the discussion above, the S-Transform-based fault detection system is further divided into S-energy calculation and the S-energy threshold value determination. These two parts are implemented on PL.

The S-energy calculation can be further divided into an FFT module, a Gaussian window module, an IFFT module, and a square root module. Besides, an S-energy threshold determination module is needed to determine the S-energy threshold. The methodology of each module and the port definitions are described in this section. The parallel pipelined FFT and IFFT modules in the introduced modules, and the Gaussian window module are converted directly from the MATLAB design. And the pipelined FFT and IFFT modules and the square root module are separately designed to fit the functional requirements of the design.

Based on the functional flow diagram illustrated in Figure 3.1, the calculation of the S-energy in three phases are separated. After the calculation is finished, the result is compared with the the threshold value. A more detailed block diagram of the system architecture on PL can be derived and is illustrated in Figure 4.1. Figure 4.1 shows that the S-energy for three phases and the threshold value are calculated separately on hardware. These values are calculated in parallel. After the calculation is finished, the S-energy values are compared with the threshold value and a decision of whether there is a fault can be made. Therefore, to make the S-energy-based fault detection algorithm work on hardware, the S-energy calculation module and the threshold determination module need to be developed.

According to the methodology of the FDST, The S-energy calculation for a single phase can be further divided into an FFT module, a Gaussian window module, an IFFT module, and a square root module. The block diagram of the S-energy calculation for a single phase is illustrated in Figure 4.2.

The parallel pipelined FFT and IFFT modules in the introduced modules, and the Gaussian window module are converted directly from the MATLAB design. And the sequential pipelined FFT and IFFT modules and the square root module are separately designed to fit in the hardware design.

Figure 4.1 shows the blocks in the hardware design on PL (one phase) and how data flow from the input to the output through each functional block. This figure illustrates the architecture of the S-energy calculation when using the parallel pipelined FFT module (more information about this parallel pipelined FFT module can be found in Section 4.2) with a configuration of using eight Gaussian window modules, IFFT modules, and Square root modules. This corresponds to configuration 1 later in Section 4.10.

In Figure 4.1, a fine arrow means only one set of data is transmitted on the line, whereas a bold arrow indicates more than one set of data is transmitted on the line. The number of data being transmitted is 16 since the number of outputs of the parallel FFT module is 16, and it stays the same until being added.

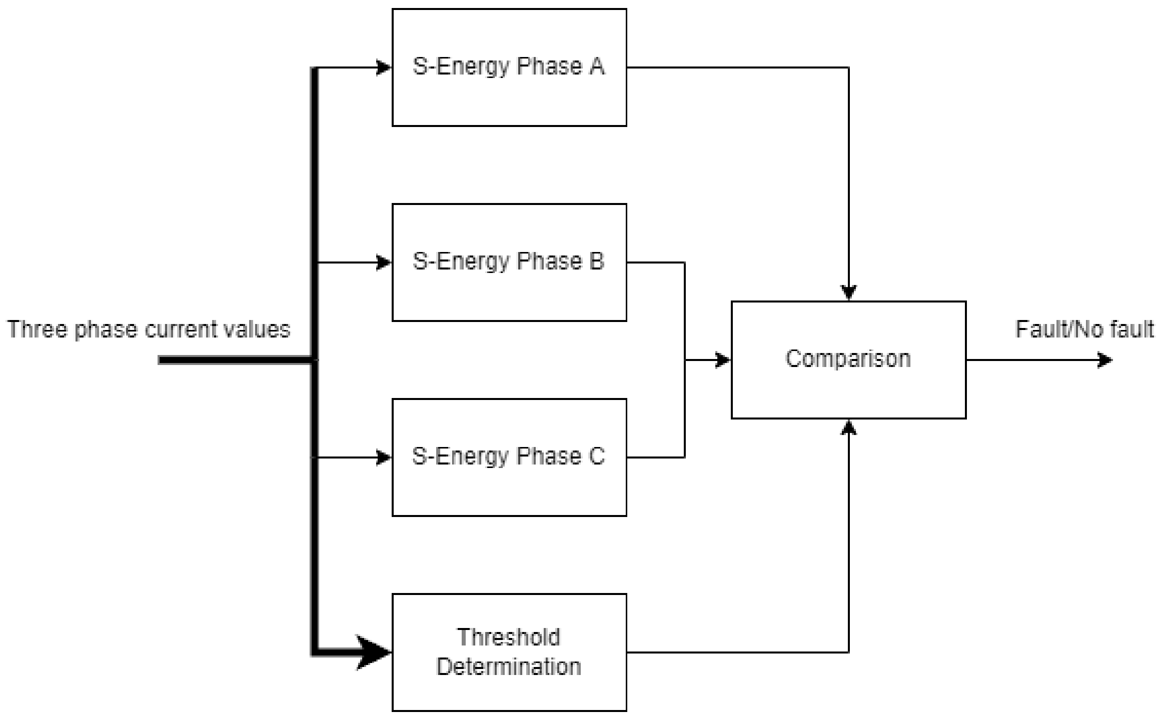


Figure 4.1: Block diagram of PL system architecture

4.2. Parallel Pipelined FFT and IFFT module

The Fast Fourier Transform converts a time-based signal into a frequency-based signal and was first introduced by Cooley and Tukey in 1965 and is widely used in the field of signal processing [46]. It provides an efficient way to compute the DFT of a series. A 2^n point FFT can be achieved by a 'divide and conquer method via two $2^{(n-1}$ point DFT. And each DFT can be divided further into smaller DFTs until only two points are left.

There are two types of FFT algorithms: decimation in time (DIT) and decimation in frequency (DIF). The data flow of the 16-point DIT FFT is illustrated in Figure 4.3.

In Figure 4.3, $x[i],(i = 0...16)$ is the input series. $X[i],(i = 0...16)$ is the output series, which is also the spectrum of the input series $x[i]$. $W[j],(j = 0...8)]$ is the twiddling factor of the FFT, which is a set of trigonometric constant coefficients that are multiplied in the FFT algorithm. This set of coefficients can be calculated by the CORDIC algorithm or stored in registers or a ROM in advance. In Figure 4.3, $W[j]$ is multiplied to the sum of each element at each node. Where there is a -1 under the node, there is a subtraction operation. There is an addition operation where there is not a -1 under the node. As shown from Figure 4.3, the order of the input series is not in the natural order, whereas the order of the output series is in the natural order. This will not affect the parallel implementation of the FFT algorithm on hardware since both the input and the output will be in parallel. However, this will affect the implementation of which the input and output series are in sequence.

The data flow of the 16-point DIF FFT is illustrated in Figure 4.4. In Figure 4.4, similar observations can be achieved as it is in Figure 4.3. $x[i],(i = 0...16)$ is the input series. $X[i],(i = 0...16)$ is the output series.$W[j],(j = 0...8)]$ is the twiddling factor of the FFT.Where there is a -1 under the node, there is a subtraction operation. There is an addition operation where there is not a -1 under the node. In contrast to the 16-point DIT FFT algorithm, in the 16-point DIF FFT algorithm, the order of the input series is in the natural order, and the order of the output series is not in the natural order.

4.2.1. Hardware Implementation of Parallel Pipelined FFT Module

The hardware implementation of the parallel pipelined FFT module is derived directly from the FFT algorithm's data flow. The FFT part is chosen to be 16 points in the proposed S-Energy calculation algorithm. Therefore, in the hardware implementation of a parallel pipelined FFT module, it is also chosen to be a 16-point FFT. In the MATLAB code of the proposed S-Energy calculation algorithm, decimation in frequency is selected. It is therefore determined in the hardware implementation as well. The twiddling factors of the

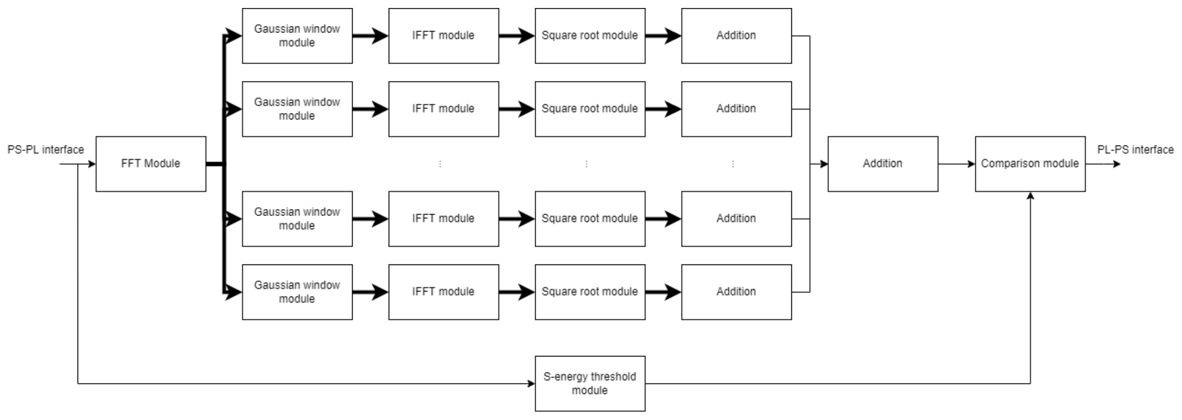


Figure 4.2: Block diagram of PL system architecture

FFT are chosen to be stored in the registers instead of generating from the CORDIC algorithm. This will save resource utilization since it reduces mathematics operations.

Considering the interfacing with PS, the input of PL (measured current values needed to be processed) is sequential. Therefore, a set of shift registers are added in front of the FFT module to transfer the input sequential data feed into parallel data. i.e., 16 inputs of the FFT are available at the same clock cycle after the shift registers.

The generic parameters of the implementation of the parallel pipelined FFT module is described in Table 4.1, and the ports of the implementation of the parallel pipelined FFT module are described in Table 4.2.

Generic	Type	Default value
`fft_length`	integer	16
`data_width`	integer	16
`in_int_width`	integer	2
`tf_real_int_width`	integer	2
`tf_imag_int_width`	integer	1
`number_window`	integer	8

Table 4.1: Generic definition of the parallel pipelined FFT module

In Table 4.1:

- `fft_length` is the length of FFT.
- `data_width` is the width of the data.
- `in_int_width` is the width of the integral part of the input data.
- `tf_real_int_width` is the width of the integral part of the stored real twiddling factors.
- `tf_imag_int_width` is the width of the integral part of the stored imaginary twiddling factors.
- `number_window` is the number of Gaussian windows.

In Table 4.2:

- `clk` and `rst` are the clock signal and the reset signal.
- `in_real_0` ~ `in_real_15` is the input series of the FFT.
- There is no imaginary part of the input series since the FFT module is receiving the current signals
- `out_real_0` ~ `out_real_15` is the real part of the output series of the FFT.
- `out_imag_0` ~ `out_imag_15` is the imaginary part of the output series of the FFT.

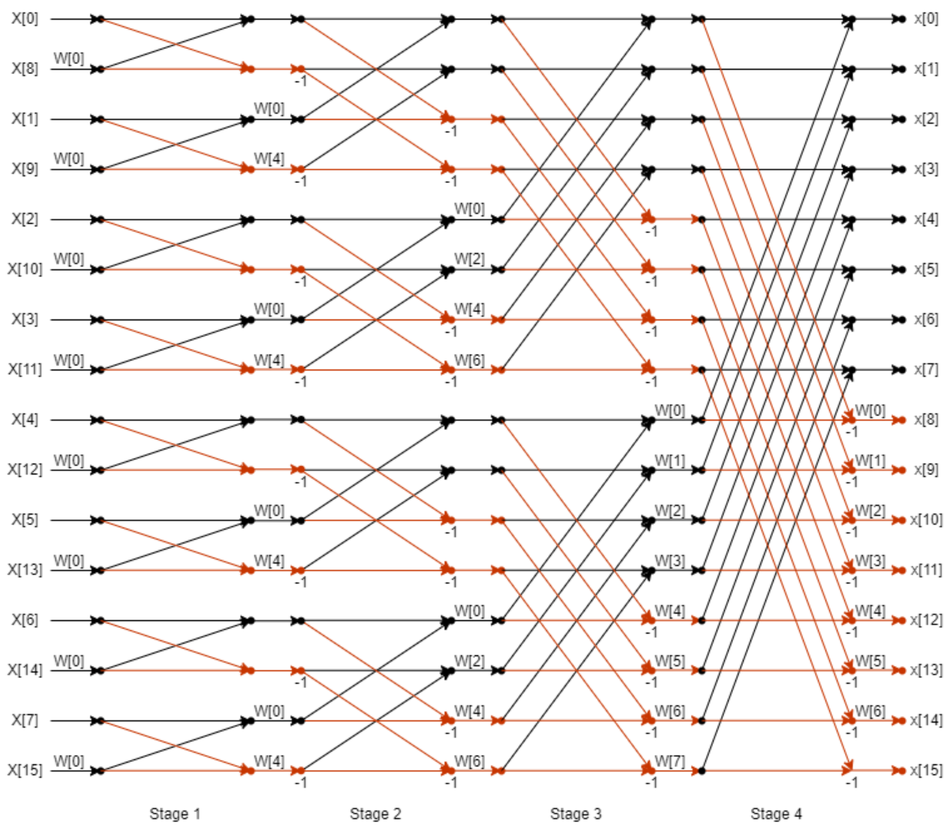


Figure 4.3: Data flow of 16-point DIF FFT

- `m_tvalid` is a signal indicating if the output of the FFT is valid and act as a master.
- `s_tstart` is a signal indicating the start of a new FFT and act as a slave.

The FFT algorithm in this module is divided into four stages in the parallel pipelined implementation. After each stage, the results are stored in registers for the usage of the next stage (therefore, it is pipelined). This will slightly increase the resource utilization in terms of flip-flops on FPGA, but it will increase the possibility of achieving a higher speed. The naming of 'parallel' FFT implementation indicates that the input and output data lines are parallel. Consequently, the FFT module generates the output series of the input series after four clock cycles (each stage takes one cycle). And `m_tvalid` signal will be set after four clock cycles of the set of `s_tstart` signal.

4.3. Sequential Pipelined FFT and IFFT Module

The parallel pipelined implementation of the FFT algorithm requires a lot of resources since it implements every mathematical operation in the FFT algorithm in separated resources. A way to deal with this problem is to reuse the processing elements inside the algorithm to reduce the size of the design in exchange of the speed of operation. The major processing element inside the FFT algorithm is the 'butterfly' element. This butterfly element operates the addition and subtraction of the data. The architecture of sequential pipelined FFT can be classified into the following types:

- Single-path Delay Feedback (SDF)
- Multiple-path Delay Commutator (MDC)
- Single-path Delay Commutator (SDC)

The radix of the FFT algorithm varies based on the power of 2. The radix-2 FFT is simple to implement, and radix-4 or higher has more advantages in higher points of FFT. Since only the 16-point FFT is implemented, radix-2 is chosen.

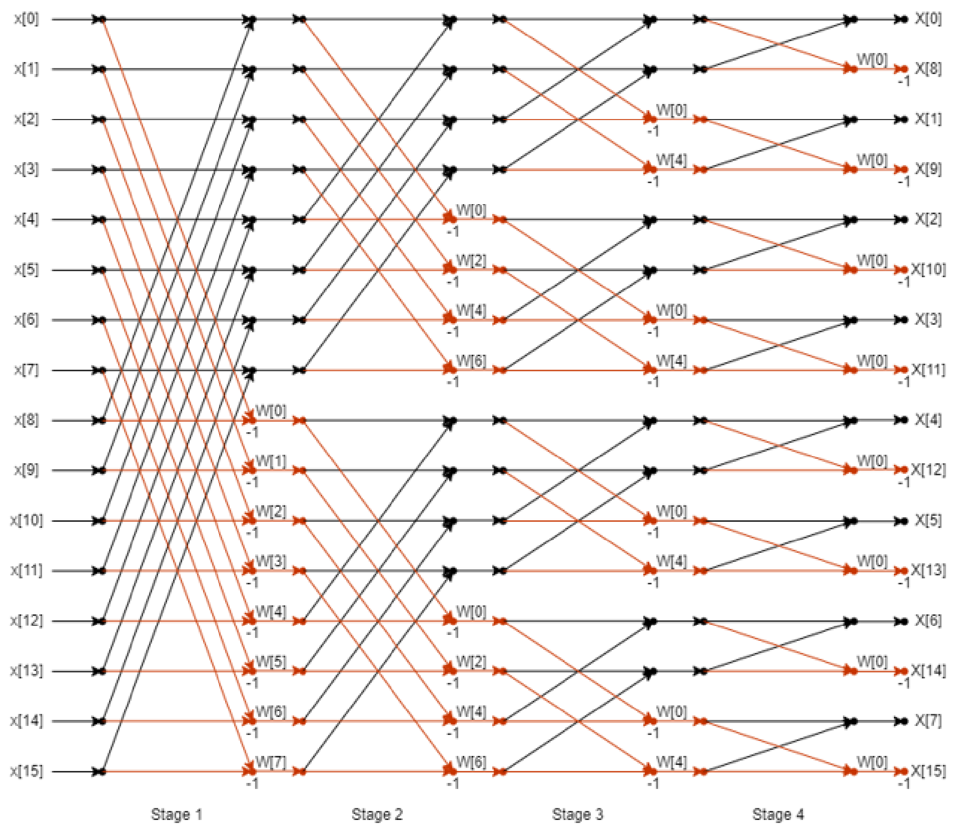


Figure 4.4: Data flow of 16-point DIT FFT

The architecture of the Radix-2 Single-path Delay Feedback (R2SDF) FFT is illustrated in Figure 4.5.

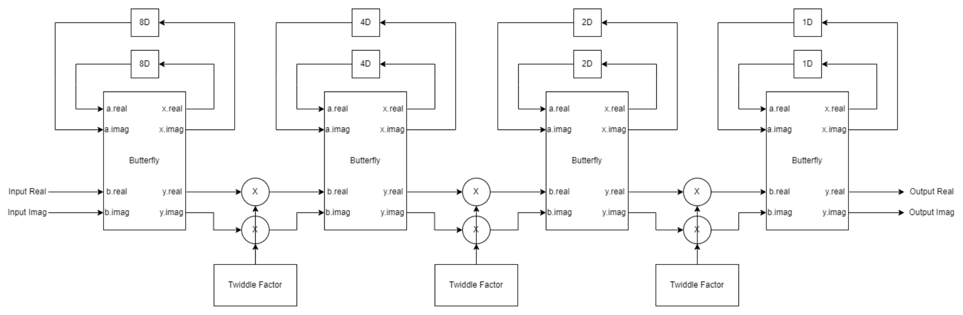


Figure 4.5: Architecture of R2SDF FFT

In Figure 4.5, it can be observed that the sub-modules in the R2SDF FFT are:

- Butterfly module, which can either pass through the data or perform the addition/subtraction
- Delay line module, which is formed by shift registers to provide delays in a certain amount of clock cycles
- Registers that store the twiddling factors
- complex multiply module, which includes a complex multiplier and performs the multiplication of the twiddling factor (is inside the butterfly module in Figure 4.5)

This architecture is referred to as sequential pipelined FFT here since it takes the input of the FFT in sequence and is still pipelined. With the above discussion, the generic parameters of the implementation of

Port	Direction	type
`clk`	input	`std_logic`
`rst`	input	`std_logic`
`in_real_0 ~ in_real_15`	input	`std_logic_vector(data_width-1 downto 0)`
`in_imag_0 ~ in_imag_15`	input	`std_logic_vector(data_width-1 downto 0)`
`out_real_0 ~ out_real_15`	output	`std_logic_vector(data_width-1 downto 0)`
`out_imag_0 ~ out_imag_15`	output	`std_logic_vector(data_width-1 downto 0)`
`m_tvalid`	output	`std_logic`
`s_tstart`	input	`std_logic`

Table 4.2: Port definition of the parallel pipelined FFT module

Generic	Type	Default value
`data_width`	integer	16
`tf_width`	integer	16
`fft_length`	integer	4

Table 4.3: Generic definition of the sequential pipelined R2SDF FFT module

pipelined R2SDF FFT are described in Table 4.3, and the ports of the implementation of pipelined R2SDF FFT are described in Table 4.4.

In Table 4.3:

- `data_width` is the width of the input data.
- `tf_width` is the width of the stored twiddling factors.
- `fft_length` is the stages of the FFT operation. This parameter has a different meaning than the `fft_length` in parallel pipelined FFT module. Here the number of points of FFT is represented as 2^{fft_length}.

Port	Direction	type
`clk`	input	`std_logic`
`rst`	input	`std_logic`
`din_real`	input	`std_logic_vector(data_width-1 downto 0)`
`din_imag`	input	`std_logic_vector(data_width-1 downto 0)`
`dout_real`	output	`std_logic_vector(data_width-1 downto 0)`
`dout_imag`	output	`std_logic_vector(data_width-1 downto 0)`
`m_dout_tvalid`	output	`std_logic`
`m_dout_index`	input	`std_logic_vector(fft_length-1 downto 0)`
`s_din_tvalid`	input	`std_logic`

Table 4.4: Port definition of the sequential pipelined R2SDF FFT module

In Table 4.4:

- `clk` and `rst` are the clock signal and the reset signal.
- `din_real` is the input series of the FFT and the real part of the input series of the IFFT.
- `din_imag` is set to `0` in FFT and is the imaginary part of input series of the IFFT.
- `dout_real` is the real part of the output series of the FFT and the IFFT.
- `dout_imag` is the imaginary part of the output series of the FFT and the IFFT.
- `m_dout_tvalid` is a signal indicating if the output of the FFT is valid and acts as a master.
- `s_din_tvalid` is a signal indicating if the data from the input of FFT is valid and acts as a slave.

- m_dout_index is a signal indicating the index of the elements from the output series of the FFT and IFFT, and is used for applying the Gaussian window.

In the R2SDF FFT, the data that are added by the butterfly are passed to the second stage, and the data that are subtracted by the butterfly are fed to the delay line. The butterfly pasess through these subtracted data when these data are fed in to the butterfly again from the delay line. The subtracted data then are passed to the complex multiplier and multiplied by the twiddling factor stored in the registers. The operations in the next two stages are similar to the first stage. In the parallel pipelined FFT implementation, a set of shift registers is implemented before the FFT operation to transform the data from sequential to parallel. This set of shift registers is no longer needed for sequential pipelined FFT implementation since this module takes one element at each clock cycle.

Based on the descriptions above, the port definitions and behaviours of each sub-modules in the FFT module can be derived.

4.3.1. Butterfly Module

The generic parameters of the butterfly module are described in Table 4.5 and the ports of the butterfly module are described in Table 4.6.

Generic	Type	Default value
data_width	integer	16
in_int_width	integer	3
out_int_width	integer	3

Table 4.5: Generic definition of the butterfly module

In Table 4.5:

- data_width is the width of the input data.
- in_int_width is the width of the integral part of the input data.
- out_int_width is the width of the integral part of the output data.

Port	Direction	type
control	input	std_logic
iu_real	input	std_logic_vector(data_width-1 downto 0)
iu_imag	input	std_logic_vector(data_width-1 downto 0)
il_real	input	std_logic_vector(data_width-1 downto 0)
il_imag	input	std_logic_vector(data_width-1 downto 0)
ou_real	output	std_logic_vector(data_width-1 downto 0)
ou_imag	output	std_logic_vector(data_width-1 downto 0)
ol_real	output	std_logic_vector(data_width-1 downto 0)
ol_imag	output	std_logic_vector(data_width-1 downto 0)

Table 4.6: Port definition of the butterfly module

In Table 4.6:

- control is the signal that controls the butterfly module to operate whether the addition/subtraction or the direct pass-through.
- iu_real and iu_imag are the signals that are the real part and the imaginary part of the upper input of the butterfly.
- il_real and il_imag are the signals that are the real part and the imaginary part of the lower input of the butterfly.
- ou_real and iu_imag are the signals that are the real part and the imaginary part of the upper output of the butterfly.

- ol_real and il_imag are the signals that are the real part and the imaginary part of the lower output of the butterfly.

The butterfly module operates the addition or the subtraction of the real part of the upper inputs or the imaginary part of the lower inputs, as well as passes the input data directly to the output without any operations. This is controlled by the control signal. The operations can be illustrated as 4.1 and 4.2.

$$upper = upper + lower \tag{4.1}$$

$$lower = upper - lower \tag{4.2}$$

4.3.2. Delay Line module

The generic parameters of the delay line module are described in Table 4.7, and the ports of the delay line module are described in Table 4.8.

Generic	Type	Default value
data_width	integer	16
delay	integer	8

Table 4.7: Generic definition of the delay line module

In Table 4.7:

- data_width is the width of the input data.
- delay is the delay in clock cycles, which is set as various values in the R2SDF FFT to achieve delays in different clock cycles.

Port	Direction	type
clk	input	std_logic
rst	input	std_logic
en	input	std_logic
d	input	std_logic_vector(data_width-1 downto 0)
q	output	std_logic_vector(data_width-1 downto 0)

Table 4.8: Port definition of the delay line module

In Table 4.8:

- clk is the clock signal.
- rst is the reset signal.
- en is the enable signal.
- d is the input data of the delay line.
- q is the output data of the delay line.

Shift registers form the delay line module to provide a certain delay (in clock cycles) for the data generated by the butterfly to feed back to itself. The generic parameter controls the number of clock cycles for the delay.

4.3.3. Twiddling Factor Register Module

The ports of the twiddling factor register module are described in Table 4.10.

In Table 4.9:

- tf_width is the width of the stored twiddling factors.
- addr_width is the width of the address signal.

In Table 4.10:

Generic	Type	Default value
`tf_width`	integer	16
`addr_width`	integer	2

Table 4.9: Generic definition of the twiddling factor register module

Port	Direction	type
`clk`	input	`std_logic`
`rst`	input	`std_logic`
`en`	input	`std_logic`
`addr_control`	input	`std_logic_vector(addr_width-1 downto 0)`
`control`	input	`std_logic`
`out_real`	output	`std_logic_vector(tf_width-1 downto 0)`
`out_imag`	output	`std_logic_vector(tf_width-1 downto 0)`

Table 4.10: Port definition of the twiddling factor register module

- `clk` is the clock signal.
- `rst` is the reset signal.
- `en` is the enable signal.
- `addr_control` is the signal indicating the address reading from the registers that store the twiddling factors.
- `control` is the signal that controls whether to apply the twiddling factors to the multiplication or not.
- `out_real` is the real part of the selected twiddling factor.
- `out_imag` is the imaginary part of the selected twiddling factor.

The twiddling factors are stored in registers. When the data generated by the butterfly do not need to be multiplied by the twiddling factor, none of the twiddling factor will be selected. When none of the twiddling factor is selected, the real part of the output is set to 1, and the imaginary part of the output is set to 0.

4.3.4. Complex Multiplier Module

The generic parameters of the complex multiplier module are described in Table 4.11, and the ports of the complex multiplier module are described in Table 4.12.

Generic	Type	Default value
`data_width`	integer	16
`in_int_width`	integer	3
`out_int_width`	integer	3
`tf_width`	integer	16
`tf_real_int_width`	integer	2
`tf_imag_int_width`	integer	1

Table 4.11: Generic definition of the complex multiplier module

In Table 4.11:

- `data_width` is the width of the input and output data.
- `in_int_width` is the width of the integral part of the input data.
- `out_int_width` is the width of the integral part of the output data.
- `tf_width` is the width of the twiddling factors.
- `tf_real_int_width` is the width of the integral part of the twiddling factors.

Port	Direction	type
`clk`	input	`std_logic`
`rst`	input	`std_logic`
`en`	input	`std_logic`
`tf_real`	input	`std_logic_vector(tf_width-1 downto 0)`
`tf_imag`	input	`std_logic_vector(tf_width-1 downto 0)`
`in_real`	input	`std_logic_vector(data_width-1 downto 0)`
`in_imag`	input	`std_logic_vector(data_width-1 downto 0)`
`out_real`	output	`std_logic_vector(data_width-1 downto 0)`
`out_imag`	output	`std_logic_vector(data_width-1 downto 0)`

Table 4.12: Port definition of the complex multiplier module

- `tf_imag_int_width` is the width of the imaginary part of the twiddling factors.

In Table 4.12:

- `clk` is the clock signal.
- `rst` is the reset signal.
- `en` is the enable signal.
- `tf_real` and `tf_imag` are the real part and the imaginary part of the twiddling factor.
- `in_real` and `in_imag` are the real part and the imaginary part of the input data of the complex multiplier.
- `out_real` and `out_imag` are the real part and the imaginary part of the output of the complex multiplier.

The complex multiplying operation can be expressed as equation 4.3.

$$(a+bi)\times(c+di) = (ac-bd)+(ad+bc)i \tag{4.3}$$

Therefore, the real and imaginary parts of the complex multiplication can be calculated separately, each with one addition and two multiplications.

4.4. Parallel Pipelined vs. Sequential Pipelined FFT Implementation

The term "parallel" FFT implementation is derived from the input data and the output data of the FFT module are in parallel. The hardware implementation itself is also pipelined. The choice of the naming is for a better reference for the comparison. Here the comparison of the parallel pipelined and R2SDF implementation is discussed.

It is easy to observe that the ports defined in the two implementations differ. There are 16 input data ports (16 extra for the imaginary part if it is IFFT) in the parallel pipelined implementation and one input port (one extra for the imaginary part if it is IFFT) in the R2SDF implementation. The circumstances for the output data ports are similar. There are 32 output data ports in total in the parallel pipelined implementation and two output data ports in total in the R2SDF implementation. This will affect the interface with the Gaussian window module connected to it. Another difference in the output ports is that the R2SDF implementation has a port that indicates the index of the output. This is because the Gaussian window module needs to know the index of the output of the FFT to apply the correct element for the multiplication.

In terms of architecture, for 16-point FFT, the parallel pipelined implementation performs 32 complex multiplications, 32 additions, and 32 subtractions. The result of each stage is stored in registers. The output of the FFT is available after four clock cycles of the input. The R2SDF implementation contains four complex multiplication, four additions, and four subtractions. The first output of the FFT is available after 18 clock cycles of the first input of the FFT.

From a broader view of the impacts on the whole system, using parallel pipelined FFT or sequential pipelined FFT implementation for FFT modules and IFFT modules also affects the interfacing between FFT module and Gaussian window module and between Gaussian window module and IFFT module. The output results from the Gaussian window module are written into a RAM where the address is generated by the index of the output signal (`m_dout_index`). This RAM is connected to the IFFT module to feed the data in the correct sequence for the IFFT. This RAM is no longer needed in the parallel pipelined FFT implementation since all 16 outputs of the Gaussian window are available at the same clock cycle. Therefore, the IFFT module can get these data simultaneously without needing to be concerned about storing and re-ordering.

4.5. Gaussian window module

The Gaussian window module takes care of applying the two dimensional Gaussian window to the output of the FFT module. The generic parameters of the Gaussian window module are described in Table 4.13, and the ports of the Gaussian window module are described in Table4.14.

Generic	Type	Default value
`data_width`	integer	16

Table 4.13: Generic definition of the Gaussian window module

In Table 4.13:

- `data_width` is the width of the input and output data.

Port	Direction	type
`clk`	input	`std_logic`
`rst`	input	`std_logic`
`fft_index`	input	`std_logic_vector(3 downto 0)`
`window_index`	input	`std_logic_vector(3 downto 0)`
`din_real`	input	`std_logic_vector(data_width-1 downto 0)`
`din_imag`	input	`std_logic_vector(data_width-1 downto 0)`
`dout_real`	output	`std_logic_vector(data_width-1 downto 0)`
`dout_imag`	output	`std_logic_vector(data_width-1 downto 0)`
`dout_index`	output	`std_logic_vector(3 downto 0)`
`s_din_tvalid`	input	`std_logic`
`m_dout_tvalid`	output	`std_logic`

Table 4.14: Port definition of the Gaussian window module

In Table 4.14:

- `clk` is the clock signal.
- `rst` is the reset signal.
- `fft_index` is the signal indicating the index of the FFT module (from 0 to 7, this signal does not exist in the parallel pipelined FFT implementation since the outputs are in parallel and the index is not needed).
- `din_real` and `din_imag` are the real part and the imaginary part of the input data of the Gaussian window.
- `dout_real` and `dout_imag` are the real part and the imaginary part of the output of the Gaussian window.
- `dout_index` is the signal indicating the index of the output of the Gaussian window (from 0 to 16, this signal does not exist in the parallel pipelined FFT implementation since the outputs are in parallel, and the index is not needed).
- `s_din_tvalid` is the signal received from the FFT module to indicate whether the input data of the Gaussian window module (output data of the FFT module) is valid.

- `m_dout_tvalid` is the signal indicating whether the output data of the Gaussian window module is valid.

The elements of the Gaussian window are stored in registers with the optimization mentioned earlier. Therefore, only the elements that are larger than 0.01 is stored, and other elements are set as 0. In the parallel pipelined FFT implementation, the output data of the FFT module are in parallel. Thus, the FFT outputs and the elements in the Gaussian window can be matched by encoding the index of the signals. Whereas, in order to match the elements in the Gaussian window and the outputs of the pipelined FFT implementation to generate the correct outcome, the index signals are needed for the Gaussian window module to understand which one is going in and is able to find the corresponding Gaussian window element to multiply. The index of the output is also generated based on the index of the input signal. In the pipelined FFT implementation, the outputs of the Gaussian window module are written into a RAM where the index of the output signal generates the address of the writing operation. This RAM is connected to the IFFT module on the other side to feed the data in the correct sequence of the IFFT so that the proper results can be calculated.

4.6. Square root module

The square root module is developed based on the non-restoring square root algorithm proposed by Yamin Li and Wanming Chu [47]. This square root algorithm iterates through the resulting bit and generates one correct bit in each iteration. The partial remainder generated in each iteration is the key point of this algorithm and is used in the next iteration to determine the operation of the next iteration. The architecture of this algorithm is illustrated in Figure 4.6.

The data bits of the operand are adjustable. The 32-bit operand is chosen to illustrate here since it will be the setting used for the square rooting in the system if the data bit of the input current values are set to 16. As shown in Figure 4.6, D are the registers that store the input operand, and the elements are shifted by 2 bits per iteration. The MSB of the remainder R determines if the operation in the next iteration is an addition or a subtraction. Q is the result of the square root operation. As can be seen from Figure 4.6, in each of iteration, one bit of the result is generated. Therefore, the result of the square root operation is available after 16 clock cycles of the input operand.

The generic parameters of the square root module are described in Table 4.19, and the ports in the square root module are described in Table 4.20.

Generic	Type	Default value
`sq_data_width`	integer	32

Table 4.15: Generic definition of the square root module

In Table 4.19:

- `sq_data_width-1` is the width of the input data.

Port	Direction	type
clk	input	`std_logic`
rst	input	`std_logic`
din	input	`std_logic_vector(sq_data_width-1 downto 0)`
dout	output	`std_logic_vector((sq_data_width/2)-1 downto 0)`
s_tstart	input	`std_logic`
m_tvalid	output	`std_logic`

Table 4.16: Port definition of the square root module

In Table 4.20:

- `clk` is the clock signal.
- `rst` is the reset signal.
- `din` is the operand of the square root operation.

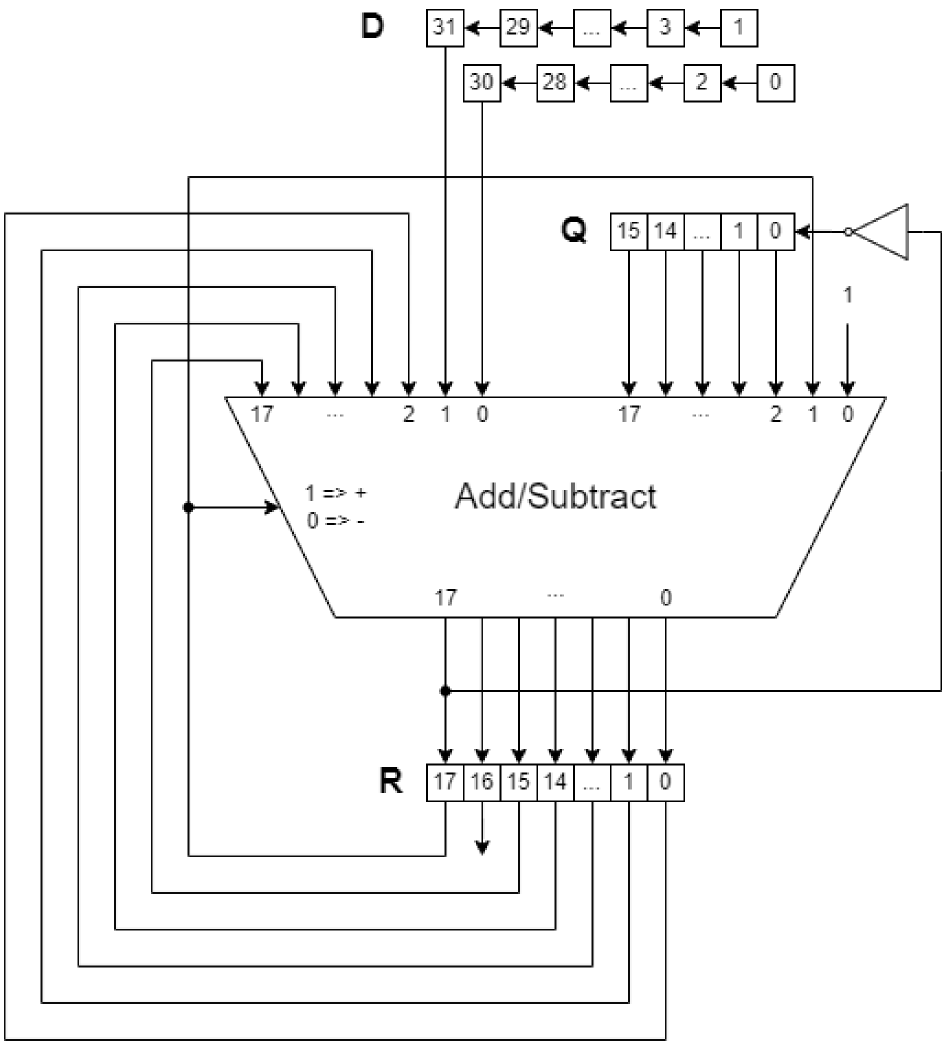


Figure 4.6: Architecture of hardware implementation of the non-restoring iterative square root algorithm[47]

- `dout` is the result of the square root operation.
- `s_tstart` is the signal controlling the module to start the square root operation.
- `m_tvalid` is the signal indicating whether the output data of the square root module is valid.

4.7. S-energy Threshold Determination

The S-energy threshold determination module is a module outside the S-energy calculation module. It takes three phase current signals into itself to determine the S-energy threshold value with a safety margin. The specific calculation is already introduced in Section 2.2.3, and is listed here for easier reference. Suppose we have I_{RMS} as the RMS value of the three-phase input signal values. The S-energy threshold value can be calculated as Equation 4.4,

$$S_{THsm} = -2.6\times10^3 I_{rms}^4 + 3.5\times10^3 I_{rms}^3 - 1.7\times10^3 I_{rms}^2 + 4.2\times10^2 I_{rms} - 45 \tag{4.4}$$

where S_{THsm} is the S-energy Threshold value.

As presented in [1], the S-energy threshold value is updated every one or two seconds. Therefore, this is implemented by a counter to count to one second until the calculation of RMS value of the current values begins. After the calculation of the S-energy threshold value based on Equation 4.4, this value is in dB, which is used to compared with the S-energy calculated by the hardware in normal values. In order to avoid logarithmic operations, further processing is applied. The relationship between the calculated S-energy threshold value in dB S_{THsm} and the original S-energy threshold value is shown in Equation 4.5.

$$S_{TH_{sm,original}} = 20\log_{10} S_{TH_{sm}} \tag{4.5}$$

To make it a better fit for the hardware implementation on an FPGA, the base of the logarithms operation is changed from 10 to 2 using:

$$\log_{10} x = \frac{\log_2 x}{\log_2 10} \tag{4.6}$$

Based on Equation 4.5 and Equation 4.6, the original value of the S-energy threshold can be found as follows.

$$\log_2 S_{TH_{sm,original}} = \frac{S_{TH_{sm}}}{20} \log_2 10 \tag{4.7}$$

Therefore,

$$S_{TH_{sm,original}} = 2^{\frac{S_{TH_{sm}}}{20} \log_2 10} = 2^{0.1661 \cdot S_{TH_{sm}}} \tag{4.8}$$

In this way, shown in Equation 4.8, the logarithmic operation can be avoided. Besides, the exponent operation is changed to the power of 2 instead of 10. The reason for doing so is that the calculation of the power of 2 can be achieved by shifting the bits on hardware, which will save significant resources. The number of bits that needs to be shifted is decided by $0.1661 \cdot S_{TH_{sm}}$, which is rounded to the nearest integer since the number of shifting bits can only be an integer. Naturally, this will introduce some deviations. These deviations are evaluated in Chapter 5.

Based on the previous discussion, the generic parameters of the S-energy threshold determination module are described in Table 4.19, and the ports are described in Table 4.20.

Generic	Type	Default value
`data_width`	integer	16
`in_int_width`	integer	3
`out_int_width`	integer	5
`sq_int_width`	integer	6

Table 4.17: Generic definition of the S-energy threshold determination module

In Table 4.17:

- `data_width` is the width of the input and output data.
- `in_int_width` is the width of the integral part of the input data.
- `out_int_width` is the width of the integral part of the output data.
- `sq_int_width` is the width of the integral part of the intermediate signal during the calculation of the RMS value.

4.8. Optimization and Parameterization of the Design

The design is parameterized in different modules in case some of the parameters need to be changed in the future. These adjustable parameters are defined as `generic` in VHDL. in the top module of the design on PL, the `data_width`, `tf_width`, and `fft_length` can be modified. `data_width` is the width of the input data and the intermediate results. `tf_width` is the width of the twiddling factors in FFT. Within each sub-module (if existed), `generic` are defined for flexibility of use in the parent module.

In the parallel pipelined FFT implementation, the parallel pipelined FFT module is implemented in the system on PL. The adjustable for the top module of this implementation is listed in Table 4.18.

As can be observed in Table 4.18, the numbers of the Gaussian windows can be modified. Since each Gaussian window module is connected to an IFFT module, this parameter also controls the numbers of the IFFT module. The default values of the generic parameters in Table 4.18 are set based on the MATLAB fixed-point converter simulation results.

In pipelined FFT implementation, the adjustable generic parameters are located in different sub-modules.

Generic	Default	Note
fft_length	16	the points of the FFT
data_width	16	the width of the fixed-point numbers
fft_in_int_width	2	the width of the integral part of the input of the FFT
ifft_in_int_width	5	the width of the integral part of the input of the IFFT
tf_int_width	1	the width of the integral part of the twiddling factors of the FFT
number_window	8	the number of Gaussian windows and IFFT modules in the architecture
sq_out_int_width	1	the width of the integral part of the output of square rooting
sum_out_int_width	5	the width of the integral part of the output of the FDST

Table 4.18: Generic definition of the parallel pipelined FFT module

4.9. Pipelining the Design

Pipelining is a widely chosen method to increase the speed of the design. The advantages of pipelining the design are the clock frequency increment and throughput. However, it will introduce more resource utilization [48]. Reviewing the current design, the FFT and IFF module, the Gaussian window module, and the S-energy threshold calculation module are already implemented in a pipelined method. However, the square root module is not currently implemented pipelined. Therefore, the pipelining of the square root module is discussed here.

The pipelined version of the square root module is also inspired and developed based on the non-restoring square root algorithm proposed in [47]. The architecture of the pipelined implementation of the non-restoring square root algorithm is illustrated in Figure 4.7.

As illustrated in Figure 4.7, there are 16 adder/subtractors in total in the pipelined hardware implementation of the non-restoring iterative square root algorithm. The algorithm is therefore pipelined into 16 stages, with the results of each stage stored in registers. Compared with the non-pipelined non-restoring square root implementation, it is evident that this pipelined implementation takes more resources. The advantage of this pipelined implementation is that it can take the input operand in every clock cycle (i,e, It can initiate a new square root operation every clock cycle). In contrast, the previous implementation of the square root module can only initiate a new square root operation every 16 clock cycles. The first output (the result of the square root operation) of the input operand series is available 16 clock cycles after receiving the first input.

The generic parameters of the square root module are described in Table 4.19, and the ports in the square root module are described in Table 4.20.

Generic	Type	Default value
`sq_data_width-1`	integer	32

Table 4.19: Generic definition of the pipelined square root module

In Table 4.19:

- `sq_data_width-1` is the width of the input data.

Port	Direction	type
clk	input	`std_logic`
rst	input	`std_logic`
din	input	`std_logic_vector(sq_data_width-1 downto 0)`
dout	output	`std_logic_vector((sq_data_width/2)-1 downto 0)`

Table 4.20: Port definition of the pipelined square root module

In Table 4.20:

- `clk` is the clock signal.
- `rst` is the reset signal.

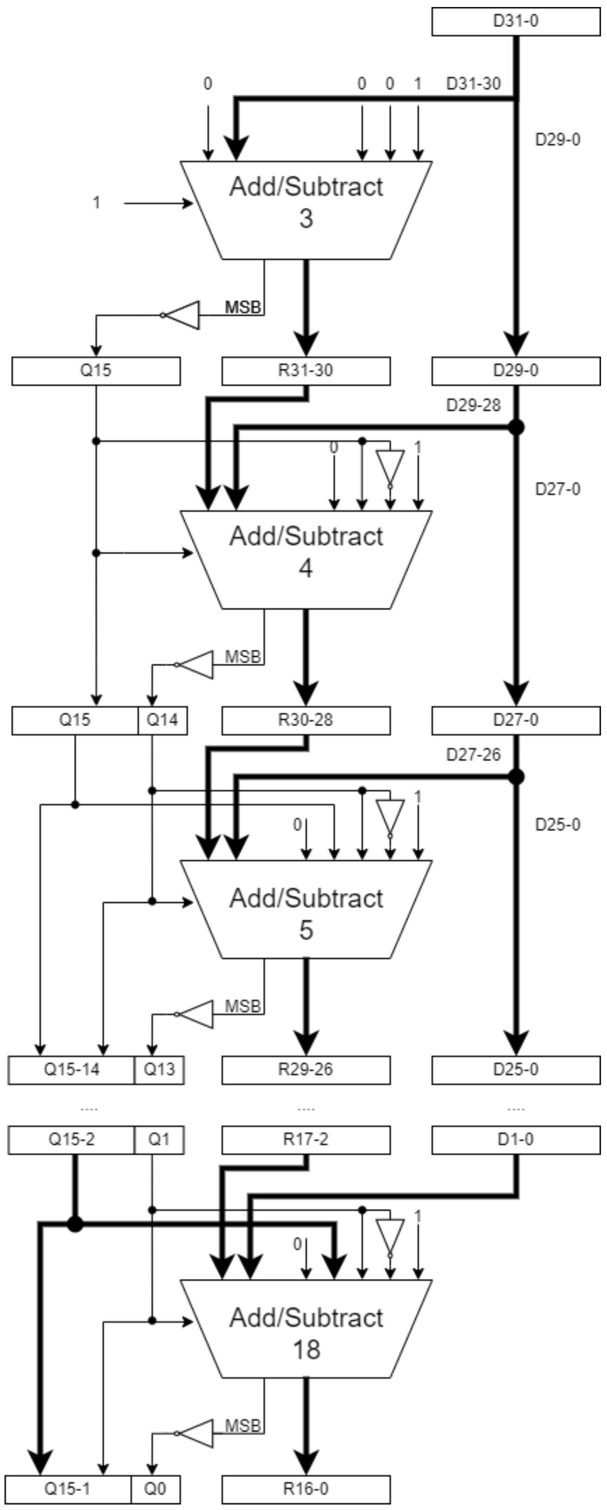


Figure 4.7: Architecture of pipelined hardware implementation of the non-restoring iterative square root algorithm[47]

- `din` is the operand of the square root operation.
- `dout` is the result of the square root operation.

Pipelining is also a method to resolve the set-up time violations. A too-long logic path normally causes these violations. So the operations cannot be finished in one clock cycle, and results cannot be stored in the next register. Adding registers in between will save the intermediate results and fix the set-up time violation.

During the tryout synthesis of the hardware design of S-energy calculation algorithm with parallel pipelined FFT and IFFT modules, when setting the clock frequency as 100MHz, the hardware design faces set-up time violation. The reason of that is during the final addition operation of the the outputs from the square root modules, the number of logic levels between two registers are too large. To solve this problem, pipelining is applied to these adders to reduce the logic levels. After making the previous one stage addition to four stages, the design does not face set-up time violation anymore.

4.10. Fitting on a smaller FPGA

The system with parallel pipelined FFT implementation takes numerous resources on PL. The most resource-utilizing module in the system is FFT/IFFT module since it contains many mathematical operations. It worsens the situation where both FFT and IFFT modules are instantiated eight times in the design.

Reusing the modules is one way to improve the high resource utilization caused by this and make it fit on a smaller FPGA., the original design contains one FFT module, eight Gaussian window modules, and eight IFFT modules. With the reuse of each module, this number can be reduced to four, two , and one, as shown in Table 4.21.

Configuration	Number of FFT module	Number of Gaussian window module	Number of IFFT module
1	1	8	8
2	1	4	4
3	1	2	2
4	1	1	1

Table 4.21: Different configurations in terms of numbers of modules

Making use of these configurations leads to other impacts and modifications in other parts of the design:

- In configuration 1, the results of the FFT module are only available for one clock cycle. However, in other configurations, the results of the FFT need to stay longer for the Gaussian window module to apply multiplications of different elements. In configuration 2, the results of the FFT need to stay available for two clock cycles. In configuration 3, the results of the FFT need to stay available for four clock cycles. In configuration 4, the results of the FFT need to stay available for eight clock cycles. This can be achieved by applying a delay after the FFT, as illustrated in Figure 4.8.
- Another control signal needs to be added to the design to control the Gaussian window and apply the correct element stored for the multiplication. This can be done by adding a state machine and letting the control signal control the generic settings of the Gaussian window modules.
- The results of the IFFT need to be stored after each calculation since the results are not all available together. The simple addition method is no longer valid. This can be achieved by applying shift registers after the IFFT module to make the sequential output of the IFFT parallel again.

After the modification of different configurations listed in Table 4.21, the latency of the S-energy calculation for configurations 2, 3, and 4 will be larger than configuration 1. However, if it is still acceptable to meet the requirements introduced by receiving SV messages and the resource utilization can be reduced by a significant amount, it is still reasonable to accept the tradeoff.

4.11. Conceptual Methods of Implementing IEC 61850 Interface

As mentioned in Chapter 2, IEC 61850 communication protocol plays a vital role in the electrical power system field. Therefore, one of the goals of this project is to implement an IEC 61850 communication interface to communicate with the RTDS in Sampled Values (to receive the current values) and GOOSE Messages (to send the indication of fault to the RTDS). To clarify:

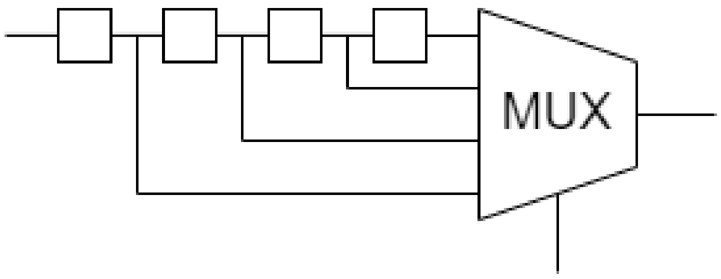


Figure 4.8: Delay implementation after FFT module for configuration 2, 3, and 4

- To receive IEC 61850 Sampled Values data packets from RTDS.
- To transmit IEC 61850 GOOSE Messages data packets to RTDS when the fault is detected.

Since IEC 61850 data frames are mapped to Ethernet frames, it is possible to receive SV data through the Ethernet port on the Zynq board.

4.11.1. Ethernet on Zynq

Zynq SoC uses the Gigabit Ethernet Controller (GEM) to achieve Ethernet connection at 10/100/1000 Mb/s in either full or half duplex operation. Recalling the data format presented in Chapter 2, the support of multi-casting and 802.1Q VLAN tagging is crucial for the feasibility of receiving data in IEC 61850 SV protocol. The multi-casting and VLAN tagging functionalities can be enabled and configured by setting the corresponding registers.

4.11.2. Gigabit Ethernet Controller (GEM)

Gigabit Ethernet Controller (GEM) is a part of the Zynq devices that implements an Ethernet MAC operating at 10/100/1000 Mbps. To operate in 100 Mbps, Either half-duplex mode or full-duplex mode is supported. However, the 1000 Mbps only supports full-duplex mode. Like many other ARM core controllers, the GEM can be controlled by assigning proper values to the corresponding registers. For programming the GEM on the Zynq device in bare-metal application to make it work, the following steps need to be followed according to Xilinx documents:

1. Initialize the controller.
2. Configure the controller based on the functions needed and the details on corresponding registers (set speed rate (100MHz), enable receiving multicast frames, setup multicast hash filtering, program the DMA configuration register and network control register to enable and set 802.1Q VLAN tagging, etc.)
3. I/O configuration. Configure the controller for MIO since the Ethernet port on the development board is hard-wired to the MIO of the SOC.
4. Configure the PHY(Ethernet physical layer). First, detect the PHY address. Second, configure the PHY, such as setting the mode.
5. Configure the buffer descriptors (BDs).
6. Configure interrupts. In this system, an interrupt condition is met when the PL detects a fault. When the interrupt occurs, the interrupt handler is called, which will send frames to RTDS in IEC 61580 GOOSE message to indicate the fault.
7. Enable the transmitter and the receiver.
8. Receiving Frames and transmitting frames.

Name	Description	Master	Slave
M_AXI_GP0	General Purpose	PS	PL
M_AXI_GP1	General Purpose	PS	PL
S_AXI_GP0	General Purpose	PL	PS
S_AXI_GP0	General Purpose	PL	PS
S_AXI_ACP	Accelerator Coherency Port for cache coherent transaction	PL	PS
S_AXI_HP0	High Performance with FIFOs	PL	PS
S_AXI_HP1	High Performance with FIFOs	PL	PS
S_AXI_HP2	High Performance with FIFOs	PL	PS
S_AXI_HP3	High Performance with FIFOs	PL	PS

Table 4.22: Interfaces between PS and PL

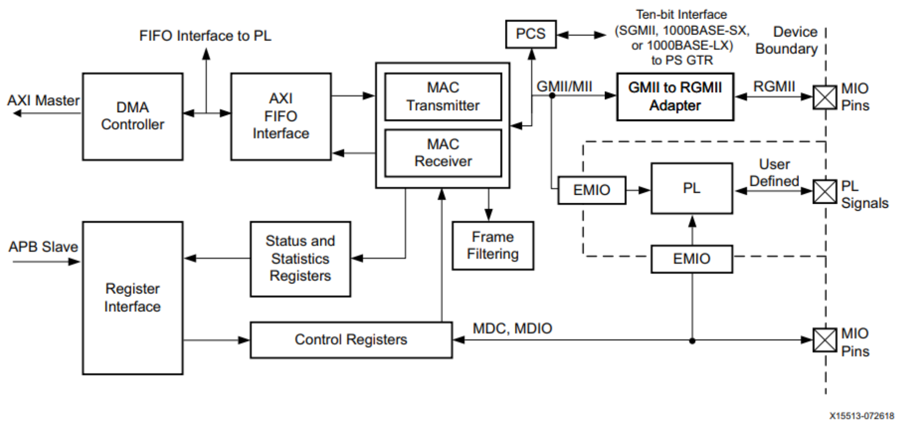


Figure 4.9: Block diagram of GEM

4.11.3. Interface between PS and PL

The values of the currents are received by Ethernet port controlled by PS. Therefore, the method of how the design on PL can access these data from PL needs to be introduced. Several interfaces are available for the data transmission between PS and PL, as listed in Table 4.22.

In Table 4.22, General Purpose ports provide maximum flexibility, and High-Performance ports provides maximum bandwidth. To avoid unwanted latency, High-Performance ports are preferable to be chosen.

4.11.4. Storing Data from Sampled Values Packets

After receiving Ethernet frames, PS can store the data (payload) in each frame in the DDR memory. Normally such storage uses the Direct Memory Access (DMA) controller of PS. However, GEM is mostly in use with the DMA of its own. The block diagram is shown in Figure 4.9 where it can be seen that there is a DMA within the GEM. If the storage is not needed, the GEM is also able to send the received data directly to PL with the FIFO interface. It is worth mention that, as the sampling rate of the RTDS is 4kHz (80 samples per cycle), and the sampling rate of the S-energy calculation algorithm is 250Hz [1] (5 samples per cycle), the data packets received between the two samples of the S-energy calculation algorithm will not be used. Apart from that, when a new value from the data packet is fed into the S-energy calculation algorithm on hardware, it needs to be transferred with the 15 values before itself (the input order is from old to new). Therefore, the values from the data packets needs to be stored in the DDR memory instead of being transferred directly into PL. Moreover, only every one of the 16 packets needs to be saved. So the software code on PS needs to count every 16 packets and send one to the DDR memory by DMA controller.

4.11.5. PL Accessing DDR Memory

PL can access the data in the DDR memory by AXI DMA IP. Due to the fact that the data received are going to be transferred into PL and used for computation in groups, these data need to be transferred in AXI-

Stream. Since the data from the Ethernet port are stored in memory, the transfer of data to PL needs to be on `S_AXIS_MM2S` slave memory map to stream ports. In this setting, data can be transferred from memory into a stream and fed to the s-Transform-based fault detection system on PL. The control of accessing data in the DDR memory from PL is taken care of PS using the DMA controller of the Zynq device.

After connecting PS and the s-energy calculation, the `M_AXIS_MM2S` via AXI DMA. The DMA itself is controlled by software running on PS. To configure the AXI DMA, the steps that need to be done are listed below:

1. Define the address within in the memory map for data transfer
2. Reset the DMA
3. Initiate the transfer
    - Memory address where the data is stored
    - Number of data to be transferred
    - The direction of transfer
4. Wait until the DMA finishes the transfer

In this case, the S-energy calculation module on PL can be fed with data from the corresponding memory address from the DDR memory. The data is transferred through `S_AXIS_MM2S` which means the data fed into PL is in a stream.

4.12. Conclusion

This chapter discusses the hardware implementation of the S-Transform-based fault detection system in detail. Based on the system architecture derived from the high-level functional flow, each module in the PL system are discussed and explained. Two versions of FFT modules (parallel pipelined and sequential pipelined) are discussed and compared, and the impact of using each of them on the whole system is introduced. The methodology of FFT is also presented in this chapter to understand the FFT architecture. The pipelining of the design is introduced to improve the speed and throughput. In the case of this design, the pipelining of the square root module is explained. Different configurations and their impacts on the whole system are introduced and discussed to explore the possibilities of fitting the PL design on a smaller FPGA.

The conceptual methods of the IEC 61850 implementation on the Zynq device are investigated. Due to the lower priority compared with the S-energy calculation algorithm, this communication part is not implemented in this project. However, the parts on a Zynq device that are necessary for the implementation to work are listed, and the steps need to be take on the PS side to build the connection and send the data to the PS are outlined.

5 Experimental Results

In this chapter, the experimental results for the hardware design are discussed. Topics covered in this chapter are:

- Experimental results for evaluation of transformation from floating-point numbers to fixed-point numbers together with the modifications of Gaussian window elements
- Experimental results for validating S-energy calculation module containing parallel pipelined FFT module
- Experimental results for validating S-energy calculation module containing sequential pipelined FFT module
- Experimental results for synthesis and resource and power estimation
- Experimental results for synthesis and resource and power estimation
- Experimental results for S-Transform-based fault detection system

5.1. Evaluation from Floating-point to Fixed-point Numbers

As described in section 3.4, the MATLAB software design of the proposed algorithm is in fixed-point numbers. It is decided to be transformed into fixed-point numbers to be implemented on the hardware design. The elements in the Gaussian window are also modified in order to save resources on the hardware. It needs to be evaluated whether the modified algorithm in fixed-point numbers can still achieve acceptable accuracy in detecting S-energy faults.

5.1.1. Experiment Setup

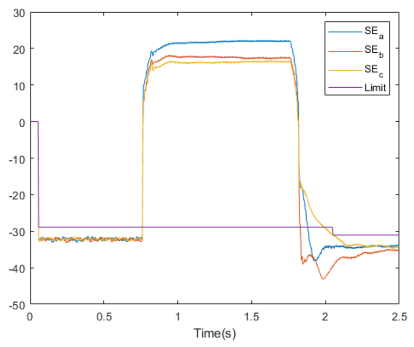
The feasibility of the fixed-point version of the proposed algorithm is validated in MATLAB. The original floating-point represented proposed S-energy calculation algorithm is converted into a fixed-point representation by the Fixed-Point Converter application. The representation of the generated fixed-point version is set to 16-point signed fixed-point numbers. And the bits for the integral part and fraction part are determined by the Fixed-Point Converter itself. This is also a reference for the bit representation in the later hardware design. The testbench is the same as the floating-point algorithm, which feeds the currents of three phases to calculate the S-energy and compares it with the threshold value.

The modification of the Gaussian window elements is also validated in this experiment. The elements in the Gaussian window with a value smaller than 0.0001 are replaced with 0. And the effect of this change on the general accuracy is observed.

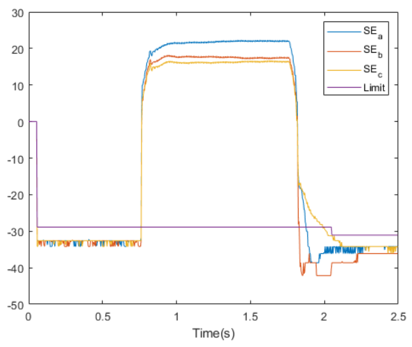
5.1.2. Experiment Results

The test results for the floating-point represented S-energy calculation algorithm is shown in Figure 5.1, where Figure 5.1b is the test results for the floating-point S-energy calculation algorithm, and Figure 5.1a is the test results for floating-point S-energy calculation algorithm.

From Figure 5.1a and Figure 5.1b it can be observed that:



(a) MATLAB test results for floating-point algorithm



(b) MATLAB test results for fixed-point algorithm

Figure 5.1: MATLAB test results for floating-point and fixed-point S-energy calculation algorithm

- The transformation from floating-point numbers to fixed-point numbers has little effect on the S-energy threshold calculation
- The most significant influence that such transformation occurs where the calculated S-energy value is below 33dB, which will not affect the accuracy of the fault determination

Based on these observations, the proposed S-energy calculation algorithm is valid after converting from floating-point numbers to 16-point signed fixed-point numbers. At the same time, replacing the values in the Gaussian window smaller than 0.0001 with 0 does not affect the accuracy. The algorithm still is able to detect the fault after the modification to the Gaussian window.

5.2. Validation for Parallel Pipelined FFT module

5.2.1. Experiment Setup

The input of the testbench of the FFT module is taken from the fixed-point values that are converted to 16-point fixed-point numbers by MATLAB Fixed-Point Converter. The input is phase A current signals in which bit 15 to bit 13 are integral, and bit 12 to bit 0 are fractional. The input is fed into the FFT module in groups of 16 in which each data is valid for one clock cycle, and each group is separated with a pause of 1000 ns. This is to emulate the behaviour that the current signals will not be fed into the hardware in every clock cycle.

5.2.2. Experiment Results

In order to speculate better on the results for the FFT module, the same set of 16 inputs for MATLAB testbench and VHDL testbench is listed and compared. The waveform of the simulation is shown in Figure 5.2.

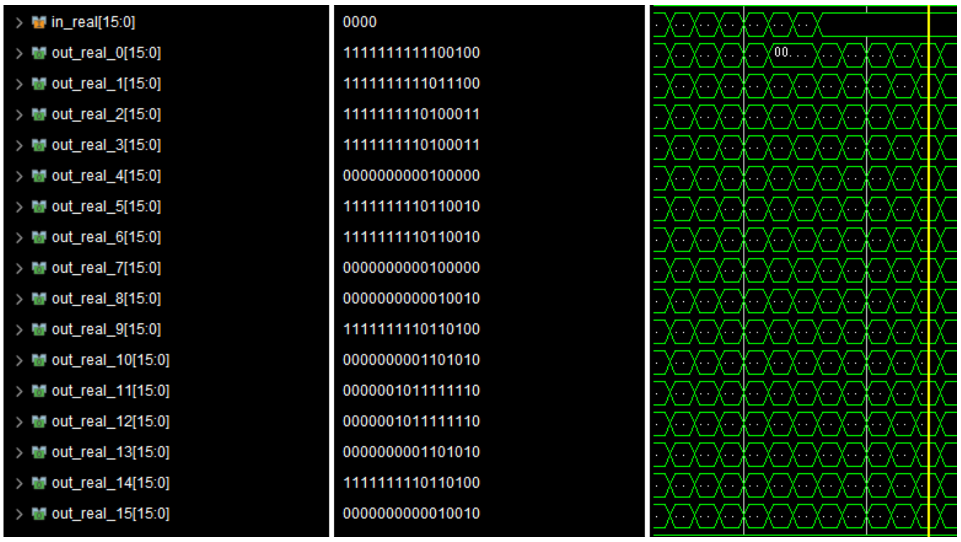
From Figure 5.2, it can be observed that the outputs of the FFT is available after five clock cycles of the last input of the FFT. At the first clock cycle after the last input of the FFT, the delay lines finish the conversion from sequential to parallel, and the output is ready after four more clock cycles.

The values of `dout_real` and `dout_imag`, as well as the results for corresponding variables in MATLAB are listed in Table 5.1 and Table 5.2. In Table 5.1 and Table 5.2, only the elements that are different in MATLAB simulation and VHDL simulation are listed to evaluate the accuracy of the FFT operation on hardware.

Index	MATLAB simulation	VHDL simulation
1	1111111111011011	1111111111011100
2	1111111110100010	1111111110100011
3	1111111110100010	1111111110100011
10	0000000001101011	0000000001101010
11	0000001011111111	0000001011111110
13	0000000001101011	0000000001101010

Table 5.1: Comparison of `dout_real` in MATLAB simulation and VHDL simulation for parallel pipelined FFT module

Table 5.1 and Table 5.2 show some minor differences between the results from the MATLAB simulation and VHDL simulation at the lower bits of the outputs because of the different behaviour of the rounding



```
in_real[15:0]       0000
out_real_0[15:0]    1111111111100100
out_real_1[15:0]    1111111111011100
out_real_2[15:0]    1111111110100011
out_real_3[15:0]    1111111110100011
out_real_4[15:0]    0000000000100000
out_real_5[15:0]    1111111110110010
out_real_6[15:0]    1111111110110010
out_real_7[15:0]    0000000000100000
out_real_8[15:0]    0000000000010010
out_real_9[15:0]    1111111110110100
out_real_10[15:0]   0000000001101010
out_real_11[15:0]   0000001011111110
out_real_12[15:0]   0000001011111110
out_real_13[15:0]   0000000001101010
out_real_14[15:0]   1111111110110100
out_real_15[15:0]   0000000000010010
```

Figure 5.2: Simulation results for pipelined FFT module

Index	MATLAB simulation	VHDL simulation
4	1111111111010010	1111111111010011
7	0000000000101110	0000000000101101
10	0000000000111110	0000000000111101
12	0000001110110010	0000001110110001
13	1111110001001111	1111110001010000
15	0000000000100111	0000000000100110

Table 5.2: Comparison of `dout_imag` in MATLAB simulation and VHDL simulation for parallel pipelined FFT module

methods. It can be observed that all of these differences occur at the least significant bit (LSB) of the value with a difference of 1, and it might affect the other bit due to the carrying. However, one at the LSB is a tolerable difference; therefore, the parallel pipelined FFT module can perform the FFT correctly.

5.3. Validation for Sequential Pipelined FFT Module

5.3.1. Experiment Setup

The input of the testbench of the FFT module is taken from the fixed-point values that are converted to 16-point fixed-point numbers by MATLAB Fixed-Point Converter. The input is phase A current signals in which bit 15 to bit 13 are the integral part, and bit 12 to bit 0 are the fractional part. The input is fed into the FFT module in groups of 16 in which each data is valid for one clock cycle, and each group is separated with a pause of 1000 ns. This is to emulate the behaviour that the current signals will not be fed into the hardware in every clock cycle.

5.3.2. Experiment Results

In order to speculate better on the results for the FFT module, the same set of 16 inputs for MATLAB testbench and VHDL testbench is listed and compared. The VHDL simulation results are shown in Figure 5.3.

where:

- `clk` and `rst` are the clock signal and the reset signal.
- `din_real` is the input series.
- `din_imag` is set to 0 in as there is no imaginary input.
- `dout_real` is the real part of the output series .
- `dout_imag` is the imaginary part of the output series T.

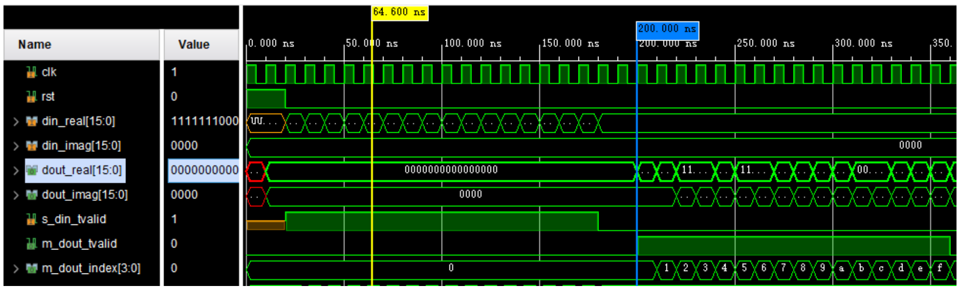


Figure 5.3: Simulation results for pipelined FFT module

- `m_dout_tvalid` is a signal indicating if the output of the FFT is valid.
- `s_din_tvalid` is a signal indicating if the data from from the input of FFT is valid.
- `m_dout_index` is a signal indicating the index of the elements from the output series.

The values of `dout_real` and `dout_imag`, as well as the results for corresponding variables in MATLAB are listed in Table 5.3 and Table 5.4. Similar to the evaluation for parallel pipelined FFT module, only the elements that are different in MATLAB simulation and VHDL simulation are listed in Table 5.3 and Table 5.4 to evaluate the accuracy of the FFT operation on hardware.

Index	MATLAB simulation	VHDL simulation
0	1111111111100100	1111111111100101
1	1111111111011011	1111111111011100
2	1111111110100010	1111111110100011
3	1111111110100010	1111111110100011
10	0000000001101011	0000000001101010
11	0000001011111111	0000001011111110

Table 5.3: Comparison of `dout_real` in MATLAB simulation and VHDL simulation for sequential pipelined FFT module

Index	MATLAB simulation	VHDL simulation
4	1111111111010010	1111111111010011
7	0000000000101110	0000000000101101
8	1111111111011001	1111111111011010
11	0000001110110010	0000001110110001
14	0000000000011100	0000000000011011
15	0000000000100111	0000000000100110

Table 5.4: Comparison of `dout_imag` in MATLAB simulation and VHDL simulation for sequential pipelined FFT module

From Table 5.3 and Table 5.4 similar differences between the results from the MATLAB simulation and VHDL simulation at the LSB of the outputs can be observed. The differences are still limited to the LSB, and some other bits are affected due to the carrying. It can be also observed that the FFT is started after the set of `s_din_tvalid` and the first input of `din_real`. The first element of the outputs `dout_real` and `dout_imag` is available after 28 clock cycles.

5.4. Validation for Gaussian window Module

5.4.1. Experiment Setup

The input of the testbench of the Gaussian window module is the output from the FFT module. The Gaussian window connected to the pipelined FFT module is evaluated here with the input data feed from the FFT module and the valid signal also from the FFT module. The output data is validated by comparing it to the MATLAB testbench. The verification for the Gaussian window module is selected to compare the outputs of the first Gaussian window (of eight in total).

5.4.2. Experiment Results

The results from the VHDL testbench for validating the Gaussian window module are shown in Figure 5.4.

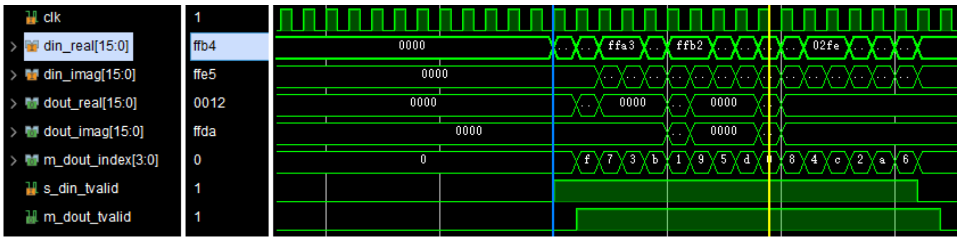


Figure 5.4: Simulation results for Gaussian window module

The values of `dout_real` and `dout_imag` are listed in Table 5.3 and Table 5.4. The index of the values are in natural sequence in the MATLAB simulation and it corresponds to the `m_dout_index` in the Gaussian window module. Note that the values that are not listed in Table 5.3 and Table 5.4 are the same in MATLAB and VHDL simulations.

Index	MATLAB simulation	VHDL simulation
15	1111111111111110	1111111111111111

Table 5.5: Comparison of `dout_real` in Gaussian window module in MATLAB simulation and VHDL simulation

Number	MATLAB simulation	VHDL simulation
0	1111111110011001	1111111110011010
1	1111111111111101	1111111111111110

Table 5.6: Comparison of `dout_imag` in Gaussian window module in MATLAB simulation and VHDL simulation

From Table 5.5 and Table 5.6 similar differences between the results from the MATLAB simulation and VHDL simulation at the LSB of the outputs can be observed. From the previous discussion, this difference does not affect the general accuracy. From Figure 5.4 it can be observed that the output of the Gaussian window module is available after one clock cycle of the input of the Gaussian window.

5.5. Validation for Parallel Pipelined IFFT Module

5.5.1. Experiment Setup

The input of the testbench of the parallel pipelined IFFT module is the output of the Gaussian window module from the last stage. The parallel pipelined IFFT module is evaluated here with the configuration of two Gaussian windows and IFFT modules to validate the behaviour of the parameterization of the design. The IFFT module is validated by comparing the outputs of the parallel pipelined IFFT module with the corresponding outputs of the MATLAB simulation.

5.5.2. Experiment Results

The results from the VHDL testbench for validating the parallel IFFT module are shown in Figure 5.5. From Figure 5.5 it can be observed that the output of the IFFT is available after five clock cycles of the input. It can also be observed that the different outputs are available for four clock cycles. This is caused by the specific configuration of the design (the one in this testbench corresponds to Configuration 3 in 4.21).

The values of `out_real_0` to `out_real_15` and `out_imag_0` to `out_imag_15` are compared with the corresponding results in the MATLAB simulation in fixed point representation. The deviations of the VHDL testbench and the MATLAB simulation are illustrated in Figure 5.6.

In Figure 5.6, `IFFT1` to `IFFT8` corresponds to the eight IFFT operations in the S-energy calculation algorithm, the 'Point' axis is the point of the IFFT operation, and the 'Deviation' axis is the absolute value of the differences between the results from VHDL testbench and MATLAB simulation. From Figure 5.6, it can be observed that the most significant deviation of the output of the IFFT module, in this case, is 8×10^{-8}, which is small enough for the conclusion that the parallel pipelined IFFT module performs the operations needed for the S-energy calculation algorithm.

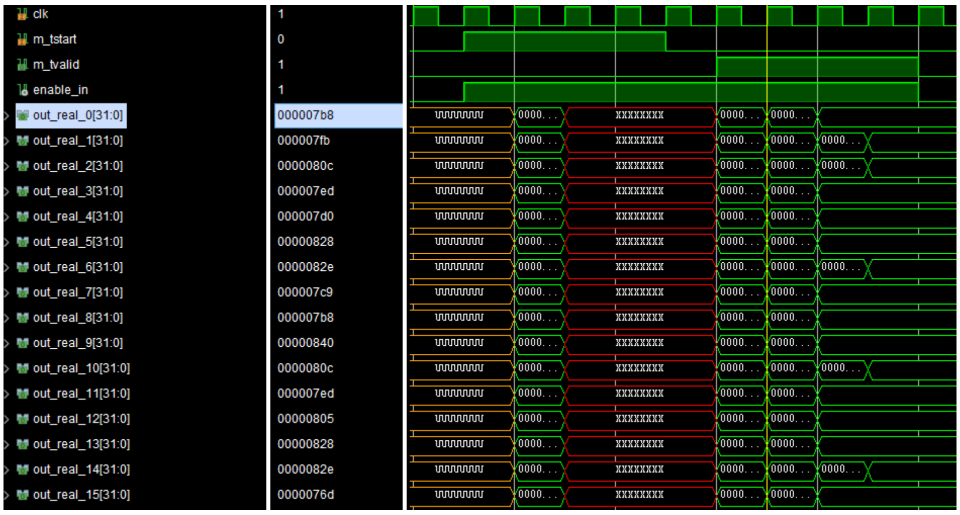

clk 1
m_tstart 0
m_tvalid 1
enable_in 1
out_real_0[31:0] 000007b8
out_real_1[31:0] 000007fb
out_real_2[31:0] 0000080c
out_real_3[31:0] 000007ed
out_real_4[31:0] 000007d0
out_real_5[31:0] 00000828
out_real_6[31:0] 0000082e
out_real_7[31:0] 000007c9
out_real_8[31:0] 000007b8
out_real_9[31:0] 00000840
out_real_10[31:0] 0000080c
out_real_11[31:0] 000007ed
out_real_12[31:0] 00000805
out_real_13[31:0] 00000828
out_real_14[31:0] 0000082e
out_real_15[31:0] 0000076d
UUUUUUUU
0000...
XXXXXXXX

Figure 5.5: Simulation results for parallel IFFT module

5.6. Validation for Square Root Module

5.6.1. Experiment Setup

The input of the testbench of the square root module is the output of the parallel pipelined IFFT module from the last stage. The square root module is evaluated here with the configuration of two Gaussian windows and IFFT modules to validate the behaviour of the parameterization of the design, as same as the evaluation in the last section for the parallel pipelined IFFT module. The square root module is validated by comparing the outputs of the square root module with the corresponding outputs of the MATLAB simulation.

5.6.2. Experiment Results

The results from the VHDL testbench for validating the sqaure root module are shown in Figure 5.7. From Figure 5.7 it can be observed that the output of the square root module is available after 16 clock cycles (160ns) of the input. It can also be observed that the different outputs are available for four clock cycles. The reason here is the same as it is for the parallel IFFT module as it is the simulation corresponding to Configuration 3 in 4.21.

The values of `dout` are compared with the corresponding results in the MATLAB simulation in fixed point representation. The deviations of the VHDL testbench and the MATLAB simulation are illustrated in Figure 5.6.

In Figure 5.8, `Square root 1` to `Square root 8` corresponds to the eight square root operations that are followed by each IFFT operation in the S-energy calculation algorithm, the 'Point' axis is the point of the IFFT operation, and the 'Deviation' axis is the absolute value of the differences between the results from VHDL testbench and MATLAB simulation. From Figure 5.8, it can be observed that the largest deviation of the output of the IFFT module, in this case, is 3×10^{-5}, which is small enough for the conclusion that the parallel pipelined IFFT module performs the operations needed for the S-energy calculation algorithm. This deviation is caused by the minimal input of the square root operation. At this case, the input is 6.52×10^{-9} (`00000000000000000000000000000111` in binary). The result from the square root operation in MATLAB is 9.16×10^{-5} (`0000000000000011` in binary), and the result from the square root module is 6.10×10^{-5} (`0000000000000010` in binary). Therefore, it can be observed that the deviation appears at the LSB of the result.

5.7. Synthesis and Resource Utilization

This section shows the synthesis results of each configuration of the system and compares them. The final choice of the hardware architecture of the S-energy calculation functionality can be made based on resource utilization and latency.

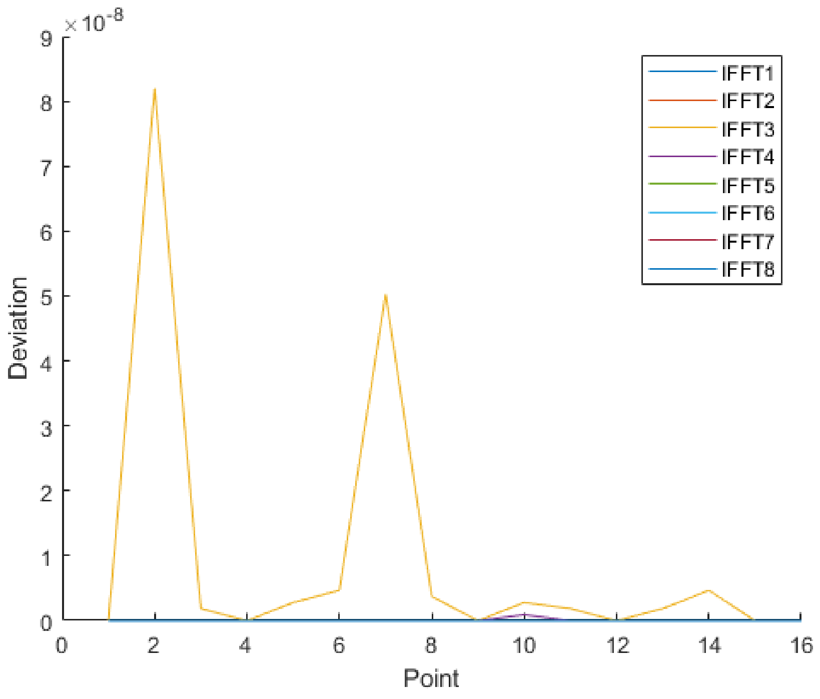


Figure 5.6: Deviations of the VHDL testbench and the MATLAB simulation for parallel pipelined IFFT module

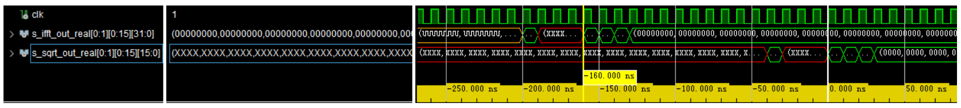


Figure 5.7: Simulation results for sqaure root module

5.7.1. Experiment Setup

In the experiment in this section, synthesis of the architectures and configurations mentioned in the previous chapters and sections are carried out. The resource utilization of each architecture is the key factor to be observed to gain more insights into the resource and area estimation of each configuration. The resource utilization estimation and the latency results contribute to the final decision of the hardware implementation of the S-energy calculation algorithm.

5.7.2. Experiment Results

The resource utilization of each configuration of the S-energy calculation hardware with parallel pipelined FFT and IFFT modules and the S-energy calculation hardware with sequential pipelined FFT and IFFT modules are shown in Table 5.7. In addition, the latency of each configuration is also listed to help with deciding which configuration to choose.

Configuration	LUT	LUTRAM	FF	DSP	Latency
1	81.80%	3.02%	13.89%	100.00%	490ns
2	28.91%	1.51%	7.85%	65.28%	510ns
3	18.39%	0.76%	4.84%	33.45%	530ns
4	13.53%	0.38%	3.64%	17.53%	570ns
Sequential	6.21%	0.19%	1.45%	12.27%	870ns

Table 5.7: Resource utilization estimation of each configuration

From Table 5.7, it can be observed that configuration 1 and 2 take too much DSP resources (configuration 1 even takes full DSP resources), which will affect other functionalities that might be needed on the same hardware in the future work. Therefore, it is not wise to choose those two configurations as the fi-

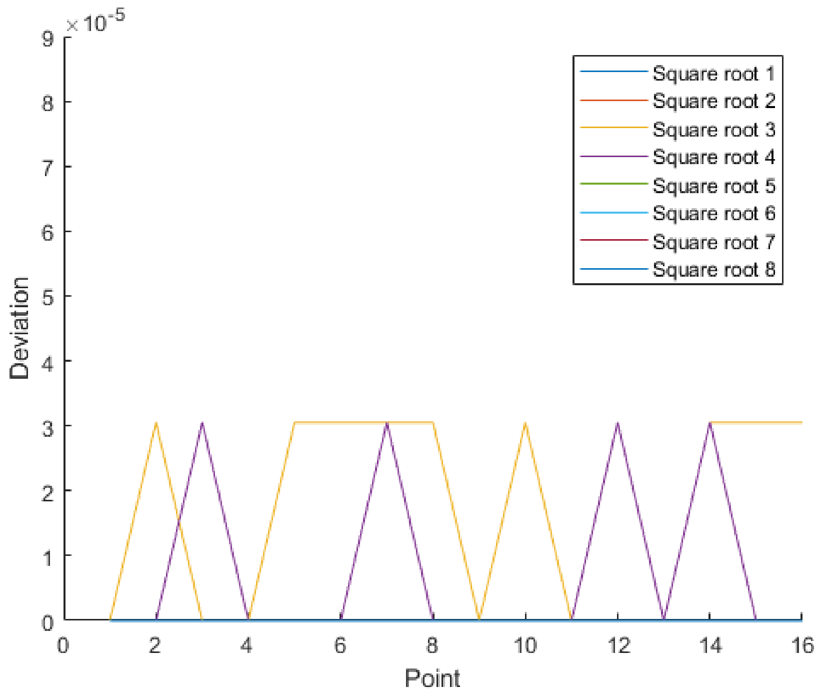


Figure 5.8: Deviations of the VHDL testbench and the MATLAB simulation for the square root module

nal implementation. Comparing the resource utilization of configuration 3 and 4, it can be observed that configuration 4 saves many resources in DSP (nearly half in this category). It can also be observed that the sequential pipelined configuration requires the least resource utilization. Table 5.7 makes it clear to decide on the choice of hardware architecture. There is only a 40ns (four clock cycles) difference in terms of latency between configuration 3 and configuration 4. Between configuration 4 and sequential FFT-based configuration, the difference in terms of resource estimation is slight, whereas the latency difference is slightly more significant compared to other configurations. Therefore, configuration 4 (one FFT module, one Gaussian window module, and one IFFT module) is chosen as the final architecture within the S-energy calculation architecture with parallel FFT and IFFT modules. After the choice is made, the implementation is run on Vivado for configuration 4. The power estimation of this configuration is 1.112W.

5.8. Validation for S-Transform-based fault detection system

5.8.1. Experiment Setup

In this experiment, the validation for the S-energy-based fault detection system implemented on hardware(PL) is simulated. The input data are three-phase current data from the MATLAB files. One is the current during a phase-to-ground (LG) fault, and the other is the current during a phase-to-phase (LL) fault. During the simulation, the calculated S-energy results from the hardware are recorded and compared with the expected outputs from the MATLAB simulation. During this process, the computed S-energy values from MATLAB simulation act as a benchmark to validate the functionality of the S-energy calculation and fault detection.

5.8.2. Experiment Results

The results for validating the S-energy calculation algorithm and fault detection in three phases during an LG fault are shown in Figure 5.9. In Figure 5.10, results for validating the S-energy calculation algorithm for each phase during an LG fault are presented. Figure 5.10a, Figure 5.10b, and Figure 5.10c show the comparison between the S-energy calculated by the hardware and their benchmarks for phase A, B, and C during an LG fault, respectively.

For the validation of the S-energy calculation algorithm and fault detection during an LL fault, the results for three phases are presented in Figure 5.11. The comparison between the S-energy calculated by the hardware and their benchmarks for phase A, B, and C during an LL fault are shown in Figure 5.12a, Figure 5.12b,

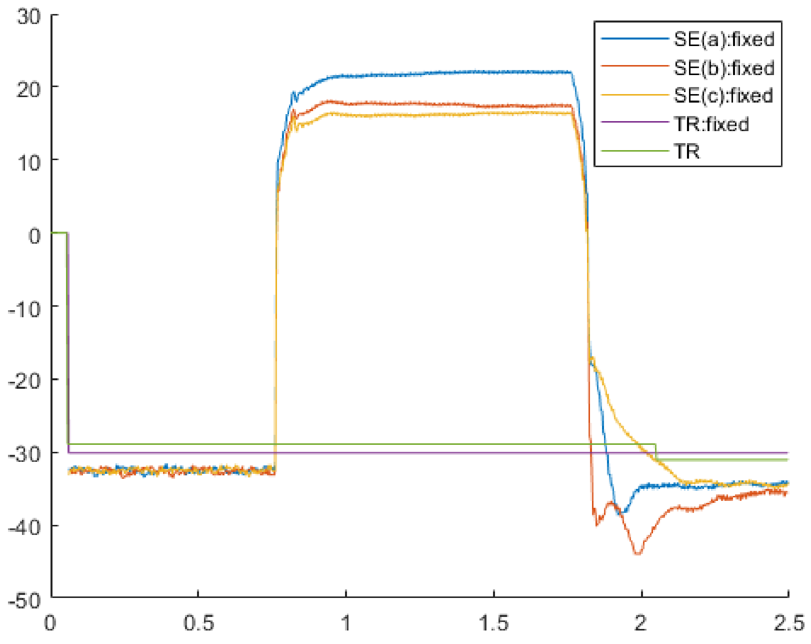


Figure 5.9: Validation for S-energy calculation algorithm and fault detection in three different phases during an LG fault

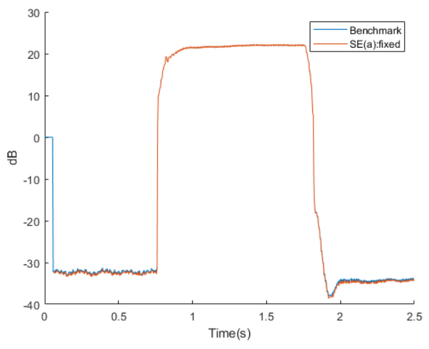
and Figure 5.12c.

Figure 5.10 and Figure 5.12 show that the S-energy calculation algorithm on hardware produces identical outputs to the MATLAB benchmarks during the LG fault and the LL fault. However, it can be observed that the S-energy threshold values calculated by hardware deviate from the results from MATLAB. The reason of the deviation in threshold voltages is that during the calculation of the S-threshold energy, the multiplication of the power of 2 is implemented by shifting a certain number of bits to the left. And during the process of determining the exact numbers to shift, the calculated result is rounded to the nearest integer. These two combined provide less sensitivity and accuracy. Based on the inception of the signals in the threshold determination module, the implemented threshold determination algorithm can provide identical thresholds in dB as MATLAB simulation. However, this accuracy is "eaten" by rounding up.

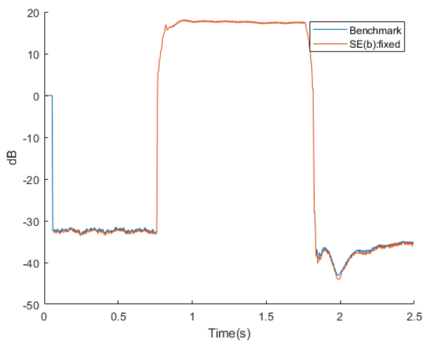
5.9. Conclusion

This chapter shows and discusses the experiments carried out in this project to evaluate and validate the hardware design for the S-energy calculation algorithm. The experiments are divided into two parts. Section 5.1, which explores the possibilities of representing the S-energy calculation algorithm in fixed-point numbers on hardware, is the experiment before the hardware design. After the confirmation of the feasibility of implementing the S-energy calculation algorithm on hardware, Section 5.2 to Section **??** are the experiments performed after the hardware design. In these sections, each key module of the hardware implementation of the S-energy calculation algorithm is simulated and validated. The inputs are taken from the sample data from the previous work. The outputs generated by the hardware design in the simulations are compared with the MATLAB testbenches. During these experiments, it can be frequently observed that due to the conversion from floating-point to fixed-point representation, the accuracy of the outputs are affected. However, these deviations mostly show at the LSBs of the signals. Therefore, the accuracy of the S-energy calculation algorithm, in general, is not much affected, and still maintains a tolerable range. Finally, the S-energy calculation and the fault detection are validated in Section 5.8, where the results show the S-Energy is calculated correctly on hardware and the S-transform-based fault detection system is able to recognize the fault.

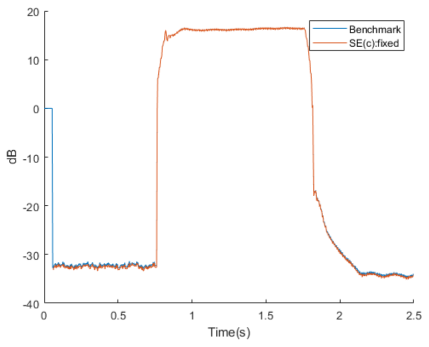
The resource utilization and the latency are evaluated and discussed in this chapter for different configurations of the hardware design of the S-energy calculation algorithm to determine which configuration to choose for the final S-Transform-based fault detection algorithm. Within the configurations evaluated, the configuration with one parallel FFT module, one Gaussian window module, one parallel IFFT module, and



(a) Verification for phase A S-energy calculation during an LG fault



(b) Verification for phase B S-energy calculation during an LG fault



(c) Verification for phase C S-energy calculation during an LG fault

Figure 5.10: Verification for S-energy-based fault detection system in each phase with its benchmark

other essential modules sits in a middle point with reasonable resource utilization and latency. This configuration can produce the S-energy result after 54 clock cycles (540ns under 100MHz clock), and the resource estimation is occupying 13.53% of LUT, 0.38% of LUTRAM, 3.64% of FF, and 17.53% of DSP. The final implementation on hardware with three S-energy modules (one for each phase) occupies 30.72% of LUT, 1.17% of LUTRAM, 11.61% of FF, and 53.24% of DSP. The power estimation of the final implementation is 2.219W.

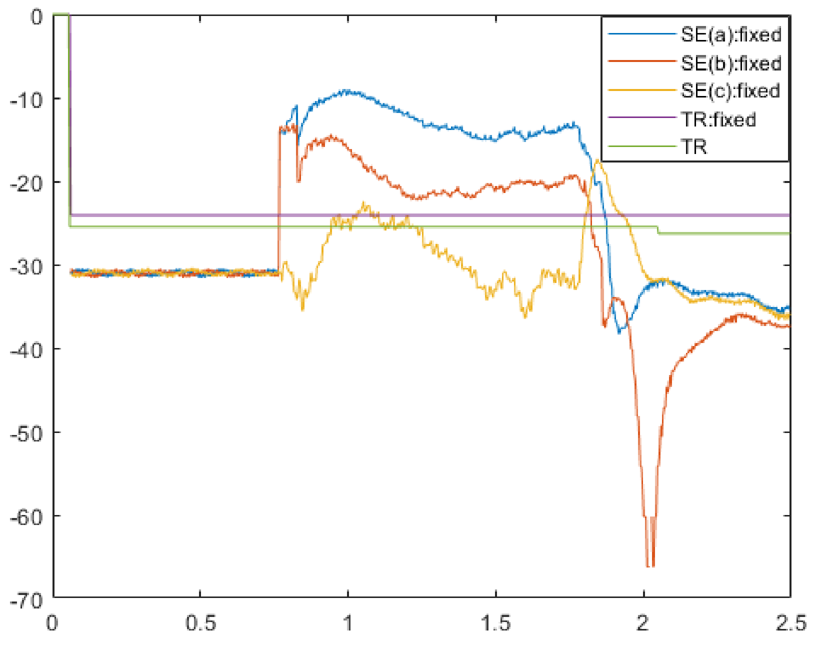
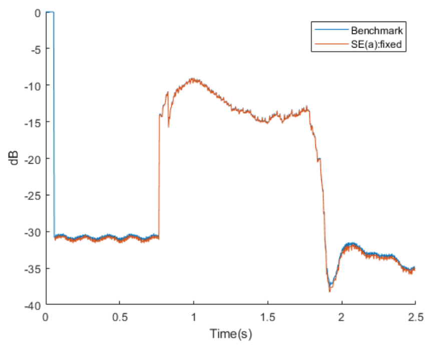
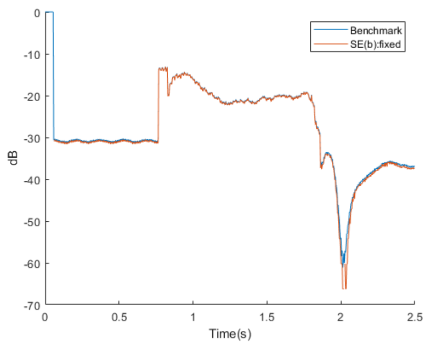


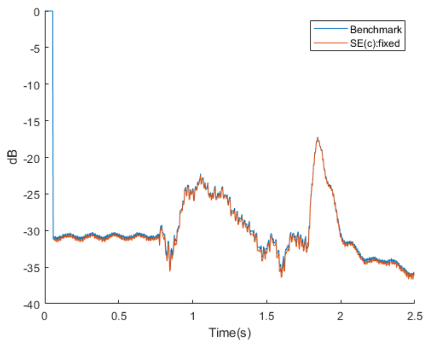
Figure 5.11: Verification for S-energy calculation algorithm and fault detection in three different phases during an LL fault



(a) Verification for phase A S-energy calculation during an LL fault



(b) Verification for phase B S-energy calculation during an LL fault



(c) Verification for phase C S-energy calculation during an LL fault

Figure 5.12: Verification for S-energy-based fault detection system in each phase with its benchmark

6
Conclusion and Future Work

6.1. Summary

In this thesis, the hardware implementation of the S-energy calculation algorithm used in an S-transform Based Fault Detection for distance protection is proposed and discussed. In Chapter 1, the goal of this project is stated as to propose a hardware implementation for the S-energy calculation algorithm used in S-Transform based fault detection algorithm. The high-level design methodology is also presented in Chapter 1. In Chapter 2, the related background is introduced. The previous work of the development of S-Transform-based fault detection algorithm is brief introduced, as well as the methodology of the FDST and the threshold value determination. The data format of IEC 61850 Sampled Values and GOOSE Messages are listed and discussed, which are mapped to ISO 8802-3 Ethernet frames. This provides a possibility of implementing the IEC 61850 communication interface on an Ethernet interface.

In Chapter 3, design choices such as high-level functional requirements, devices choice, and fixed-point representation are determined. In order to meet the design requirements derived from the characteristics of the project, the hardware design is chosen to be implemented on Xilinx ZCU 104 MPSoC development board. The high-level functional flow proposed from the design requirements provides guidance to the hardware architecture design. Following this, the hardware implementation can be divided into two parts, S-energy calculation hardware design on PL, and IEC 61850 communication interface on PS, namely. In order to make sure that it is possible to efficiently implement the S-energy calculation algorithm on hardware in a fixed-point representation, the transformation from floating-point numbers to fixed-point numbers is explored. The methodology to optimize the Gaussian window applied in the originally proposed S-energy calculation algorithm is also discussed. At the end of Chapter 3, an architecture for the S-Transform-based fault detection algorithm is illustrated.

In Chapter 4, the hardware design of the S-energy calculation algorithm is proposed. The core function of the FDST is FFT, which is carefully designed in this hardware architecture. Both parallel pipelined FFT module and sequential pipelined FFT module are developed on hardware for the S-energy calculation algorithm. To reduce the significant increment of the hardware utilization brought by the recursive IFFT operations, different configurations of the system implementing parallel pipelined FFT and IFFT modules are developed. Pipelining on hardware is applied to avoid the set-up time violation. The methods of programming on the PS in order to make it work as an IEC 61850 communication interface with MAC address filtering, as well as the ways to transfer the receiving data into the DDR memory for the S-energy calculation algorithm on hardware to fetch is also introduced.

In Chapter 5, experiments are carried out to validate the hardware design. Before the hardware design, the experiments of converting the floating-point numbers to fixed-point numbers are discussed to ensure the possibility to implement the fixed-point S-energy calculation algorithm on hardware. After the hardware design, simulations are carried out for validating the functionalities of each modules and the whole system on PL. After the validation, all of the different configurations using parallel pipelined FFT and IFFT modules, as well as the architecture using sequential pipelined FFT and IFFT modules are able to produce the desired output as S-energy. By comparing the resource utilization estimation from synthesis and the latency from simulation, the area-latency trade-off finally contributes to the conclusion of the choice of the configuration with one parallel FFT module, one Gaussian window module, one IFFT module, and other essential parts as the final S-energy calculation hardware implementation on PL. Unfortunately, due to the lack of time with the development board, the IEC 61850 communication interface is not successfully implemented on PS.

6.2. Main Contributions

The goals for this work is described in Section 1.1. To revisit the goals:

1. Evaluate the possibilities of implementing the proposed algorithm on hardware
2. Propose a hardware implementation for the S-energy calculation algorithm used in the proposed S-Transform-based fault detection method
3. Implement the IEC 61850 communication interface on hardware and establish a data path between it and the S-energy calculation algorithm

In order to meet the goals in 1.1, first the evaluation for the possibilities of transferring the proposed S-energy calculation in MATLAB to hardware is carried out. This is done by converting the floating-point algorithm to fixed-point representation, and then running the same testbench to compare the behaviour and accuracy. The motivation of this is that fixed-point representation saves much resource utilization comparing to floating-point representation. Once the accuracy requirements are met, the next step is to 'translate' the S-energy in software to a possible hardware implementation. During this process, the word 'translate' means to adjust the proposed S-energy calculation algorithm into a hardware design considering the algorithm operates in sequence in software, whereas, the operations on hardware is in parallel. The possible modifications for the S-energy calculation algorithm are also performed at this time to further reduce the resource utilization.

To translate the proposed S-energy calculation algorithm in software to hardware implementation, the algorithm is broken down from high to low level. The high level block view provides a good understanding of the necessary modules to perform different operations in hardware and the form of the input and output data. The high-level architecture of the hardware design is also derived at this point. With the further breakdown of the algorithm, the architecture of each sub-module can also be derived. Possible modifications are performed to make the proposed S-energy calculation algorithm fit on a hardware design. The proposed S-energy calculation algorithm is firstly implemented on hardware in a straight way. Afterwards, possible improvements are implemented, such as pipelining, and paramitization to make the proposed S-energy calculation algorithm fit better on hardware and provide some flexibility.

At the same time, the hardware implementation of the calculation of the S-energy threshold value is also developed for to be compared with the calculated S-energy by the implemented algorithm. The outcome of such comparison is the result for whether there is a fault in currents.

Alongside with the S-energy calculation algorithm on hardware, the concecptual methods of making a IEC 61850 communication interface are investigated. This work focuses on how to receive the current values with IEC 61850 standard in Sampled Values, and send the 'yes' or 'no' message indicating if there is a fault with IEC 61850 standard in GOOSE Messages. IEC 61850 communication standard defines the exchange of information and configuration in substations for the purpose of monitoring, controlling, and protection, etc. It provides the possibilities to transfer the information in substations in a digital format. In this work, the algorithm implemented on hardware receives the current values in Sampled Values, which reduce the complexity of the system since the Analog-to-Digital Converter (ADC) is no longer needed. The IEC 61850 formatted messages are mapped to ISO 8802-3 Ethernet frames. This leads to a possible implementation over a Ethernet communication interface to achieve the data transmitting the receiving for IEC 61850 communication.

After the hardware implementation of the S-energy calculation is developed, several experiments are carried out to validate the hardware design. The main focus of these experiments is to check whether the hardware implementation of the S-energy calculation algorithm performs the same operation (i.e., generates the same output) as the converted fixed-point version of the proposed S-energy calculation algorithm. The submodules in the hardware design is validated first to ensure it gives the expecting results to perform the operations in the whold hardware system. After each sub-module are validated. The top-level validation focuses on if the S-energy calculation algorithm on hardware can provide the 'yes' or 'no' result to detect the fault in currents. The hardware implementation is validated in the same dataset in MATLAB testbenches in order to validate the functionality of the S-energy algorithm implementation.

Therefore, the major contributions of this work are:

- Evaluated and confirm the possibilities to convert the S-Energy calculation algorithm from floating-point to fixed-point representation.
- Developed a hardware design for S-Energy-based fault detection algorithm
- Validated the proposed hardware design with the dataset from MATLAB testbenches
- Investigated the possibilities and conceptual methods of implementing an IEC 61850 communication interface on the same hardware

This means the goals 1 and 2 are successfully achieved. The resource utilization of the developed S-Transform-based fault detection system occupies 30.72% of LUT, 1.17% of LUTRAM, 11.61% of FF, and 53.24% of DSP. The power estimation of the implemented system is 2.219W. However, goal 3 is not completed. The possibilities for achieving goal 3 is positive based on the investigations carried out.

6.3. Future Work

As this work is part of the complete work of the enhanced relay hardware implementation, the future work may also include implementing other functionalities from the protective relays, such as phase selection, direction determination, zone determination, etc [1].

Another possibility to continue this work is to continue working on the hardware implementation of the IEC 61850 communication interface. It is investigated in this work but not implemented. The investigation shows the Zynq devices with Ethernet ports have the potential to provide the functionality for IEC 61850 communication.

Bibliography

[1] Jose J Chavez et al. “S-Transform based fault detection algorithm for enhancing distance protection performance”. In: *International Journal of Electrical Power & Energy Systems* 130 (2021), p. 106966.

[2] Edmund O Schweitzer III and Jeff Roberts. “Distance relay element design”. In: *proceedings of the 46th Annual Conference for Protective Relay Engineers, College Station, TX*. 1993.

[3] Siniša J Zubić and Milenko B Djurić. “A distance relay algorithm based on the phase comparison principle”. In: *Electric Power Systems Research* 92 (2012), pp. 20–28.

[4] M Ramamoorty. “Application of digital computers to power system protection”. In: *Journal of Inst. Eng.(India)* 52.10 (1972), pp. 235–238.

[5] A Gómez Expósito, JA Rosendo Macias, and JL Ruis Macias. “Discrete Fourier transform computation for digital relaying”. In: *International Journal of Electrical Power & Energy Systems* 16.4 (1994), pp. 229–233.

[6] Peter Grant McLaren and MA Redfern. “Fourier-series techniques applied to distance protection”. In: *Proceedings of the Institution of Electrical Engineers*. Vol. 122. 11. IET. 1975, pp. 1301–1305.

[7] MS Sachdev and M Nagpal. “A recursive least error squares algorithm for power system relaying and measurement applications”. In: *IEEE Transactions on Power Delivery* 6.3 (1991), pp. 1008–1015.

[8] AP Morais et al. “Numerical distance relaying algorithm based on Mathematical Morphology and Least-Squares Curve Fitting method”. In: *Electric power systems research* 81.5 (2011), pp. 1144–1150.

[9] AS AlFuhaid and MA El-Sayed. “A recursive least-squares digital distance relaying algorithm”. In: *IEEE Transactions on Power Delivery* 14.4 (1999), pp. 1257–1262.

[10] V Terzija, M Djuric, and B Kovacevic. “A new self-tuning algorithm for the frequency estimation of distorted signals”. In: *IEEE Transactions on Power Delivery* 10.4 (1995), pp. 1779–1785.

[11] Francisco Martin and Jose A Aguado. “Wavelet-based ANN approach for transmission line protection”. In: *IEEE Transactions on power delivery* 18.4 (2003), pp. 1572–1574.

[12] AH Osman and OP Malik. “Transmission line distance protection based on wavelet transform”. In: *IEEE Transactions on Power Delivery* 19.2 (2004), pp. 515–523.

[13] D Chanda, NK Kishore, and AK Sinha. “A wavelet multiresolution analysis for location of faults on transmission lines”. In: *International journal of electrical power & energy systems* 25.1 (2003), pp. 59–69.

[14] Kleber M Silva, Washington LA Neves, and Benemar A Souza. “Distance protection using a wavelet-based filtering algorithm”. In: *Electric Power Systems Research* 80.1 (2010), pp. 84–90.

[15] Hong-Seok Song and Kwanghee Nam. “Dual current control scheme for PWM converter under unbalanced input voltage conditions”. In: *IEEE transactions on industrial electronics* 46.5 (1999), pp. 953–959.

[16] Chong Ng, Li Ran, and Jim Bumby. “Unbalanced grid fault ride-through control for a wind turbine inverter”. In: *2007 IEEE Industry Applications Annual Meeting*. IEEE. 2007, pp. 154–164.

[17] Jeff Roberts and Armando Guzman. “Directional element design and evaluation”. In: *proceedings of the 21st Annual Western Protective Relay Conference, Spokane, WA*. Citeseer. 1994.

[18] Donald D Fentie. “Understanding the dynamic mho distance characteristic”. In: *2016 69th Annual Conference for Protective Relay Engineers (CPRE)*. IEEE. 2016, pp. 1–15.

[19] Edmund O Schweitzer III. “New developments in distance relay polarization and fault type selection”. In: *16th Annual Western Protective Relay Conference*. 1989, pp. 24–26.

[20] Yanwei Wang and Jeff Orchard. “Fast Discrete Orthonormal Stockwell Transform”. In: *SIAM Journal on Scientific Computing* 31.5 (2009), pp. 4000–4012. DOI: 10.1137/080737113.

[21] KR Krishnanand and PK Dash. “A new real-time fast discrete S-transform for cross-differential protection of shunt-compensated power systems”. In: *IEEE Transactions on Power Delivery* 28.1 (2012), pp. 402–410.

[22] C Robert Pinnegar and Lalu Mansinha. “Time-local Fourier analysis with a scalable, phase-modulated analyzing function: the S-transform with a complex window”. In: *Signal Processing* 84.7 (2004), pp. 1167–1176.

[23] Pengfei Qi and Yanchun Wang. “Seismic time–frequency spectrum analysis based on local polynomial Fourier transform”. In: *Acta Geophysica* 68.1 (2020), pp. 1–17.

[24] Zhang Sen, Wang Jiulong, and Yu Dengyun. “Frequency estimation using nonlinear least squares and S-transform”. In: *2010 3rd International Congress on Image and Signal Processing*. Vol. 1. IEEE. 2010, pp. 138–142.

[25] Carine Simon et al. “The S-transform and its inverses: Side effects of discretizing and filtering”. In: *IEEE transactions on signal processing* 55.10 (2007), pp. 4928–4937.

[26] PK Dash, BK Panigrahi, and G Panda. “Power quality analysis using S-transform”. In: *IEEE transactions on power delivery* 18.2 (2003), pp. 406–411.

[27] João F Martins et al. “The application of S-transform in fault detection and diagnosis of grid-connected power inverters”. In: *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE. 2012, pp. 5247–5252.

[28] Aditya A Lokhande. “A survey on S-transform”. In: *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)* (2017), pp. 35–39.

[29] R.G. Stockwell, L. Mansinha, and R.P. Lowe. “Localization of the complex spectrum: the S transform”. In: *IEEE Transactions on Signal Processing* 44.4 (1996), pp. 998–1001. DOI: 10.1109/78.492555.

[30] Lingmei Ai, Jiaofen Nan, and Li hua Pu. “Application of S-transform in EEG analysis”. In: *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*. Vol. 2. IEEE. 2010, pp. V2–453.

[31] Mitsuo Takeda and Kazuhiro Mutoh. “Fourier transform profilometry for the automatic measurement of 3-D object shapes”. In: *Applied optics* 22.24 (1983), pp. 3977–3982.

[32] H Zhu et al. “Space-local spectral texture map based on MR images of MS patients”. In: *MS: Clin. Lab. Res* (2001).

[33] Ralph E Mackiewicz. “Overview of IEC 61850 and Benefits”. In: *2006 IEEE Power Engineering Society General Meeting*. IEEE. 2006, 8–pp.

[34] David R Coelho. *The VHDL handbook*. Springer Science & Business Media, 2012.

[35] “IEEE Standard for VHDL Language Reference Manual”. In: *IEEE Std 1076-2019* (2019), pp. 1–673. DOI: 10.1109/IEEESTD.2019.8938196.

[36] Juan J Rodriguez-Andina, Maria J Moure, and Maria D Valdes. “Features, design tools, and application domains of FPGAs”. In: *IEEE Transactions on Industrial Electronics* 54.4 (2007), pp. 1810–1823.

[37] Harry D Foster. “2018 FPGA functional verification trends”. In: *2018 19th International Workshop on Microprocessor and SOC Test and Verification (MTV)*. IEEE. 2018, pp. 40–45.

[38] Jie Jiang, Daoyin Yu, and Zheng Sun. “Real-time image processing system based on FPGA for electronic endoscope”. In: *IEEE APCCAS 2000. 2000 IEEE Asia-Pacific Conference on Circuits and Systems. Electronic Communication Systems.(Cat. No. 00EX394)*. IEEE. 2000, pp. 682–685.

[39] Hoang Nguyen, Johnson I Agbinya, and John Devlin. “FPGA-based implementation of multiple modes in near field inductive communication using frequency splitting and MIMO configuration”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 62.1 (2014), pp. 302–310.

[40] Jassim M Abdul-Jabbar and Sara W Abboodi. “Fpga implementation of modified fractional wavelet transforms”. In: *2013 International Conference on Electrical Communication, Computer, Power, and Control Engineering (ICECCPCE)*. IEEE. 2013, pp. 95–100.

[41] Umar Alqasemi et al. “FPGA-based reconfigurable processor for ultrafast interlaced ultrasound and photoacoustic imaging”. In: *IEEE transactions on ultrasonics, ferroelectrics, and frequency control* 59.7 (2012), pp. 1344–1353.

[42] H. den Boer et al. “FPGA-based Deep Learning Accelerator for RF Applications”. In: *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*. 2021, pp. 751–756. DOI: 10.1109/MILCOM52596.2021.9652891.

[43] M Ben Said et al. “Standard FPGA-based or full cSoC controllers for three-phase PWM boost rectifier, two viable solutions”. In: *2012 15th International Power Electronics and Motion Control Conference (EPE/PEMC)*. IEEE. 2012, DS2c–16.

[44] Slim Ben Othman et al. “MPSoC design approach of FPGA-based controller for induction motor drive”. In: *2012 IEEE International Conference on Industrial Technology*. IEEE. 2012, pp. 134–139.

[45] “IEEE Standard for Floating-Point Arithmetic”. In: *IEEE Std 754-2008* (2008), pp. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.

[46] James W Cooley and John W Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[47] Yamin Li and Wanming Chu. “A new non-restoring square root algorithm and its VLSI implementations”. In: *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*. IEEE. 1996, pp. 538–544.

[48] Eduardo Boemo. “Pipelining on FPGAs: A Tutorial”. In: *2019 X Southern Conference on Programmable Logic (SPL)*. IEEE. 2019, pp. 53–60.