FPGA Based DFT System Design, Optimization and Implementation Using High-Level Synthesis

Shensheng Tang*[1], Monali Sinare[2], Yi Xie[3]

[1]Department of Electrical and Computer Engineering, St. Cloud State University, St. Cloud, MN 56301, USA; stang@stcloudstate.edu

[2]Department of Electrical and Computer Engineering, St. Cloud State University, St. Cloud, MN 56301, USA; mksinare@go.stcloudstate.edu

[3]School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou 510006, China; xieyi5@mail.sysu.edu.cn

*Corresponding author

Abstract: In this paper, a discrete Fourier transform (DFT) algorithm is designed and optimized for the FPGA implementation using the Xilinx VIVADO High-Level Synthesis (HLS) tool. The DFT algorithm is written by C++ programming and simulated for functional verification in the HLS and MATLAB. For hardware validation, the DFT module is packaged as an IP core and tested in a VIVADO project. A Xilinx SDK application written by C language is developed and used for testing the DFT module on a Zynq FPGA development board, ZedBoard. For visualization of the DFT magnitude spectrum generated in FPGA, a GUI is developed by C# programming and related commands/data can be communicated between the GUI and ZedBoard over the serial port. Experimental results are presented with discussion. The DFT module design, optimization and implementation as well as the VIVADO project development methods can be extended to other FPGA applications.

Keywords: FPGA; DFT; IP core; VIVADO HLS; C/C++; Verilog; C#; Optimization; Hardware Validation.

Reference to this paper should be made as follows: Tang, S., Sinare, M., and Xie, Y. (20xx) 'FPGA Based DFT System Design, Optimization and Implementation Using High-Level Synthesis', Int. J. Computer Applications in Technology, Vol. xx, No. xx, pp.xxx–xxx.

Biographical notes: Shensheng Tang is an Associate Professor in the Department of Electrical and Computer Engineering in St. Cloud State University, USA. He received his Ph.D. from University of Toledo, USA. He has eight years of product design and development experience, as hardware engineer, system engineer, and manager respectively, in the electronics and wireless industry. His current research interests include embedded systems, networking (wireless, wired), Internet of things (IoT), and modelling and performance evaluation. He has served or is serving as editor or guest editor for international journals and technical program committee (TPC) member of international conferences. He has produced about 100 peer-reviewed publications in the above areas. He is a senior member of IEEE.

Monali Sinare is a student at the Department of Electrical and Computer Engineering, St. Cloud State University, Minnesota, pursuing a Master of Science in Electrical Engineering. She has received a Master of Science degree in Electronics Science from Savitribai Phule Pune University, Pune, Maharashtra, India, in 2007. She has nine years of experience in FPGA based control system design and development. She has worked on the design and development of Verilog and VHDL modules, testing, and integration of various submodules for FPGA based systems designed for control and monitoring of medical instruments. Her research interests include digital signal processing, image processing, high-level synthesis, hardware-software co-design, and embedded systems.

Dr. Yi Xie is currently an associate professor at the School of Computer Science and Engineering, Sun Yat-sen University. He received the B.S., M.E. and Ph.D. degrees from Sun Yat-sen University, Guangzhou, China. He was a visiting scholar at George Mason University and Deakin University during 2007 to 2008, and 2014 to 2015, respectively. He won the outstanding doctoral dissertation award of the Chinese Computer Federation (CCF) in 2009. His recent research interests include networking, cyber security and behavior modeling. Some of his works have been published in IEEE top journals, such as ToN, TPDS, TBD, TCSS and Sensors. He has received eight research grants and has served as a young Associate Editor for a Springer journal named Frontiers of Computer Science.

1 Introduction

Parallel processing in case of designs involving complex computation and processing of large datasets can improve the performance of the design with respect to speed of the computation. FPGAs (Field-Programmable Gate Arrays) [1] provide large scale programmable logic arrays which can be used for parallel programming. The FPGA techniques have been applied to many areas such as electronic circuit implementation [2] and telecommunication systems [3]. Moreover, the availability of high-level synthesis (HLS) tools [4] makes it easier to develop designs targeted for FPGAs using high level programming languages such as C/C++. Discrete Fourier transform (DFT) [5] is one of the most powerful tools in a large number of fields such as spectrum analysis [6]-[9], fast convolution [10]-[12], data compression [13]-[15], polynomial multiplication [16][17], and matrix multiplication [18][19].

Spectrum analysis is an important DFT application for practical computation of the frequency content of real-world signals. In [6], algorithms for the nonuniform-time discrete Fourier transform (NUT-DFT) were developed and evaluated for taking nonuniform-time domain signals and producing a uniform sampled spectrum. In [7], a fast Fourier transform (FFT) was used in the spectral analysis of Electroencephalography (EEG) signals for the detection of alpha rhythm in subjects with open and closed eyes. In [8], the robustness of the warped discrete Fourier transform (WDFT)-based cepstral features was investigated for continuous speech recognition with the speech spectrum warped using the Mel-scale filterbank. In [9], the sliding window algorithm was applied in computing the discrete time fractional Fourier transform (DTFrFT) and the hopping DTFrFT algorithm was proposed to obtain a continuous fractional spectrum.

DFT has been considered by the machine learning community to be a natural approach to fast convolution [10]. In [11], an algorithm of computing convolutions using discrete Fourier transforms was presented to accelerate training a large convolutional network by a significant factor compared to existing state-of-the-art implementations. In [12], spectral representations were employed to model and train convolutional neural networks (CNNs).

In [13], a block-encoding method based on a windowed Discrete Fourier Transform (DFT) was proposed to improve the coding efficiency for the transfer of compressed data in sensor networks. In [14], Stochastic Compressive Data Aggregation (S-CDA) was used for wireless sensor networks under random deployment. The random deployment was modelled by the Poisson point process and the signal recovery was based on the random discrete Fourier transform (RDFT) which reveals the frequency content of smooth signals, such as temperature or humidity maps. In [15], an algorithm of adaptive bit encoding was proposed for electrocardiogram (ECG) data compression using the conventional discrete Fourier transform. The simplicity and low cost infrastructural requirement of the algorithm makes it suitable for implementation on an embedded platform to be used in mobile devices.

In [16], a truncated version of the classical Fast Fourier Transform was presented to be used for polynomial multiplication and the multiplication of multivariate polynomials, and a logarithmic factor was gained with respect to previously known algorithms. In [17], the Fast Fourier Transform (FFT) with a linearithmic complexity of $O(n\,logn)$ was exploited in the design of a high-speed polynomial multiplier. A constant geometry FFT datapath was used in the computation to simplify the control of the architecture. In addition, some generalized discrete Fourier transforms are studied in matrix multiplication. In [18], one-dimensional and two-dimensional generalized discrete Fourier transforms were shown to require the same number of operations to be computed on a vector and matrix if the vector is fractured into the matrix. In [19], a unified approach of the generalized discrete Fourier transform (GDFT) matrices was investigated for the computation of family of discrete sinusoidal transforms.

In this paper, a DFT module is designed and optimized for FPGA using Xilinx VIVADO HLS tool [20] as well as implemented in the HLS for Xilinx Zynq-7020 SoC (System on Chip) device [21]. The DFT algorithm written by C++ programming is simulated for functional verification in the HLS and packaged as an IP (Intellectual Property) core for hardware validation in a VIVADO project (via VIVADO Design Suite [22]) and tested on a Zynq FPGA development board, ZedBoard [23]. For visualization of the DFT magnitude spectrum generated in the FPGA, a graphical user interface (GUI) is designed by C# programming through Visual Studio IDE [24] for sending commands to the ZedBoard and receiving the processed DFT data from it over a UART (Universal Asynchronous Receiver Transmitter) serial port. The Xilinx DDS compiler IP core [25] is modified along with custom logic to generate the sinusoidal data samples. The main contribution of the paper is detailed as follows:

- Design and develop a DFT module using VIVADO HLS with the algorithm written by C++ code.
- Simulate and verify the functional behaviour of the DFT module using data generated from MATLAB [26].
- Optimize the DFT design by applying various HLS directives, and perform synthesis and C/RTL Co-simulation for functional verification as well as package the design as a DFT IP (ready to be used at a VIVADO project).
- Design and develop a VIVADO project to test the developed DFT IP core.

- Develop other IPs (DDS IP, FIFO IP and Tlast generator IP) for the VIVADO project to make it as a complete testing system.
- Develop a Xilinx SDK (Software Development Kit) [27] application program by C language, which works with the hardware design created with VIVADO Design Suite.
- Design and develop a C# GUI to test the VIVADO project on ZedBoard and visualize the results.

The remainder of the paper is organized as follows: Section 2 describes the design, optimization and implementation of the DFT module; Section 3 details the design and implementation of a VIVADO project (system) on the FPGA that incorporates the DFT module and other IPs; Section 4 implements a GUI using C# programming that communicates with the FPGA design; Section 5 presents the hardware/software system operation and experimental results; Finally, Section 6 concludes the paper.

2 Design, Optimization and Implementation of the DFT Module

2.1 DFT Basics

DFT is used to convert a discrete time domain signal into its corresponding frequency domain representation and vice versa. The conversion of a time domain signal into frequency domain is called DFT or forward DFT. The conversion of a frequency domain signal into time domain is called inverse DFT. If x(n) is a discrete signal of length N, its corresponding frequency domain representation using DFT is expressed as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \text{ for } k = 0, 1\ldots., N\text{-}1. \quad (1)$$

The frequency domain representation involves complex values and is usually represented as the real and imaginary parts, which can be calculated separately:

$$X[k] = \sum_{n=0}^{N-1} x(n) \cos\left(\frac{2\pi}{N}kn\right) - \sum_{n=0}^{N-1} x(n) j \sin\left(\frac{2\pi}{N}kn\right). \quad (2)$$

The magnitude information of the input signal in frequency domain can be calculated as

Real:

$$R(k) = \sum_{n=0}^{N-1} x(n) \cos\left(\frac{2\pi}{N}kn\right), \text{ for } k = 0,1, \ldots, N\text{-}1. \quad (3)$$

Imaginary:

$$I(k) = -\sum_{n=0}^{N-1} x(n) \sin\left(\frac{2\Pi}{N}kn\right), \text{ for } k = 0, 1, \ldots, N\text{-}1. \quad (4)$$

Magnitude:

$$|X(k)| = \sqrt{R^2(k) + I^2(k)}, \text{ for } k = 0, 1, \ldots, N\text{-}1. \quad (5)$$

2.2 DFT Module Design and Simulation in HLS

The DFT module design takes N sample points as input and produce the corresponding DFT magnitude data. The algorithm of the DFT flow in VIVADO HLS is shown in Figure 1 along with the DFT calculation loop implemented by C/C++ programming. The input data points are read through a memory interface and stored locally. The sine and cosine coefficient matrices are stored in a local memory as half precision, which is 16-bit floating point numbers. The coefficients are read as signed fixed-point numbers with precision <16, 2>, which means 16 bit data with 2 bits representing numbers on the left side of the decimal point and 14 bits representing numbers on the right side. The input data are considered as signed 16-bit integer data. The fixed-point multiplication of the two variables local_in (a 16 bit signed integer) and temp_c or temp_s with precision <16, 2> generates a fixed-point result with precision <32, 18>. To use the fixed-point data type, the ap_fixed.h library provided in the HLS is used [20]. To use arbitrary integer precision, the ap_int.h library in the HLS is used [20]. After the execution of the inner loop, the real and imaginary data are used to calculate the DFT magnitude data based on Equation (5). Here the real and imaginary values are of 32-bit integers, hence the result of multiplication is stored in 64-bit integer variables (r3 and r4 in Figure 1). Finally, the square root function is used to calculate the 64-bit sum. In the HLS, the square root function is not defined for arbitrary integer precision, hence long integer type casting is used.

For C simulation, a testbench is written by C++ language. The input of the testbench includes 200 data points of a 100 Hz sinusoidal wave with amplitude of 10^4 and sampling frequency of 1 kHz generated by MATLAB. The testbench code calls the DFT function that is implemented in the HLS according to Figure 1 and generates corresponding DFT results. The results are printed on the console of the HLS software and used for comparison with the MATLAB simulation. Figure 2 compares the DFT data generated from the HLS logic and from the fft() function in MATLAB under the same input signal with multiple frequencies (100 Hz, 250 Hz, and 400 Hz). It is observed that the plots from both the HLS logic and the MATLAB simulation are perfectly matched in all three testing frequencies.

2.3 DFT Module Optimization and Package in HLS

The above C simulation verifies the correctness of the DFT algorithm design. It is now ready to do the synthesis on the DFT module. The latency of the design is in millisecond level due to the nested loops as shown in Figure 3. We can apply various HLS directives [20] to the DFT design for optimization. By analysis, we know there are three main loops in the DFT logic, RD_loop, Outer_loop and WR_loop. The RD_loop reads N data points from the input interface and WR_loop writes N/2 output data points to the output interface. The outer_loop has an inner_loop inside it which

has a tripcount of N. The outer_loop has a trip count of N/2. From initial tries, the inner_loop needs 2 DSP slices and the outer_loop takes 6 DSP slices for implementation. This introduces the limitation for unrolling the loops completely, since the targeted device ZedBoard has only 220 DSP Slices. Additionally, the outer_loop is dependent on the inner_loop. By considering the dependency and device limitation, we choose to unroll the inner_loop by a factor of 10 (the larger the factor, the more usage of the FPGA resources). The WR_loop reads the 32-bit data from the local memory and writes the data to the output interface. Applying the PIPELINE directive is better than the UNROLL directive in terms of latency. For the input and output data interface, the AXI-stream interface (specified as axis) is used [28]. As the sine and cosine coefficients are stored as constants, a single port ROM is used to store the coefficients. Figure 3 shows different directives applied to the elements of the design logic.

After the optimization processing is performed, the latency of the logic is reduced from 1.43 ms to 366.05 μs. Figure 4 shows the performance comparison of the latency of the DFT module, where solution 1 is the performance estimate before applying the directives and solution 2 is after applying the directives. Figure 5 shows the loop latency in the design. There are 200 data points in the simulation, the default trip count for inner_loop is 200. As the inner_loop is unrolled by a factor of 10, the 200 iterations are divided into 10 parts and thus the trip count is 20.

Finally, the C/RTL co-simulation is run to verify the functionality in the register-transfer level (RTL). The co-simulation report shows that the duration taken for the DFT calculation is 366.05 μs from the time instant of reading input to the instant of writing output, which can also be found in the RTL simulation waveforms generated in a pop-out VIVADO window (which is triggered from HLS), as shown in Figure 6. This shows that the DFT design is successful and ready to be packaged as a custom IP by the HLS. The DFT IP can be added to a VIVADO repository and will be used by a VIVADO project for system construction.

3 Design and Implementation of a VIVADO Project

3.1 Design Description

In order to validate the DFT IP packaged by the HLS, a testing system is developed through building a VIVADO project. Figure 7 shows the block diagram of the VIVADO project including the DFT IP and a few other IPs for the project. The VIVADO design incorporates two parts, the programmable logic (PL) design and the processing system (PS) design. The DFT IP is integrated in the PL logic design along with the test data generation and data transfer logic. The PS monitors and controls the flow of the PL design.

As the DFT IP is designed to take digitized signal samples as the input, the Xilinx DDS compiler IP core is modified to generate the sinusoidal data. The DFT IP needs N data samples and has the AXI-stream interface at input. Hence, a combinational logic is introduced to store N consecutive data samples generated from the DDS to an AXI-stream data FIFO IP. Once the N data samples are written into the FIFO IP, the DFT IP is started. The DFT IP reads the sample points stored in the FIFO and calculates the DFT magnitude. Once the calculation is completed, the DFT output is sent to the processor through a DMA (Direct Memory Access) IP core [29]. Both the output interface of the DFT IP and the input interface of the DMA IP are of AXI-stream type. However, the DMA input interface needs two additional signals: Tlast and Tkeep signals. Hence, a custom IP (i.e., Tlast generator IP) is developed to generate Tlast and Tkeep signals to work with the Tvalid signal of the DFT IP output interface. The N/2 output data points from the DFT IP output are sent to the PS through the DMA IP. The PS transfers the data to the GUI over the serial port for display.

3.2 Project Implementation in VIVADO

Figure 8 shows the VIVADO block design. The PS provides 100 MHz clock and a reset signal to the PL design. The PS communicates with the PL side IPs through the AXI-Lite interface. The DMA IP is connected to the PS through the high performance AXI port (S_AXI_HP0). The DDS_sinegen IP is connected to the PS through the AXI-Lite interface for receiving the sine wave frequency settings. The output of the DDS_sinegen IP is connected to the AXIS_data_fifo. The DFT IP reads input data from the FIFO port. The data transfer from the DDS IP to the DFT IP is controlled and monitored by the PS using an AXI GPIO IP and a combinational logic implemented from the utility vector logic blocks. The output of the DFT IP is passed to the DMA IP through the Tlast_gen IP. The DMA IP transfers the DFT data to the PS. Each block in the design is briefly described as follows.

- DDS_sinegen IP

This is a custom AXI-Lite Slave IP core which includes the Xilinx LogicCORE DDS IP core along with custom logic. It has the following ports:

- AXI-Lite slave interface: connected to the processor for controlling data input.
- data_enable: input signal to control the output data to be valid.
- fifo_ready_in: input signal coming from the AXI-stream data FIFO IP, which sends the data out only when this port is ready to take the data input.
- Sig_out[15:0]: 16 bit sinusoidal waveform data output.
- fifo_sig_valid: valid signal for data output.

The Xilinx LogicCORE DDS IP has the phase parameter to the sinusoidal waveform data generator. With appropriate phase setting, the DDS IP generates the sinusoidal waveform samples with a specific frequency [25]. In this design, the input samples are generated internally by the DDS IP. The DDS IP core needs a phase increment value to be set to convert it to the sinusoidal waveform. With an appropriate phase increment value, the output frequency is given by following equation:

$$f_{out} = f_{clk} \cdot \Delta\theta / 2^{B}, \quad (6)$$

where f_{out} is the output frequency, f_{clk} is the system clock frequency (which is 100 MHz in the design), $\Delta\theta$ is the phase increment value and B is the phase width that is the number of bits used to set the phase increment value.

The frequency resolution can be achieved by

$$\Delta f = f_{clk} / 2^{B} . \quad (7)$$

In this design, the system clock is 100 MHz. When the phase width is set as 26, the resolution will be obtained as 1.49 Hz. The phase increment value needs to be entered to get the desired frequency output. The phase increment value for a given frequency can be calculated as

$$\Delta\theta = f_{out} \cdot 2^{B} / f_{clk} . \quad (8)$$

For example, for a 2 MHz sinusoidal output waveform, the phase increment value to be entered is $\Delta\theta = 2MHz \cdot 2^{26}/100MHz = 1342177$. The DDS IP core is configured as shown in Figure 9.

In this design, the frequency generation from the DDS IP core is controlled by the GUI through the processor. Since the corresponding phase increment value is transferred from the processor to DDS IP core, the DDS IP core is added to a VIVADO design with AXI-Lite slave interface and packaged as an IP (i.e., DDS_sinegen IP). In the AXI-Lite slave interface code, the 0^{th} bit of slave register0 is used as valid signal and slave register1 is used for entering phase data value. Figure 10 shows the slave register connections to the phase data input of the DDS IP core. A custom logic is added to control the data to be valid at the output. It takes a data enable signal input that is controlled from the processor along with a combinational logic so that it stays logic high only for the period of N data samples. Here N is set as 200 in the experiment.

- AXI FIFO IP

The sinusoidal waveform sample data are stored in an AXI-stream data FIFO. This FIFO has AXI-stream interface for input as well as output. The FIFO IP is configured for 16-bit 256 words depth. A programmable full signal is enabled for which the threshold is set at 199. Here the FIFO IP gives the first word by default at the output interface, thus an additional value of 199 is set for 200 data words.

- AXI DMA IP

The AXI DMA IP in the VIVADO library provides the direct memory access between the DDR memory and the AXI-stream peripherals of the PL side [29] through an AXI_Lite interface. In this design, the DMA IP is used to transfer the DFT IP output data to the processor. The DMA writes the data directly to the DDR, from which the processor can read the data and send them to the GUI via the serial port for display. The processor must initiate the data transfer by setting the source address, destination address in the DDR, and the transfer length [30]. The DMA reads the data from the source address and writes the data to the destination address.

The DMA IP core has two channels (i.e., read and write) for data transfer; only one channel can be enabled at one time. In this design the data is only transferred from the PL to the DDR, thus only write channel is enabled.

- Tlast_gen IP

The DMA AXI-stream input interface needs two additional signals, i.e., Tlast and Tkeep signals, which are not directly generated from the AXI-stream output interface of the DFT IP. A custom IP is needed to generate the Tlast and Tkeep signals. The Tlast signal indicates the last word in the stream data transfer, which is generated by taking the Tvalid signal from the DFT IP as input with shifting by a clock pulse. The Tkeep signal indicates the valid data bytes. In this design, the data width is 32 bits (4 bytes), thus the Tkeep width is set to 4.

- AXI GPIO IP and Combinational Logic

Since the PS monitors and controls the flow of operation of the VIVADO design. An AXI GPIO IP is introduced to connect the control and monitor signals from the PS to the PL. The processor issues two control signals. One is the data enable signal, which initiates the DDS output data passing to the AXI FIFO IP; the other is the DFT enable signal, which starts the DFT IP operation. The processor reads one signal for monitoring which is the FIFO full flag. Once the FIFO is full, the processor disables the enable signal of the DDS_sinegen IP and issues an enable signal to the DFT IP.

This IP has two GPIO channels enabled, one 2-bit channel for the two control signals and one 1-bit channel for the monitor signal. The two control signals are differentiated in the PL design using two Xslice IPs provided in the VIVADO library.

To allow the workflow of the VIVADO design, some combinational logic IPs are used. One control signal from the GPIO IP is ANDed with the inverted FIFO full signal for the DDS_sinegen IP; once the FIFO is full, the DDS_sinegen IP will immediately stop sending data. Similarly, the other control signal from the GPIO IP is ANDed with the FIFO Tvalid signal and connected to the Tvalid input of the DFT

IP; the DFT IP will recognize the FIFO Tvalid only when the PS issues the related control signal. This way it enables that the DFT IP reads the N data samples only when the data block is ready at the FIFO, rather than read them in between.

3.3 SDK Application Design

The Xilinx SDK [27] is an Integrated Development Environment (IDE) that works with the hardware design created by the VIVADO Design Suite. As mentioned earlier, the processor in ZedBoard controls and monitors the functionality of the VIVADO design and performs serial communication with the GUI running on a PC. Hence, an SDK application project needs to be created to work with the VIVADO design.

The application design written by C language initializes the peripherals at the start of the application project creation. Specifically, it initializes the PS UART, UART interrupt, AXI GPIO, and AXI DMA. The flowchart of the SDK application program is shown in Figure 11. The PS UART in the Xilinx SDK has the default baud rate of 115200, data width 8 bits, stop bit of 1 and no parity. The PS UART can work in four modes, Normal, Local loopback, Remote loopback and Automatic echo. In this design, the UART is set to work in Normal mode.

The UART interrupt is configured to generate an interrupt when a data item is received on the serial port. The interrupt subroutine reads the received data and raises a flag. The flag is polled in the main function continuously. If a data item is received on the serial port, it will be checked and a corresponding action will be taken. The data item includes a number of command characters that are defined to represent corresponding frequencies to be set for the input waveform from the DDS_sinegen IP. A special character 'e' is defined to represent the start of the DFT operation. In the Zynq FPGA device, there is a Generic Interrupt Controller (GIC) that controls the interrupt request from the peripherals [30].

The GIC is configured for monitoring the UART interrupt. The UART can generate an interrupt on multiple events. It includes an interrupt mask register that can enable or disable particular interrupt for the design [21]. A mask value is generated and loaded in the interrupt mask register, which enables the interrupt for the FIFO-full event, transmit buffer empty event, overrun error event, framing error event, parity error event and Timeout error event. The Timeout error occurs when the receiver has remained idle for more than the time set in the timeout register.

As mentioned in VIVADO design description, the PS issues two control signals, one for FIFO write enable and one for DFT IP start. If the DFT IP start command is received, a FIFO write enable signal is set. It allows the DDS to write N data points into the FIFO. The FIFO full signal is polled in the processor. If the FIFO full signal goes high, the PS lowers the FIFO write enable signal. Next, the processor starts the DFT IP by setting the enable bit high. It starts the DFT operation and produces the DFT output. The processor then initializes the AXI DMA transfer for receiving data from the PL. It polls if the AXI DMA is receiving the data. Once the data are received, they are transferred through the serial port to the GUI for display.

4 Design and Implementation of a GUI

A graphical user interface (GUI) is designed using windows form application in .net framework [24]. Figure 12 shows the GUI panel developed for this design, which includes the following functionalities:

- Communicating with the ZedBoard over the serial port
- Setting the frequency
- Start the DFT computation
- Stop the DFT computation
- Display the processed DFT data in frequency domain
- Display the received frequency calculated from the processed DFT data

The communication between the ZedBoard and the GUI is through the UART serial port. A menu option named *Serial Port Settings* is provided on the top-left corner on the GUI. A separate form is added in the design that pops out (as shown in Figure 13) when the user clicks on the serial port settings menu. The user can select the communication port, baud rate, data bits, parity, stop bits and flow control. The communication port number may be different for different devices. After selecting the serial port settings, the user should click on the Apply Settings button for activation.

The GUI communicates with the Zynq-7020 SoC processor in the ZedBoard over the serial port. There is a command structure used for communication. In this design, different frequencies can be selected for the frequency setting of the input signal. The selected frequencies are available between 0 and 50 MHz, which are available in the drop-down combo box labelled *Select Frequency* on the GUI panel. Table 1 shows the frequencies selected in the drop-down box of the GUI and their respective command indices. The index of the selected frequency is sent to the ZedBoard over the serial port. The command character of the *Start Acquisition* is defined as the letter "e". When the *Start Acquisition* button is clicked, the letter "e" is sent to the ZedBoard.

Table 1 Selected frequency and index

Index	Frequency
0	2MHz
1	4MHz
2	6MHz
3	7.5MHz
4	10MHz
5	12.75MHz
6	15MHz

7	18MHz
8	18.5MHz
9	20MHz

The ZedBoard transmits the DFT output data to the GUI over the serial port. The data words are separated by ";" and ends with a new line "\n" character. In the GUI program, the ReadLine() function is used to read the complete data until the new line character. The received data are then separated using the Split() function and separated data strings are converted to double for later use. To plot the received DFT output data, a Zedgraph class [31] is used, which is an open source class library, user control, and web control for .net, written in C# language. The Zedgraph class provides methods for the 2D plot in the C# GUI. The AddCurve() method is used to initialize the graph pane. When the DFT output data are received from the ZedBoard, the list of the data to be plotted on the graph pane is updated. The updated list is plotted on the graph pane by updating the list using the zedhraph.invalidate() and zedgraph.axischange() methods.

To calculate the x-axis value in the form of frequency on the GUI, the following equation is used:

$$f = i \cdot F_s/N\,, \tag{9}$$

where f is the frequency to be calculated corresponding to the received data point; i is the index of data point; F_s is the sampling frequency, which is 100 MHz in this design; N is the total number of data point, which is 200 in this design. Using Equation (9), the frequency data points are calculated and used as the x-axis in the graph pane.

To detect the frequency from the received DFT output data, the maximum magnitude from the received 100-point magnitude data is detected. The index of the data point with the maximum magnitude is retrieved using the Array.IndexOf() method; by this way we get the index of the maximum magnitude. This index is put into the above equation to calculate the corresponding frequency. The calculated value is then displayed in a textbox labelled *Received Frequency* on the GUI panel.

Once the *Start Acquisition* button is pressed, it sends the start command to the FPGA functionality on the ZedBoard and sets a flag to indicate continuous data acquisition. The FPGA functionality completes the operation and sends the DFT output data to the GUI. Upon the reception of the data from the FPGA, the GUI plots the data on the Zedgraph pane and checks for the flag set for continuous acquisition. If the flag is set, the start command is sent over the serial port again. To stop the continuous acquisition, the user must click the *Stop Acquisition* button. The *Stop Acquisition* button clears the continuous acquisition flag and thus stops the acquisition. A flowchart of the C# program for the GUI is given in Figure 14.

5 System Operation and Results

After the VIVADO project and the GUI program are completed, we can run the whole testing system. First, we run the synthesis and implementation of the VIVADO project and generate the bitstream file. Next, we configure the ZedBoard boot mode jumpers (JP7-JP11) to the JTAG boot mode and connect the ZedBoard power cable, UART cable, and USB-JTAG cable respectively as shown in Figure 15. The power-LED (green LED) should light on. Then, we export hardware and launch an SDK application from the VIVADO project. In the opened SDK window, we program the FPGA by clicking the command "Program FPGA" on the SDK menu; the done-LED (blue LED) on the ZedBoard should light on. Finally, right click on the SDK application and run the command "Launch on hardware". Now the FPGA is ready to take commands from the GUI over the serial port.

The following are the steps to operate the C# GUI and test the DFT IP response.

- Once the GUI starts, a message pops out reminding you to select a serial port, set the serial port from the *Serial Port Settings* menu on the GUI panel.
- Select the frequency from the dropdown combo box with labelled *Select Frequency*.
- Start the DFT computation by clicking on *Start Acquisition* button. The Zedgraph pane *DFT DATA* will show a plot of the DFT magnitude received from the FPGA; and the textbox labelled *Received Frequency* will show the frequency of the input signal calculated from the received DFT data.
- To stop the DFT computation, click on the *Stop Acquisition* button.

The following figures show the GUI snapshots for different frequencies. Figure 16 shows the magnitude response for an input signal with frequency 7.5MHz. It can be observed that the peak of the magnitude is at 7.5 MHz in the plot and the received frequency textbox also shows 7500000 Hz. Similarly, Figure 17 shows the magnitude response for an input signal with frequency 20 MHz.

In Figure 18, we intentionally set the frequency of the input signal as 12.75MHz, which requires the DDS generate a signal of frequency 12.75 MHz. However, the minimum frequency interval in the x-axis in this design is F_s/N = 100 MHz/200 = 500 kHz. Hence, the minimum change in the detected frequency would be 500 kHz. In this case the input frequency is 12.75 MHz; the detected frequency should vary between 12.5 MHz and 13.0 MHz. In the experiment, it is verified that the frequency indeed dynamically jumps between 12.5 MHz and 13.0 MHz, and one of them is randomly captured for display in the received frequency textbox.

6 Conclusions

This work involved developing a complete FPGA based digital system including design and optimization of a DFT computation algorithm using the HLS, a VIVADO project that tests and validates the DFT IP, and a C# GUI that visualizes the results of the DFT output data. The process of design and development of the DFT IP in the HLS provided hands-on experience for using the HLS design techniques for parallelization and optimization. Additionally, the optimized usage of floating point, fixed point and arbitrary integers is handled during the DFT IP design. The optimized DFT IP reduced the latency of the logic from 1.43 ms to 366.05 μs. This work also provided hands-on experience for developing a VIVADO project for validation of a custom IP packaged by the HLS. The Xilinx provided IP cores such as DDS and AXI DMA were used in developing the testing system. The Zynq-7020 SoC processor was configured to control the functional flow of the FPGA design and transfer the DFT output data over a UART serial port. A GUI was designed using the windows form application in C# programming to display the DFT output data. The Zedgraph technique was used to visualize the DFT magnitude spectrum plotting on the GUI. Experimental operations and results were presented with discussion. For the future work, we consider to apply the same development method to the FPGA implementation on the inverse DFT and other applications such as image processing. The DFT IP design, optimization and implementation as well as the VIVADO project development methods can be extended to other FPGA applications.

Acknowledgement

The authors would like to acknowledge the support of the project development from St Cloud State University, MN, USA through the Early Career Grant (No. 211129) and the support from the Department of Electrical and Computer Engineering through providing its equipment and software.

References

[1] S. Brown and J. Rose, "FPGA and CPLD architectures: A tutorial", IEEE Design and Test of Computers, 13(2):42–57, 1996.

[2] S. Vaidyanathan, E. Tlelo-Cuautle, A. Sambas, L. G. Dolvis, and O. Guillén-Fernández, "FPGA design and circuit implementation of a new four-dimensional multistable hyperchaotic system with coexisting attractors", International Journal of Computer Applications in Technology, Vol. 64, No. 3, pp. 223-234, 2020.

[3] S. Vaidyanathan, I. Pehlivan, L. G. Dolvis, K. Jacques, M. Alcin, M. Tuna, and I. Koyuncu, "A novel ANN-based four-dimensional two-disk hyperchaotic dynamical system, bifurcation analysis, circuit realisation and FPGA-based TRNG implementation", International Journal of Computer Applications in Technology, Vol. 62, No. 1, pp. 20-35, 2020.

[4] R. Nane, V.M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y.T. Chen, H. Hsiao, and S. Brown, "A Survey and Evaluation of FPGA High-Level Synthesis Tools", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 35 (10): 1591–1604, 2016.

[5] J.G. Proakis and D.G. Manolakis, Digital Signal Processing: Principles, Algorithms and Applications, 3rd Edition, Prentice Hall, Oct. 1995.

[6] D. Bland, T. Laakso and A. Tarczynski, "Analysis of algorithms for nonuniform-time discrete Fourier transform", IEEE International Symposium on Circuits and Systems. Circuits and Systems Connecting the World (ISCAS 96), Vol. 2, pp. 453-456, 1996.

[7] S. Valipour1, A.D. Shaligram and G.R.Kulkarni, "Spectral analysis of EEG signal for detection of alpha rhythm with open and closed eyes", International Journal of Engineering and Innovative Technology (IJEIT), Vol. 3, No. 6, pp. 1-4, Dec. 2013.

[8] M. J. Alam, P. Kenny, P. Dumouchel and D. O'Shaughnessy, "Robust speech recognition using warped DFT-based cepstral features in clean and multistyle training", 2014 22nd European Signal Processing Conference (EUSIPCO), Lisbon, Portugal, pp. 1791-1795, 2014.

[9] Y. Liu, F. Zhang, H. Miao and R. Tao, "The hopping discrete fractional Fourier transform", Signal Processing, Vol. 178, Available online August 2020. https://doi.org/10.1016/j.sigpro.2020.107763

[10] Y. Bengio and Y. LeCun, Scaling learning algorithms towards AI, Chapter 14, pp. 321-360. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, Large-Scale Kernel Machines, MIT Press, 2007.

[11] M. Mathieu, M. Henaff and Y. LeCun, "Fast training of convolutional networks through FFTs", CoRR, abs/1312.5851, 2014. http://arxiv.org/abs/1312.5851

[12] O. Rippel, J. Snoek and R.P. Adams, "Spectral representations for convolutional neural networks", Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS 2015), Vol. 2, pp. 2449-2457, Dec. 2015.

[13] F. Qu, F. Guo, W. Jiang and X. Meng, "Data Compression Based on DFT for Passive Location in Sensor Networks", Procedia Engineering, Vol. 29, pp. 3091-3095, 2012.

[14] G. Pastor, I. Norros, R. Jäntti and A.J. Caamaño, "Compressive Data Aggregation from Poisson point process observations", 2015 International Symposium on Wireless Communication Systems (ISWCS), Brussels, Belgium, pp. 106-110, 25-28 Aug. 2015.

[15] D. Sadhukhan, S. Pal and M. Mitra, "Electrocardiogram data compression using adaptive bit encoding of the discrete Fourier transforms coefficients", IET Science, Measurement & Technology, Vol. 9, No. 7, pp. 866-874, 2015.

[16] J. van der Hoeven, "The truncated fourier transform and applications", Proceedings of the 2004 international symposium on Symbolic and algebraic computation (ISSAC), pp. 290–296, July 2004. https://doi.org/10.1145/1005285.1005327

[17] D.D. Chen, N. Mentens, F. Vercauteren, S.S. Roy, R. C. Cheung, D. Pao, and I. Verbauwhede, "High-Speed Polynomial Multiplication Architecture for Ring-LWE and

SHE Cryptosystems", IEEE Transactions on Circuits and Systems I, vol. 62, no. 1, pp. 157-166, 2015.

[18] G. Bongiovanni, P. Corsini and G. Frosini, “One-dimensional and Two-dimensional Generalized Discrete Fourier Transform”, IEEE Trans. Acoust. Speech Signal Process. Vol. ASSP-24, pp. 97-99, Feb. 1976.

[19] V. Britanak and K. R. Rao, “The Fast Generalized Discrete Fourier Transforms: A Unified Approach to The Discrete Sinusoidal Transforms Computation”, Signal Processing, vol. 79, pp. 135-150, Dec. 1999.

[20] Xilinx, Inc, “Vivado Design Suite User Guide: High-Level Synthesis”, UG902, v2020.1, June 2020.

[21] Xilinx, Inc, “Zynq-7000 SoC: Technical Reference Manual”, UG585, v1.12.2, July 2018. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[22] Xilinx, Inc, “Vivado Design Suite User Guide: Using the Vivado IDE”, UG893, v2020.1, June 2020. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug893-vivado-ide.pdf

[23] Digilent, Inc, "ZedBoard Hardware User’s Guide", Ver 2.2, Jannuary 2014. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

[24] Microsoft Corp., Visual Studio 2019. Available: https://visualstudio.microsoft.com/

[25] Xilinx, Inc, “DDS Compiler v6.0: LogiCORE IP Product Guide”, Dec. 2017. Available: https://www.xilinx.com/support/documentation/ip_documentation/dds_compiler/v6_0/pg141-dds-compiler.pdf

[26] B. Hahn and D. Valentine, Essential MATLAB for Engineers and Scientists, 7th Edition, Academic Press; April 2019.

[27] Xilinx, Inc, “Getting Started with Xilinx SDK”, v2016.2. Available: https://www.xilinx.com/html_docs/xilinx2016_2/SDK_Doc/index.html

[28] Xilinx, Inc, “Vivado Design Suite: Vivado AXI Reference”, UG1037, v4.0, July 2017. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf

[29] Xilinx, Inc, “AXI DMA v7.1: LogiCORE IP Product Guide”, June 2019. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

[30] M.A. Enderwitz, R.A. Elliot, C.H. Louise, and R.W. Stewart, The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC, First Edition, Strathclyde Academic Media, 2014.

[31] ZedGraph. Available: https://sourceforge.net/projects/zedgraph/

Appendix: All figures

Input: N data samples of a digitized signal
Output: N/2 processed DFT data points

1. Read and store N input data samples from input interface;
// DFT calculation loop
2. Outer_loop:
{ Inner_loop:
{ Process N data samples to generate a real and imaginary DFT data points via Eqs (3) and (4);
}
Process real and imaginary parts to get the magnitude data points via Eq (5);
}
3. Write N/2 magnitude data points to output interface;

```
//DFT Calculation loop
Outer_loop:
for( i = 0; i < Data_points_div2; i++)
{
    temp_real = 0;
    temp_imag = 0;
    inner_loop:
    for(n = 0; n < Data_points; n++)
    {
        temp_s = sin_c[i][n];
        temp_c = cos_c[i][n];
        mul1 = (((local_in[n]) * temp_c));
        mul2 = (((local_in[n]) * temp_s));
        temp_real = temp_real + ap_int<18>(mul1);
        temp_imag = temp_imag + ap_int<18>(mul2);
    }
    r3 = temp_real * temp_real; // 32 bit multiplication
    r4 = temp_imag * temp_imag;
    r5 = r3 + r4;               // 64 bit addition
    local_out[i] = (int)(sqrtf((long long)r5));
}
```

Fig 1. Algorithm of computing the DFT magnitude and critical implementation code

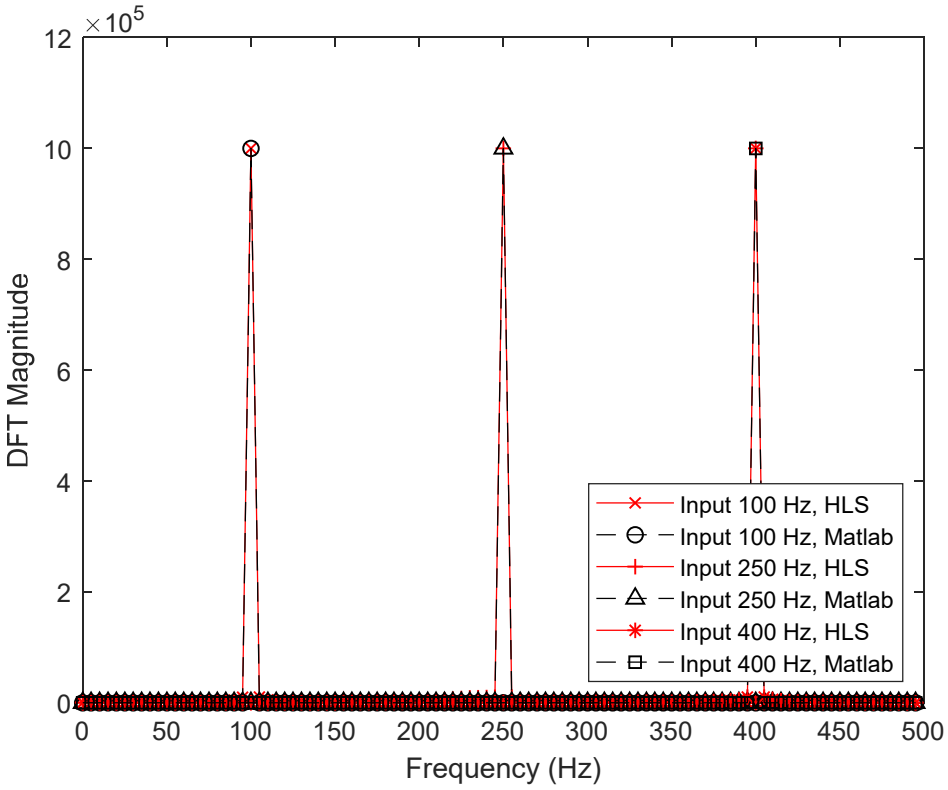

Fig 2. Comparison of DFT results from HLS Logic and MATLAB simulation for 100Hz, 250 Hz and 400 Hz sinusoidal inputs

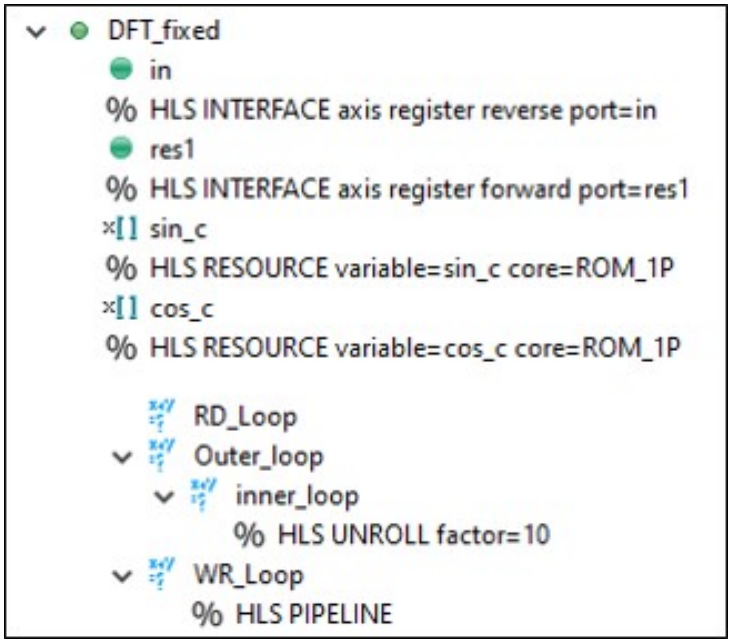

Fig 3. Applying different HLS directives to the DFT Logic elements

Performance Estimates

Timing (ns)

Clock		solution1	solution2
ap_clk	Target	10.00	10.00
	Estimated	8.719	8.719

Latency (clock cycles)

		solution1	solution2
Latency	min	142903	36605
	max	142903	36605
Interval	min	142903	36605
	max	142903	36605

Fig 4. Performance comparison of the latency of the DFT module before and after optimization

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- RD_Loop	200	200	1	-	-	200	no
- Outer_loop	36300	36300	363	-	-	100	no
+ inner_loop	340	340	17	-	-	20	no
- WR_Loop	101	101	3	1	1	100	yes

Fig 5. Loop latency of the DFT logic after applying the directives

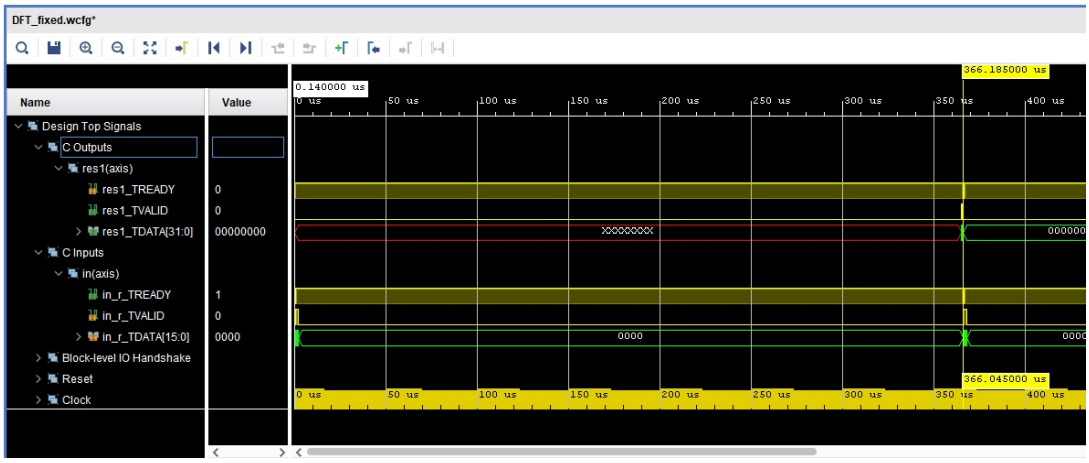

Fig 6. The DFT module RTL simulation waveforms

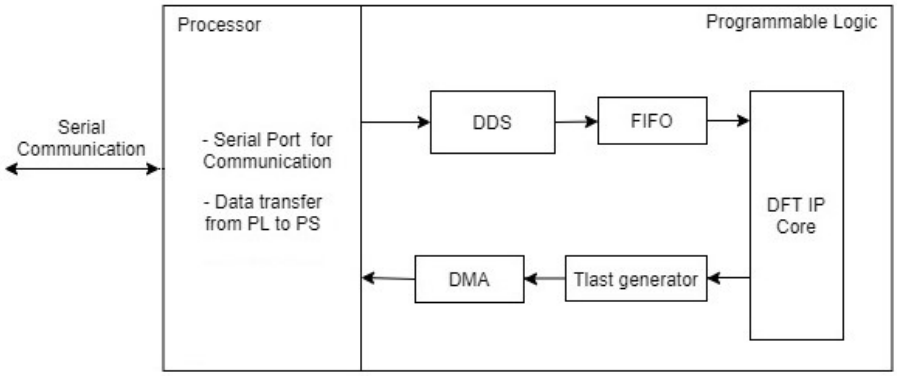

Fig 7. The VIVADO project block diagram

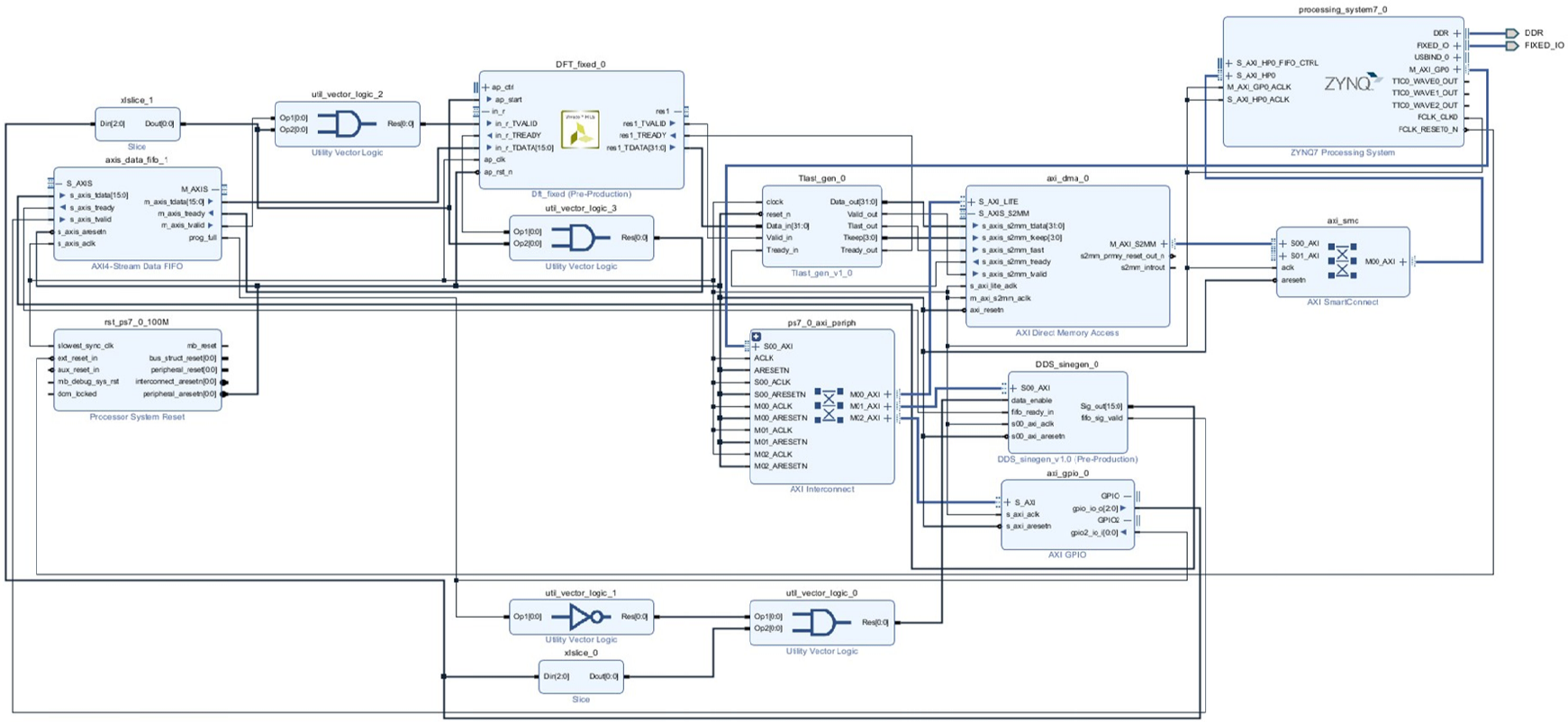

Fig 8. The VIVADO project system construction

Component Name: dds_compiler_0

Configuration | Implementation | Detailed Implementation | **Summary**

Output Width	16 Bits
Channels	1
System Clock	100 MHz
Frequency per Channel (Fs)	100.0 MHz
Noise Shaping	None
Memory Type	Block ROM (Auto)
Optimization Goal	Area (Auto)
Phase Width	26 Bits
Frequency Resolution	1.4901161193847656 Hz
Phase Angle Width	16 Bits
Spurious Free Dynamic Range	96 dB
Latency	7
DSP48 slice	0
BRAM (18k) count	15

Fig 9. The DDS IP core configuration summary

```
// Add user logic here
assign phase_valid = slv_reg0[0]; //Phase data valid signal
assign phase_data = slv_reg1;     //32 bit phase data from PS
// User logic ends
```

Fig 10. AXI-Lite register usage for phase programming (Verilog code)

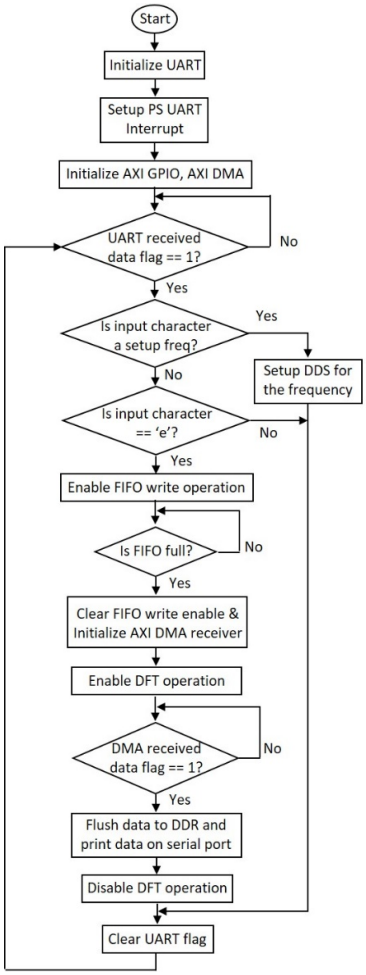

Fig 11. Flowchart of the SDK application program

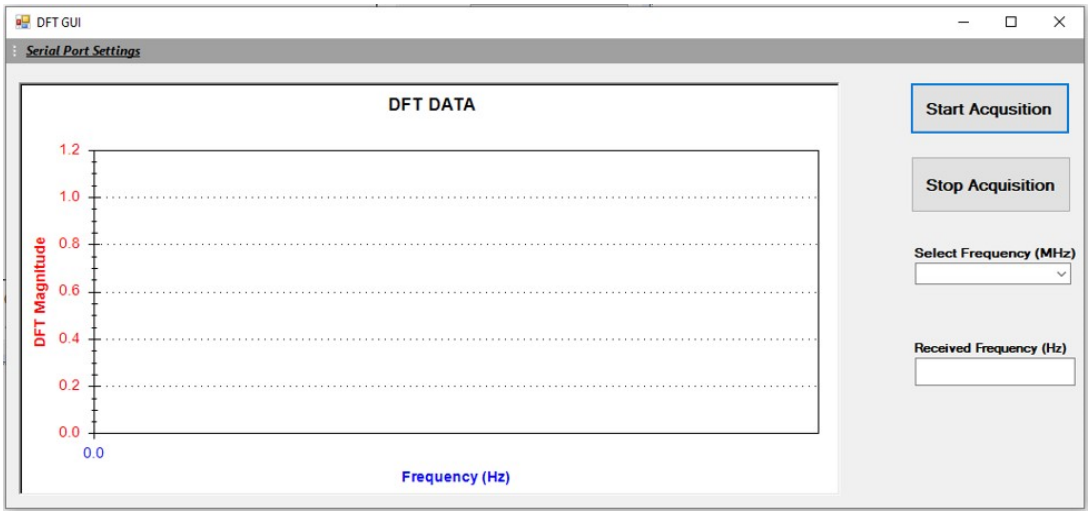

Fig 12. The GUI panel for communicating with the ZedBoard

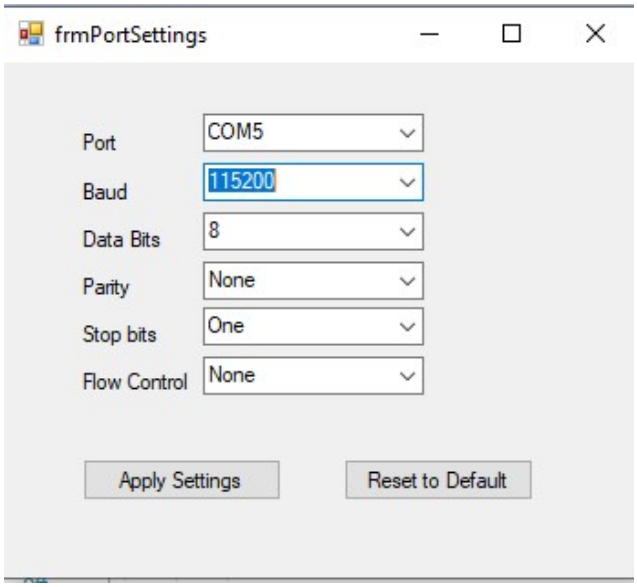

Fig 13. The serial port settings menu

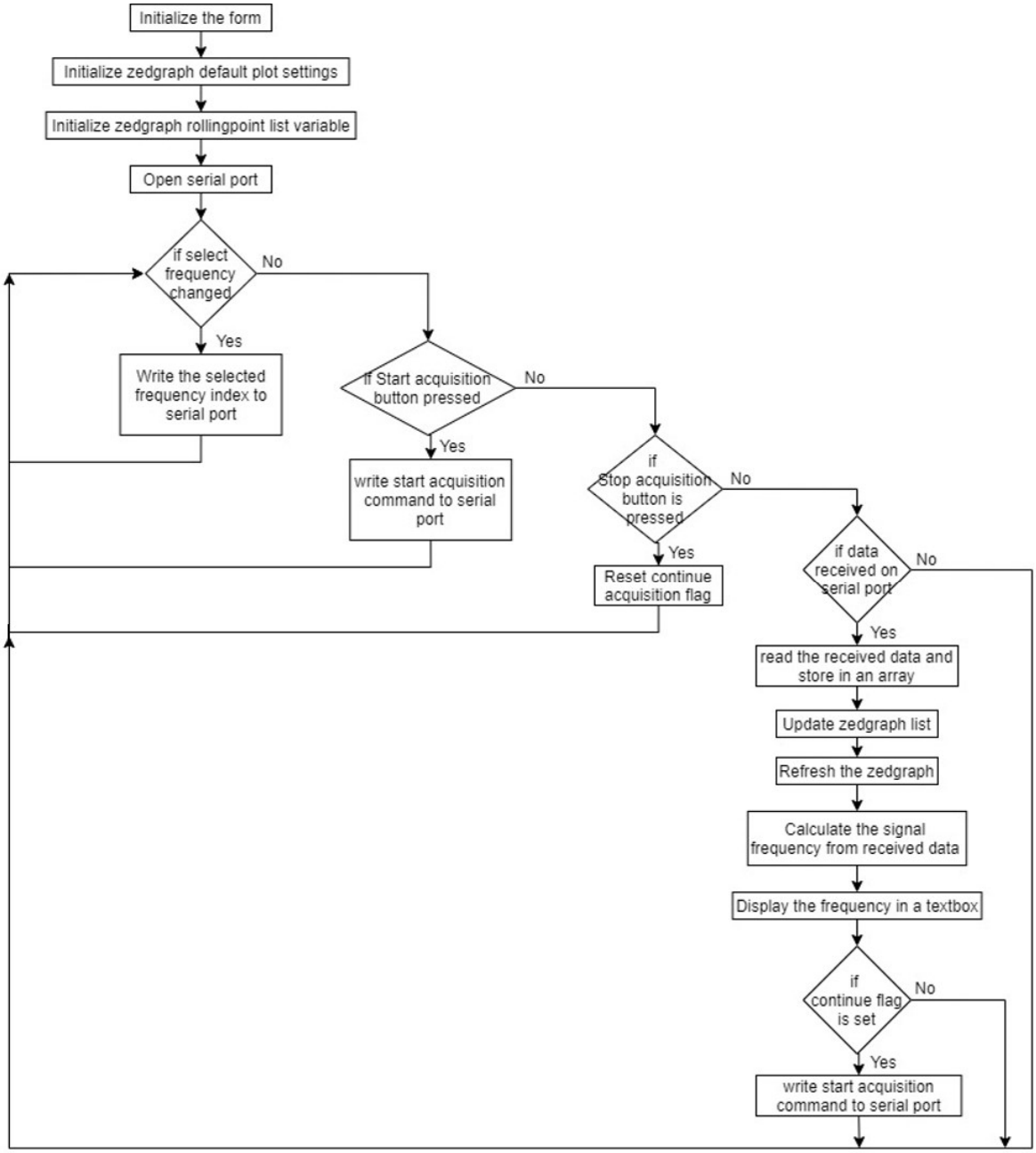

Fig 14. Flowchart of the GUI program

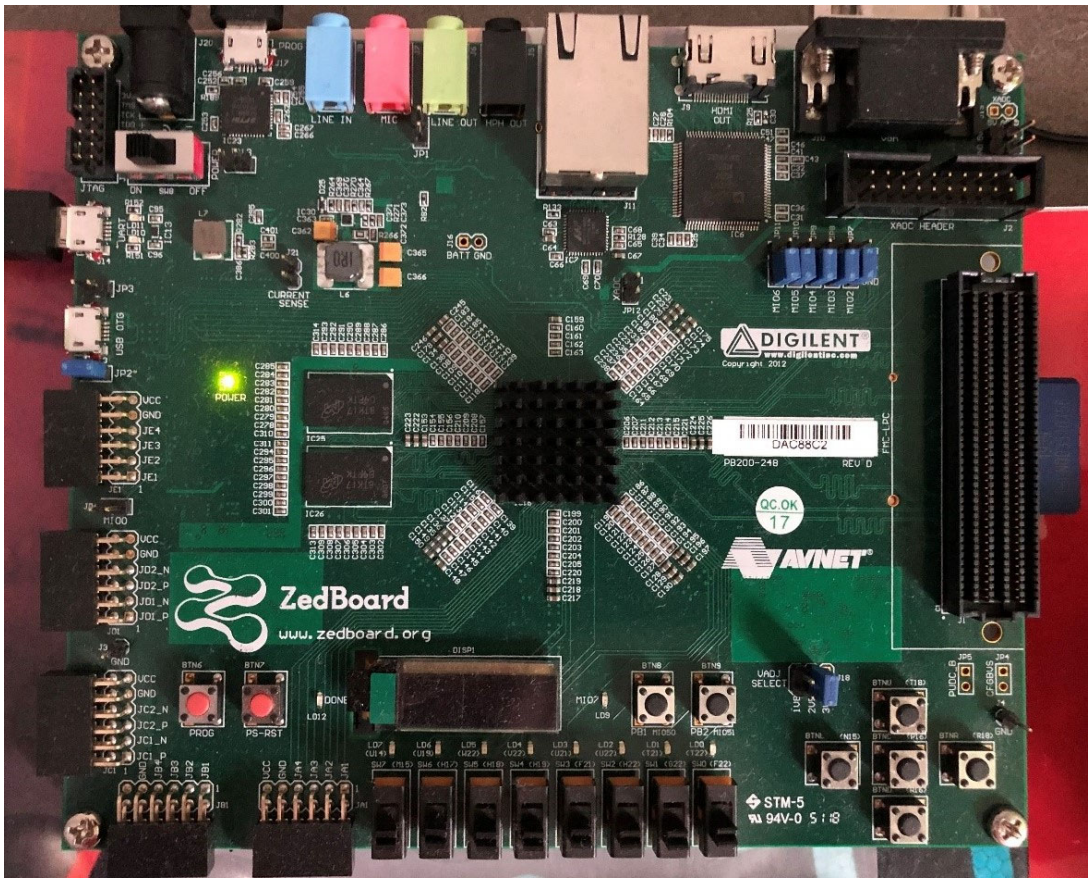

Fig 15. The ZedBoard in running

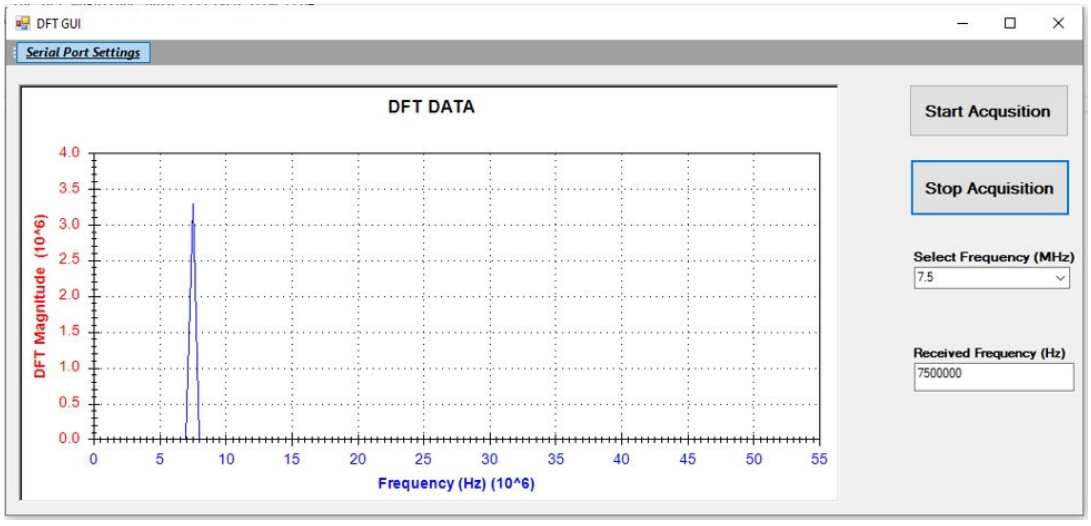

Fig 16. DFT magnitude for an input signal with frequency 7.5MHz

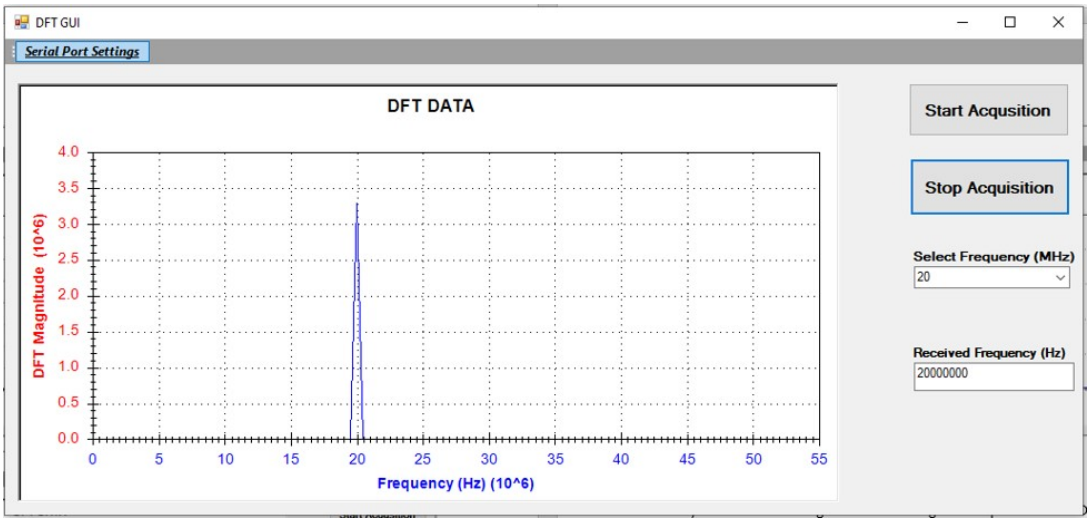

Fig 17. DFT magnitude for an input signal with frequency 20MHz

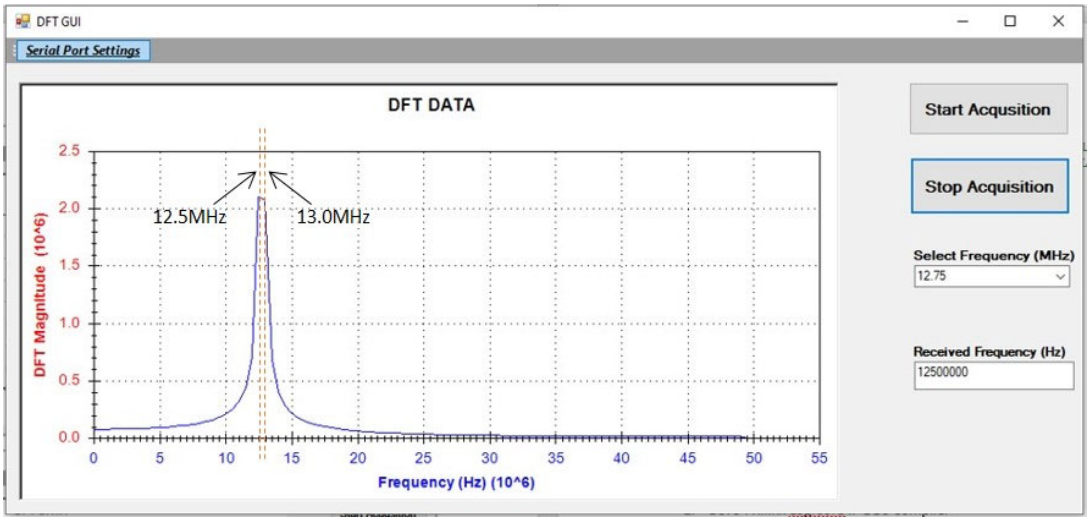

Fig 18. DFT magnitude for an input signal with frequency 12.75MHz