The Development of a High Performance Digital RF Transmitter for NMR

By

Shaw Wei Cheoo

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Electronic Engineering.

Victoria University of Wellington
2012

Abstract

This Master's thesis consists of the development of a Nuclear Magnetic Resonance (NMR) Radio Frequency (RF) transmitter, which is a core electronic subsystem of an NMR system. The main purpose of this research is to contribute to the application of NMR, which is a new sensing technology that has yet to be fully implemented into the everyday world. One of the barriers to adopting this technology is its complexity. However, the invention of high speed digital FPGAs (Field Programmable Gate Array) such as the Spartan series has made it easier to develop high performance NMR systems over recent years. The major contribution to this research is the development of faster digital signal processing hardware, and methodologies that have been implemented on a single chip. This has reduced the size and the cost of the electronic subsystem and contributed towards the evolution of NMR as a general tool. This thesis introduces the concept of implementing a high-speed NMR RF multi-frequency transmitter by using multiple Direct Digital Synthesis (DDS) cores to generate sine-waves, which range from 100 kHz to 750 MHz. The research required three stages to be achieved, beginning with conceptual design of a high-speed transmitter using MATLAB-Simulink, RTL-level (Register-Transfer Level) simulation and hardware implementation, which included hardware testing on a prototype board. This Master's research is to seek a solution to building a multi-core DDS module in an FPGA device. In other words, the research work focuses on finding an alternative solution to constructing a DDS system. The project involves building up the VHSIC Hardware Description Language (VHDL) program to work beyond the hardware limitation of an FPGA device. Hence, the final solution does not consider any noise impact due to the structure of the developed system.

Acknowledgements

I wish to express my sincere thanks to my supervisor and co-supervisor, Dr. Robin Dykstra and Dr. Paul Teal for providing me with all the necessary suggestions toward my research.

I place on record my sincerest gratitude to the committee members Prof Dale Carnegie, Dr. Christopher Hollitt and Dr. Alex Potanin in helping me in the conversion to Master's level and for their extreme patience in the face of numerous obstacles.

I also want to thank my fellow PhD students for their constant encouragement, support, feedback and friendship. I am extremely grateful and indebted to them for their expert, sincere and valuable guidance extended to me.

I take this opportunity to record my sincere thanks to my family in Malaysia for their unceasing encouragement and support.

Finally yet importantly, I would like to thank everyone who, directly or indirectly, has lent me their hand while I encountered problems in New Zealand.

Table of Contents

List of Tables

List of Figures

Table of Abbreviations

ASIC	Application-Specific Integrated Circuits
BUFG	Global Clock Buffers
BUFPLL	High Speed I/O Clock Buffer
CE	Chip Enable
CIC	Cascaded Integrator-comb
CLB	Configurable Logic Blocks
CLKOUT	Clock Output
CMT	Clock management tiles
CS	Chip Select
CORDIC	Coordination Rotation Digital Computer
CP	Charge Pump
DAC	Digital-to-analog converter
DACCLK	DAC Master Clock
DCI	Input clock pin
DCM	Digital Clock Manager
DCO	Output clock pin
DDR	Double data rate
DDS	Direct Digital Synthesis
DLL	Delay Lock Loop
DSP	Digital signal processing
EMF	Electromotive Force
EMI	Electromagnetic Interference
FA	Full adder
FFT	Fast Fourier Transform
FID	Free Induction Decay
FIR	Finite Impulse Response

FPGA	Field programmable gate array
GTP	Gunning Transceiver
IBUFG	Single-ended Input Global Clock Buffer
ILOGIC	Input Logic
IOB	Input/output Block
IOI	I/O interface
IP	Intellectual Property
ISE	Integrated Software Environment
LAB	Logic Array Blocks
LF	Loop Filter
LSB	Least Significant bit
LUT	Look-up tables
LVDS	Low-voltage differential signal
MADD	Multiply-Add
MCU	Microcontroller unit
MSB	Most significant bits
MUX	Multiplexer
NAND	Negated ANDs
NMR	Nuclear Magnetic Resonance
OLOGIC	Output Logic
OSERDES	Output Serialisation or Deserialisation
PFD	Phase-Frequency Detector
PIC	Peripheral interface controller
PLD	Programmable Logic Device
PLL	Phase locked loop
PROM	Programmable Read-Only Memory
RF	Radio Frequency

ROM	Read-only memory
RTL	Register-transfer level
SCLK	Serial Clock
SCLR	Synchronise Clear
SDIO	Serial Data Input Output
SDO	Serial Data Output
SDR	Single data rate
SERDES	Serialisation or de-serialisation
SFDR	Spurious free dynamic range
SNR	Signal-to-noise ratio
SOPC	System on Programmable Chip
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
TTL	Transistor logic
VCO	Voltage Controlled Oscillator
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Chapter 1
Introduction

This chapter introduces the problem statement, an overview of the proposed NMR RF transmitter, the motivation, research goals, and organisation of this thesis.

1.1 Problem Statement

In the communication field, an RF transmitter of a device has been associated with the frequency and modulation of a carrier signal. These techniques are transferable to NMR where an amplitude-modulated stimulus is applied to a probe.

In the past few decades, the implementation of a traditional NMR RF transmitter has been achieved using analog construction. Although it has many advantages such as high precision, there are several drawbacks are still associated it such as cost, size, complexity and portability [6]. After 40 years of research, developments of microelectronics brought a technological revolution to the transceiver system. The availability of commercial products in recent years has resulted in a dramatic switch from classical architecture towards a digital domain [11].

The momentous changes in the architectural conception of an NMR RF transmitter, which used to be exclusively granted by the technological possibilities of hardware integration, can now be realised by modern microelectronic devices. Most of the current NMR RF transmitters try to merge the analog and digital approach in order to get the best performance from both worlds.

Assorted RF transceivers are developed for different applications such as wireless communications, digital radiation spectrometers and so on. Despite the fact that not all the RF transmitters are fitted to the NMR system, but these RF transmitters can be altered to suit different applications in NMR spectroscopy. Modifications of the RF transmitter enable the NMR system to meet the objectives of this thesis research.

Several NMR RF transmitters have been proposed in recent years resulting in innumerable solutions that generally tend to sacrifice some features like overall system throughput or real-time capability in favour of an increase in system versatility [25, 26, 41, 63]. This thesis proposes the implementation of an NMR RF transmitter on a single FPGA chip. The proposed solution contains internal DDS modules, which could run in parallel to generate high-speed samples. In addition, this can help overcome the frequency limitation of a standard NMR RF transmitter generating analog sine-wave signals ranges from 100 kHz to 750 MHz at 2 GHz sampling rate. Moreover, the proposed system could support an output signal of up to four distinct frequencies. This overcomes the phase coherence problem.

1.2 Overview of Proposed NMR RF Transmitter

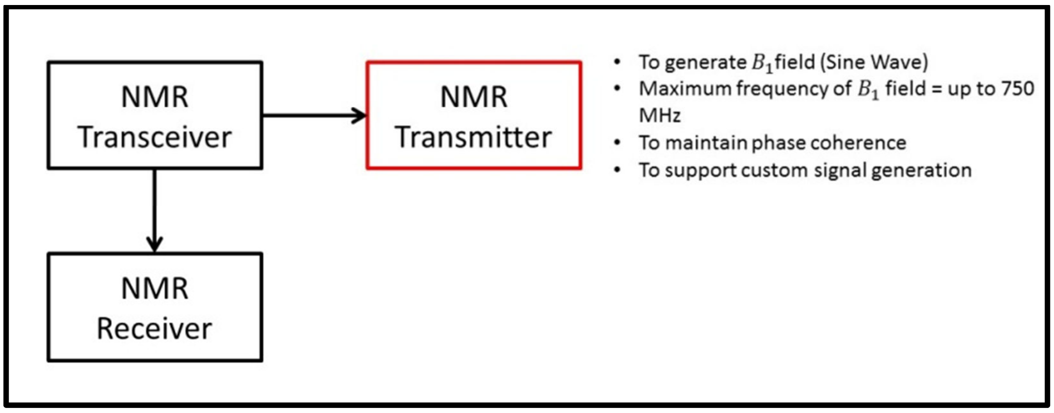

NMR Transceiver
NMR Transmitter
NMR Receiver
To generate B_1field (Sine Wave)
Maximum frequency of B_1 field = up to 750 MHz
To maintain phase coherence
To support custom signal generation

Figure 1.0: Overview of proposed RF transceiver.

The NMR RF transmitter proposed in this thesis is part of the NMR transceiver as shown in Figure 1.0. This NMR transmitter can generate a variable sine-wave (B_1) from 100 kHz to 750MHz. 100 kHz is the minimum frequency we aim for the system to generate. 750 MHz is selected, as the upper limit output frequency after considering that sample of high frequency signal might be insufficient to maintain the signal shape as sinusoidal.

Usually, the signal generated by the transmitter can be represented by the formula below:

$$x(t) = A_{Gain}\sin[2\pi f_c t] \tag{1.0}$$

A_{Gain} is a unity gain, which is used to determine the amplitude of the signal. In this thesis, it will be treated as '1'. f_c is the signal frequency while 't' is the sampling time.

The purpose of an NMR RF transmitter focuses on preserving the phase coherence of a signal. The phase coherence lost is a limitation of most of the NMR transmitters available within the NMR field. Most of the systems lose the original signal phase after switching to another frequency.

1.3 Introduction of NMR RF Transmitter in NMR Spectroscopy

NMR spectroscopy is a technique used to study the physical, chemical, and biological properties of materials by probing the nuclear dipole moments in the material [23]. It was found and measured using molecular beams by Isidor Rabi in 1938 [32]. He received the Noble Prize in Physics in 1944 for his work [1]. Two years later, Felix Bloch and Edward Mills Purcell expanded the technique to liquids and solids. They used it as an analytical tool in chemistry and physics in 1946. They too were awarded the Nobel Prize in Physics in 1952 [2, 17].

NMR is a phenomenon that occurs when the nuclei of certain atoms are immersed in a static B_0 field and exposed to an oscillating B_1 field [16, 23]. In a sample unit, the nuclei are in equilibrium with the stationary field when they are completely aligned. When a B_1 field is applied, they will be disturbed from equilibrium. After the RF transmitter is turned off, they will begin to return to equilibrium by transferring energy to the surrounding molecular environment. This process is called 'relaxation' [18].

The B_1 field is a sine-wave signal generated by an RF transmitter (RF pulse). It is applied to stimulate the spins, which would generate a decaying resonance signal known as a Free Induction Decay (FID). Figure 1.1 is the example of a FID [23].

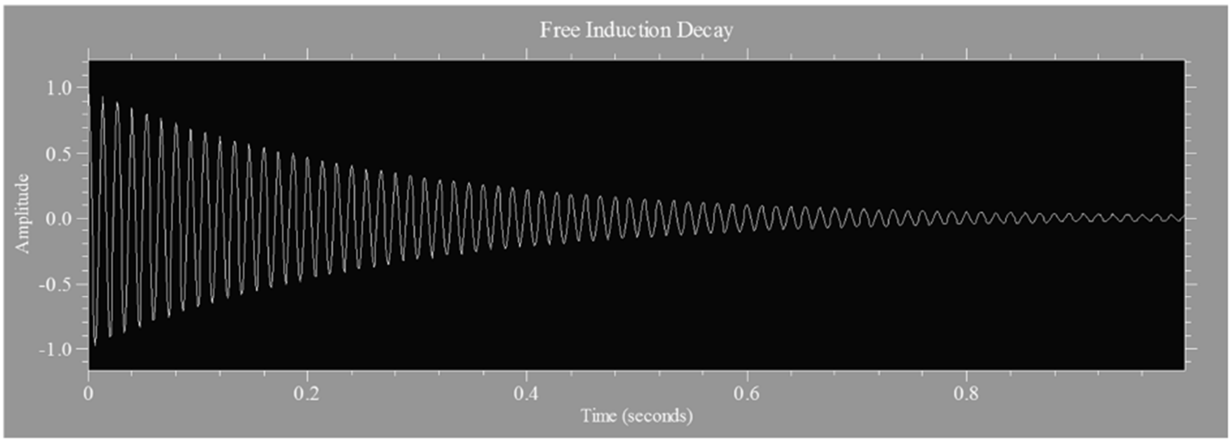


Figure 1.1: An FID signal.

1.3.1 Phase Coherence

Before the RF pulse is applied to the system, nuclei are in its characteristic frequency, but are out of phase. At a certain time when the RF pulse is transmitted to the system, all the nuclear magnetic moments become coherent. This forces the magnetisation vector M to spiral down around the z-axis as shown in Figure 1.2.

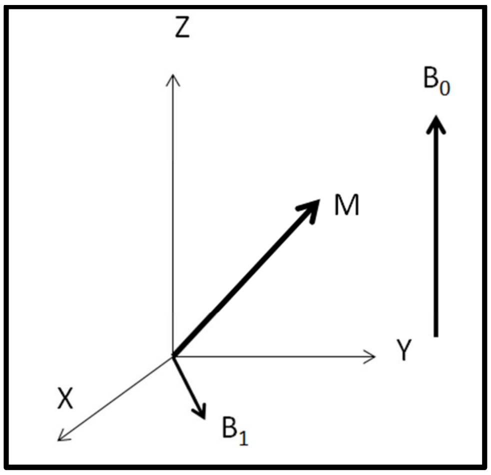


Figure 1.2: Magnetisation vector M spiralling down into the x-y plane due to an applied RF pulse.

The magnetisation vector now precesses in the x-y plane at the Larmor frequency. When it reaches the x-y plane, the RF field is removed, and the precessing magnetisation can then be observed as an induced Electromotive Force (EMF) in a coil surrounding the sample. At that moment, the phase coherence of the nuclear spins is lost. The nuclear magnetic moments return to random arrangement around the z-axis. In other words, there is no net x-y magnetisation. This process is known as the ‘relaxation process’ and is recorded by the NMR spectroscopy.

In summary, the NMR RF transmitter makes the excitation of nuclei at different times as possible while the phase coherence is maintained. Therefore, the RF pulse transmitted to the magnet system must be able to maintain coherence before and after switching to other frequencies.

1.4 Thesis Motivation

Zhang and Dong constructed a multi-channel adjustable frequency system in their research [66]. They used a NOIS II processor from Altera Corporation. In their paper, they proposed a custom DDS module, which contains three output channels. Nios II is powerful but the application of Nios II in their proposed system has many redundancies in the chip because plenty of features remain unused. A similar solution can be implemented by using low-level FPGA devices such as the Cyclone series from Altera or Spartan series from Xilinx to optimise the utilisation of the processing unit. Furthermore, their proposed DDS system operates at 210 MSPS, and the DDS system is classified as a low speed system. The limitations of their system lie behind the motivations of the research work presented in this thesis.

1.4.1 Challenges of an NMR RF Transmitter for NMR Applications

The design of an NMR RF transmitter based on an 'All in One' modular concept is difficult to implement by fitting multi-DDS cores into a single chip. The challenges to the research are because of:

1. Many NMR RF transmitters are formed by different combinations of electronic components such as DDS chips, memory chips and so on. The present NMR RF transmitter uses too many external components, which can be substituted by an FPGA device. This traditional design of a standard NMR RF transmitter limits the improvement and enhancement of the existing system in the future [33, 41, 47, 63].

2. Various commercial products and research works on NMR RF transmitters are carried out by utilising external DDS chips. The advantage of utilising the additional DDS chips is that it could make the design process of the system easier. Unfortunately, the system encounters phase coherence lost after the switching of an output signal from one frequency to another [38, 39].

3. Many present commercial products and academic researchers emphasise a 1 GHz sampling rate or lower for the RF transmitter. The achievable throughput is still far away from the possibility of a high-resolution digital RF transceiver. The hardware components, which are used in the system, have restricted the maximum signal frequency to a 1 GHz sampling rate. Recently, advances in electronics and computing

make it much easier to design a more compact system. The activities within this area have increased to make the 2 GHz signal sampling feasible [7, 8, 25, 40, 41, 45, 46, 47, 63].

1.4.2 Limitations of Current of RF Transmitter

1.4.2.1 Deficiency of NMR RF Transmitter

Most of the NMR RF transmitters use discrete components to build up an entire RF system. The implementation is deficient because the circuitry often takes up a lot of board space. It is also prone to signal drift, interference, distortion and so on. Moreover, these NMR RF transmitters are bulky, semi-portable, and more costly. Discrete components cause the system board to have a complicated circuit design, and the whole RF system lacks visibility to control signals flexibly. These factors are the motivation to look for alternative solutions.

From commercial and industrial points of view, it is necessary to have a flexible design that can be adapted easily to suit different requirements. However, this is hard to achieve because the performance of a DDS system relies on the hardware components on the circuit board. If further improvements are needed, a system upgrade might not be achievable if the forthcoming works are beyond the ability, which the components could support. Furthermore, the system board may not be able to extend its life cycle if damaged or missing parts on the board become no longer available. At that moment, the whole system needs to be re-designed to meet the new requirements [33].

Many researchers have integrated an FPGA into the RF system. However, not all the modules within the RF system are implemented on a single FPGA chip. In this case, the FPGA chip is only used for minor processing purposes such as signal latching, digital switching, clock synthesis and so on. External DDS chips are still applied in the RF system to generate high-speed sine-waves. A similar methodology has been proposed and in work by Takeda. In his paper, he utilised an FPGA chip together with an external DDS chip, AD9740, to generate a sine-wave signal [41].

1.4.2.2 NMR RF Transmitter with External DDS

Another publication of Takeda describes, "A highly integrated FPGA-based NMR spectrometer". His concepts can be achieved by using an FPGA device to incorporate a pulse programmer and a receiver. A DDS interface, which is connected to an external DDS chip, could support the high frequency generation of up to 400 MHz. His original pulse programmer used several tens of integrated circuits on a few circuit boards and there are numerous wire connections among them [41].

In the latest NMR products, FPGAs have been added to the products of several companies. These companies used the FPGA devices as digital signal processing units. A general-purpose pattern generator system known as the "Programmable transistor-to-transistor logic (TTL) and DDS RF Pulse Generator" has been designed and manufactured by SpinCore Technologies Inc. The PulseBlasterTM processor provides all the timing for pulse synchronisation. However, a DDS chip is used to provide the RF excitation from DC to 300MHz [39].

Another FPGA-NMR product by SpinCore Technologies Inc. is a radio processor called the "Complete RF Acquisition and excitation system with Digital Detection, Real-Time signal Processing and Signal Averaging". It is a general-purpose, broadband, RF data acquisition and excitation system. It is a complete NMR subsystem with a frequency range from 0 to 100 MHz. A DDS chip is used to generate an RF output with 14-bit resolution and up to 300 MSPS clock rate [38].

The Magritek Kea2 is another FPGA-based NMR spectrometer system. It could be found on a T-Rex v2.0 board, which was developed at Victoria University of Wellington. This board utilises an AD9910 1 GHz sample rate DDS chip to generate a sine-wave from DC to 400 MHz [27, 47].

The disadvantage of using multiple DDS chips is the loss in phase coherence especially after the sine-wave frequency is varied. Usually, an external DDS chip comes with a built-in digital-to-analog converter (DAC). Numerous commercial products operate at sampling frequencies of up to 1 GHz because it is the maximum clock speed of the built-in DAC available on the market. The sampling frequency can be improved to 2 GHz if a dual data port DAC chip is used together with an FPGA chip. This DDS system is currently on the T-Rex v4.0 board. It is fast enough to carry out a 2 GHz DDS operation [46].

1.4.2.3 FPGA-based NMR RF Transmitter

Some companies like SpinCore Technologies, Agilent Technologies, and Magritek have introduced RF transceivers with an on-board external DDS chip. This structure can support a few hundred MHz to 1 GHz sampling frequency [3, 27, 38, 39]. Huang, Ren, and Mao from Beijing Institute of Technology proposed a method in DDS design by using discrete components (not the FPGA device). Their DDS system is merged with phase lock loop (PLL) outputs to produce a signal generator. Their signal generator can operate at 2728 MHz sampling rate and is capable of generating a signal of up to ~341MHz [66]. However, the entire system is bulky. The size of the system can be reduced by using an FPGA device to replace most hardware components.

The invention of a pulse generator should not simply rely on a faster DDS chip. The design should consider using a customised DDS module in an FPGA chip to achieve better productivity than a system built with many external DDS chips. A DDS module is dependent on the frequency of a DAC chip. All the data generated by the DDS module must synchronise with the conversion process of the DAC chip. Thus, the research must focus on the methodologies to attain such a high clock rate. The system should benefit from the maximum potential across the device.

Recently, many researchers have exploited an FPGA chip to develop an NMR transceiver. Nevertheless, the FPGA chip is not fully utilised to make a transceiver. External DDS chips are not replaced by the FPGA chip due to the DDS core being generated by an FPGA chip such as Xilinx FPGA, which could only support up to 250 MHz. It could not meet the trend of the DDS system to operate at 1 GHz.

1.5 Research Goals

An overall goal of the research is to investigate a new approach to the NMR RF transmitter. The detail research objectives are as below:

1. Design a high-speed NMR RF transmitter. The goal is to explore the potential of a built-in DDS core of an FPGA chip to generate a dual-channel output at a 1 GHz sampling frequency of each. This output will be processed by a dual-input DAC chip for a total of 2 GHz sampling rate. The entire system should be able to generate the output signal ranging from 100 kHz to 750MHz. To accomplish this goal, it is necessary to study and to investigate the structures of these features to see how they could be utilised as a main core in a pulse generator. However, due to the limitations of FPGA to go beyond an allowable clock speed in DDS, the challenges of this problem lead to four objectives as below:

   a. Design a new topology for an NMR RF transmitter by using Matlab-Simulink.

      The Simulink is used to design the structures of a high-speed RF transmitter. The structures are verified through the Matlab-Simulink simulation.

   b. Simulate the designs in 1(a) above in the register-transfer level (RTL-level) simulation by using the Xilinx VHDL compiler.

      The simulations in Matlab do not consider the timing constraints within the DDS system. Matlab simulations are used to verify the functionality of the design and to analyse the flow from input to output. Therefore, it is necessary to simulate the design in RTL-level to ensure the behavioural structures meet the timing constraints.

   c. Design a new methodology to run a dual-output DDS module at 1 GHz each. The 1GHz frequency is beyond the clock limitation of an internal DDS core in an FPGA chip [29, 44].

      According to the datasheet provided by Xilinx, a DDS core has a single sine or/and cosine output signal. The DDS core can operate at a maximum clock of 250 MHz. A 250 MHz clock speed is insufficient to generate outputs at 1 GHz sampling frequency. Consequently, the proposed custom DDS system is expected to quadruple the clock speed of the system by implementing multiple DDS core modules within the system.

d. Analyse and investigate the relationship between Nyquist zones and output frequencies.

   If a desired signal is beyond 125 MHz, an alternative solution should be investigated. The new solution should be able to generate the samples of high-speed signal (outside of the 1st Nyquist zone) by using its folded frequency.

e. Develop a custom signal generator by using a distributed memory block.

   A signal in any form can be generated by using this method because it is DDS free and fully customised by users.

f. Develop an innovative methodology to implement an RF generator.
   The RF generator should support switching of four frequencies to a single output. This solution can maintain phase coherence of a signal when the signal is switching from one frequency to another.

2. After the simulations, the next task is to transfer the simulated solutions to a prototype board. This section leads to four objectives as below:

   a. To configure the on-board DAC and clock chip.

      The evaluation board could not function properly without the configuration of DAC and clock chip. The DAC chip should be programmed to enable a 2 GHz processing rate. It must be able to convert dual-input digital data from DDS to a single analog output at a 2 GHz sampling rate. The clock chip should be programmed to generate a 100 MHz clock to an FPGA chip and a 2 GHz to a DAC chip.

   b. To transfer the VHDL coding of an 8-core DDS module to a prototype board.

      DDS modules will be modified to support multi-frequency generation. The analog signal obtained from the evaluation board will be measured by using a Tektronix MSO 4204 Mixed Signal Oscilloscope.

c. To insert a distributed memory block into the system.

   A coefficient file with custom data can be loaded into a distributed memory block. The data will be retrieved and sent to a DAC chip. The output is measured by using a Tektronix MSO 4204 Mixed Signal Oscilloscope.

d. To examine and to observe the phase coherence of the system.

   Once the system manages to generate an output signal, the phase coherence of the signal is tested. The results are observed and recorded in Chapter 4.

Notes:

Theoretically, the Nyquist frequency of an 8-core DDS system is 1 GHz because the output signal is at a 2 GHz sampling rate. However, it is impossible to generate an output with 1 GHz because an ideal low-pass filter does not exist in real life.

1.6 Organisation of Thesis

Chapter 2: Background

Chapter 2 contains the essential background of an NMR RF transmitter and an FPGA device. The review covers the preliminary concepts of a standard RF transmitter, and the basic components used to form a DDS oscillator. The background of an FPGA device includes the fundamental concepts and the architecture of the device is stated in this section.

Chapter 3: Design and Development

Chapter 3 emphasises the design and development of an 8-core DDS system. The research begins by using Matlab-Simulink to simulate a single DDS system. From the single DDS system, more modules are added to create an 8-core DDS module. After that, the 8-core DDS module is modified to form a multi-frequency generator, which could support the switching of an output signal between four frequencies. After the Matlab simulation, Simulink designs are transferred to the RTL-level simulation. The simulation is done by utilising Xilinx ISim simulator. A distributed memory block was inserted into the system to generate a random signal.

Chapter 4: Hardware Implementation, Testing and Results

Hardware testing of an 8-core DDS system is implemented on an evaluation board. The on-board DAC and clock chip are configured so that the entire system can generate an analog signal at 2 GHz sampling rate. After the configuration, several random frequencies are chosen to test the developed system on the evaluation board. Then, the phase coherence of the output signal is tested and observed. The last part of the hardware testing is to generate a signal by using a distributed memory block. All the results are recorded and presented in this chapter.

Chapter 5: Conclusion and Future Work

Chapter 5 is the final chapter of the thesis. A conclusion is presented to wrap up the entire thesis. The concluding part concerns the future work, which can be carried out to enhance and improve the NMR RF transmitter developed in this thesis.

Chapter 2

Background

This chapter reviews the background of an NMR RF transmitter, which is used as a pulse generator in the NMR transceiver. The background covers the existing techniques, essential background, and basic concepts of an NMR RF transmitter. Lastly, it discusses the background and basic architecture of an FPGA device.

2.1 NMR Transceiver

An NMR spectrometer requires a magnet, a coil, and some electronic instrumentation such as a transceiver and a computer. These instruments provide the stimulus and to be subsequent detect the FID signal [37]. Figure 2.0 depicts a simplified NMR system.

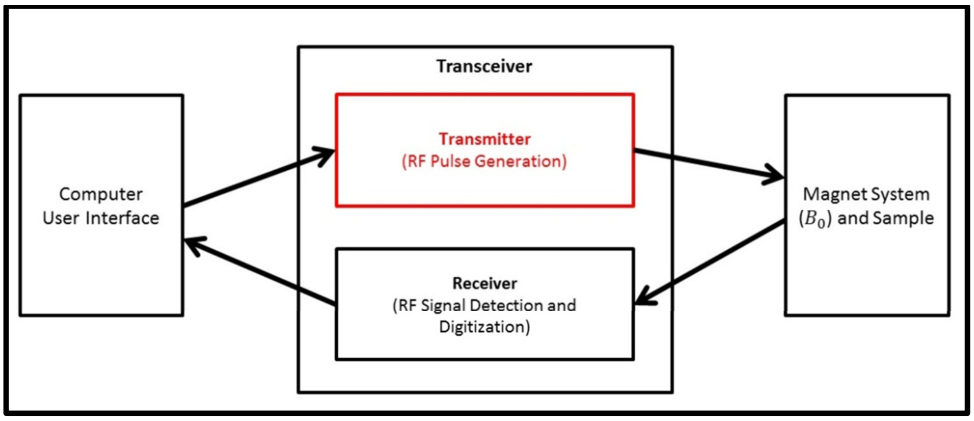


Figure 2.0: Block diagram of an NMR system [5].

Typically, the most expensive and difficult part of the system is the magnet system. The cost of the system is proportional to the magnetic strength. The better the sensitivity and spectral resolution of the system, and hence more expensive is the magnet.

The first block to the left in Figure 2.0 refers to the digital part of the system. This usually involves a computer to control all the sequencing of events, processing and storing of data. It serves as the medium of communication between the user and the spectrometer. The transceiver is responsible for the generation and detection of analog RF signals. There are two modules in the transceiver, a transmitter, and a receiver. The transmitter performs the tasks of generating real-time RF pulses to stimulate the B_1 coil in the magnet system. This part is the main research core of this thesis. The receiver acts as a detector to detect the returning FID. Firstly, the receiver amplifies the small RF FID signal and then digitises the signal. The digital data is processed to capture the NMR information. However, this thesis focuses only on the NMR RF transmitter, which is highlighted in red. The computer interface, magnet system and receiver module in the transceiver are not discussed in this thesis.

2.2 Fundamentals of the Sinusoidal Oscillator

A sinusoidal oscillator is also known as a sine-wave generator and is used to generate a sine waveform. The synthesis process is known as digital sine oscillation. The summary below shows examples of sinusoidal oscillators presented over the past few years.

2.2.1 Recursive Oscillators

Recursive oscillators use the traditional methods to produce cosine or sine signals. In the early 1900s, the German physicist Heinrich Barkhausen modelled an oscillator as an amplifier. His model has the output feedback to input via a phase shifting network. The Barkhausen criteria for the oscillator require the function gain to be equal to one, and the total loop phase shift needs to be a multiple of 2π radians [43].

As a result, several oscillators have been designed with this approach such as [43]:

- Direct Form oscillator
- Coupled Form Oscillator
- Digital Waveguide Oscillator and many more

This thesis concentrates on the development of a sinusoidal oscillator using the built-in modules of an FPGA device. The sinusoidal oscillator can be implemented by either using Coordination Rotation Digital Computer (CORDIC) or DDS modules in the FPGA devices.

2.3 Direct Digital Synthesis

DDS usually refers to look-up table (LUT) based sinusoid generators. A DDS is a technique to generate frequency- and phase-tunable output signals. It uses digital data processing blocks with a fixed-frequency precision clock to generate a sinusoidal signal [5]. In Figure 2.1, a simple DDS can be created with a precision reference clock, an address counter, a Programmable Read-Only Memory (PROM) LUT and a DAC chip. In this basic system, PROM stores the amplitudes of a complete sine-wave. The address counter is used to generate memory addresses to access the PROM. A memory address is an identifier for a memory location in the PROM. The memory address will retrieve the digital amplitude from the PROM to the DAC chip.

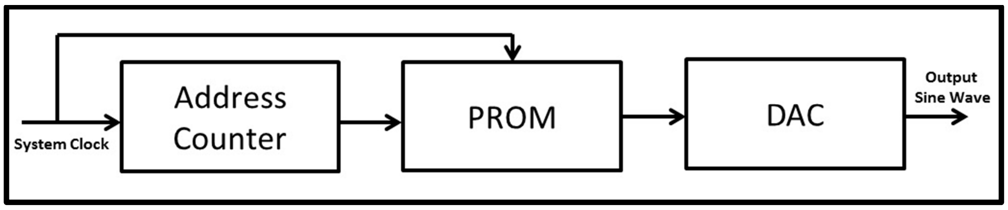


Figure 2.1: Basic DDS structure [8].

There are two factors to determine the output frequency generated by a DDS block:

- Frequency of the DDS system clock
- Sine-wave step size

The advantage of using the DDS based system is that the simplistic architecture of the DDS system is less susceptible to the tuning flexibility. The reason is the output frequency can only be changed by modifying the system clock speed or by re-programming the step size in the PROM.

2.3.1 Phase Accumulator

The main part of the DDS system is the phase accumulator whose contents are updated once on each clock cycle. Each time the phase accumulator is triggered, the tuning word or phase increment, M is added to the contents of the phase accumulator. Let us assume that the initial content of the phase accumulator is 00…00 and M=00… 01. The phase accumulator is updated by 00…01 on each clock cycle.

A phase accumulator replaces the address counter in Figure 2.1 to introduce a function known as a 'phase wheel'. This 'phase wheel' can be visualised as a vector rotating around the phase circle as illustrated in Figure. 2.2.

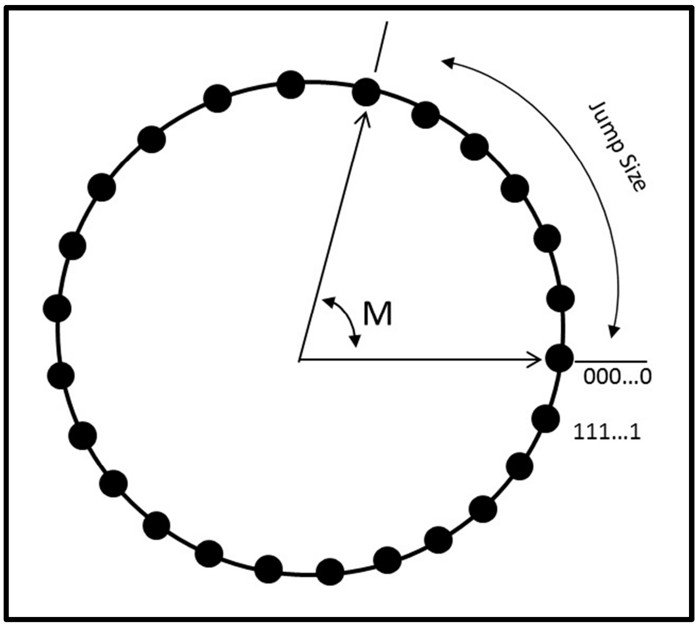


Figure 2.2: Digital phase wheel [5].

In Figure 2.2, every point of the phase wheel corresponds to the sample phases of a full sine-wave. When the vector rotates around the wheel, a phase accumulator is visualised to generate equivalent phases of a sine-wave signal. One revolution of vector rotation of the phase wheel at a constant speed produces a finished cycle of an output sine-wave. The revolution of the phase wheel represents an overflow in the phase accumulator. Nevertheless, this complete output sine-wave is linear and is unsuitable to be applied directly to generate a sine-wave signal. Hence, a sine LUT is used to convert the output phases to sine-wave digital amplitudes. A DAC chip is then used to convert these amplitudes to an analog signal [5].

Each time the phase accumulator receives a clock pulse, the phase accumulator increases the stored number in the phase register by adding a tuning word to that register. The magnitude of phase increment is based on the frequency tuning word. This frequency tuning word forms a ‘hopping’ process to skip the N-bit phase points of the phase wheel. These N-bit phase points are known as phase step size or phase jump size. The control over the jump size constitutes the frequency tuning resolution of the DDS system. The larger the jump size, the faster the phase accumulator overflows to complete the generation of a sine-wave cycle [5].

Any changes in the frequency tuning word result in immediate changes in the output frequency. The output frequency is inversely proportional to the number of samples per cycle. When the output frequency increases, the total samples per unit cycle decrease. However, the Nyquist sampling theory dictates a complete sine-wave must have at least two samples per cycle to construct the output waveform. Thus, the maximum frequency of an output sine-wave is half of the DDS system clock.

In summary, the frequency of a sine-wave signal can be expressed by the equation (2.0) below:

$$f_{out} = \frac{M}{2^N} f_{system}$$

$$So, M = \frac{f_{out}\ x\ 2^N}{f_{system}} \tag{2.0}$$

$f_{out} = the\ frequency\ of\ an\ output\ sine - wave\ signal$

$M \quad = frequency\ tuning\ word$

$N \quad = N - bit\ of\ a\ phase\ accumulator$

$f_{system} \quad = system\ frequency$

‘M’ is also known as a digital number for phase hopping in an accumulation. ‘N’ is the total bits of a phase accumulator, so 2^N is the amount of steps available in the accumulator. The division of f_{system} by 2^N represents the frequency resolution of the system.

2.3.2 Phase-to-Amplitude Converter

In this thesis, the phase-to-amplitude converter or LUT is configured as a sine LUT. If B represents the width of a truncated phase, a sine LUT can be simply a read-only memory containing 2^B samples that create the sample-domain waveform from the truncated phase received from the phase accumulator. In other words, it converts phases from a phase accumulator to amplitudes. These digital phases are also the memory addresses used to retrieve the corresponded amplitudes from the sine LUT.

A PROM can be used to implement a sine LUT. A PROM based LUT is easily implemented due to the simplicity of the PROM circuit. A LUT contains one complete cycle of the waveform to be generated. The LUT translates the truncated phases into the sample amplitudes which are then converted to an analog signal by DAC.

If a high precision signal is required, the LUT needs a very large capacity as more bits are required to store the amplitudes. This increases the hardware cost and the implementation is limited by the capacity of the PROM. Practically, only the ‘high bits’ of an accumulated phase are used as a memory address to access the LUT. Generally, these high bits are known as the truncated phase. The size of a LUT is 2^B words, therefore $B < N$; N is the width of accumulated phases.

2.3.3 Phase Truncation

The N-bit of a phase accumulator determines the frequency resolution of a DDS system. The greater the value of N means the better the resolution of the system. However, the high frequency resolution of the DDS system can increase the capacity of the LUT block. Considering that each entry of the LUT block has an N - bit of precision data, it then requires a huge capacity of LUT. For example, for the LUT block of Xilinx FPGA devices, the phase truncation is done in the LUT block before converting the phase to amplitude. Generally, the width of a phase accumulator block of Xilinx devices can be up to 48 bits. If all the 48 bits are used, then the LUT block requires 281.147 X 10^{12} bits of memory.

In a practical DDS system, all the bits of the phase accumulator are not passed on to the LUT. These bits are truncated, leaving only some of most significant bits (MSB) for amplitude conversion. This method reduces the size of the LUT and does not affect the frequency resolution.

A phase truncation technique uses a fraction of the MSB of the accumulator output to provide phase information. If a phase accumulator has data length = N-bit and no phase truncation has applied to the accumulated phases, then the LUT needs 2^N depth to store the amplitudes. If the phase truncation is applied, the lower N'-bit of the accumulated phase is ignored. Then, the accumulated phase remains the upper N-N'-bit. This N-N'-bit is the data width of a truncated phase which means N-N' = B. Therefore, the sine LUT needs only $2^{(N-N')}$ or 2^B bits to store the data.

The implication of phase truncation can be depicted by the concept of the 'digital phase wheel'. In order to give a clear explanation of the digital phase wheel, assume a simple 8-bit phase accumulator of which only the upper 5 bits are used as input to sine LUT. The result is the phase resolution represented by the outer circle tic marks. After the truncation, the resolution is identified by the inner circle of tic marks [5]. The phase resolution of an 8-bit phase accumulator is $360^o/2^8 = 1.406^o$.

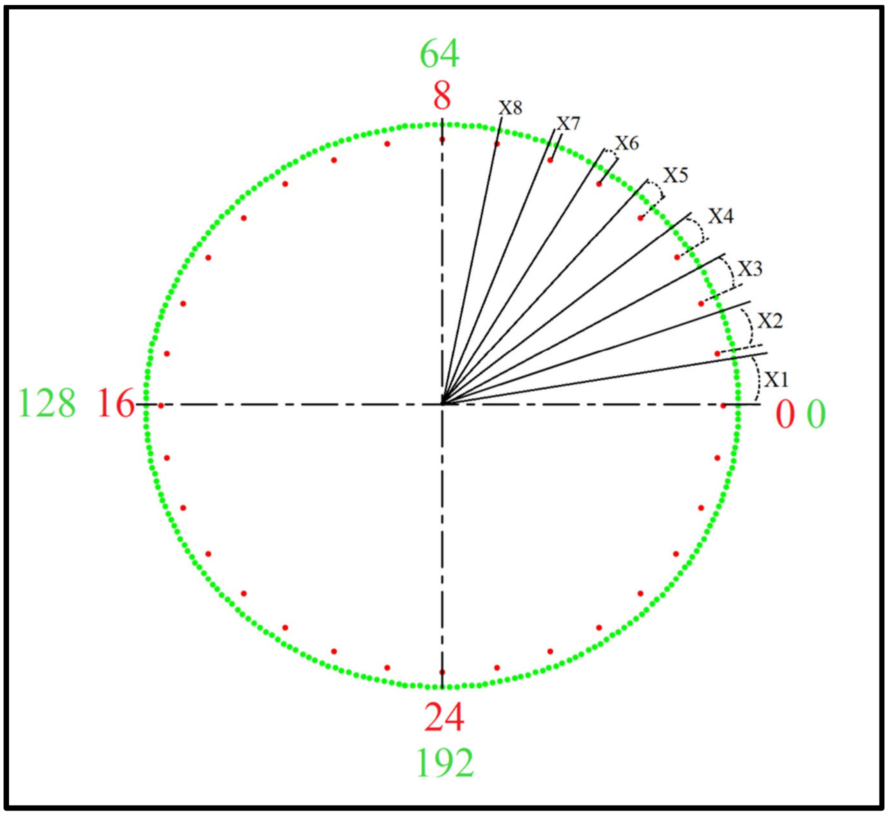


Figure 2.3: Phase truncation error [5].

If a tuning word of 7 is selected, the accumulator would have increments of 7. In Figure 2.3, the first eight phase angles correspond to the count steps of the phase accumulator. After the first count on the outer circle (increment by 7), the phase step falls short of the first inner tick mark. It causes a discrepancy between the phases of the outer circle (8-bit resolution) and inner circle (5-bit resolution). The phase error due to this discrepancy is 7 X 1.406^{o} = 9.842^{o} as noted by X1 in Figure 2.3.

After the 2nd phase step, 7 more counts on the outer circle. The accumulated phases fall between the 1st and the 2nd tick marks on the inner circle. Again, there is a discrepancy between the phases of the outer circle (8-bit resolution) and inner circle (5-bit resolution). The phase error is 6 X 1.406^{o} = 8.436^{o} as noted by X2 in Figure 2.3.

The phase errors due to the discrepancy between the inner and outer circles on every clock cycle are listed below:

On the 3rd phase step (X3), phase error is 5 X $1.406^o = 7.03^o$

On the 4th phase step (X4), phase error is 4 X $1.406^o = 5.624^o$

On the 5th phase step (X5), phase error is 3 X $1.406^o = 4.218^o$

On the 6th phase step (X6), phase error is 2 X $1.406^o = 2.812^o$

On the 7th phase step (X7), phase error is 1 X $1.406^o = 1.406^o$

On the 8th phase step, phase error is 0 X $1.406^o = 0^o$. The reason is that the accumulator phase (outer circle) and the 5-bit resolution phase (inner circle) coincide. As a consequence, there is no discrepancy between these two circles and so no phase error. This pattern continues as the accumulator increments by 7 (each time) which are represented by the outer circle.

Hence, the phase errors due to the truncation will result in errors in amplitude during the phase-to-amplitude conversion. Fortunately, the phase errors are periodic. In other words, the accumulator phase and truncated phase will coincide after some revolutions of the phase wheel regardless of the tuning word.

Although the phase truncation leads to the truncation error, it can be reduced by applying a noise shaping technique. The noise shaping technique used in this Master research is the Taylor Series Corrected DDS.

- Taylor Series Corrected DDS

In the Taylor series corrected DDS; the previously discarded fractional bits have been used to calculate corrections to the phases. The corrections are added to the look-up table values to produce outputs with very high spurious free dynamic range (SFDR). This noise shaping technique can maintain the LUT to a reasonable size. The Taylor series corrected approximation is the expansion of a function of:

$$f(x) = f(a) + f'(a)\frac{(x-a)}{1!} + f''(a)\frac{(x-a)^2}{2!} + \cdots \qquad (2.1)$$

Hence, the Taylor expansion of $\sin\left(\frac{\pi}{2}q\right)$ (quarter-wave storage in LUT) is as below:

$$\sin\left(\frac{\pi}{2}q\right) = sin\left(\frac{\pi}{2}r\right) + k_1(q-r)\cos\left(\frac{\pi}{2}r\right) - \frac{k_2(q-r)^2\sin\left(\frac{\pi}{2}r\right)}{2} + \cdots \qquad (2.2)$$

The total q-bit phase is divided into the upper phase ‘r’ and the lower phase ‘q-r’ [48]. After the phase truncation, only the upper phase ‘r’ is used for the phase information, so the Taylor series is performed around ‘r’.

The phase values have angular units so the constant k_n (n=1,2,3…) is used to adjust the units of each series term. k_n includes a multiple of π/2 to compensate for the phase units. It is also used to make sure the expression is always positive [15]. From equation (2.2), k_1 is $\pi/2=1.57$ and k_2 is $-(\pi/2)^2 = -2.47$. The Taylor series approximation takes up to three terms. In fact, additional terms can be employed but the contribution to the accuracy is very small [48]. This architecture needs two read-only memory (ROM) blocks to store $sin\left(\frac{\pi}{2}r\right)$ *and* $cos\left(\frac{\pi}{2}r\right)$. The unit conversion factor k_n is included in the values stored in the sine and cosine ROMs.

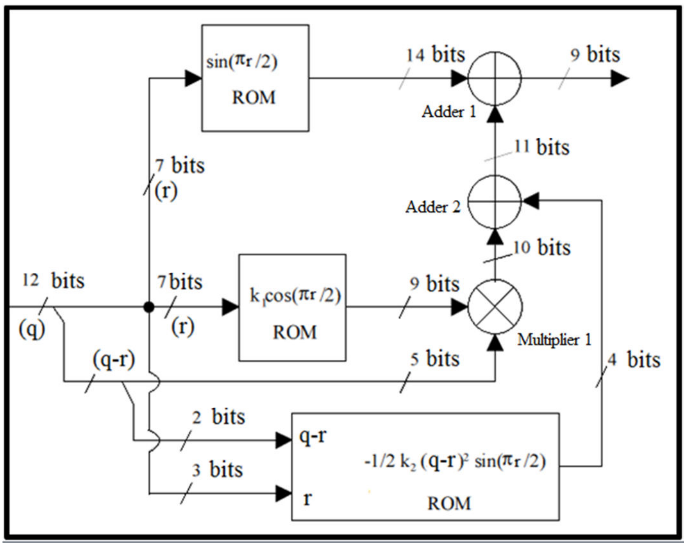


Figure 2.4: Taylor series approximation for quarter sine converter [48].

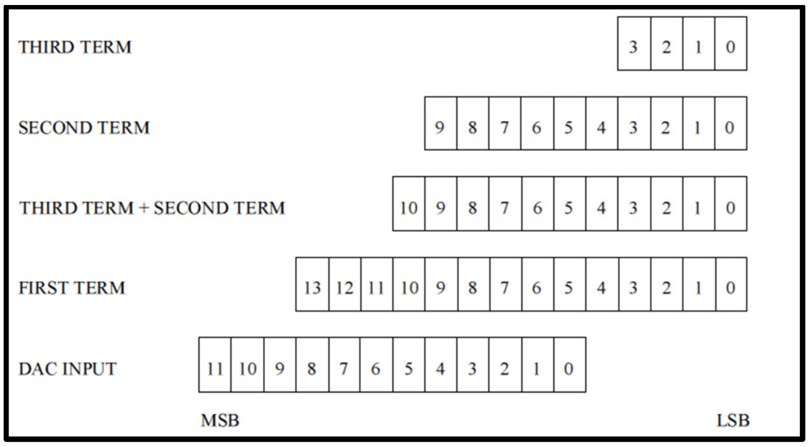


Figure 2.5: Relative bit position of data word implemented in Figure 2.4[48].

For example, an input phase (q) has 12 bits (Figure 2.4). The upper 7 bits of ‘q’ are used as the upper phase increment ‘r’. This ‘r’ is transferred to both sine ROM and cosine ROM at the same time. The upper 7 bits are the address signals and hence determine the number of entries in the ROMs. More bits in the upper phase address result in a larger LUT with decreased speed and greater cost. The sin (πr/2) ROM is the first term of the Taylor series expansion. The output from the sine ROM will then sum the other terms from the expansion to re-construct the final phase (Adder 1).

The output of the cos (πr/2) is configured to incorporate the predetermined unit conversion value k_1. The cosine ROM is less significant than the amplitude term. Thus, the output can be at lower accuracy and so 9-bit output number is used. The lower 5 bits (q-r) are used to compute the other two terms used in the Taylor series expansion. The ‘q-r’ is multiplied with the output from cosine ROM to generate the second term (Multiplier 1). The second term has 10-bit output number.

The MSB of output from the ‘multiplier 1’ has a bit alignment with the DAC input Least Significant bit (LSB). Because of the alignment, the output number ranges from 1 LSB to 1/1024 LSB of the final 10-bit DAC input used. Not all the bits after the ‘multiplier 1’ are needed to maintain the accuracy of the system. Therefore, the system truncates the multiplication product to 10 bits. In any case, truncation to 10 bits after ‘multiplier 1’ was used in the example in Figure 2.4. Any bit widths can be used as desired as long as the principles of the design are met.

The third expansion term is computed in a ROM by combining the derivative of sin (πr/2) and (q-r)^2. This is done by selecting the upper 2 bits of ‘q-r’ and the lower 3 bits of ‘r’ as a portion of the address for the ROM. This is possible because the last expansion term only contributes roughly 1/4 LSB to the DAC input (see Figure 2.5). In Figure 2.4, only the ‘r’ is needed for this degree of accuracy. Finally, the third term ROM output will combine with the ‘multiplier 1’ output in ‘Adder 2’. Subsequently, the result will combine with the first term ROM output in the ‘Adder 1’.

2.3.4 CORDIC

CORDIC is an algorithm used for computing the hyperbolic and trigonometric functions, where the initial vector is rotated by a predetermined sequence of sub-angles. The summation of the rotations approaches the given angle by using minimal hardware such as shift, add, subtract and compare [14, 50]. When comparing CORDIC with the ROM-based DDS, CORDIC does not incur an exponential growth of the hardware as the output word size increases. However, it is difficult to achieve high throughput because of the sequential nature of the algorithm [14].

Two architectural configurations are available for the CORDIC core:

1. A fully parallel configuration with single-cycle data throughput at the expense of the silicon area
2. A word serial implementation with multiple-cycle throughput but occupying a small silicon area

According to Volder, an algorithm is derived from the general rotation. A transform rotates a vector (X, Y) through an angle (Ø). This rotation yields a new vector (X', Y') in a Cartesian plane. The equations below represent the condition of the rotation [50]:

$$X' = Xcos\emptyset - Ysin\emptyset \tag{2.3}$$

$$Y' = Ycos\emptyset - Xsin\emptyset \tag{2.4}$$

$$\emptyset' = 0$$

A CORDIC can operate in two major modes. The first is rotation mode and the second is vectoring mode. The CORDIC algorithm introduces a constant scale factor, Z_n, which is only dependent on the number of iterations (n) and can be compensated by multiplying a $1/Z_n$ [10].

2.3.4.1 Rotation Mode

CORDIC rotates a vector by a specified angle, also known as micro-rotations. These micro-rotations are the sequence of successively smaller rotations, each of the $\text{atan}\left(2^{-i}\right)$. The entire process can be expressed as a simple shift and add/ subtraction operations. Equation (2.5) to (2.7) shows the expression for an i^{th} iteration, where i is the iteration index from 0 to n.

$$x_{i+1} = x_i - \alpha_i \,.\, y_i \,.\, 2^{-1} \tag{2.5}$$

$$y_{i+1} = y_i - \alpha_i \,.\, x_i .\, 2^{-1} \tag{2.6}$$

$$\emptyset_{i+1} = \emptyset_i + \alpha_i . \text{atan}(2^{-i}), \tag{2.7}$$

$$\alpha_i = \pm 1 \; indicates\ the\ direction\ of\ rotation$$

If the series of ‘n’ micro-rotations are performed by the CORDIC algorithm, the expression in the n^{th} iteration can be summarised using equation (2.8) to (2.10) below:

$$X' = \prod_{i=1}^{n} \cos\ (\text{atan}\left(2^{-i}\right))(X_i - \alpha_i Y_i 2^{-i}) \tag{2.8}$$

$$Y' = \prod_{i=1}^{n} \cos\ (\text{atan}\left(2^{-i}\right))(Y_i - \alpha_i X_i 2^{-i}) \tag{2.9}$$

$$\emptyset' = \sum_{i=1}^{n} \emptyset - (\alpha_i . \text{atan}\left(2^{i}\right)) \tag{2.10}$$

The explanation above can be simplified by the diagram of vector rotation in Figure 2.6 below.

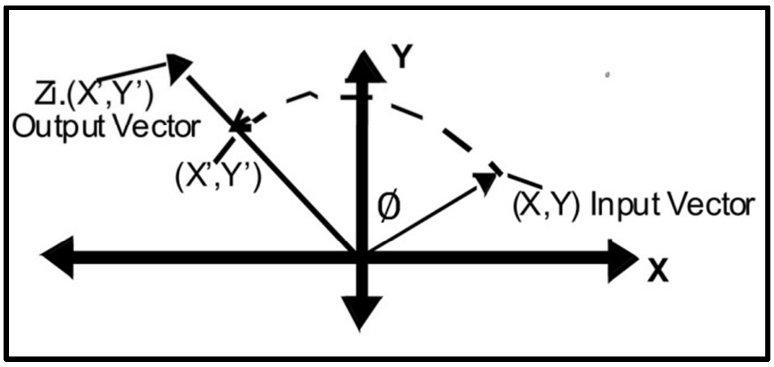


Figure 2.6: Diagram of vector rotation [58, 60].

In Figure 2.6, the vector *(X, Y)* is rotated through the angle (Ø) to a new vector *(X’, Y’).* The vector rotation is dependent on the value α_i . The direction of the rotation α_i should be selected so that $\emptyset'$ converges towards zero. For example, if $\emptyset_{i+1} \geq 0, \alpha_i = -1$ and if $\emptyset_{i+1} < 0$, $\alpha_i = +1$. The coordinate components of the output vector are computed after the given coordinates component of an input vector are rotated to the desired angle (Ø). The equations (2.11) to (2.13) show the expression of these coordinates. [50]:

$$X' = Z_n.Xcos\emptyset - Ysin\emptyset \quad (2.11)$$

$$Y' = Z_n.Ycos\emptyset + Xsin\emptyset \quad (2.12)$$

$$\emptyset' = 0$$

$$CORDIC\ Scale\ Factor, Z_n = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}} \quad (2.13)$$

The vector rotation mode is used for the conversion of polar coordinates to the rectangular coordinates. However, it is also used to calculate the sine and cosine functions.

2.3.4.2 Vectoring Mode

The vector translation is the modification of the vector rotation algorithm. In vector translation, CORDIC rotates the input vector towards the x-axis while accumulating an angle. The attribute of the vector translation can calculate the magnitude and phase of the input vector. Thus, it is used to convert the rectangular coordinates to be polar coordinates. Figure 2.7 below shows the basic operation of the vector translation.

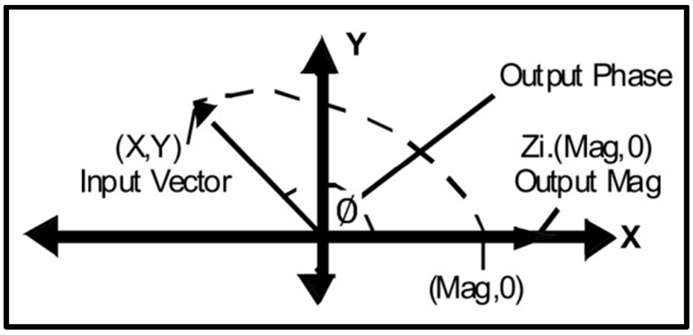


Figure 2.7: Diagram of vector translation [58, 60].

In Figure 2.7, the vector translation rotates the input vector (X, Y) around the circle until the angle aligns the result vector with the X-axis (Y=0) [49]. This is the point where the magnitude X' and angular argument Ø' of (X, Y) are computed. The direction of the rotation α_i should be selected so that Y' converges towards zero. For example, if $\emptyset_{i+1} \geq 0$, then $\alpha_i = -1$ and if $\emptyset_{i+1} < 0$, then $\alpha_i = +1$. The CORDIC equations for the vector translation are [50]:

$$X' = Z_n\sqrt{X^2 + Y^2} \qquad (2.14)$$

$$Y' = 0$$

$$\emptyset' = \operatorname{atan}\left(\frac{X}{Y}\right) \qquad (2.15)$$

$$CORDIC\ Scale\ Factor, Z_n = \prod_{i=0}^{n}\sqrt{1 + 2^{-2i}} \qquad (2.16)$$

Both vector rotation and vector translation are limited to the rotation angles between $\pm 90^o$. This restriction is due to the use of 2^0 for tangent in the 1st-iteration. A complete sine-wave is symmetric from quadrant to quadrant. Every sine value from 0 to 2π can be represented by reflecting or inverting the first-quadrant. If any rotation angles are greater than 90^o, coarse rotation and inverse coarse rotation are needed. The coarse rotation is used to shift the input samples from the full circle into the first-quadrant. The inverse coarse rotation is used to shift the output samples to the correct quadrant [10, 58, 60].

Several functions can be operated by the CORDIC such as:

- Sine and cosine function
- Polar-to-rectangular transformation
- Arctangent function
- Vector magnitude function
- Cartesian to polar transformation
- Inverse CORDIC function and so on

Usually, only sine and cosine functions are used for sinusoidal oscillation [44].

2.3.4.3 Sine and Cosine Function

Sine and cosine functions of a particular input angle can be computed using vector rotation mode to generate a 2π sinusoidal waveform. However, the CORDIC algorithm is only valid for the first-quadrant. Therefore, an optional course and inverse rotation module is used to extend the range of the input angle (X) to the full circle in order to cover the four quadrants [58, 60].

Initialising the input vector component Y=0 into equations (2.17) and (2.18) reduces the rotation mode.

$$X' = Z_n X_0 \cos \emptyset_0 \tag{2.17}$$

$$Y' = Z_n X_0 \sin \emptyset_0 \tag{2.18}$$

By letting $X_0 = \frac{1}{Z_n}$, the un-scaled sine and cosine of an angle argument $\emptyset_0$ will be produced to modulate the magnitude value. A rotator gain is used to scale the output of CORDIC. If the gain X_0 does not produce satisfying results, a single multiplier can be used to place an additional gain constant before the CORDIC rotator. This step can produce un-scaled results.

2.3.5 DAC

The phase accumulator generates phases, which are the addresses for a sine LUT. These digital phases cannot be used in NMR applications because the results are linear and must be converted to the corresponding values of analog voltage or current.

All modules of the DDS system are synchronised to each other by a reference clock. The clock topology allows the phase generation, phase-to-amplitude, and digital-to-analog conversion to operate simultaneously. Usually, a low-pass filter is used to suppress the effects of the image responses in the output spectrum.

2.4 Comparisons between LUT-based DDS and CORDIC Algorithm

The difference between CORDIC and LUT-based generators is that LUT-DDS has an unsigned modulus 2^N accumulator. The accumulator output addresses the phase-to-amplitude conversion. In a CORDIC based generator, the phase accumulator is a 2's complement modulus-π accumulator. The generated phases are in radians, and they can be converted to amplitude by the CORDIC algorithm.

A general CORDIC generator requires two shifters [12]. In the CORDIC system, no multipliers or memories are needed unlike in the ROM-based LUT. This is because the hardware complexity of a CORDIC rotator approximates to a single multiplier. In other words, the CORDIC performs multiplication as part of the rotation operation. So, it gets rid of the need for a pair of explicit multipliers [10]. The other advantages of using the CORDIC based DDS with respect to the LUT-based DDS are that it can achieve both high phase resolution and precision at lower hardware cost [13, 44].

The main problem of the CORDIC based DDS is the limitation of its convergence ranges. The magnitude and the sequential nature of the CORDIC algorithm make it difficult to achieve the higher throughput design. Unlike the ROM-based DDS, all the required sine/ cosine samples of the CORDIC algorithm are limited between $\pm\pi$ [2, 44]. Different approaches have been presented since the last decade. These approaches include the method to expand the convergence range of the CORDIC algorithm. By repeating certain iterations of the algorithm, accurate results can be obtained.

The processing time to compute a better precision result is unnecessarily excessive if a large number of iterations are required [20, 24]. Under these circumstances, the ROM-based DDS is more suitable for higher throughput, and they are easier to implement. The nature of CORDIC induces a quantisation error during the conversion process. However, the quantisation error can be monitored by controlling the total iterations in the CORDIC algorithm.

Due to the natural characteristics of CORDIC, it generates amplitude values by using algorithms. Hence, it needs time for arithmetic computation. In certain architecture types, CORDIC requires a large number of iterations for accurate results. Therefore, the speed is slow and the time delay is high. The negative impacts from the CORDIC algorithms make the system unsuitable for high-speed operation. Hence, LUT methods are generally quicker than this CORDIC algorithm.

2.5 FPGA

In recent years, FPGA has gradually replaced the analog control system in order to manage analog control function digitally. A traditional RF system with analog circuitry has many disadvantages such as lacking control flexibility. On top of that, a close-loop control structure is difficult to implement in a microcomputer system. The tremendous growth in digital communication techniques has resulted in both DDS and FPGA, which are useful, especially in the RF systems [18].

Nowadays, many fields use FPGAs to implement custom instruments because of the merits of these advanced chips such as high flexibility, programmability, and full customisation. The development of nanotechnology enables more logic gates, and Intellectual Property (IP) cores are integrated into a single small FPGA chip. The integrated features in the FPGA device can be tailored and customised depending on the design specifications. These key features enable the FPGA device to achieve fast real-time processing on a single chip. An embedded programmable operation system can be integrated into a high-level FPGA device, called a System on Programmable Chip (SOPC), to provide steadier performance. Another advantage is to satisfy the specification of the design by reconfiguring the FPGA device with different IP cores. This cannot be done if a dedicated hardware is used because it would likely require component changes. Hence, the FPGA device blurs the boundary between software and hardware.

FPGA devices are flexible; its user-customisable functions can be added or removed as required by the project objectives. This is something that cannot be done in microcontrollers in general. For executing digital signal processing (DSP) algorithms, FPGAs are preferable because they are better suited for real-time applications. Besides that, FPGAs can optionally process multiple random signals in parallel or take sequential functionality like adding soft processor cores. However, microcontrollers always operate in a sequential manner. As an example, microcontroller-computing expressions with more than two terms need a sequence of arithmetic and/ or logical operations. However, FPGAs can increase the performance of systems by having such computations in parallel with a reasonable amount of combinational logics.

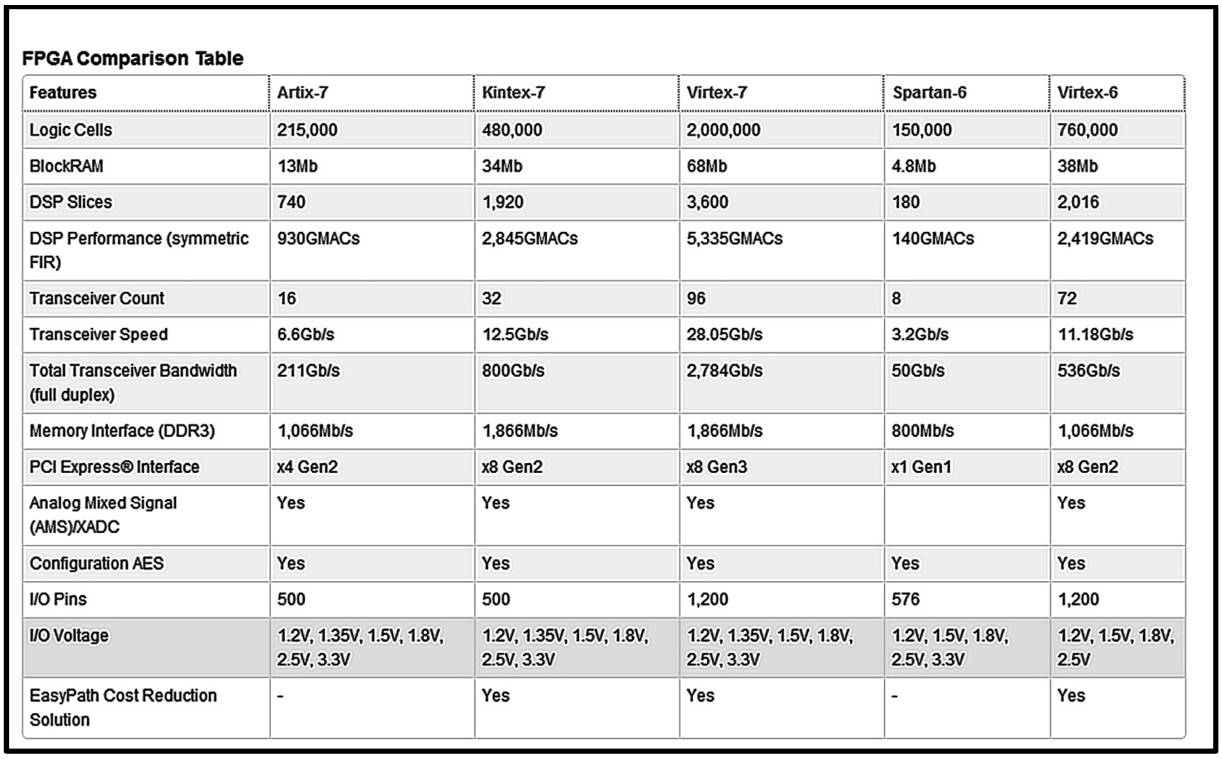
Microcontrollers change too often and there is a lot of rework required in order to keep pace with changing technology. This is necessary to save the design from being obsolete. The lifetime of FPGA-based development is longer, and can be adopted for advanced chip if needed. Other than that, microcontrollers are low-cost (much lower than FPGAs) and are more power efficient. This is true for small applications and large quantities. Microcontrollers are preferable when a project requires few or no hardware multipliers. If the project involves a vast number of gates to implement multiplication, FPGAs are more suitable for this application.

Usually, microcontrollers are sufficient for specific application requirements. The required application is normally developed by using software. In developing such a system, hardware interfacing is required. Generally, it is implemented using standard specialised chips. Unfortunately, it leads to a fixed system. This fixed system cannot be altered without hardware design. However, FPGAs are very versatile [27, 42].

FPGA devices are always used as a co-processor for existing processors, accelerating time-critical operations or extending the operations of the standard ALU [38, 39]. FPGAs have many built-in devices, which are very convenient for the research presented here. The devices have serialisation or de-serialisation (SERDES) for data transmission at 1 GSPS, LUTs, phase accumulators etc. Therefore, no additional circuitry is required to implement these devices. Although phase accumulators and LUTs can be created by using a microcontroller, the microcontroller needs code to develop these two features. In contrast, these features can be added to the FPGA devices using the Xilinx IP Core generator.

The projects implemented using FPGA are more costly than a simple microcontroller unit (MCU) is. This is because programming an FPGA device to operate in a specific task takes a lot longer than programming an MCU. For an MCU such as a peripheral interface controller (PIC), the packages that are geared towards a certain task can be purchased from the manufacturer. However, developing efficient VHDL code can be very time consuming but the code developed for FPGA is reusable if the hierarchy of the VHDL program is planned and written properly. Hence, the FPGA is suitable for design and development processes or for use in products that have a high degree of complexity with only low demand. Once the demand increases, mass production is necessary, FPGA will be replaced by Application-Specific Integrated Circuits (ASIC) for better production cost.

A single FPGA chip can feed many modules such as DDS, DDC (Direct Digital Control), PLL, Finite Impulse Response (FIR), and Cascaded Integrator-comb (CIC) filtering, SERDES, DCM (Digital Clock Manager) and so on for multi-carrier applications. These modules will be very useful for developing a complete NMR electronic instrument on the same evaluation board in the future. Furthermore, one or multi-modules can be implemented in parallel on an FPGA device so that multiple channels can be attained with or without additional circuitry. Therefore, an FPGA platform is preferable in this research as a target device.



FPGA Comparison Table

Features	Artix-7	Kintex-7	Virtex-7	Spartan-6	Virtex-6
Logic Cells	215,000	480,000	2,000,000	150,000	760,000
BlockRAM	13Mb	34Mb	68Mb	4.8Mb	38Mb
DSP Slices	740	1,920	3,600	180	2,016
DSP Performance (symmetric FIR)	930GMACs	2,845GMACs	5,335GMACs	140GMACs	2,419GMACs
Transceiver Count	16	32	96	8	72
Transceiver Speed	6.6Gb/s	12.5Gb/s	28.05Gb/s	3.2Gb/s	11.18Gb/s
Total Transceiver Bandwidth (full duplex)	211Gb/s	800Gb/s	2,784Gb/s	50Gb/s	536Gb/s
Memory Interface (DDR3)	1,066Mb/s	1,866Mb/s	1,866Mb/s	800Mb/s	1,066Mb/s
PCI Express® Interface	x4 Gen2	x8 Gen2	x8 Gen3	x1 Gen1	x8 Gen2
Analog Mixed Signal (AMS)/XADC	Yes	Yes	Yes		Yes
Configuration AES	Yes	Yes	Yes	Yes	Yes
I/O Pins	500	500	1,200	576	1,200
I/O Voltage	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V
EasyPath Cost Reduction Solution	-	Yes	Yes	-	Yes

Figure 2.8: Key differences between different Xilinx FPGA families.

Figure 2.8 shows the key differences between the Xilinx FPGA families. Considering that the development of the custom DDS system in this research is part of the NMR system, the total I/O pins required by the entire project are 295. Besides that, a PCI Express interface and two I/O voltage levels (1.3 V and 3.3 V) are needed. Therefore, Spartan-6 will be the best option as the exact functionality that is needed is provided by the chip. Based on past projects in similar fields, the estimated amount of logic cells and DSP slices required for this research are covered by a Spartan-6 device.

2.6 Spartan-6 (XC6SL45T)

This research is part of the overall project belong to the NMR research group. The aim of the project is to implement an NMR RF transmitter for NMR application purposes. Some research components are inherited from past research projects. Therefore, a standard FPGA chip, Spartan-6, is used to prevent compatibility problems and to avoid the necessity to re-design and re-develop the overall system. This decision will reduce the research timeline and improve the research cost saving by reusing the available resources.

Many DSPs have difficulty in meeting the specifications of an RF system. For example, a waveform at 2GHz sampling time is hard to achieve because it lacks flexibility. Fortunately, FPGAs have very good coordination ability to deal with this limitation. The characteristics of FPGAs enable them to have high speed data access realised. By combining the flexibilities of a DSP processor and an ASIC, the DSP circuit, which is composed of an FPGA device, can work in parallel as well as in the sequential manner (state machine). The instructions executed by a standard DSP will cost more cycles to finish than an FPGA device running in parallel processing. Overall, the processing speed of FPGAs is faster than the general DSP available on the market [65].

Spartan-6 has a large array of configurable logic blocks (CLB) ranging from 3840 to 147443 logic cells within the Spartan-6. This huge amount of cells gives great flexibility together with its speed [56]. For many DSP applications such as NMR spectrometers, speed is important for initial processing of the data. After that, the data rate is reduced and it becomes more convenient to manage. In this research, the low-cost and low power Spartan-6 can be reconfigured so that it can be used for multi-purpose implementation to make sure the hardware has a higher utilisation factor.

Generally, the maximum I/O clock frequency of a Spartan-6 is 540 MHz [57]. However, the clock speed can be extended to reach 1 GHz by using specific I/O resources in the device. The I/O resource used for the high speed signalling process is known as I/O SERDES [36]. This feature would be further explained in Chapter 3.

The Spartan-6 has adequate features to make up a complete RF transmitter. All the digital jobs required in the RF transmitter can be done in a single FPGA chip. By utilising Spartan-6, the project cost decreased while the portability increased. Spartan-6 has good scalability to avoid becoming obsolete and to shorten the time needed to design a new-product [4]. The internal FPGA modules are reusable and re-configurable to meet the project changes. The Spartan-6 chip can run its modules in parallel, and these modules can be overwritten as many times as necessary using only a computer and a download cable.

2.6.1 Basic Architecture of Spartan-6

Generally, a common FPGA consists of logic blocks called Configurable Logic Blocks (CLB) or Logic Array Blocks (LAB) depending on the manufacturer, I/O pads and routing channels (as shown in Figure 2.9). The logic blocks are used to implement combinational and sequential logics. These logic blocks are connected to I/O blocks via wires, which are known as interconnection resources. These I/O blocks are special logic blocks at the periphery of the device for external connections.

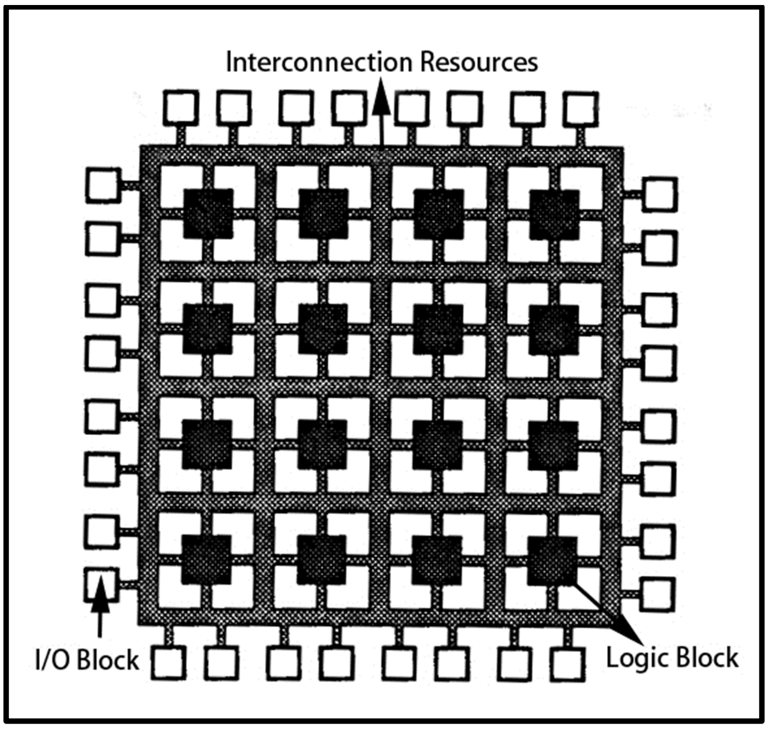


Figure 2.9: FPGA architecture [64].

In general, all the routing channels have the same number of wires. Multiple I/O pads may fit into the height of one row or the width of one column in the array. Currently, FPGAs employ logic blocks, which are based on the following [64]:

- Multiplexers
- Transistor Pairs
- Gates such as two input Negated ANDs (NAND) or exclusive-ORs
- LUTs
- Wide fan-in AND-OR structures

The programming process of an FPGA is done via electrically programmable switches and is almost the same as a traditional Programmable Logic Device (PLD). However, FPGAs can achieve much higher levels of integration than PLDs. Even so, PLD routing architectures are very simple but highly inefficient crossbar-like structures. Every output of a PLD is directly connectable to every input through one switch. In contrast, each I/O connection of a typical FPGA is usually passed through several switches. Hence, FPGA routing architectures are more efficient than PLDs. The logics of a PLD are implemented using predominantly two-level AND-OR logic with wide input AND gates. On the other hand, the logics of an FPGA are implemented using multiple levels of lower fan-in gates and are much more compact.

Figure 2.9 shows the architecture of a common FPGA device. An FPGA routing architecture incorporates wire segments of varying lengths that can be interconnected via electrically programmable switches, which can affect the density achieved by an FPGA [35]. In practice, optimum wire segments and lengths are used in every project. If insufficient wire segments are used, it will result in poor density because only a small fraction of the logic blocks can be utilised. On the other hand, the use of extreme wire segments that go unused is impractical because of the waste of board area. If an excessive length of wire is used, implementing local interconnections becomes too costly in the area and delay. If all the segments used are short, long interconnections are implemented with many switches in series. This condition will cause impractical large delays. In fact, unused routing tracks increase the cost and decrease the performance of the part without showing any benefit. Thus, most of the FPGA manufacturers provide just enough tracks so that most of the designs can fit in terms of LUTs and I/Os can be routed.

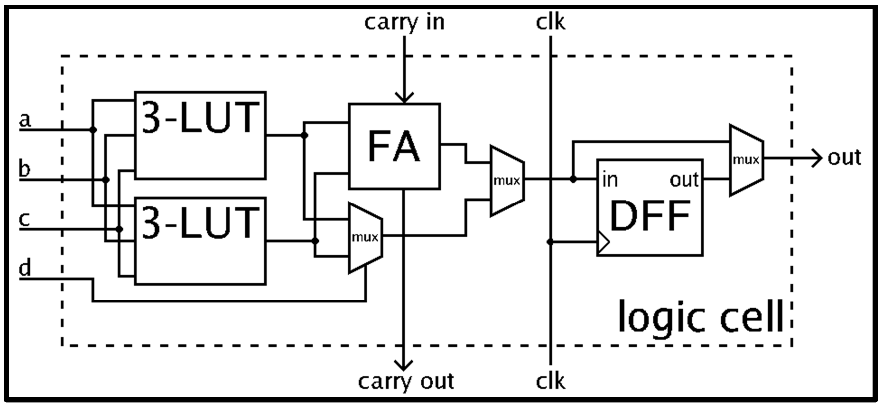
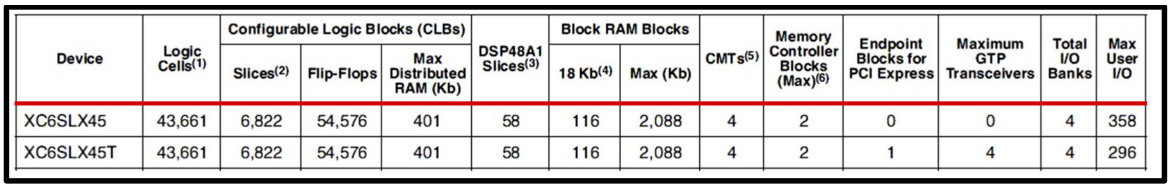


Figure 2.10: Simplified block of a logic cell [21].

A general logic block consists of a few logic cells, which are normally known as slices. A typical cell has a 4-input LUT, a full adder (FA) and a D-type flip-flop (DFF) (Figure 2.10). In practice, the entire or part of the FA is placed into LUT in order to save space. In normal mode, the 3-input LUTs (a, b and c) are combined with 'd' into a 4-input LUT via a left multiplexer (MUX). In arithmetic mode, the outputs of 3-LUTs are sent to FA. These two modes can be selected by programming the middle MUX. The MUX to the right can also be programmed to decide if the output will be either synchronous or asynchronous [21]. In recent years, manufacturers started to improve LUTs by moving from a 3-input LUT to a 6-input LUT. It is claimed that this can increase the performance of an FPGA device.

In this research, a Spartan-6 (XC6SLX45T) is used to carry out the signal processing of an RF signal. Spartan-6 logic cell ratings reflect the increased logic cells due to the introduction of the new 6-input LUT architecture. In CLBs, each Spartan-6 FPGA slice contains four LUTs and eight flip-flops with each DSP48A1 slice containing an 18 x 18 multiplier, an adder, and an accumulator. The DSP38A1 slices support DSP algorithms and high levels of DSP integration. Minimal use of general FPGA logic leads to low power, very high performance, and efficient silicon utilisation.



Device	Logic Cells(1)	Configurable Logic Blocks (CLBs)			DSP48A1 Slices(3)	Block RAM Blocks		CMTs(5)	Memory Controller Blocks (Max)(6)	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices(2)	Flip-Flops	Max Distributed RAM (Kb)		18 Kb(4)	Max (Kb)						
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296

Table 2.0: Features of Spartan-6 [56].

A Spartan-6 device has four clock management tiles (CMT) and 116 block RAMs (see Table 2.0). Each CMT contains two DCMs and one PLL. Each block RAM is 18Kb in size. However, each block can be used as two independent 9Kb blocks. The difference between XC6SLX45 and XC6SLX45T are that the XC6SLX45 has a 1.0 voltage core but no Gunning Transceiver (GTP). In contrast, XC6SLX45T has a 1.2V voltage core, GTP and extra endpoint block for PCI express feature [56]. A GTP Transceiver supports multi-gigabyte data rates in high-speed data transmission.

2.6.2 CLB

CLBs are responsible for implementing sequential and combinatorial circuits in FPGA devices. The connections among CLBs are done via a programmable interconnection. Each of the CLBs is connected to a switch matrix and the switch matrix provides access to the general routing resources as indicated in Figure 2.11. A CLB element has two slices. These two slices do not connect to each other directly. Each slice is organised as a column. A slice in the column has an independent carry chain.

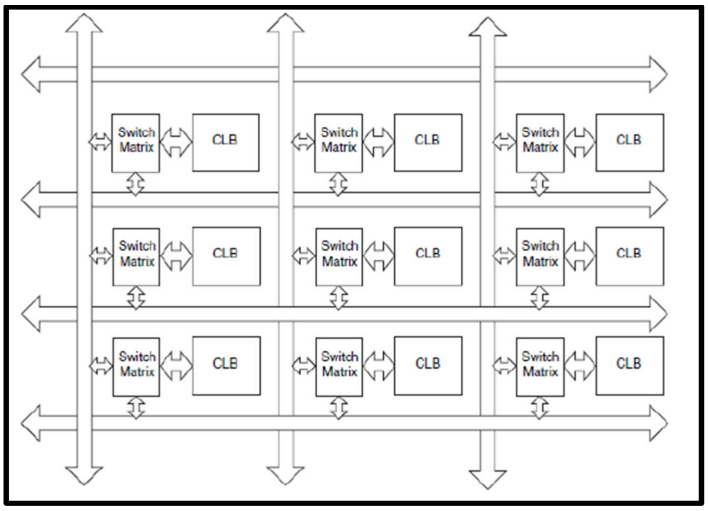


Figure 2.11: CLB and interconnect channels [55].

A CLB has a pair of slices, which are not connected to each other as shown in Figure 2.12. The slice in the bottom of the CLB is labelled as SLICE (0), and the slice in the top is labelled as SLICE (1).

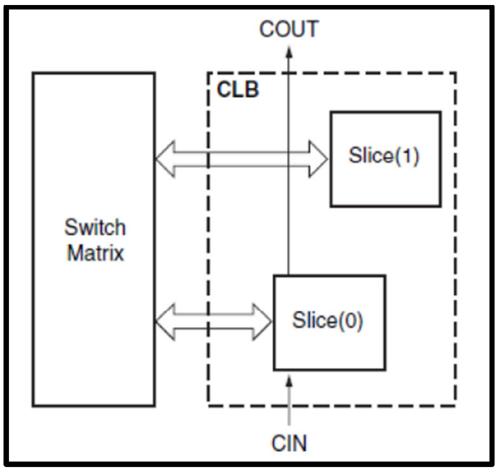


Figure 2.12: Slices in a CLB [55].

Each slice contains four logic-function generators or LUTs and eight storage elements. These elements are used by all types of slices to provide logic and ROM functions. SLICEX is a basic slice. It contains 6-input LUTs and 8 flip-flops so it can be used for logic only. Other slices such as SLICEL and SLICEM contain additional features such as wide-function multiplexers and carry logic, which allow SLICEL and SLICEM to provide logic, arithmetic, and ROM functions. However, SLICEM has two additional features. SLICEM has distributed RAM, shift registers that allow the ability to use the LUTs as the 64-bit distributed RAM, and as variable-length shift registers with maximum 32-bit. The main difference between SLICEX, SLICEL, and SLICEM is that SLICEX can only be used for logic, SLICEL only for logic and arithmetic, and SLICEM can be used for logic, arithmetic, and memory.

Each column of CLBs has two slice columns. One is SLICEX and another one is either SLICEL or SLICEM. SLICEX occupies 50% and each of the other two slices occupies 25% of the available slices. Table 2.1shows all the features of these three types of slices [56] and Table 2.2 shows the summary of logic resources for each CLB in Spartan-6 devices.

Feature	**SLICEX**	**SLICEL**	**SLICEM**
6-Input LUTs	√	√	√
8 Flip-Flops	√	√	√
Wide-Function Multiplexers		√	√
Carry Logic		√	√
Distributed RAM			√
Shift Registers			√

Table 2.1: Slice features [56].

Slices	**LUTs**	**Flip-flops**	**Arithmetic and Carry Chains** (2)	**Distributed RAM** (1)	**Shift Registers** (1)
2	8	16	1	256 bits	128 bits

Table 2.2: Logic resources of a CLB [56].

Notes:

1. *SLICEM only, SLICEL, and SLICEX do not have distributed RAM or shift registers.*
2. *SLICEM and SLICEL only.*

2.6.3 LUT

The function generators in Spartan-6 are implemented as 6-input LUTs from two 5-input LUTs. In Figure 2.13, there are six independent inputs (A1 to A6) and two independent outputs (O5 and O6). The combination of two 5-input LUTs (two LUT5s) can implement a 6-input function generator (LUT6) with only an O6 output. Two arbitrary LUT5 function generators can implement two independent 5-input function generators with O5 and O6 outputs. The wide-function multiplexers of SLICEL and SLICEM can be used to combine two LUT6s to implement a 7-input function generator or to combine four LUT6s to implement an 8-bit input function generator *[55]*.

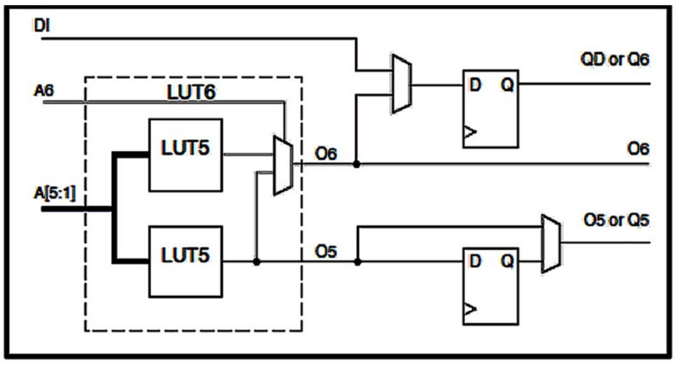


Figure 2.13: LUT of Spartan-6 [55].

A function generator produces outputs via LUTs. The Static Random-Access Memory (SRAM) cells store the content of a LUT. The input addresses of the function generator are used to select the desired SRAM cells. Figure 2.14 shows the working area of a 3-input LUT (a,b and d). Of course, the actual LUT is different; the figure is simplified and just for clarifying the working idea.

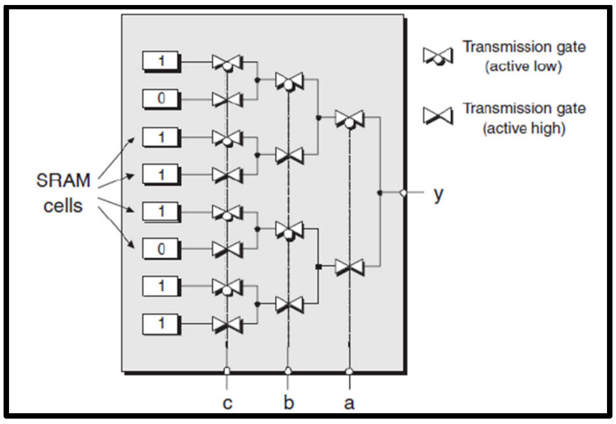

Transmission gate (active low)
Transmission gate (active high)
SRAM cells
1
0
1
1
1
0
1
1
y
c
b
a

Figure 2.14: A transmission gate-based LUT [28].

2.6.4 Storage Elements

There are eight storage elements (two sets) per slice in a Spartan-6 device. The logics of a slice are stored by flip-flops (each set contains four flip-flops). The first set can be configured as either edge-triggered D-type flip-flops or level-sensitive latches but the second set can only be edge-triggered D-type flip-flops. When four flip-flops are configured as level-sensitive latches, the remaining flip-flops cannot be used [55].

2.6.5 Multiplexers

As shown in Table 2.1, the wide-function multiplexer is available only for SLICEL and SLICEM. The multiplexer allows LUT combinations of two or four LUTs in order to implement 7-input or 8-input general-purpose function generators [55].

2.6.6 Carry Logic

As shown in Table 2.1, carry logic is available only for SLICEL and SLICEM. They are used to perform fast arithmetic addition and subtraction in a slice.

2.6.7 Distributed RAM

In Spartan-6 FPGA devices, SLICEM has additional data input and write-enable signals. These two key features allow LUTs to use SRAM cells as distributed RAM in FPGA devices. The size of the RAM can be created by combining multiple LUTs in a SLICEM to store a large amount of data. Since the flip-flops are from the same slice, the performance of distributed RAM is therefore fast and is suitable for small data buffers [55].

2.6.8 Shift Registers

A 32-bit shift register can be configured by the LUT of SLICEM without using any flip-flops in a slice. Multiple LUTs can be cascaded to produce delays of up to 128 clock cycles. As examples, two, three, and four LUTs can be configured as 64-bit, 96-bit, and maximum 128-bit shift registers due to only four LUTs in a slice. However, more than one SLICEM can be cascaded to create a shift register, which is more than 128-bit [55].

2.6.9 I/O Architecture of Spartan-6

The Input/output Block (IOB) of Spartan-6 supports a wide variety of standard interfaces via programmable high performance I/O drivers and receivers. Each IOB can be configured to be input, output, or tri-state. The IOBs are grouped into four or six banks depending on device size and package. A Spartan-6 FPGA device such as XC6SLX45T has four banks, one on each side of the device (see Figure 2.15). Each bank has various standards, which include slew rate, output drive voltage and on-chip termination. Differential I/O standards can be configured by grouping adjacent IOBs together in one tile [54].

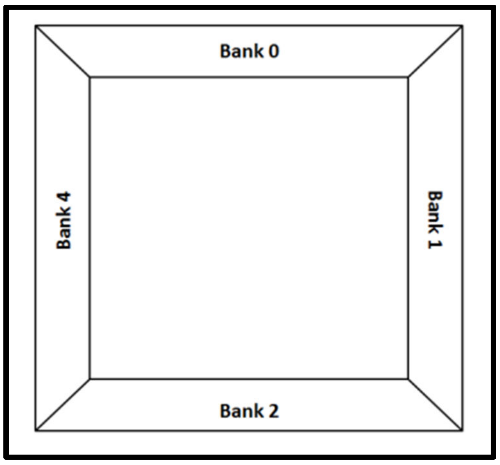


Figure 2.15: Spartan-6 XC6SL45T I/O banks [54].

Figure 2.16 shows a Spartan-6 FPGA I/O tile containing two IOBs, two Input Logics (ILOGIC), two output logics (OLOGIC) and two IODELAYs.

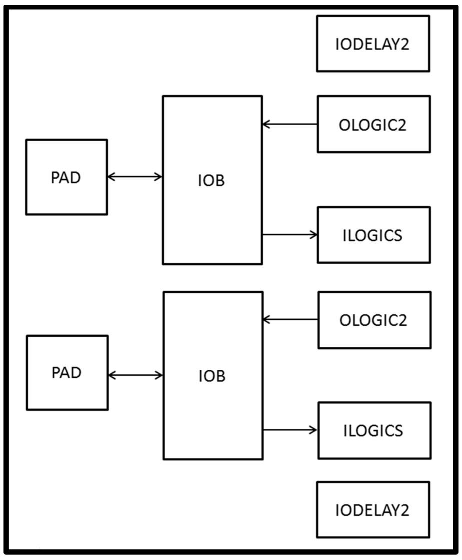


Figure 2.16: I/O tile of Spartan-6 [54].

2.6.9.1 IOB

The IOB in Figure 2.17 is used to drive or to receive signals from a physical pin of a Spartan-6 device. The IOB can be configured as input, output, or tri-state by using SelectIO drivers. Each IOB has direct connection to ILOGIC/ OLOGIC pairs containing the input and output logic resources for data and tri-state control. Both ILOGIC and OLOGIC can be configured as input serdes (ISERDES) and output serdes (OSERDES) for serialising and de-serialising data to allow higher operational speeds. Only one IOB is needed to configure a single-ended I/O standard and two adjacent IOBs for a differential I/O standard [54].

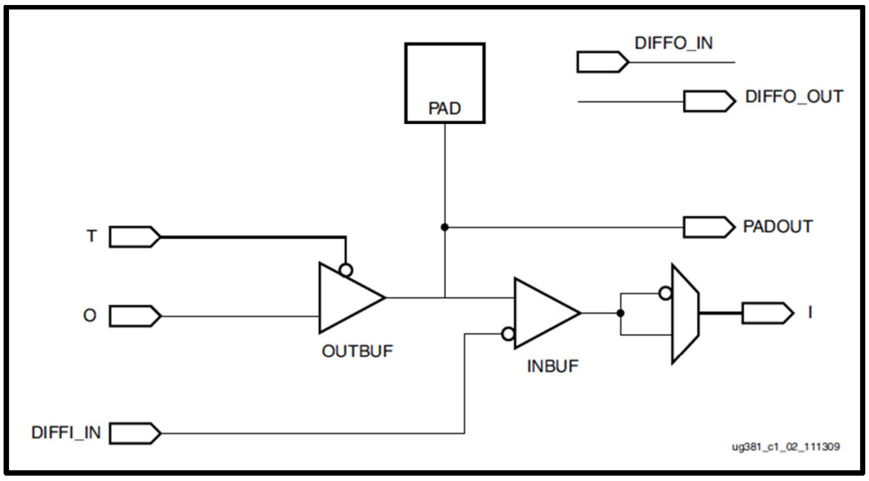


Figure 2.17: Basic IOB diagram [54].

2.6.9.2 Input/ Output Logic Resources

An I/O interface tile (IOI) has two IOLOGICs, which are Master IOLOGIC and Slave IOLOGIC. These two IOLOGICs can be concatenated in differential I/O standard or operate independently as two single-ended I/O blocks. In Figure 2.18, the master I/O buffer drives the pad 'P' and the slave I/O buffer drives the pad 'N' separately. Thus, it results in two single-ended blocks. When the resources of these master and slave I/O buffers are combined, a pair of the differential signal path on pad 'P' and pad 'N' can be created. Moreover, these I/O drivers can be configured to implement a parallel-to-serial (serialisation) or serial-to-parallel (deserialisation) conversion in differential mode. The conversion can produce a rate of up to twice that of a single I/O buffer [54].

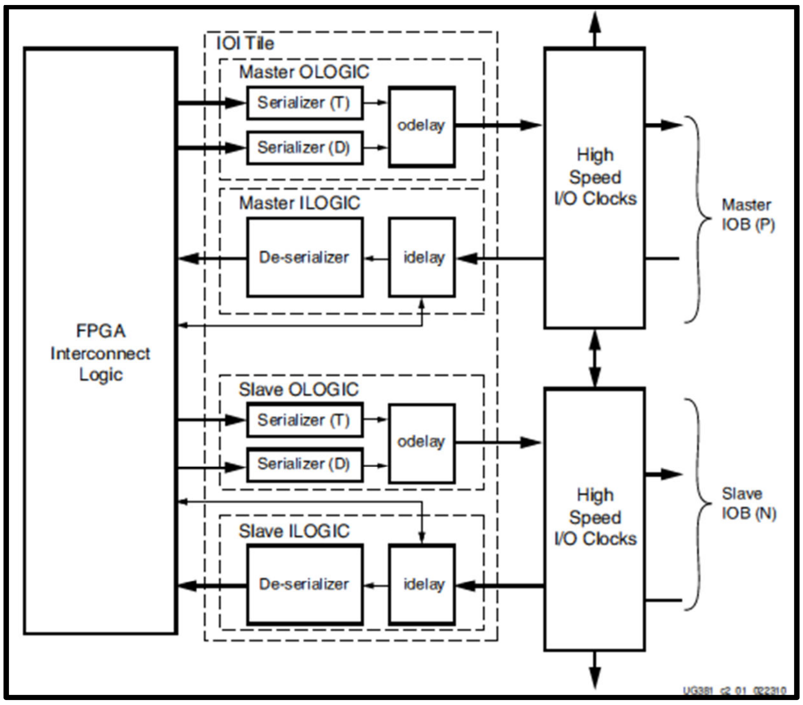


Figure 2.18: Resources of master and slave IOLOGICs [54].

The IODELAY in the master and slave IOLOGIC blocks are optional. The amount of delay can be fixed or variable depending on the programming to Spartan-6 FPGA devices.

2.6.10 Input/ Output Serialisation and De-serialisation (ISERDES and OSERDES)

In Spartan-6 FPGA devices, each IOB contains a 4-bit ISERDES and OSERDES block. The IOSERDES block allows higher operational speeds in data transmission. In order to obtain more bits in IOSERDES (more than 4-bit), adjacent serdes blocks can be cascaded to create serdes ratios from 2:1 to 8:1. These are applicable to both single data rate (SDR) and double data rate (DDR) modes. There are some precautions for using ISERDES in SDR mode when compared to ISERDES in DDR mode. In DDR mode, master and slave logic blocks with two sets of serdes registers are used to create a differential signalling standard. In SDR mode, two data lines cannot enter the device in adjacent master and slave IOBs when serdes is cascaded to achieve more than 4-bit serdes. For serdes ratio of four or less in SDR mode, only one SERDES block is needed and so the phase detector is not required because no serdes block is cascaded [36].

In de-serialisation, the received data stream is a multiple of the rate of the incoming clock and the clock signal is used as a framing signal for the received data. Therefore, there will be multiple changes in the state of the data lines occurring during a clock period. If the clock signal is in SDR mode, the data stream is at the same rate as the receiver clock. If the clock signal is in DDR mode, the data stream is at twice the rate of the receiver clock because serdes receives DDR data on the rising edge and the falling edge of the receiver clock [36].

In serialisation, the process depends on the desired format of the transmitted data stream and a forwarded clock, and both must be regenerated from the same serdes system clock. Consequently, both data and clock can change state at the same time. If the forwarded clock is SDR, the data stream changes state twice for each data bit transition [36].

2.6.11 Clock Resources

All the synchronous signal processes inside an FPGA device from the outside world occur through specific clock pins. The connection of the external clock to the FPGA is established via clock routing resources in order to eliminate any potential for timing hazards. Figure 2.19 shows a simple clock tree of an FPGA device.

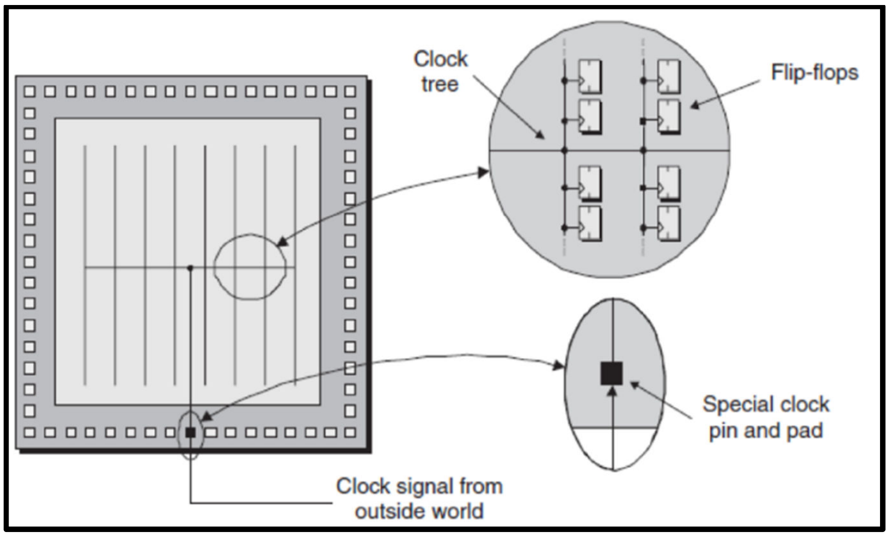


Figure 2.19: A simple clock tree [28].

There are two processes to manage the clocking resources in FPGA devices. The first process it to set up a path for the clock signals from external clock sources to FPGAs through global clock routing resources. The second process is the management of the clock signal inside the FPGA device through CMTs. The CMTs provide very high performance clocking skew for synchronous signalling process inside FPGAs. For this there is dedicated routing within CMTs. By using these dedicated routes, global resources can be freed up for other design elements as well as reducing the chances for noise coupling as these routes are handled locally. Each CMT block contains two DCMs and one PLL. Therefore, a Spartan-6 (XC6LS45T) device contains four CMTs, eight DCMs and four PLLs [28].

Any output clock from the CMT can minimise clock skew by using a Global Clock Buffer (BUFG). In case the BUFGs are insufficient, a CMT clock can be used without any BUFG but all logic must be placed within the clock region [61].

DCMs are advanced clocking systems. They are integrated directly into the global clock distribution network for high performance and high frequency application. DCMs can improve system performance by eliminating the clock skew either within the device or to external components. It is also used to eliminate the clock distribution delay within the system. Besides this, DCMs have optional features to phase shift the clock outputs. It can also delay the received clock either by a fixed fraction of the clock period or by incremental amounts. In addition, DCMs can act as a clock distributor to synthesise a new clock frequency, optionally multiply, or divide the incoming clock frequency. Other benefits provided by DCMs are listed below [61]:

- Free-running oscillator
  In free-running oscillator mode, an external clock signal needs to be connected to a Spartan-6 until DCM is locked, then the external clock signal no longer needs to be presented and can be removed. At this moment, the internal clock generated by the DCM module will continue to run at the same frequency. If temperature or voltage levels fluctuate, the frequency can vary.
- Provide a clean 50% duty cycle output clock
- Convert the incoming signal to different I/O standards
- Acts as a filter for clock input jitter

PLL serves as a frequency synthesiser for a wide range of frequencies. It also acts as a jitter filter for either an external or an internal clock in conjunction with the DCMs. Figure 2.20 shows the overview of a PLL and the entire process can be summarised as below.

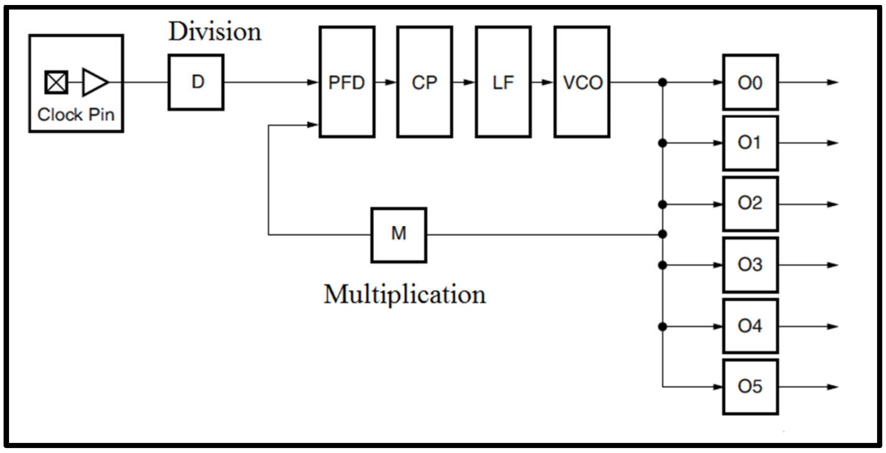


Figure 2.20: Overview of a PLL in Spartan-6 [61].

Each clock input has a programmable counter D to perform a division on a clock signal. This division will lower the clock frequency. The Phase-Frequency Detector (PFD) block is used to generate a signal proportional to the phase and frequency between the reference clock (input clock) and the feedback clock. Thereafter, the Charge Pump (CP) and Loop Filter (LF) blocks will process the signal from the PFD and will generate a reference voltage to the Voltage Controlled Oscillator (VCO). The feedback clock of the PLL will be controlled by a programmable counter M [61].

Later on the multiplication stage (M), the signal will be fed to the PFD. An up or down signal will be created in the PFD to the CP. The purpose of the LF is to determine whether the VCO should operate at a higher or lower frequency. The entire process allows a wide range of frequency synthesis.

2.6.12 Other Resources of Spartan-6

Besides the modules discussed above, there are other resources available for the Spartan-6 FPGA [56]:

- Block RAM
- Memory Controller block
- Soft 8-bit PicoBlaze Processor
- Soft 32-bit MicroBlaze Processor
- GTP transceivers
- Integrated Endpoint block for PCI Express Design

One can choose a suitable FPGA module to satisfy their design requirements from the available resources in an FPGA device.

2.7 Development Tools

Every FPGA device manufacturer introduces its own design tools to support FPGA devices. Examples of design tools introduced by two major FPGA manufacturers are 'Xilinx Integrated Software Environment (ISE)' by Xilinx and 'Quartus II' by Altera. These design tools enable the project developers and designers to control the FPGA-based project development cycle. The main advantage of using FPGA devices is that their associated development tools reduce the time-to-market. This is because the development tools are able to generate many parts or functions to meet the design requirements. These functions can be obtained by using core wizards such as IP Core Generator from Xilinx or Mega-functions from Altera. Generally, these two manufacturers provide a free version of development tools to support basic capabilities. The non-free version comes with advanced capabilities, which are commonly used for the development of embedded systems, DSP systems and so on. In the research presented in this thesis, Xilinx ISE is used for the synthesis and analysis of VHDL designs on the Spartan-6 device.

This ISE development tool enables developers or designers:

- To synthesise (compile) their FPGA-based designs
  Synthesis is the process of constructing a gate level netlist from a model of a circuit described in VHDL.
- To perform timing analysis
- To examine RTL Diagrams
- To simulate a design's reaction to different stimuli
- To configure the target device

2.8 Summary

This chapter reviewed the structures necessary to build an RF transmitter. The pulse generator is the main core in the RF transmitter, and it is designed to generate a sine-wave output with a desired frequency. The ROM-based LUT is more reliable and more economical than the CORDIC algorithm in terms of hardware cost and hardware resources. The limitations of the present RF transmitter lie behind the motivation for this research:

- Not all the components of the present RF transmitter are completely implemented in the FPGA. The features of the FPGA are not fully utilised by most solutions, and it results in many redundancies in the FPGA.
- The limitations and constraints of an NMR spectrometer depend on the availability of the hardware components on the market. A good design should not rely too much on faster hardware devices because different methodologies should be tried to operate the system beyond the limitations in order to achieve the objectives of the research.

Chapter 3
Design and Development

Chapter 3 will discuss the conceptual design of an advanced DDS system and its simulation results. This chapter contains two sections. The first section deals with Matlab-Simulink simulation and the second with RTL-level simulation. Matlab-Simulink is used as a high-level modelling tool to interpret the overall functionality of a DDS system, especially from the flow of input to output. The primary purpose of using Simulink is to simulate the basic structural pattern of a custom DDS system. The custom DDS system can handle two types of signal synthesis processes. The basic type generates a signal from the 1st Nyquist zone based on over-sampling theory. The second type generates a signal which is outside of the 1st Nyquist zone with under-sampling theory. After Matlab simulation, the custom DDS modules undergo RTL-level simulation. RTL-level simulation involves a VHDL compiler and an ISim simulator. The RTL-level simulation is a behavioural simulation. It helps to understand the architecture and the operation of the entire custom DDS system.

3.1 High-level Simulation of a DDS core with Matlab-Simulink

There is always some confusion in setting up the periods for simulation. In the DDS Simulink simulation, four types of period settings could be found:

1. **FPGA Clock Period (ns):**

   This is the FPGA clock period setting of the Xilinx system generator block. The setting of this period, however, does not affect the output frequency of the DDS block. It is solely used for FPGA synthesis and implementation by using Mat-lab Simulink. In this research, the FPGA synthesis and implementation are handled by using Xilinx ISE. Consequently, the FPGA clock period setting in this research is inessential.

2. **Simulink System Period:**

   This is the period setting used in the Simulink simulation. Usually, it refers to the sampling period of the Simulink block (except Xilinx Simulink blocks). The minimum sampling period is equal to 1.

3. DDS System Clock (MHz):

In this research, it is set to 250 MHz. This value is used to calculate the phase increment in the DDS module (phase accumulator).

4. Explicit Sample Period:

This explicit sample period refers to the sample period of the DDS module (only for Xilinx Simulink blocks). If any Simulink system period is set to more than '1', the explicit sample period must be 'Checked'. This will make sure the explicit sample period is set to the same value as the Simulink system period. The aim is to prevent the Simulink using the sample period of 1 for the DDS blocks. This is to avoid the system generating an inaccurate output frequency.

The FPGA clock period setting is not important here because the research presented here does not involve FPGA synthesis using Simulink. All the structural designs that are simulated by using Simulink would be transferred to RTL-level simulation (Xilinx ISE) manually at a later stage.

3.1.1 Design Requirements

A single DDS should satisfy the following requirements:

System Clock, f_{system}	: 250 MHz
Signal-to-noise ratio (SNR)	: ~86 dB
Input width of Phase Accumulator	: 25-bit
Input width of LUT	: 25-bit
Output width of LUT	: 14-bit
Frequency Resolution, Δf	: 7.45 Hz

Notes:

With Xilinx FPGA devices, the phase truncation is done by LUT block before phase-to-amplitude conversion.

Estimated SNR of the truncated output

SNR = 6.02B + 1.76 dB where B is the total bits to the DAC chip (truncated) (3.0)

86 = 6.02 (14) + 1.76 dB

B = 86 dB

Depth of sine LUT

The table depth is $= 2^B$ (3.1)

$= 2^{14}$

= 16384 samples

Frequency Resolution

$$\Delta f = \frac{f_{system}}{C \ x \ 2^N}$$, where N is the N-bit phase accumulator (3.2)

C is the total output sample over a unit time

$$\Delta f = \frac{250 \ MHz}{1 \ x \ 2^{25}}$$, $C = 1$ because only one output over a unit time

Δf = 7.45 Hz

Depth of the phase accumulator

The depth of the phase accumulator is $= 2^N$ (3.3)

$= 2^{25}$

= 33554432

A Xilinx phase accumulator supports an input width from 3-bit up to 48-bit. As a LUT can support from a minimum of 12-bit to 25-bit of input width, a 25-bit phase accumulator is selected to match the LUT. A 14-bit truncated output is selected so that the output width of the LUT matches the input width of a DAC chip used on an evaluation board.

The phase accumulator has a 25-bit input to generate phase information. The depth of the phase accumulator is 2^{25} = 33554432. However, this 25-bit will be truncated to 14-bit to reduce the capacity of a LUT. After the truncation, the total entries of a LUT decrease from 33554432 to 2^{14} =16384 samples. In other words, it is reduced to 0.0488% of the initial capacity.

Note:

In Xilinx FPGA devices, the selection of quarter or full wave storage is performed automatically by FPGA devices to get the most efficient implementation [53].

3.1.2 Matlab-Simulink of a Basic DDS System

A single DDS module consists of a phase register, a phase accumulator and a sine LUT [5]. The system-level model of a single DDS system is shown in Figure 3.0.

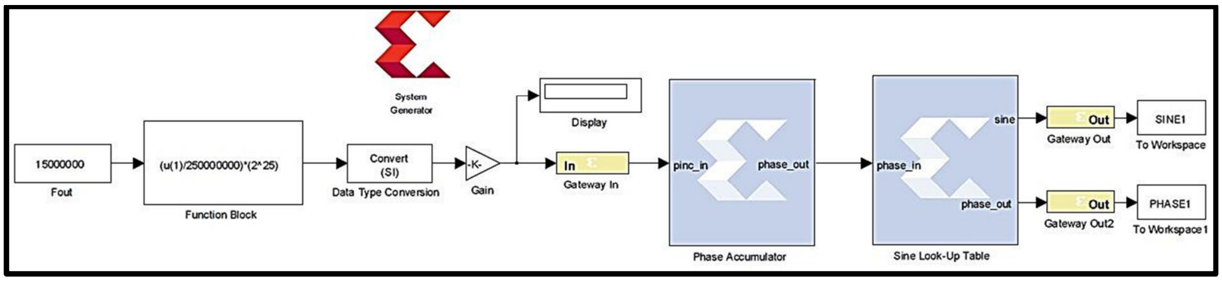


Figure 3.0: Matlab-Simulink of a single DDS.

As is illustrated above, a Xilinx system generator is a system-level modelling tool that extends Matlab-Simulink to cover the FPGA hardware design [31]. 'Fout' is a constant block used to provide the information of a signal frequency to the DDS system which is measured in Hz. The Xilinx Gateway blocks are added to the Simulink in order to interface the nominal Simulink modules with the Xilinx block sets. The 'Gateway In' represents the input port into FPGA while the 'Gateway Out' is the output port from the FPGA.

The Xilinx 'Gateway In' and 'Gateway Out' blocks are used to handle data conversion between Matlab-Simulink and the Xilinx block set. The conversion is necessary because the data processing in Matlab-Simulink uses double-precision floating-points, while the Xilinx block set uses fixed-point precision.

The phase accumulator and sine LUT functions are both Simulink blocks which have the same environment as the modules in the Xilinx FPGA devices. In Figure 3.0, the output from the accumulator block is in 25-bit width while the output from the LUT block is truncated to 14-bit.

The remaining two blocks 'SINE1' and 'PHASE1' are used to capture all the data generated throughout the simulation. The data are used to plot two graphs via Matlab workplace. These two graphs show the relationship between accumulated phases and signal amplitudes.

On every clock pulse, the phase accumulator generates a phase, which serves as a memory address to the sine LUT. The 25-bit phase accumulator can be filled gradually until it overflows. Each time the phase accumulator overflows, the output of the sine LUT will achieve a revolution of a complete sine-wave. The rate at which the phase accumulator overflows depends on the data width of the phase accumulator and the frequency tuning word.

The sine LUT is a module used to convert the digital phase to amplitude. If all the 25 bits of the phase are used for conversion, the sine LUT needs a lot of memory to store the amplitudes. Although the Spartan-6 has no problem in handling the output width of a 25-bit LUT, a lower output width can be used to achieve the same frequency resolution and output frequency. The common method is to use a phase truncation approach to extract the upper 14-bit data from the 25-bit data and rounding the 11-LSB. The upper 14 bits are the final phase. Despite the fact that a small degree of noise would add to the output waveform due to the loss of the low 11-bit data, it can be compensated by noise shaping in the LUT block. The Xilinx LUT block provides a noise shaping technique to compensate the spur effects.

The noise shaping technique utilised in this research is known as the Taylor series corrected DDS method (the only option provided and recommended by Xilinx). This noise shaping technique applies previously discarded fractional bits to calculate the correction that can be added to the sine LUT. It does not consume any of logic fabrics but requires two embedded multipliers, one constant coefficient multiplier and four adders. This compensation method can produce output with superior SFDR [52].

3.2 Conceptual Design of an 8-core DDS System

The Simulink module does not operate the simulation of 2 GSPS in real-time. This is because the simulation does not consider the actual timing needed by a system. The Simulink is more suitable for verifying and understanding the functionality of a module instead of simulating the actual timing of the system. Hence, 'unit time' is used to measure the timing of an output signal because every tick in a scope is 1 clock cycle. After the simulation, the conceptual designs of the DDS system would be transferred to the RTL-level simulation.

3.2.1 Design Requirements

The frequency resolution of an 8-core DDS module is 0.93 Hz. It can be calculated by using equation (3.2). The calculation is equivalent to dividing the frequency resolution of a single DDS core, 7.45 Hz, by eight. The denominator is eight because the 8-core DDS module generates eight outputs over unit time. An 8-core DDS module should satisfy the following requirements:

System Clock, f_{system}	: 250 MHz
SNR	: ~86 dB
Frequency Resolution, Δf	: 0.93 Hz
Input width of LUT	: 25-bit
Output width of LUT	: 14-bit
Depth of sine LUT	: 16384 samples
Input Width of Phase Accumulator	: 25-bit
Depth of Phase Accumulator	: 33554432
Total Phase Offsets	: 8 sets with 25-bit of each

The frequency resolution, Δf of 8-core DDS module can be calculated by

$$\Delta f = \frac{250\ MHz}{8\ x\ 2^{25}}$$
$$= 0.93\ Hz$$

3.2.2 Simulink of an 8-core DDS Module

An 8-core DDS module does not really contain eight individual DDS cores. It is labelled as an 8-core module because it generates eight samples at a time. The productivity of the 8-core DDS system is similar to the system formed by eight DDS cores. The benefit of using an 8-core DDS module is that it composes a sine-wave with 8 times more samples than a single DDS core system.

Figure 3.1 shows the Simulink diagram of an 8-core DDS module. The module has three major parts. The first part on the left is a phase register used to calculate the frequency tuning word for phase increment. The second part is a 2 GHz DDS system that performs phase accumulation and phase-to-amplitude conversion. The final part on the right is a mock system of a DAC chip, which is used to construct the sine-wave. The DAC block is not really a digital-to-analog converter. It is a pseudo module used to receive deinterleaved samples from DB0 and DB1. In the DAC block, these deinterleaved samples would be reconstructed as a complete sine-wave to SINE output port.

Notes:

'PHASE' is an optional port, which is used to output the sample phases of an output signal. Both SINE and PHASE provide a clear idea of the relationship between a phase and sine amplitude.

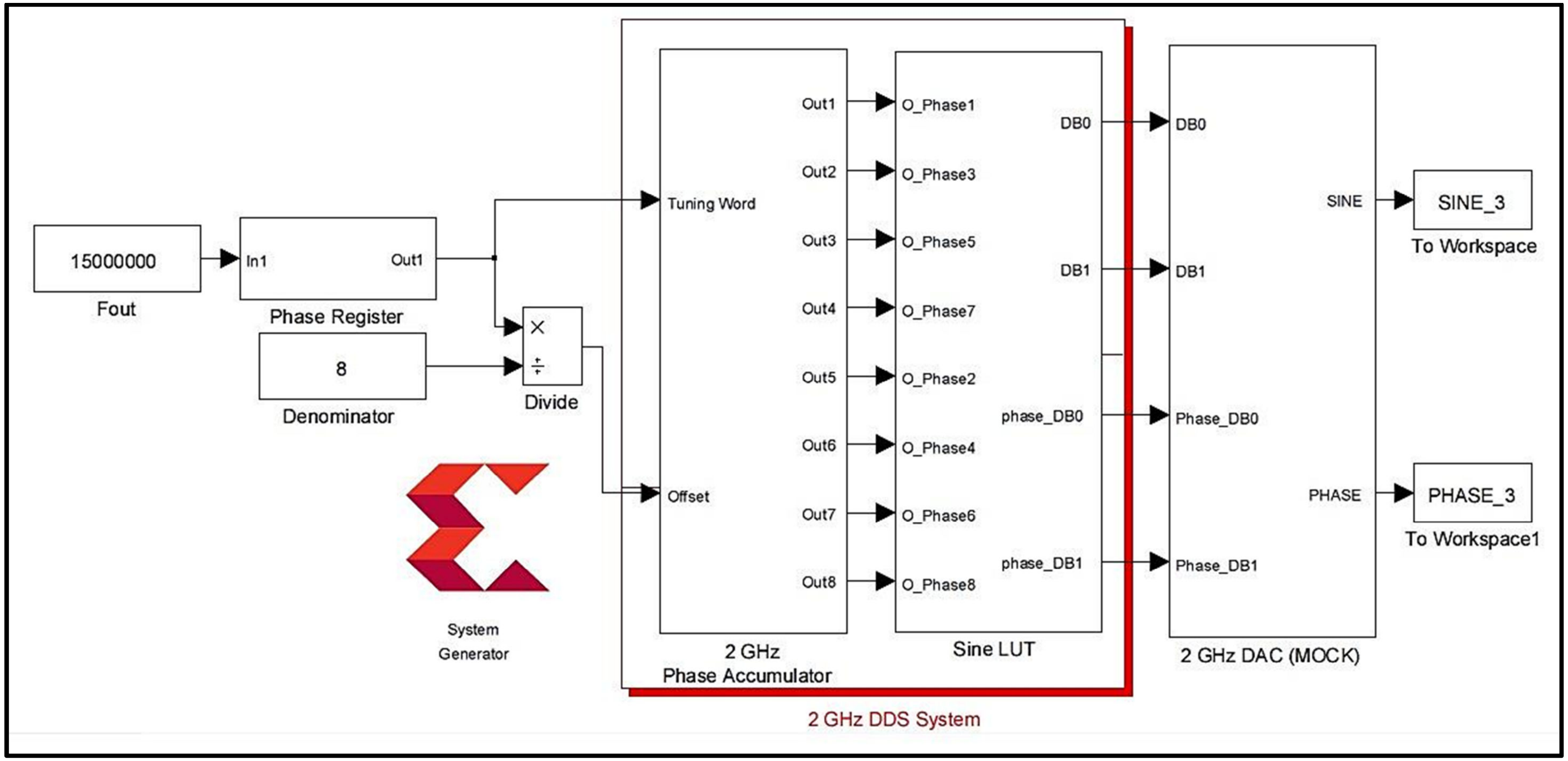


Figure 3.1: Conceptual design of a 2GHz sampling transmitter.

3.2.3 Phase Register and Minimum Phase Offset Calculation

3.2.3.1 Phase Register

The phase register is used to calculate the frequency tuning word based on the ‘Fout’ block. The phase accumulator has a unique output channel which runs at a maximum frequency of 250 MHz. This means the output signal has a sampling rate of 250 MSPS.

In a single DDS core, the total numbers of samples generated by the core to create a sine-wave are inversely proportional to the output frequency. The reduction of the samples is due to the increment of the jump size (hopping data points) between each sample phase. The bigger the jump size, the faster the phase accumulator overflows. The impact is that fewer sample phases are generated before the accumulator overflows. Insufficient samples can affect the signal intensity and the signal stability. This is because there will be inadequate samples to sustain the shape of the output as a sine-wave. Fortunately, the 8-core DDS module has the capability to insert 8 times more samples to the signal. This increases the total samples in a unit sine-wave.

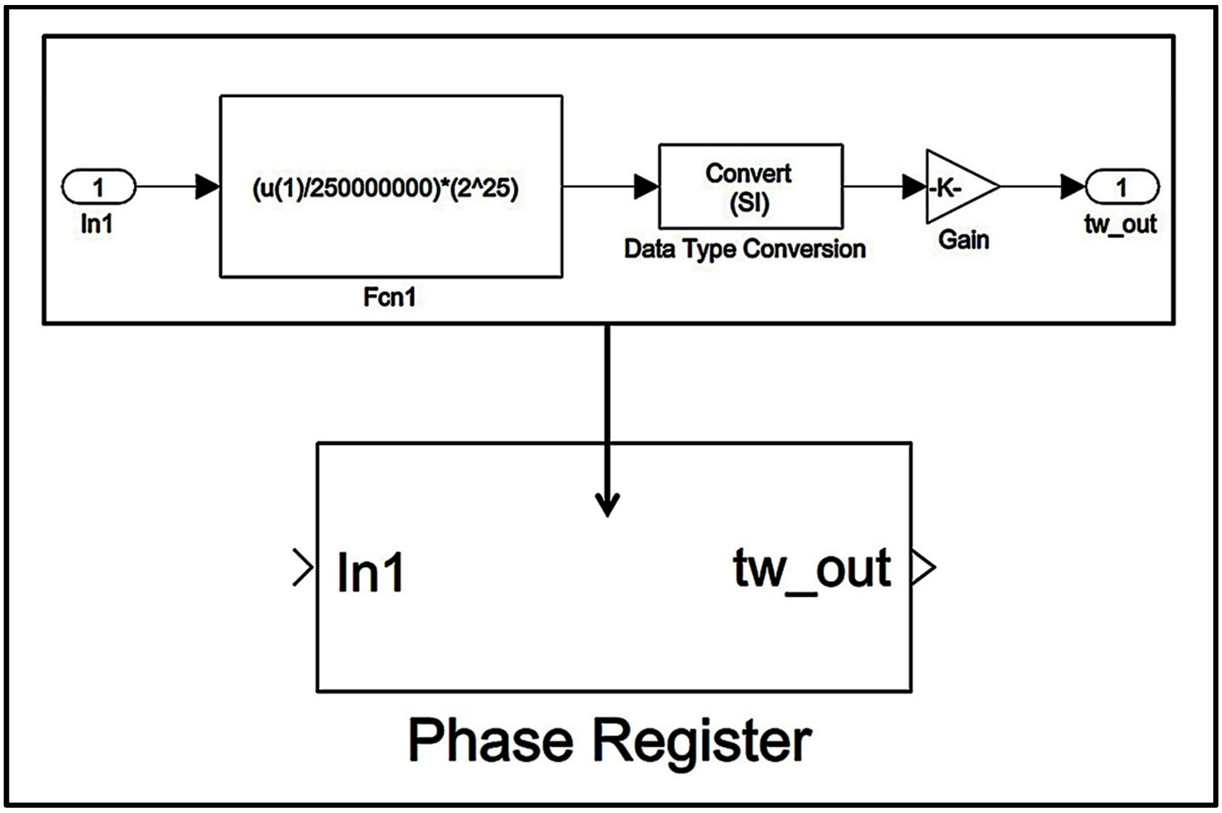


Figure 3.2: Phase register Simulink block.

Figure 3.2 shows the internal structure of a phase register. The function block, Fcn1, contains a mathematical equation (2.0) for calculating the frequency tuning word of an output frequency. For example, the frequency tuning word for a 15 MHz sine-wave is M = 2013265.

$$M = \frac{15\ MHz * 2^{25}}{250\ MHz} = 2013265$$

The gain block provides a gain factor to normalise the tuning word so that the amplitude falls within -1 to 1 on a display (scope). The gain factor is obtained by:

$$gain\ factor = \frac{1}{2^N} \qquad (3.4)$$

For a 25-bit phase accumulator, the gain factor = 1/33554432 = 29.802 x 10^{-9}

3.2.3.2 Eight Phase Offsets and Step Size

An 8-core DDS module has eight offset values, which are known as phase offsets. The phase offsets are not a fixed number. Its values depend on the frequency tuning word of an output signal. Phase offsets can be calculated by first determining the step size of an output signal. Then, the step size obtained is further calculated to determine the values of eight phase offsets.

Let us assume that the original sample is the accumulated phase generated by a phase accumulator before being shifted by any offsets (Figure 3.3). 'Out1' until 'out8' are the eight sample points after being shifted by the phase offsets. 'O' is the offset value which has the same value as the step size of the output signal.

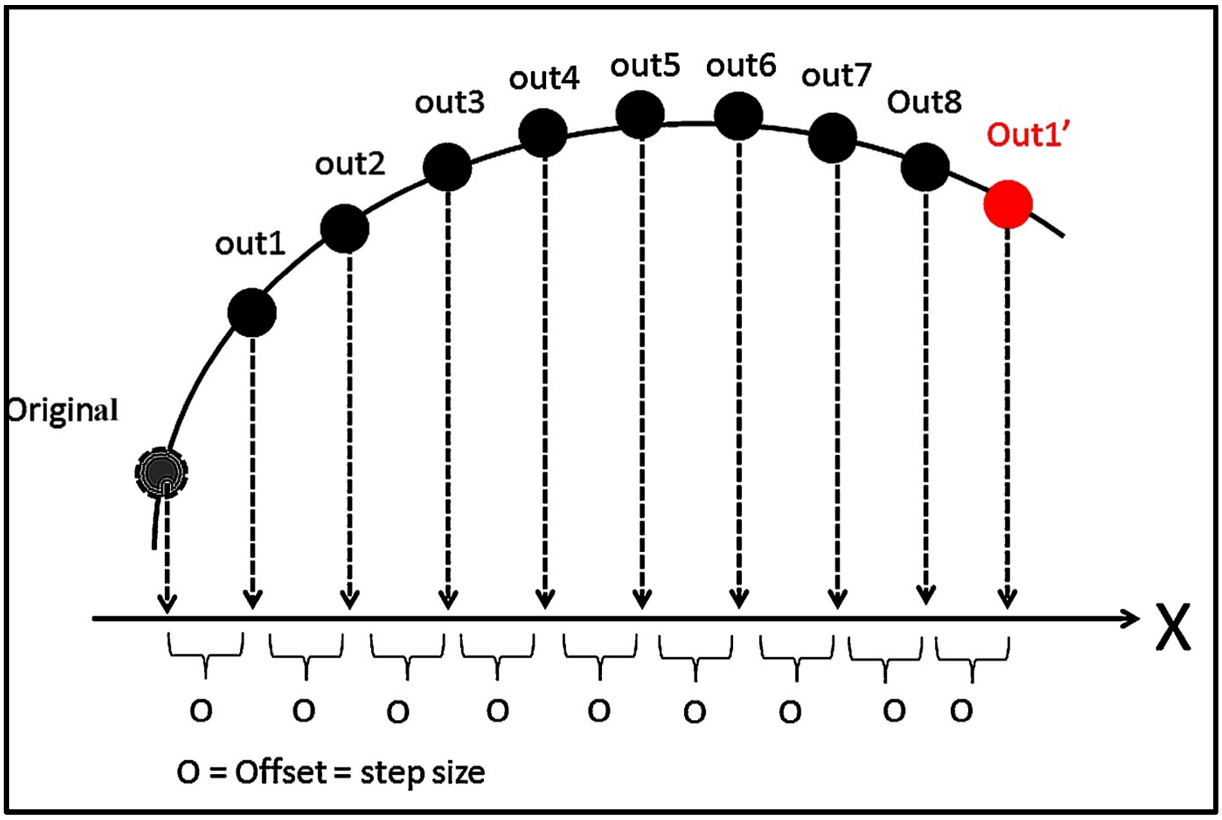


Figure 3.3: Phase offsets.

An output signal has a phase increment in the same amount as its tuning word. If an accumulated phase is generated by a single DDS core, the 'original' sample is increased by its tuning word and directly shifted to out1' without producing samples at out1 until out8.

With the 8-core DDS module, the system shifts the accumulated phase from the phase location labelled as 'original' to 'out1' until 'out8' by using eight phase offsets. In Figure 3.3, the phase distance of ‘out1’ from the ‘original’, ‘out2’ from ‘out1’, ‘out3’ from ‘out2’ and so on are measured by the equal phase, ‘O’. The offset values that are used to reposition the ‘original’ sample can be determined by summing the respective offset values ‘O’. The offset of ‘out1’ has the same value as the step size of the output signal, ‘out2’ has an offset which is double that of ‘O’, ‘out3’ has an offset which is three times that of ‘O’ and so on. After the phase offsets are identified, they are added with an accumulated phase to generate eight sample phases over a unit time. The process is repeated on the next clock cycle where the Out1’ is the first sample of the subsequent signal.

The step size of an output signal is calculated based on the frequency tuning word of the output signal. The phase distance of each sample determines the phase offset values needed to shift an accumulated phase to eight sample phases. Phase offsets make the samples distributed in a fixed sequence so that a sine-wave can be formed. The equation for calculating the step size is:

$$\frac{phase\ increment}{C} = phase\ offset \qquad ; C\ is\ the\ total\ output\ sample\ over\ a\ unit\ time$$

$$= step\ size \tag{3.5}$$

The phase increment is the same as the frequency tuning word of a signal. The equation for calculating eight phase offsets is:

Let q be the order of eight sample phases ranging from 1st to 8th,

$$\text{Phase offset to obtain a q} - \text{th order of sample phase} = q\ x\ step\ size \tag{3.6}$$

For example, at 15 MHz frequency, the tuning word is 2013265. The step size of a 15 MHz signal is

$$step\ size = \frac{2013265}{8} = 251658.125 \approx 251658$$

The denominator, C is eight because eight sequential sample phases are generated over unit time. The step size is rounded to the nearest integer because of the phase increment value is an integer.

From the step size obtained, the eight phase offsets are:

q-th Phase Offset	**Phase Offset Value**
1st	Step Size = 251658
2nd	2 x step Size = 503316
3rd	3 x step Size = 754974
4th	4 x step Size = 1006632
5th	5 x step Size = 1258290
6th	6 x step Size = 1509948
7th	7 x step Size = 1761606
8th	8 x step Size = 2013264

Table 3.0: Calculation of Phase Offset for 1st to 8th sequential phases.

Eight samples are generated on every clock cycle. After an accumulated phase has been added to the 1st to 8th phase offsets, the adding processes would be repeated on the next clock cycle to generate the subsequent eight samples. The repeated processes allow the DDS system to generate a continuous output signal. In Table 3.0, the 8th phase offset (2013264) has one digit slightly different from the tuning word (2013265). This is mainly due to the round-off error in the step size. However, it does not affect much the signal frequency because the 25-bit phases would be truncated to upper 14 bits at the output.

3.2.4 2 GHz DDS System

A 2 GHz DDS system generates eight samples at a unit time based on the 250 MHz system clock. Each sample is generated at 250 MSPS so the entire DDS system produces the final output in parallel, which gives a result at 2 GSPS. A 2 GHz DDS system has two blocks, which are 2 GHz phase accumulator and Sine LUT blocks.

3.2.4.1 2 GHz Phase Accumulator

The 2 GHz phase accumulator block has eight adders as shown in Figure 3.4. Each adder outputs a sample phase by adding a phase offset with an accumulated phase from the accumulator. The notion of generating eight consecutive sample phases (forming the samples in numerical order) is similar to a DDS system with eight DDS cores operating in parallel.

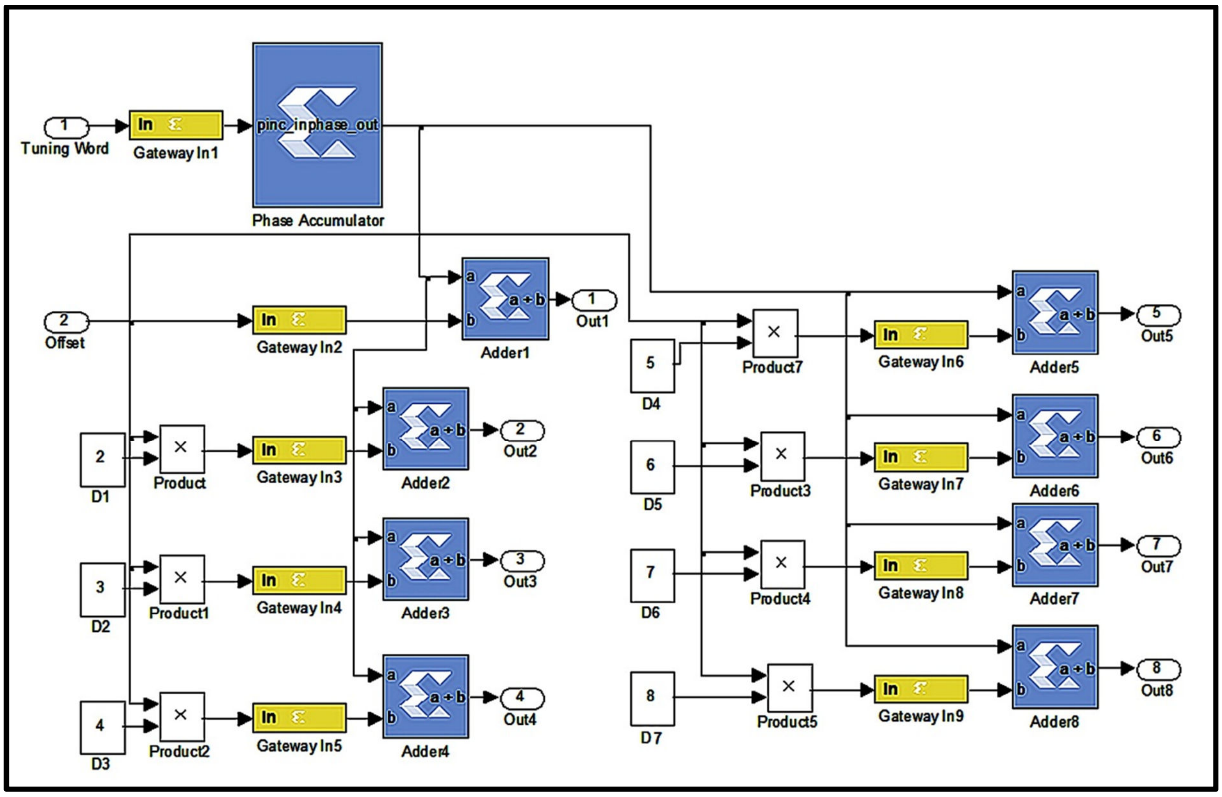


Figure 3.4: Internal structure of a 2 GHz phase accumulator block.

In a 2 GHz phase accumulator, the phase offsets are calculated by multiplying the step size of an output signal via a product block. When the clock is on the rising edge, the accumulated phases will be increased by the tuning word in the phase accumulator. The output phase from 'phase_out' of the accumulator block is copied to 'Adder1' until 'Adder8'. The first phase offset is directly connected to the Adder 1. Following this, the 2nd to 8th phase offsets are computed by multiplying the step size of the output signal with constant numbers from D1 to D7 blocks. After this, phase offsets are combined with an accumulated phase via Adder1 until Adder 8. Eight adders operate in parallel so that eight sample phases appear on the out1 to out8 Simulink ports at the same time.

The 2 GHz phase accumulator block is not exactly running at 2 GHz. It runs at a maximum of 250 MHz instead. However, the internal structure of the module could make the system generate a total number of phases, which are similar to a single 2 GHz accumulator. The phase accumulator generates an accumulated phase on every 250 MHz. The accumulated phase is shifted to eight sample phases by eight phase offsets over a unit time. Therefore,

8 x 250 MHz = 2 GHz sample rate

3.2.4.2 Sine LUT Module

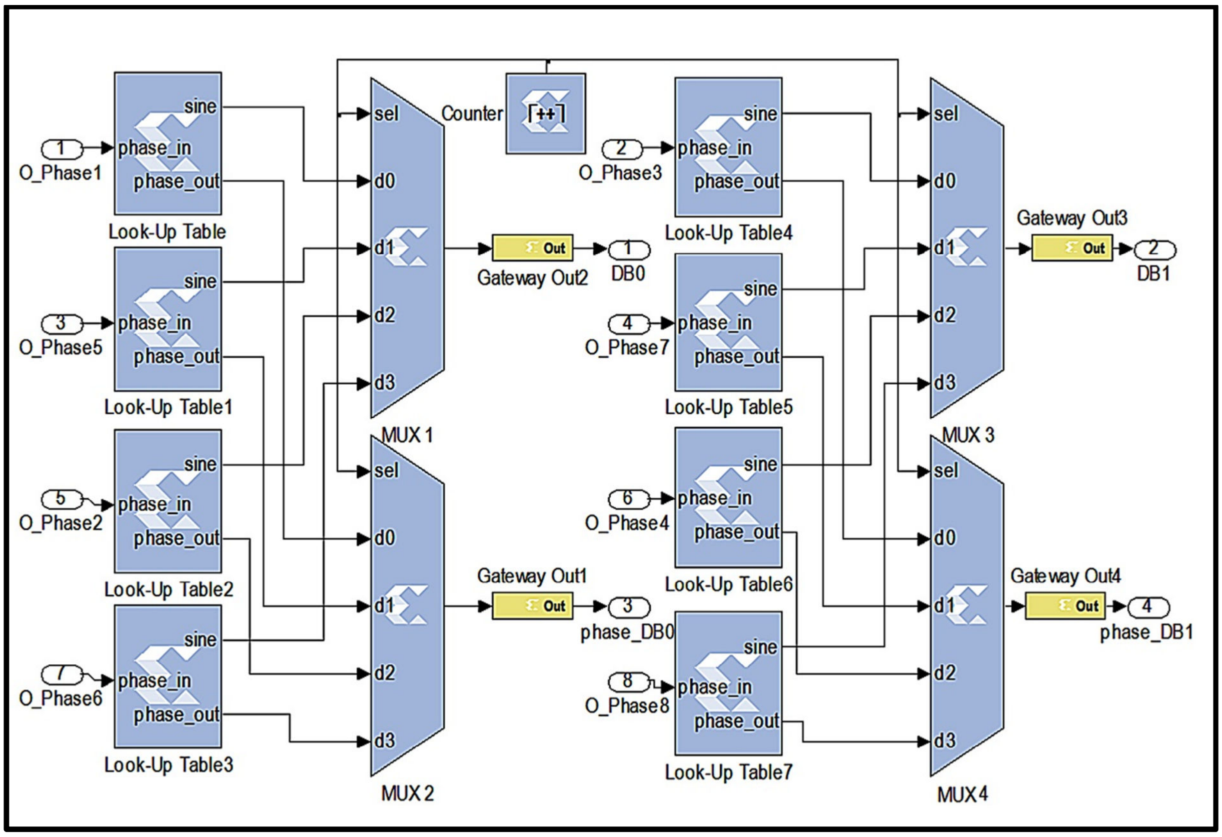


Figure 3.5: Internal structure of a sine LUT module.

In Figure 3.5, the sine LUT module requires four multiplexer blocks, eight LUT blocks and a counter block to build up the system. The reason for using eight sets of sine LUTs is to make sure that the conversion of phases into amplitudes is at the maximum 250 MHz speed [57].

When a clock is triggered, the counter starts to count in ascending order. The 1st and 2nd samples are transferred to the DAC block through MUX 1 and MUX 3 via d0 when the counter is zero. When the counter counts 1 on the second clock cycle, the 3rd and 4th samples are transferred to the DAC block through MUX 1 and MUX 3 via d1. On the third clock cycle, the counter counts 2. The 5th and 6th samples are transferred to the DAC block through MUX 1 and MUX 3 via d2. Lastly, 7th and 8th samples are shifted out from MUX 1 and MUX 3 via d3 when the counter counts 3.

The DAC block requires that the data must be deinterleaved into two data paths so that it would be interleaved by DAC internally at 2 GHz. In order to fulfil the timing requirement by the DAC block, eight sets of LUTs are used to make up 2 GSPS. In that case, each LUT processes a sample phase on every clock cycle. The two data paths are outputted by utilising MUX 1 and MUX 3 blocks. Meanwhile, MUX 2 and MUX 4 are optional blocks used to output the accumulated phases of a signal. In fact, these accumulated phases are unwanted outputs in the work presented but they have been included in order to show the relationship between the phases and the amplitudes of the output signal.

3.2.5 2 GHz DAC Module (Mock)

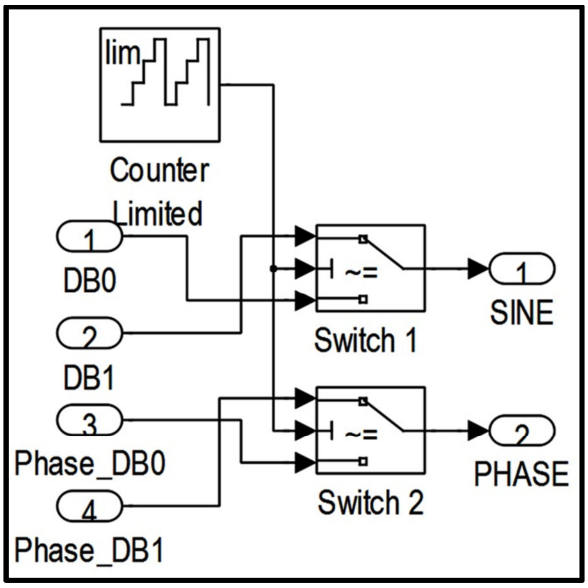


Figure 3.6: DAC module (Mock).

The 2 GHz DAC block in Figure 3.6 is only a mock module. It does not actually work like a real DAC chip. But it does multiplex four data paths (DB0, DB1, Phase_DB0 and Phase_DB1) into two outputs (SINE and PHASE). In Figure 3.6, the input DB0 and DB1 connected to Switch 1 contain the information of signal amplitudes while Phase_DB0 and Phase_DB1 connected to Switch 2 contain the information of signal phases. Each data path runs at 1 GSPS. Consequently, Switch 1 interleaves the data on DB0 and DB1 to a single output on SINE and Switch 2 interleaves the data on Phase_DB0 and Phase_DB1 to a single output on PHASE. The switching process doubles up the sampling frequency of 1 GSPS to 2 GSPS. Matlab plots the results of both SINE and PHASE to provide a clear idea of the relationship between a phase and sine amplitude of an output signal.

In the DDS design, the operation in DAC determines the final sample rate of the output signal. In order to achieve a fast sampling rate, the DAC block must be the fastest unit among the DDS components. The counter in Figure 3.6 is set to 1 sample time which is the fastest sampling period in Simulink blocks. In Figure 3.5, the counter in the LUT module sets to 2 explicit periods for multiplexing the eight LUT outputs via two multiplexers. In Figure 3.3, the phase accumulator sets to 8 explicit sample periods so that LUT can complete its conversion before the accumulator is triggered again. In summary, the ratio in the sample period for the phase accumulator module, LUT module and the DAC module are 8:2:1. By considering this ratio, the actual frequency of each module can be calculated.

DAC is 2 GHz which is equivalent to 1 sample period in DAC, so

2GHz = 0.5 ns

The LUT module rate is

2 unit explicit periods X 0.5 ns = 1 ns (1 GHz)

The frequency of phase accumulator is

8 unit explicit periods X 0.5 ns = 4 ns (250 MHz)

Hence, the Matlab-Simulink simulated 8-core DDS system is required to generate phases at a 250 MHz sample rate. The LUT module provided two output channels to DAC at 1 GHz each. Lastly, the DAC block interleaved the incoming amplitudes at 2 GSPS.

Table 3.1 lists the sample period settings for the major blocks in Simulink. For any blocks which are not listed in the table, the setting is default (refer to Appendix for parameter settings).

2 GHz Phase Accumulator Module		**Sine LUT Module**		**DAC Module (Mock)**	
Block	**Sample Period**	**Block**	**Sample Period**	**Block**	**Sample Period**
Gateway_In	8	Counter	2	Counter Limited	1
Phase Accumulator block	8				

Table 3.1: Sample period of Simulink block.

3.3 Conceptual Design of a Multi-Frequency RF Generator

A system with multi-channel adjustable frequency and phase is extremely expensive. Moreover, this system is even more difficult to control if the system made of a single DDS core [66]. In this thesis, a multi-frequency RF generator has been designed by modifying the existing 8-core DDS module to handle multi-frequency generation. The supported output frequency is still the same as in the 8-core DDS module, which ranges from 100 kHz to 750 MHz. The multi-frequency RF generator can avoid the phase coherence loss of signals because the system supports switching between four distinct frequencies. The generator can self-sustain the phases of these frequencies while its outputs are inter-switching from one frequency to another.

3.3.1 Design Requirements

The settings of the sample period in the multi-frequency RF generator remain the same as in an 8-core DDS module. The design of a multi-frequency RF generator should satisfy the following requirements:

System Clock, f_{system}	: 250 MHz
SNR	: ~86 dB
Frequency Resolution, Δf	: 0.93 Hz
Input Width of LUT	: 25-bit
Input Width of LUT	: 14-bit
Depth of sine LUT	: 16384 samples
Input width of the Phase Accumulator	: 25-bit
Depth of Phase Accumulator	: 33554432
Total Phase Offsets	: 8 sets with 25-bit of each

3.3.2 Simulink of Multi-Frequency RF Generator

Figure 3.7 shows an overview of the Simulink structure designed for a multi-frequency RF generator. The sine LUT and DAC modules are the same function blocks used in the 8-core DDS module. The new added feature is the 'Multi-Frequency DDS' block. The multi-frequency DDS block has five input ports. Four inputs are used to read the frequency values of output signals labelled as f1 to f4. Another input (Select) is to read the logic status of 'Switch-A and Switch-B' blocks.

'Fout_1' to 'Fout_4' are the constant blocks, which contain the frequencies of four output signals. For example, 50 MHz, 30 MHz, 15 MHz, and 5 MHz are randomly chosen to support the simulation of a multi-frequency RF generator in a later stage. These constant blocks connect to f1 until f4 respectively for phase accumulation in accumulators. However, only one set of phases would be processed by the 'Sine LUT' block. The 'Sine LUT' block provides the information of signal amplitudes and phases on the output ports, which are known as SINE and PHASE.

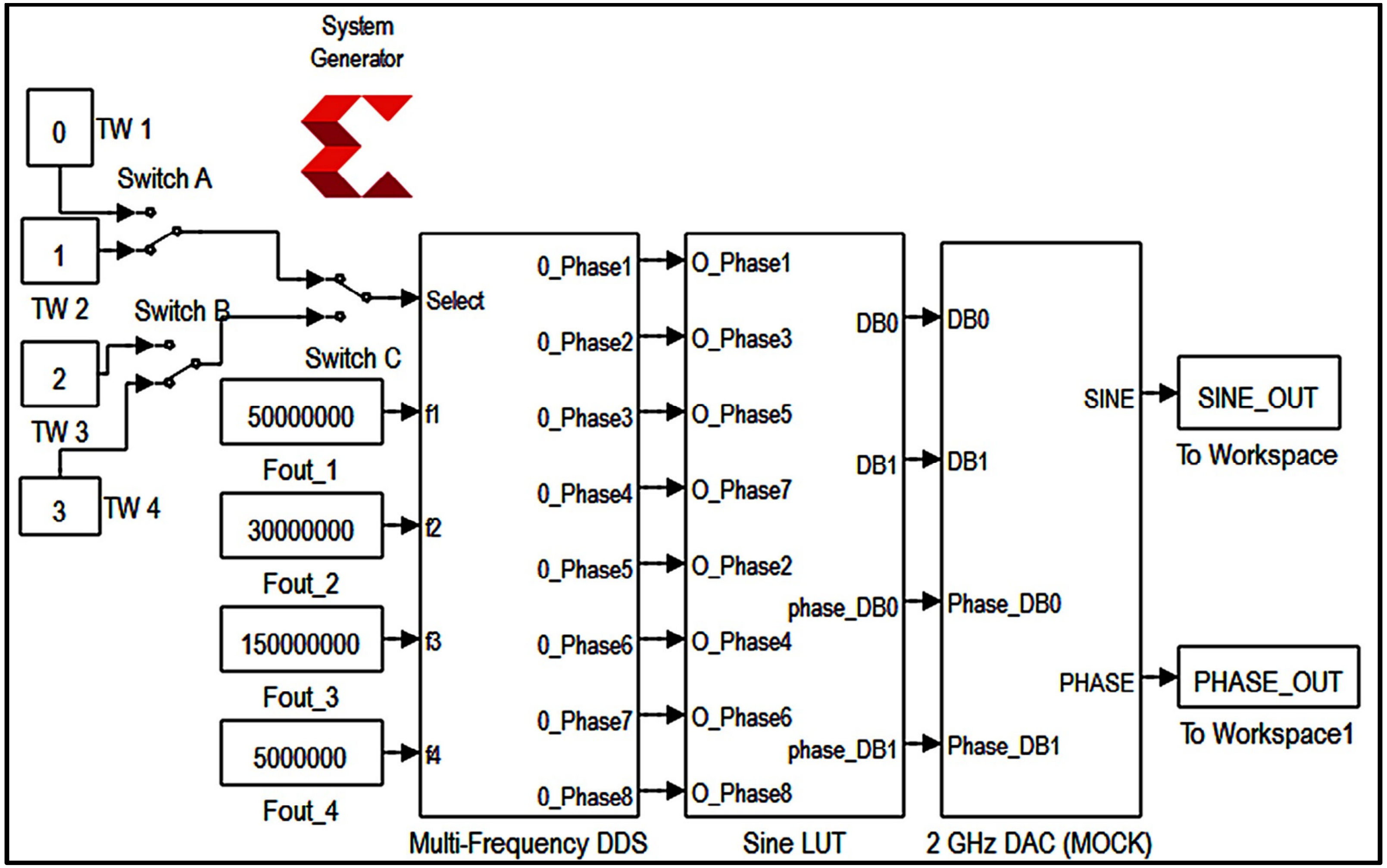


Figure 3.7: Simulink model of Multi-Frequency DDS module.

3.3.2.1 Multi-Frequency DDS Block

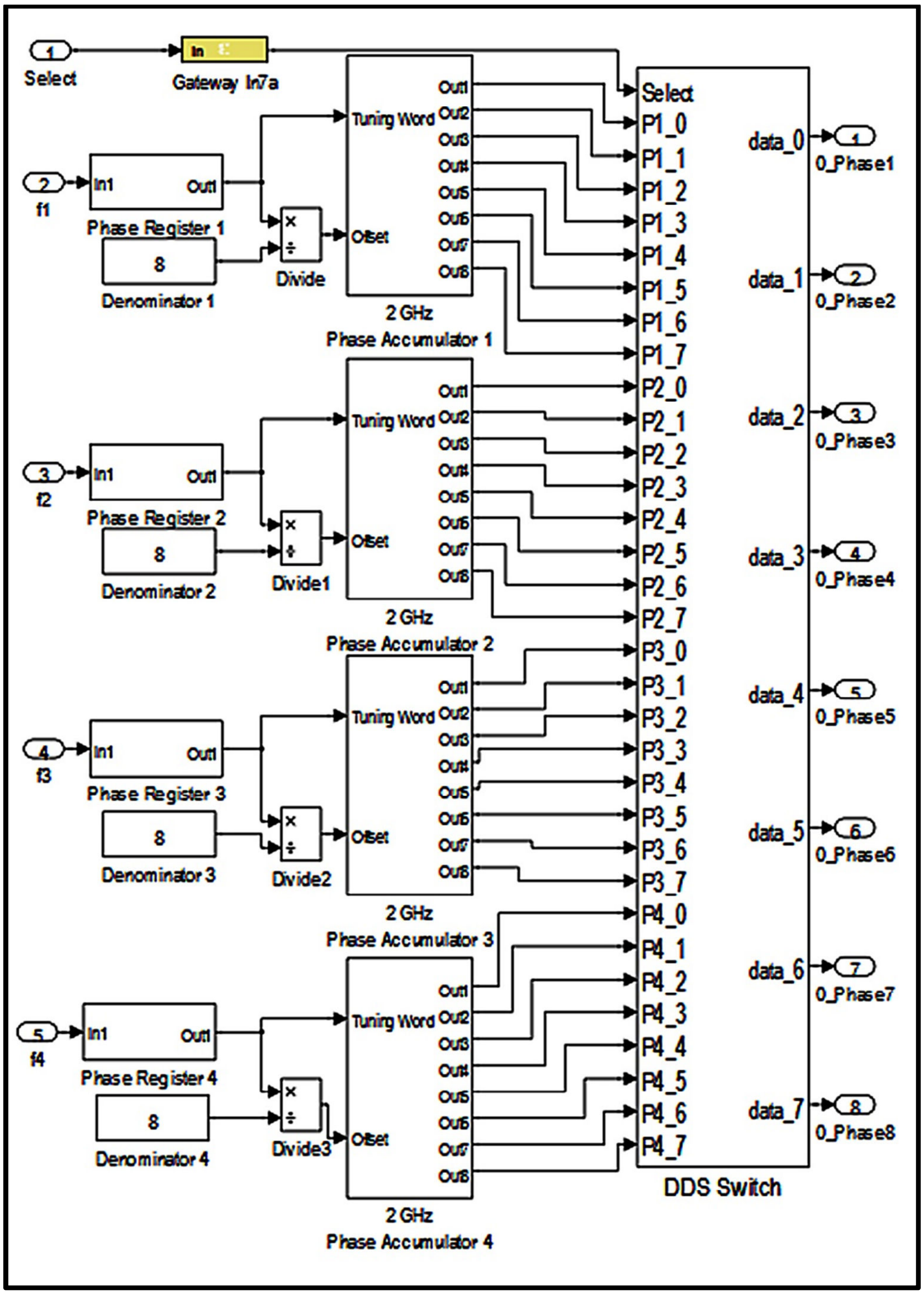


Figure 3.8: Internal structure of a multi-frequency DDS function block.

Figure 3.8 shows the internal structure of a multi-frequency DDS block. There are three main parts in this block. The first part contains the phase register blocks, which are used to calculate the phase offsets and tuning words. The second part comprises the 2 GHz phase accumulator blocks, which are used to perform the phase accumulation of four signals. The last part comprises the newly added block, which is a DDS switch block used to select the phases among the four signals to LUTs.

In the multi-frequency DDS block, the inputs of f1 until f4 will be converted to the frequency tuning word in Phase Register 1 to Phase Register 4 blocks. Step size of each signal is calculated by dividing its tuning words with Denominator 1 to Denominator 4, respectively. The values of step size would be then processed by the 2 GHz phase accumulator blocks to calculate the eight phase offsets of each signal. The phase offsets of each signal would combine with the accumulated phases to reproduce 8 times more samples on the outputs. Four phase accumulators operate simultaneously. Therefore, the samples of four signals are available at the same time to the DAC switch block. However, only the sample phases of one signal would be shifted to the LUT module. The selections among these signals are controlled by the logic status of the 'Select' port.

Phase registers and accumulators are the customised blocks inherited from an 8-core DDS module. Therefore, this section explains the features inside the DDS switch block.

- *DDS Switch block*

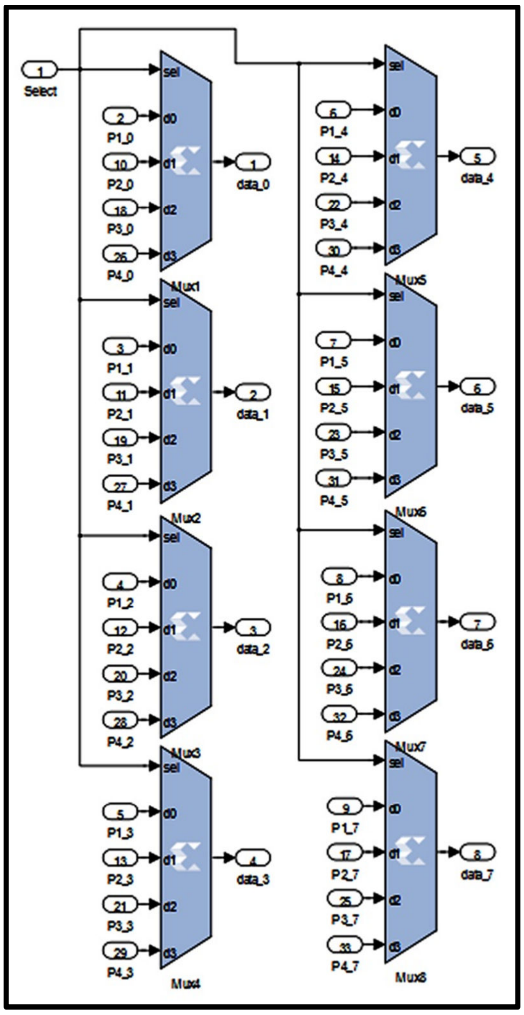


Figure 3.9: Internal structure of the DDS switch block.

The DDS switch block is very simple; it is formed by eight multiplexers. As shown in Figure 3.9, each multiplexer has four inputs. The inputs are the phases from 2 GHz phase accumulators. Since there will be eight samples generated over unit time, the connections of these sample phases to the multiplexer blocks are as below:

i. 1st consecutive samples among the four signals (out1) connect to Mux1

ii. 2nd consecutive sample among the four signals (out2) connects to Mux2

iii. 3rd consecutive sample among the four signals (out3) connects to Mux3

iv. 4th consecutive sample among the four signals (out4) connects to Mux4

v. 5th consecutive sample among the four signals (out5) connects to Mux5

vi. 6th consecutive sample among the four signals (out6) connects to Mux6

vii. 7th consecutive sample among the four signals (out7) connects to Mux7

viii. 8th consecutive sample among the four signals (out8) connects to Mux8

Notes:

1. *Each 2 GHz phase accumulator has an output connected to the multiplexer.*
2. *The out1 to out8 are the outputs from each 2GHz phase accumulator.*
3. *A 1st consecutive sample from each 2GHz phase accumulator is connected to Mux1. A 2nd consecutive sample from each 2GHz phase accumulator is connected to Mux2 and so on (the same connection manners are repeated on 3rd to 8th consecutive samples).*

Logic status of 'Select' determines which signal samples among the four frequencies will be the final output. Table 3.2 shows the selection of the signal samples based on the status of the 'Select' port.

Select port	**Phases from**
0	2 GHz Phase Accumulator 1
1	2 GHz Phase Accumulator 2
2	2 GHz Phase Accumulator 3
3	2 GHz Phase Accumulator 4

Table 3.2: Selection of output signal via Switch-A and Switch-B.

3.4 Matlab-Simulink Simulation Results

15 MHz is randomly chosen as the frequency to test a single DDS core and an 8-core DDS module so that the outputs generated from each module can be compared easily in this thesis. Besides that, 10 MHz, 30 MHz, 45 MHz, and 90 MHz are randomly chosen as the frequencies to test a multi-frequency RF generator. Two random signals (10 MHz and 30 MHz) are randomly chosen to examine the phase coherence of these two signals. All the sample periods of Simulink blocks remain default unless these have been specified in the specification.

3.4.1 Simulation Results of a Single DDS Core

The purpose of running the simulation on a standard DDS core is to observe the relationship between the phases and the amplitudes generated by the core. The explicit sample period of the phase accumulator and LUT is set to 8 in order to standardise the sampling period between a single DDS core and an 8-core DDS module. Consequently, the outputs of both simulations can be compared (both outputs have the same sampling period).

The simulation is based on the Simulink design in Figure 3.0. The parameters of the simulation are:

Fout	: 15000000 Hz
Phase Accumulator System Clock, f_{system}	: 250 MHz
Explicit Sample Period of Phase Accumulator core	: 8
Simulation Period	: 500

The simulation period is set to 500 unit time so that the design system can generate a couple of complete sine-waves within this period. When the simulation is executed, 15000000 Hz is converted to the frequency tuning word in the phase register block. The step size of the output equals the phase increment of the output frequency. Based on equations (2.0) and (3.5), the tuning word is 2013265. Hence, the step size is 2013265. The frequency resolution is fixed at 7.45 Hz for a single DDS core with 250 MHz system clock and 25-bit phase accumulator.

$$tuning\ word = \frac{15000000\ x\ 2^{25}}{250\ M} = 2013265$$

$$step\ size \quad = \frac{2013265}{1} \quad = 2013265$$

Figure 3.10 is the result obtained after the simulation. The results show how a unit cycle of sine-wave matches with the overflow of the 25-bit sample phase. The 1st accumulated phase is 0.06 because the tuning word, 2013265, is normalised by the gain factor in the phase register.

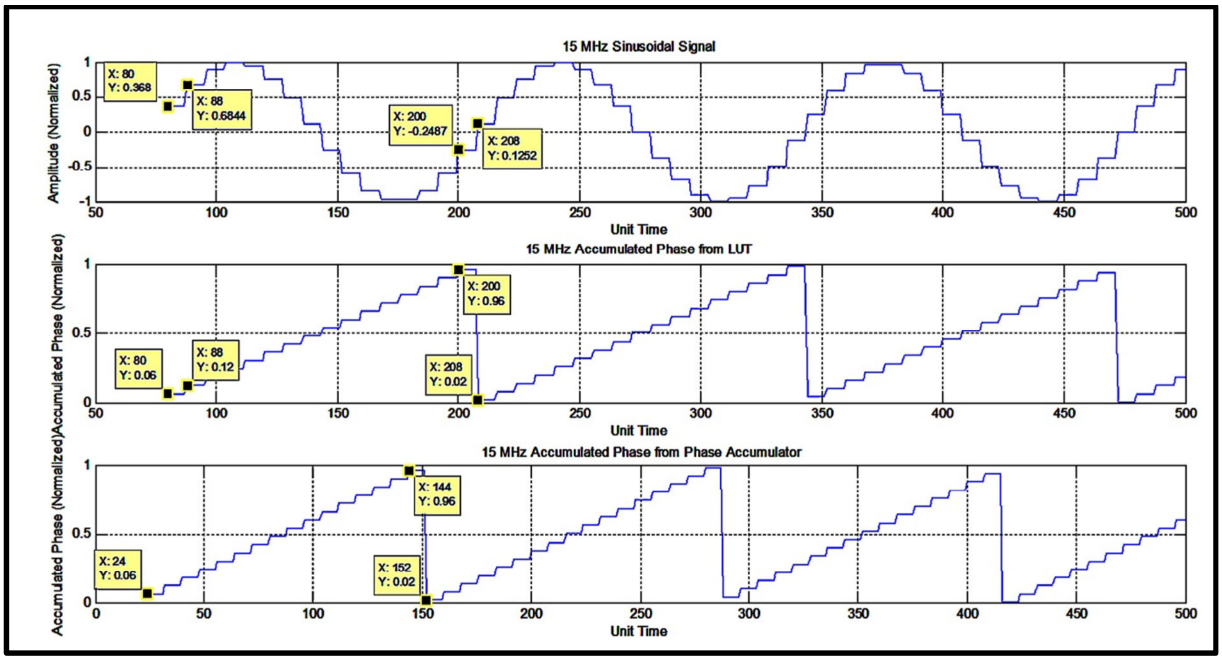


Figure 3.10: Outputs of 15 MHz sine-wave and accumulated phases.

In Figure 3.10, the first plot (top) is a 15 MHz sinusoidal signal. The second plot (centre) is 15 MHz accumulated phases captured in the output of a LUT. The third plot (bottom) is 15 MHz accumulated phases captured in the output of a phase accumulator. 15 MHz accumulated phases on the third plot have been redistributed by a LUT block. Therefore, the first samples on the second (X=80) and third plots (X=24) are located in a different time (80-24=56 latencies).

In the third plot, the first sample is generated at X=24. In other words, the phase accumulator has a latency of 24 unit time due to the 8 explicit sample periods (a phase accumulator with one explicit sample period requires three latencies). In Figure 3.10, the first output phase is equivalent to the normalised tuning word 0.06. After LUT conversion, 0.06 phase is converted to amplitude 0.368 at X=80 unit time. The LUT spent 56 unit time to convert the phase-to-amplitude. The entire 8-core DDS module spent 80 unit time to have the first amplitude appear on the output line of DAC block (DAC has zero latency in Simulink).

The second sample is at X=88 which is 8 unit time apart from the first sample. It is because the explicit sample period of the phase accumulator is eight so the accumulator generates a subsequent accumulated phase every 8 unit time.

The accumulated phases of X=80 to X=200 from the second plot are the same as X=24 to X=144 from third plot. These phases contain the phase information of a unit cycle of sine-wave. The amplitude '0.368' is converted from the accumulated phase 0.06 at X=80. Moreover, the amplitude '-0.2487' is converted from the accumulated phase 0.96 at X=200. These are the first and last amplitudes of a full sine cycle. A unit cycle of sine-wave is measured from the first samples until the last sample before the phase accumulator overflow.

Once the phase accumulator overflows, the carry out bit is discarded. Thus, the total phase drops to a minimum value which is the carry forward phase 0.02 (X=208) from the past. At this moment, X=208 becomes the primary sample phase of the subsequent sine-wave. When the entire processes are going on, a continuous sinusoidal signal can be produced.

In order to make sure the timing of a single DDS core matches the timing of an 8-core DDS module, the explicit sample period of a phase accumulator is set to 8 so that the accumulator generates samples on every 8 unit time. The purpose is to make sure a single DDS core has the same timing as the 8-core DDS module so that results generated by a single DDS and 8-core DDS can be compared. A complete 15 MHz sine-wave generated by a single DDS core requires 128 unit time which can be computed by taking the absolute unit time between the first and last samples of a sine cycle.

200-80+8 = 128 unit time

While calculating the total unit time required to generate a cycle of sine-wave, the sample at X=80 must be included because it is the first sample of a signal. Since the sample period is 8, the total samples required by a unit cycle of sine-wave is 128/8 = 16 samples.

The frequency from Figure 3.10 can be calculated by:

$f_{system} - system\ frequency$

The time period of a sample, $t_{sample} = \frac{1}{f_{system}}$

Let us assume:

Total samples of a sine-wave = $T_{samples}$

Output frequency = $\frac{1}{T_{samples} \; X \; t_{sample}}$

So, output frequency, $f_{out} = \frac{f_{system}}{T_{samples}}$ (3.7)

Therefore, the output frequency of the sine-wave in Figure 3.10 is

$f_{system} \; = 250 \; MHz$

$T_{samples} = 16$

$$\therefore \frac{250 \; MHz}{16 \; samples} = 15.625 \; MHz$$

The additional 0.625 MHz is the round-off error due to the overflow between X=200 and X=208. The exact samples required to obtain a 15 MHz frequency are $\frac{250 \; MHz}{15 \; MHz} = 16.6667 \; samples$. This value falls between the 16th sample and 17th sample. Since the number of total samples is an integer, so the $T_{samples}$ are rounded to 16. That is the approximated sample before the phase accumulator overflows.

The calculation of the total samples above is based on a unit cycle of sine-wave. When it comes to a continuous signal, the fraction is equally distributed on the subsequent signal which would blur the boundary between the first and the subsequent signals.

3.4.2 Simulation of an 8-core DDS Module

The simulation carried out in this section is based on Figure 3.1. Two simulations have been planned in this section. The first simulation shows the relationship between a tuning word and eight phase offsets. The second simulation is to examine the impact of the eight phase offsets on the output waveform. The parameters of the simulation are:

Fout	= 15000000 Hz
Phase Accumulator System Clock, f_{system}	= 250 MHz
Explicit Sample Period of Phase Accumulator	= 8
Explicit Sample Period of LUT	= 1
Simulation Period	= 500

The tuning word of 15 MHz is 2013265, which was calculated by using equation (2.0). The step size of each output sample of an 8-core DDS module, 25168, was calculated by using equation (3.5). The tuning word and step size of a 15 MHz frequency generated by an 8-core DDS module are

$$Tuning\ Word = \frac{15\ M\ x\ 2^{25}}{250\ M} = 2013265$$

$$Step\ size \quad = \frac{2013265}{8} \quad = 251658$$

3.4.2.1 Simulation Results of Phase Offset

Figure 3.11 shows part of the Simulink blocks, which are used to calculate the tuning words and the phase offsets (highlighted in red). In Figure 3.11, the tuning word of 15 MHz has been calculated and normalised in a phase register. The output would be sent to a 2 GHz phase accumulator block through the 'tw_out' port. The normalised frequency tuning word is used to generate the phases of a sine-wave signal in the phase accumulator block.

The normalised tuning word on the phase register block is measured as 0.06 (Display_tw). This 0.06 is divided by eight (denominator) to produce 0.0075 (normalised step size) via a divider block. 0.0075 is then multiplied by 2 (D2), 3 (D3), 4 (D4), 5 (D5), 6 (D6), 7 (D7) and 8 (D8) to produce 2nd to 8th phase offsets. Each phase offset is combined with an accumulated phase from the 'phase_out' through the respective Adder1 until Adder8. This methodology generated eight continuous samples 'Out1' to 'Out8' over a unit time.

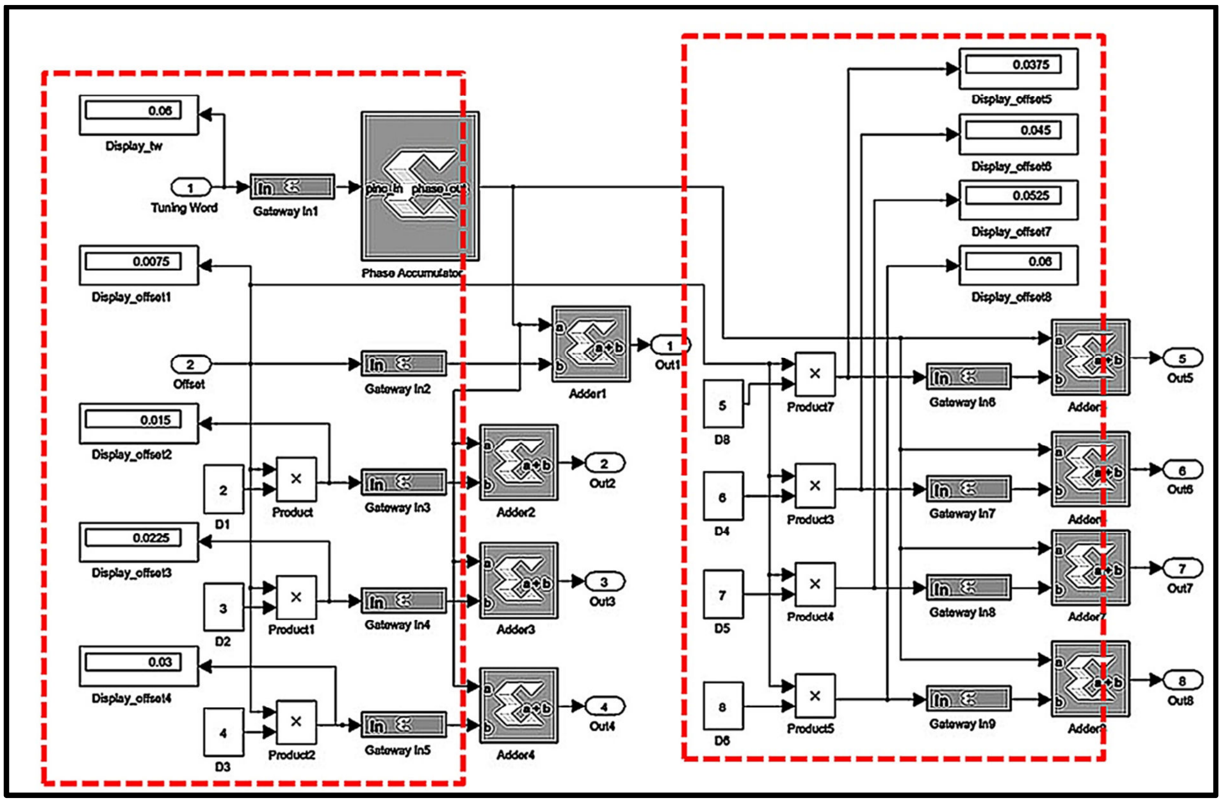


Figure 3.11: Matlab-Simulink model for phase offset simulation.

The results shown on the displays in Figure 3.11 can be used to trace the actual phase values from the normalised outputs. The results are shown in Table 3.3.

Display	**Simulink**	**Mathematical Calculation (Integer)**
Display_tw	0.060	0.060 X $2^{25} \approx$ 2013265
Display_Offset1	0.0075	0.0075 X $2^{25} \approx$ 251658
Display_Offset2	0.0015	0.015 X $2^{25} \approx$ 503316
Display_Offset3	0.0225	0.0225 X $2^{25} \approx$ 754974
Display_Offset4	0.03	0.03 X $2^{25} \approx$ 1006632
Display_Offset5	0.0375	0.0375 X $2^{25} \approx$ 1258291
Display_Offset6	0.045	0.045 X $2^{25} \approx$ 1509949
Display_Offset7	0.0525	0.0525 X $2^{25} \approx$ 1761607
Display_Offse8	0.06	0.06 X $2^{25} \approx$ 2013265

Table 3.3: Comparison between Simulink and mathematical calculation results.

The results from the Simulink and mathematical calculations are identical. The normalised outputs can be restored to their original values by multiplying each normalised result with 2^{25} (inverse of gain factor).

3.4.2.2 Simulation Results of an 8-core DDS Module

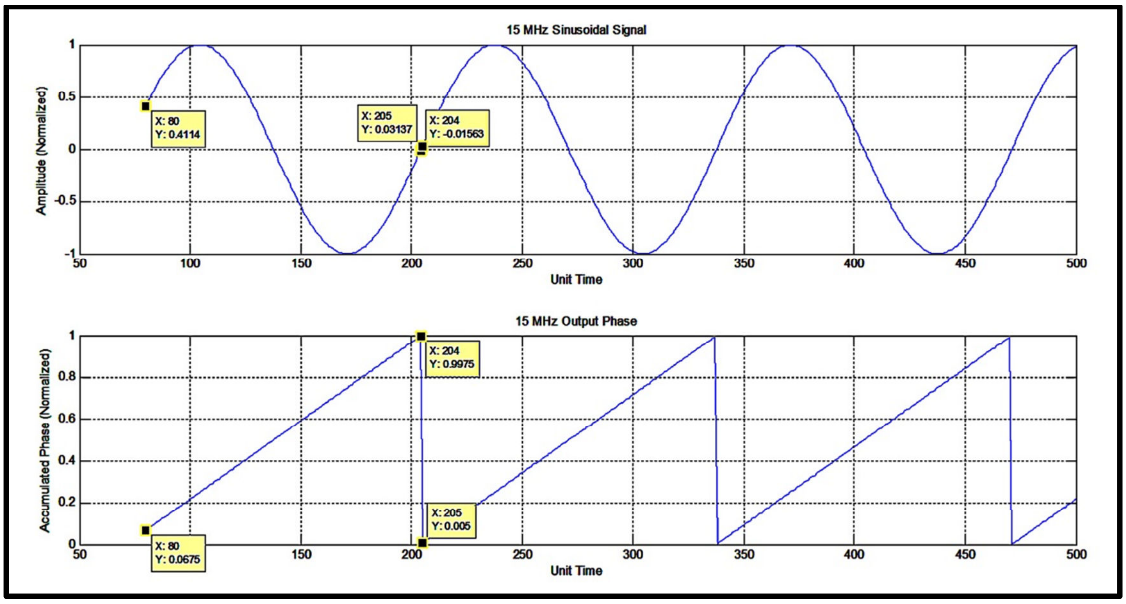


Figure 3.12: Simulation results of an 8-core DDS module captured at sine LUT module.

Figure 3.12 shows the simulation results of a 15 MHz sine-wave and its accumulated phases. These two plots are captured in the output of a sine LUT block by using the Matlab workplace. Therefore, both have the same output delays. The delays of the output are caused by the latency of the phase accumulator and the LUTs in the system.

A 2 GHz phase accumulator uses 24 unit time in the accumulation. When phases are passed to the sine LUT module, the conversion of the phase-to-amplitude takes 56 unit delays. Hence, the actual time of a phase sample before being processed by the LUT module can be obtained by deducting the unit time with 56.

In Figure 3.12, the initial sample phase appeared at 80 unit time on the LUT output line. The final sample phase of a full sine-wave appeared at 204 unit time. After converting to amplitude, a complete sine cycle is depicted by X=80 to X=204 on the X-axis. X=204 is the unit time just before the phase accumulator overflows. The phase accumulator overflows at X=205. Hence, the accumulated phase drops to the minimum. The accumulation keeps going on to generate samples for the subsequent sine-wave.

By comparing the plots in Figure 3.10 and Figure 3.12, a 15 MHz sine-wave generated by an 8-core DDS module requires 125 samples. In contrast, a single DDS core has only 16 samples to form a sine-wave. The 8-core DDS module improved the precision of the signal by contributing 125-16=109 more samples. It is equivalent to increasing the aggregated samples by 8 times.

The output frequency of the sine-wave in Figure 3.12 can be calculated by using the equation (3.8) below:

An 8-core DDS module generates eight samples over a unit time. Therefore, total samples generated by an 8-core DDS module for a signal are 8 times more than a single DDS module for the same frequency.

From the equation (3.7):

$$Total\ samples, T_{samples} = 8\ x\ \frac{f_s}{f_{out}}$$

So, output frequency, $f_{out} = \frac{8\ x\ f_s}{T_{samples}}$ (3.8)

Considering the system frequency and the total samples obtained from Figure 3.12:

$$f_{system} = 250\ MHz$$

$$T_{samples} = 125$$

$$\therefore Output\ frequency = \frac{8\ x\ 250\ MHz}{125}$$

$$= 16\ MHz$$

1 MHz is the round-off error due to the overflow of the phase accumulator. The phase accumulators of a single DDS core and an 8-core DDS module have the same tuning word and sample period. Thus, both overflow at the same time. Therefore, the accumulated phases before and after the phase accumulator overflow are 16th and 17th. Hence, 16 X 8 =128th and 17 X 8 = 136th sample phases. The multiplication by eight is because eight sample phases are produced by the 8-core DDS module over unit time.

In an 8-core DDS module, the exact amount of samples required to obtain the 15 MHz frequency is $\frac{8\,x\,250\,MHz}{15\,MHz} = 133.333$ samples. This value falls between 128th and 136th samples. However, a complete sine cycle (15 MHz) generated by an 8-core DDS module contains only 125 samples which are 8.333 lesser than the calculated 133.333 samples. This is because one or more of the eight sample phases overflow at 126th sample phase after adding the accumulated phases of 15 MHz with its phase offsets. Therefore, a subsequent sine-wave is created after 125th sample phase.

The 16 MHz output frequency is computed based on the total samples of a unit sine cycle. When it comes to a continuous signal, the DAC blurs the boundary between the first and the subsequent signals. Therefore, the signal is always measured at 15 MHz.

Figure 3.13 shows the comparison of the phases generated by a single DDS core and an 8-core DDS module.

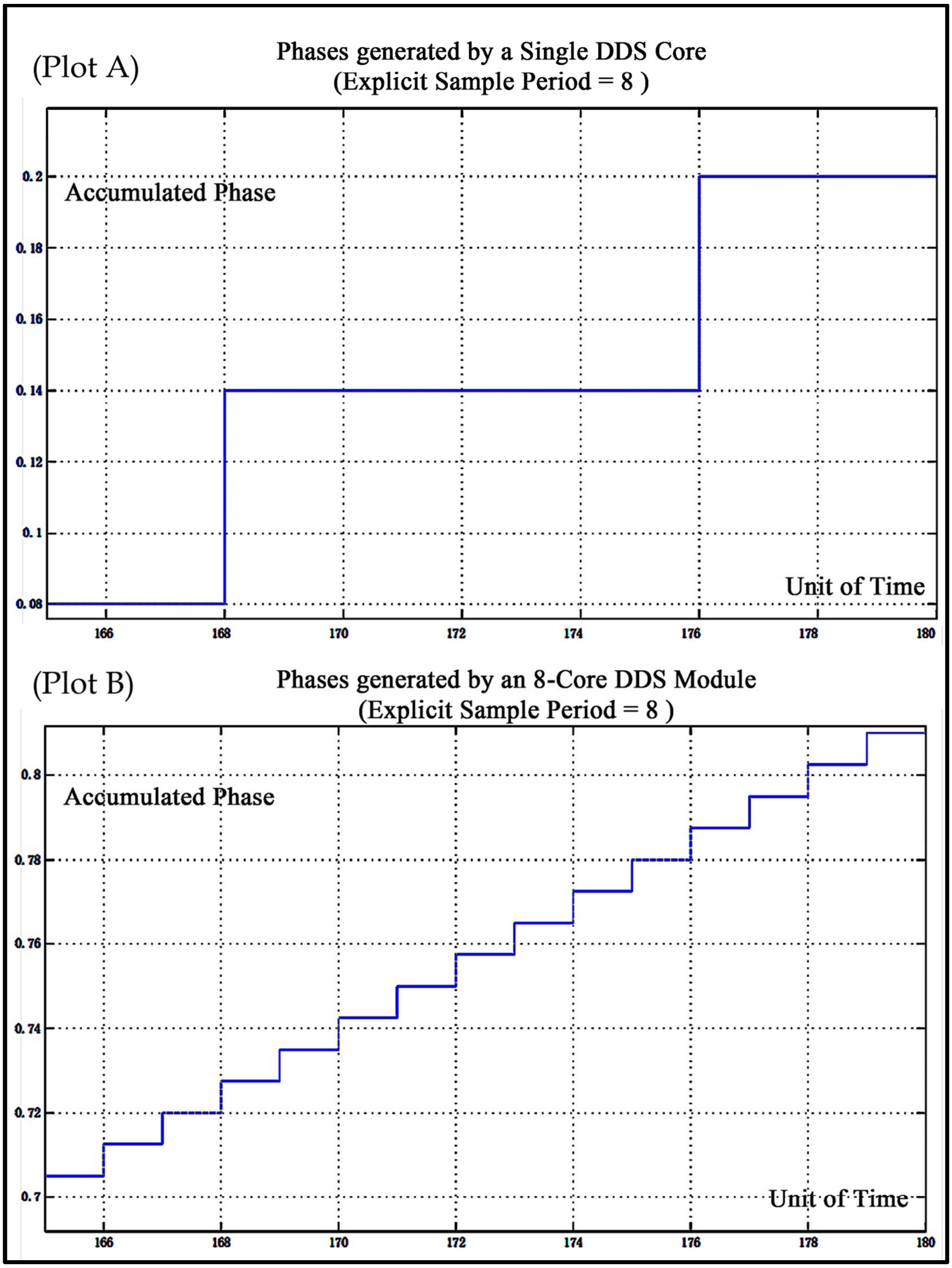


Figure 3.13: Phase steps of a single DDS core and an 8-core DDS module (15 MHz).

In Figure 3.13, it can be seen that a single DDS core has only one phase step during 168 to 176 unit time. However, an 8-core DDS module managed to insert eight samples within the same period. The step size shown in plot A is 2013265 while in plot B it is 251658. This means the step size of an 8-core DDS module is 8 times narrower than a single DDS core and 8 times faster than a single DDS core over a unit time.

3.4.3 Simulation Results of a Multi-Frequency RF Generator

The parameters of the simulation are:

Phase Accumulator System Clock, f_{system}	: 250 MHz
Explicit Sample Period of Phase Accumulators	: 8
Explicit Sample Period of LUTs	: 1
Simulation Period	: 7000

The structural design of a multi-frequency module from Figure 3.7 is used to run the simulation on four random frequencies. These four frequencies are:

- Fout_1 block :10 MHz
- Fout_2 block :30 MHz
- Fout_3 block :45 MHz
- Fout_4 block :90 MHz

The frequency tuning words of 10, 30, 45, and 90 MHz are 1342177, 4026531, 6039797, and 12079595, respectively. These tuning words can be calculated based on equation (2.0) and the step size of each frequency can be calculated by using equation (3.5). A 10 MHz signal has a step size of 167772, a 30 MHz signal has a step size of 503316, a 45 MHz signal has a step size of 754974, and a 90 MHz signal has a step size of 1509949.

Figure 3.14 shows the simulation results of a multi-frequency RF generator. Four signals generated by the generator can be selected by three switches labelled as Switch-A, Switch-B, and Switch-C. The interconnections between the switches are below:

- Switch-A is connected to either TW1 (10 MHz) or TW2 (30 MHz)
- Switch-B is connected to either TW3 (45 MHz) or TW4 (90 MHz)
- Switch-C is connected to either Switch-A or Switch-B

The default connections between the switches are shown in Table 3.4.

Switch	Connection
A	TW1
B	TW3
C	Switch-A

Table 3.4: Default connection of switches.

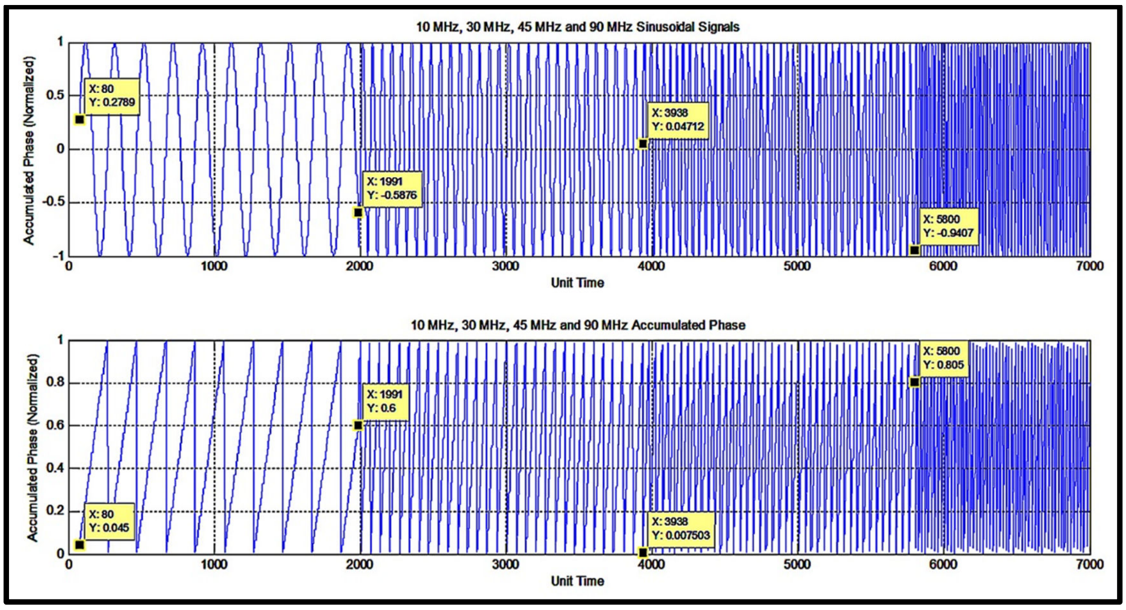


Figure 3.14: Four signals generated by a multi-frequency RF generator.

In Figure 3.14, the initial signal has a frequency of 10MHz. Switch-A changes from TW1 to TW2 after 1911 unit time. This causes the output signal to be switched to 30 MHz at X=1991. The signal oscillation is carried on until it reaches 3937 unit time. At X=3938, the frequency is changed to 45 MHz by Switch-C. Again, the output signal switches to 90 MHz at 5800th unit time after changing Switch-B to TW4. The simulation results show the design system can produce four frequencies once it was executed.

3.4.3.1 Simulation results of Phase Coherence for Multi-Frequency RF Generator

The parameters of the simulation are:

Output Frequencies	: 10 MHz and 30 MHz
Phase Accumulator System Clock, f_{system}	: 250 MHz
Explicit Sample Period of Phase Accumulators	: 8
Explicit Sample Period of LUTs	: 1
Simulation Period	: 7000

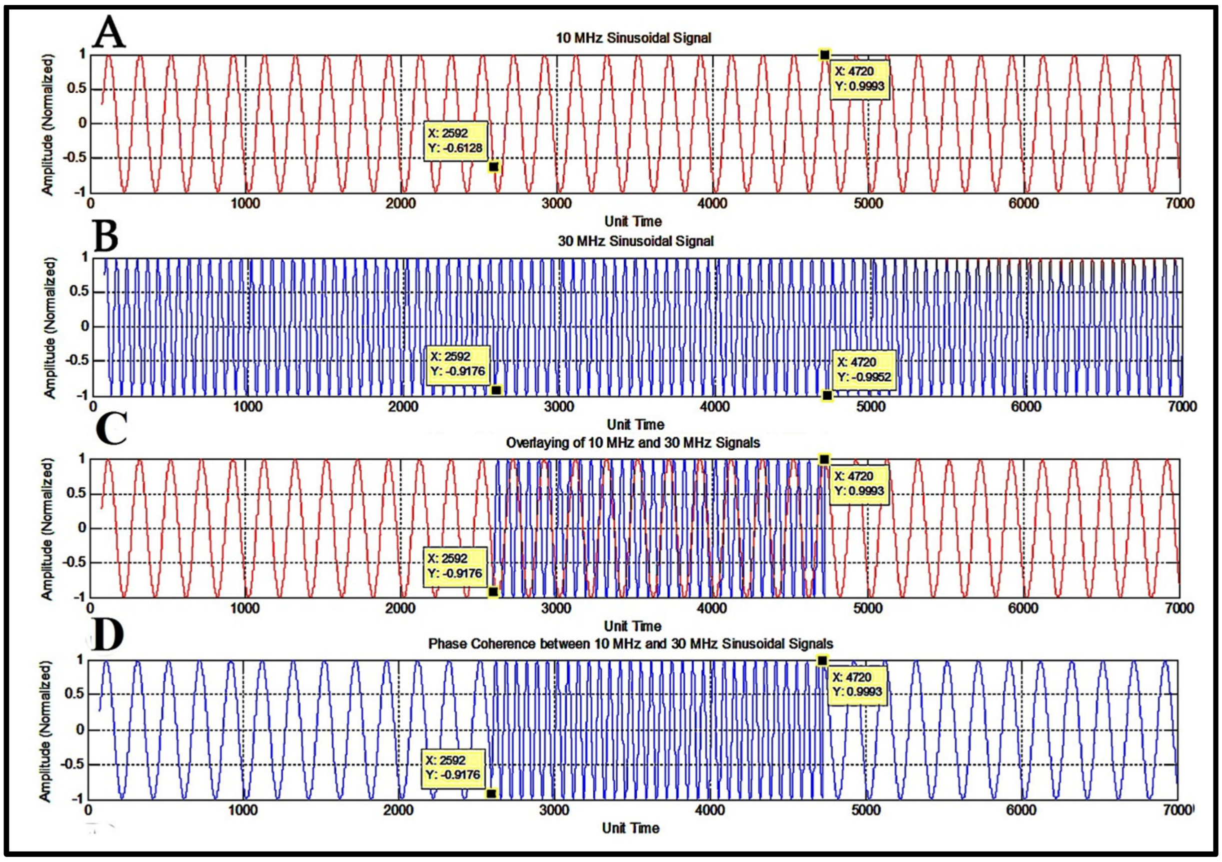


Figure 3.15: Phase coherence between 10 MHz and 30 MHz signals.

In Figure 3.15, the signal in Plot A is 10 MHz while the signal in Plot B is 30 MHz. This assumes that plot A and plot B are two independent signals running in parallel. The result in plot C can be obtained by overlaying plot A with plot B. Plot C approaches the waveform as shown in plot D.

Plot D is the output obtained while the multi-frequency RF generator switches between 10 MHz and 30 MHz sine-waves. A 10 MHz signal is oscillated by default. When the 10 MHz signal reaches X=2591 unit time, the output is toggled to 30 MHz at X=2592. Phase change occurs during the transition from 10 MHz to 30 MHz. The 30 MHz signal oscillates until X=4719. From X=4720 onwards, the output signal switches back to 10 MHz. Again, phase change occurs in the transition from 30 MHz to 10 MHz signals. 10 MHz signals before X=2592 and after X=4719 are in-phase. 10 MHz signals before and after switching are in phase coherence.

The multi-frequency RF transmitter makes the excitation of nuclei at different times while the phase coherence is maintained. This can be used by physicists to examine the physical and chemical properties of molecules or molecules of a sample under analysis in an NMR system.

3.5 RTL-Level Simulation of DDS Using FPGA

In this section, the entire DDS system is modelled in the behavioural level simulation with VHDL. The very first step is to simulate a simple DDS module by using the Xilinx ISE simulator before adding extra modules to make up an 8-core DDS system. The model helps in understanding the behavioural operations of each block in the module.

3.5.1 **Architecture of PLL**

Figure 3.16 shows the block diagram of an FPGA-based PLL module. Xilinx has recommended this module for high-speed operation. The main purpose of the PLL is to serve as a frequency synthesiser for a wide range of frequencies, and to serve as a jitter filter for either external or internal clocks. The PLL_BASE block in the PLL module is used to capture a differential 100 MHz clock signal from the clock chip and regenerate a 250 MHz clock to DDS components and a 10 MHz clock to the DAC chip for Serial Peripheral Interface (SPI) configuration.

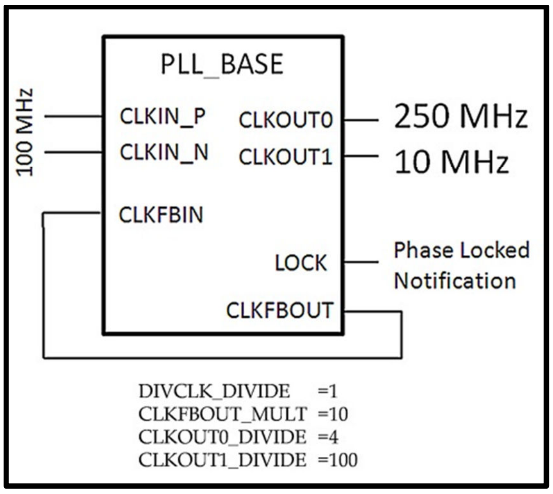


Figure 3.16: PLL module [61].

The PLL_BASE provide jitter control over the output clock signals by reducing jitter inherent on the reference clock. It filters the external clock before the clock is driven into other fabrics. The jitter filtering is adjustable by configuring the PLL attribute BANDWIDTH to be low, high or optimised [61]. The setting of the PLL_BASE can be configured through the clocking wizard or manually in VHDL coding. The attributes of the PLL module are:

Attribute	Value	Description
BANDWIDTH	Optimised	To specify the PLL programming algorithm affecting the jitter, phase margin and other characteristics of the PLL.
CLKFBOUT_MULT	10	To specify the amount to multiply all CLKOUT clock outputs by to a different frequency.
CLKFBOUT_PHASE	0	To specify the phase offset of the degree of the clock feedback output.
CLK_FEEDBACK	CLKFBOUT	To specify the clock source to drive CLKFB_IN.
CLKIN_PERIOD	10 (ns)	To specify the input period in ns to the PLL CLKIN1 input. Resolution is down to the ps.
DIVCLK_DIVIDE	1	To specify the division ratio for all output clocks with respect to the input clock.
CLKOUT0_DIVIDE	4	To specify the amount to divide the associated CLKOUT0 clock output by if a different frequency is desired.
CLKOUT0_DUTY_CYCLE	0.5	Specifies the Duty Cycle of the associated CLKOUT0 clock output in percentage (i.e., 0.50 will generate a 50% duty cycle).
CLKOUT0_PHASE	0	To allow specification of the output phase relationship of the associated CLKOUT0 clock output in number of degrees offset. For example 90 indicates 90^o. When setting CLK_FEEDBACK = CLKOUT0, phase shifting results in a negative phase shift of all remaining clock outputs.
CLKOUT1_DIVIDE	100	To specify the amount to divide the associated CLKOUT1 clock output by if a different frequency is desired.
CLKOUT1_DUTY_CYCLE	0.5	Specifies the Duty Cycle of the associated CLKOUT1 clock output in percentage (i.e., 0.50 will generate a 50% duty cycle).
CLKOUT1_PHASE	0	To allow specification of the output phase relationship of the associated CLKOUT0 clock output in number of degrees offset. For example 90 indicates 90^o. When setting CLK_FEEDBACK = CLKOUT0, phase shifting results in a negative phase shift of all remaining clock outputs.

Table 3.5: Attributes of PLL module.

The BANDWIDTH affects the jitter filtering of the PLL module. Greater jitter filtering can be achieved by setting the PLL attribute BANDWIDTH to low. However, an increase can incur in the static offset of the PLL. Therefore, Xilinx recommends setting the attribute to optimise.

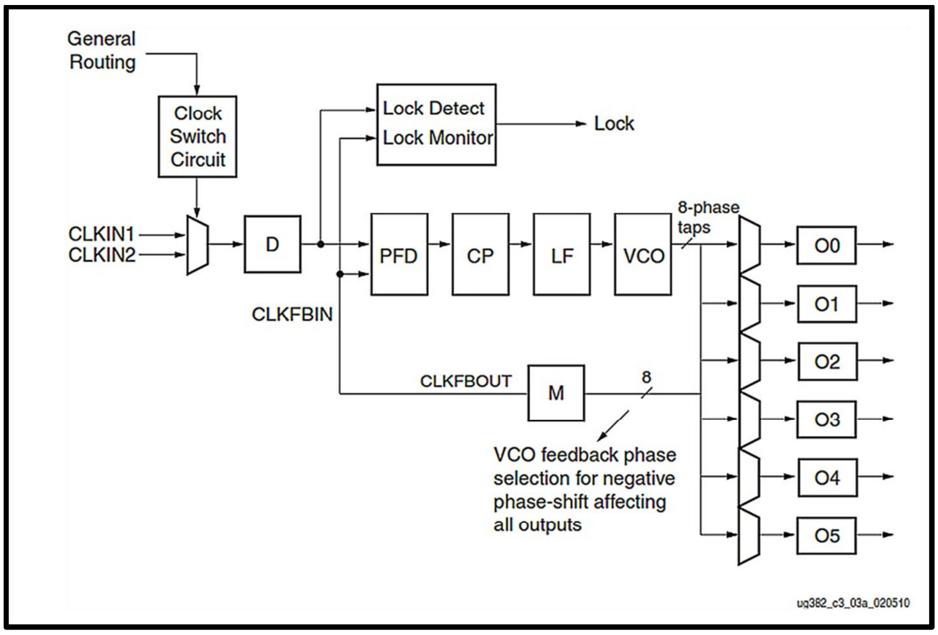


Figure 3.17: PLL block diagram: CLK_FEEDBACK = CLKFBOUT.

In the PLL structure, the differential input is converted to a single-ended clock via the Single-ended Input Global Clock Buffer (IBUFG). Each PLL has a programmable counter D that is labelled as DIVCLK_DIVIDE in Table 3.5. The PFD generates a signal proportional to the phase and frequency between the input clock and the feedback clock. The signal generated by the PFD provides an up or down signal to the CP and LF. The signal determines whether the VCO should operate at a higher or a lower frequency. If the VCO operates at high frequency, the PFD activates a down signal. This causes the control voltage to be reduced. As a result, it decreases the VCO operating frequency. If the VCO operates at extremely low frequency, the PFD activates a high signal to increase the voltage. The VCO in the PLL produces eight output phases. Each of the output phases can be used as a reference clock to the output counters. Counter M that is labelled as CLKFBOUT_MULT in Table 3.5 can be programmed to multiply the feedback clock to a different frequency. In the design, O0 (CLKOUT0) and O1 (CLKOUT1) are used to regenerate 250 MHz and 10 MHz output clocks. The 250 MHz clock is the main clock to operate the DDS module while the 10 MHz clock is for serial programming to clock and DAC chips.

In Figure 3.17, all the O [0:5] outputs can be independently programmed. As examples, O0 and O1 can be programmed to do divide-by-four (CLKOUT0_DIVIDE) and divide-by-hundred (CLKOUT1_DIVIDE). The VCO operating frequency is the same for all the outputs because only one VCO component in the PLL drives all the counters. The VCO operating frequency can be computed by the following equations:

In Table 3.5, CLK_FEEDBACK=CLKFBOUT is expressed by equation (3.9):

$$\boldsymbol{f_{VCO} = f_{CLKIN} x \frac{M}{D}} \tag{3.9}$$

Equation (3.10) shows the expression of output frequency for CLKFBOUT.

$$\boldsymbol{f_{OUT\_CLKFBOUT} = \frac{f_{VCO}}{M}} \tag{3.10}$$

Equation 3.11 shows the expression of output frequency O [0:5]

$$\boldsymbol{f_{O\,[0:5]} = \frac{f_{VCO}}{\mathrm{O\ [0:5]}}} \tag{3.11}$$

In an 8-core DDS system, the f_{VCO} and $f_{OUT\_CLKFBOUT}$ can be calculated by using equation (3.9) and (3.10):

$$\boldsymbol{f_{VCO} = 100\ MHz\ x \frac{10}{1} = 1000\ \mathrm{MHz}}$$

$$\boldsymbol{f_{OUT\_CLKFBOUT} = \frac{1000\ MHz}{10} = 100\ MHz}$$

Therefore, the CLKOUT0 and CLKOUT1 can be calculated by using equation (3.11):

$$\boldsymbol{CLKOUT0, f_{O0} = \frac{1000\ MHz}{4} = 250\ MHz}$$

$$\boldsymbol{CLKOUT1, f_{O1} = \frac{1000\ MHz}{100} = 10\ MHz}$$

3.5.2 OSERDES Module

A Spartan-6 comes with 4-bit input for each ISERDES and OSERDES. However, only OSERDES is used in this 8-core DDS module to transfer data to DAC in 500 MSPS. Figure 3.18 shows the structure of the OSERDES block in the DDS transmitter [36].

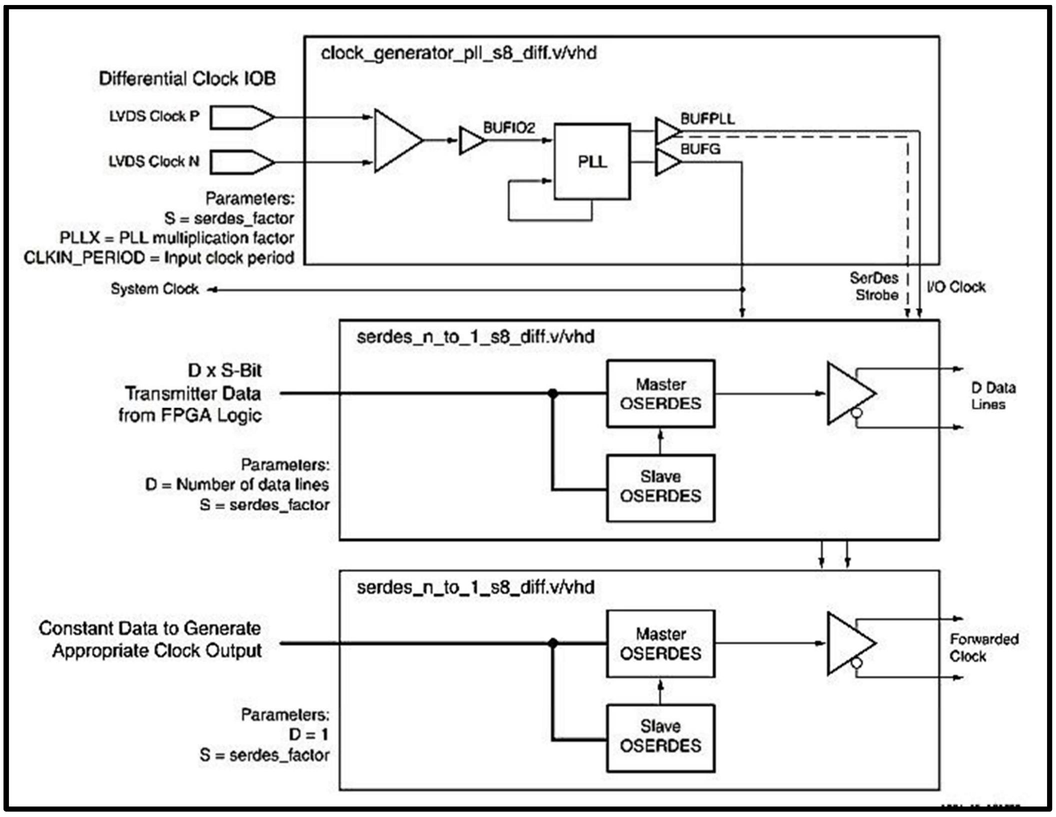


Figure 3.18: Data transmission with one SDR forwarded clock [36].

Three essential elements of OSERDES are required to implement high-speed data transmission in the 8-core DDS module. These elements are a clock generator, a high speed OSERDES 56 to 14 output lines and a forwarded clock. A BUFIO2, PLL, and High Speed I/O Clock Buffer (BUFPLL) are used to capture a differential 500 MHz clock from a DAC chip. This clock will be used to regenerate 125 MHz and 500 MHz clocks. The regenerated 500MHz clock is used as a forwarded clock for driving data from Spartan-6 to DAC in DDR mode. On the other hand, the 125 MHz clock is used to clock the serialisation of 56-to-14 bit data before transmitting out from the FPGA device [36]. Since a DAC chip has dual ports (each port operates at 1 GSPS, so dual ports have 2GSPS), an OSERDES module contain two OSERDES blocks.

The 500 MHz clock from DAC steps down to a 125 MHz internal clock by serdes factor = 4. Every time the 125 MHz clock is toggled, a 14-bit data point will be shifted to DAC. Therefore, 8-sets of data points will be transmitted on every rising edge and falling edge of the 500 MHz forwarded clock.

8-sets of 14-bit data X 125 MHz = 1 GSPS (for one OSERDES block)

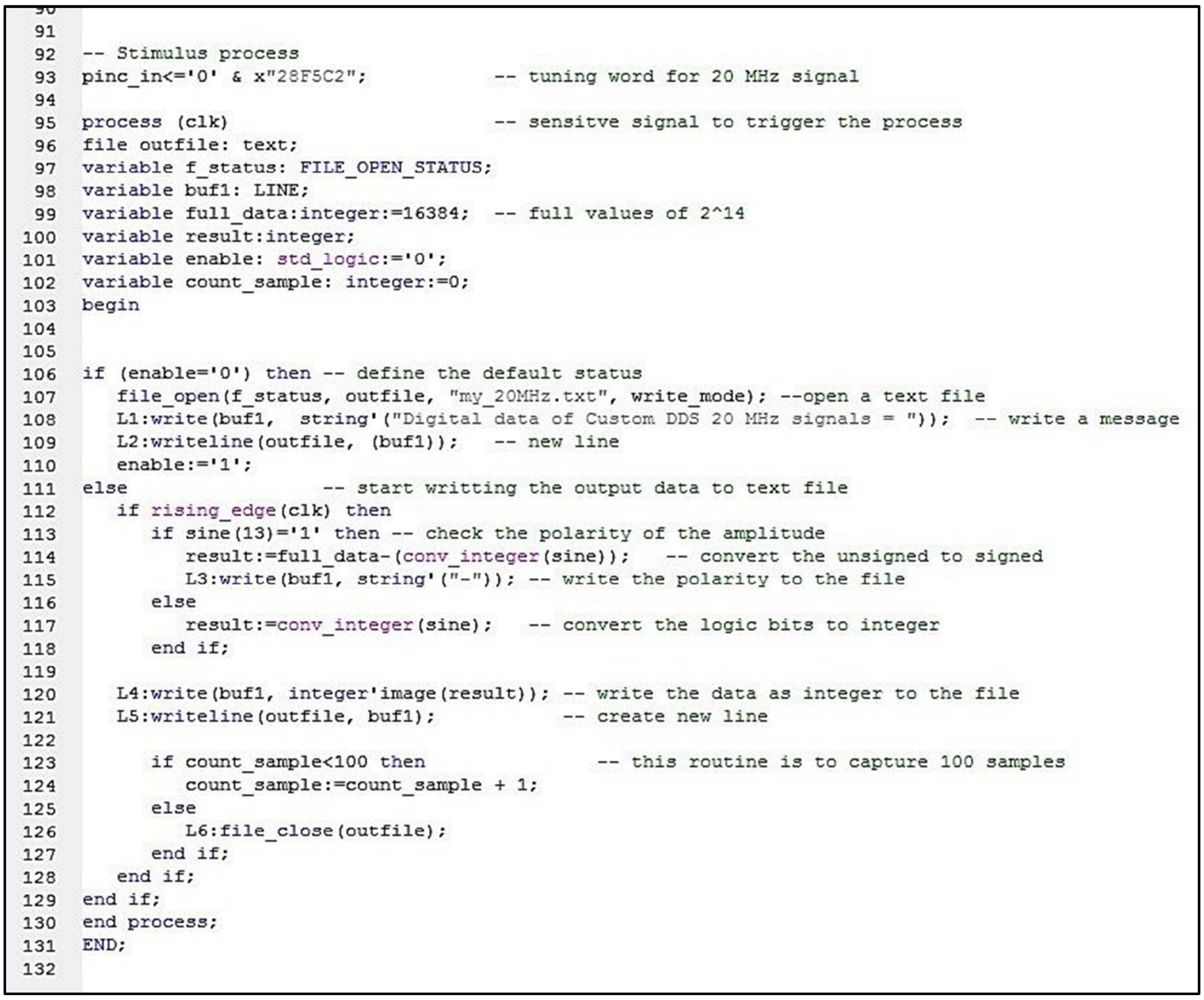
= 500 MHz in DDR

Thus, two OSERDES blocks are needed to achieve 2 GSPS.

A 500 MHz forwarded clock can be generated by driving a 4-bit repeated "0101" pattern bit to the DAC chip in the same manner as the data being transmitted by the OSERDES [36]. Effectively, the forwarded clock synchronises with the transmissions of data to DAC.

3.5.3 Development of a Test-bench File

The simulation cannot plot a sine-wave graph because ModelSim does not support the Xilinx VHDL compiler. The same problem happens to the default ISim simulator, which is unable to generate a graph by using the simulated data obtained. This obstacle can be solved by developing a test-bench file to capture all the results and record them into a text file. The text file would then be transferred to Microsoft Excel manually.



```vhdl

-- Stimulus process
pinc_in<='0' & x"28F5C2";           -- tuning word for 20 MHz signal

process (clk)                       -- sensitve signal to trigger the process
file outfile: text;
variable f_status: FILE_OPEN_STATUS;
variable buf1: LINE;
variable full_data:integer:=16384;  -- full values of 2^14
variable result:integer;
variable enable: std_logic:='0';
variable count_sample: integer:=0;
begin


if (enable='0') then -- define the default status
   file_open(f_status, outfile, "my_20MHz.txt", write_mode); --open a text file
   L1:write(buf1,  string'("Digital data of Custom DDS 20 MHz signals = "));  -- write a message
   L2:writeline(outfile, (buf1));   -- new line
   enable:='1';
else                 -- start writting the output data to text file
   if rising_edge(clk) then
      if sine(13)='1' then -- check the polarity of the amplitude
         result:=full_data-(conv_integer(sine));   -- convert the unsigned to signed
         L3:write(buf1, string'("-")); -- write the polarity to the file
      else
         result:=conv_integer(sine);   -- convert the logic bits to integer
      end if;

   L4:write(buf1, integer'image(result)); -- write the data as integer to the file
   L5:writeline(outfile, buf1);           -- create new line

      if count_sample<100 then               -- this routine is to capture 100 samples
         count_sample:=count_sample + 1;
      else
         L6:file_close(outfile);
      end if;
   end if;
end if;
end process;
END;

```

Figure 3.19: Test-bench code to capture output data.

To implement the test-bench as shown in Figure 3.19, a library must be declared in the test-bench file [30].

USE STD. textio. ALL;

The first operation in the test-bench is to create a text file. The next process is to identify the polarity of 2's complement data. If the captured data are negative, a negative '-'

sign will be added to the integer. This process will not be triggered if the data captured by the program are positive. After this, the integers will be recorded into a text file. The entire process is repeated to capture as much data as is desired. This test-bench file is not synthesisable, and it is developed only to evaluate the performance on the VHDL DDS system. That is, the test-bench itself will not be executed by any FPGA devices [30].

3.5.4 RTL-Level of a Single DDS Core

The specifications of a single DDS core:

System Clock, f_{system}	: 250 MHz
Frequency Resolution, Δf	: 7.45 Hz
Input Width of LUT	: 25-bit
Width of Truncated Output	: 14-bit
Depth of sine LUT	: 16384 samples
Input width of the Phase Accumulator	: 25-bit
Depth of Phase Accumulator	: 33554432

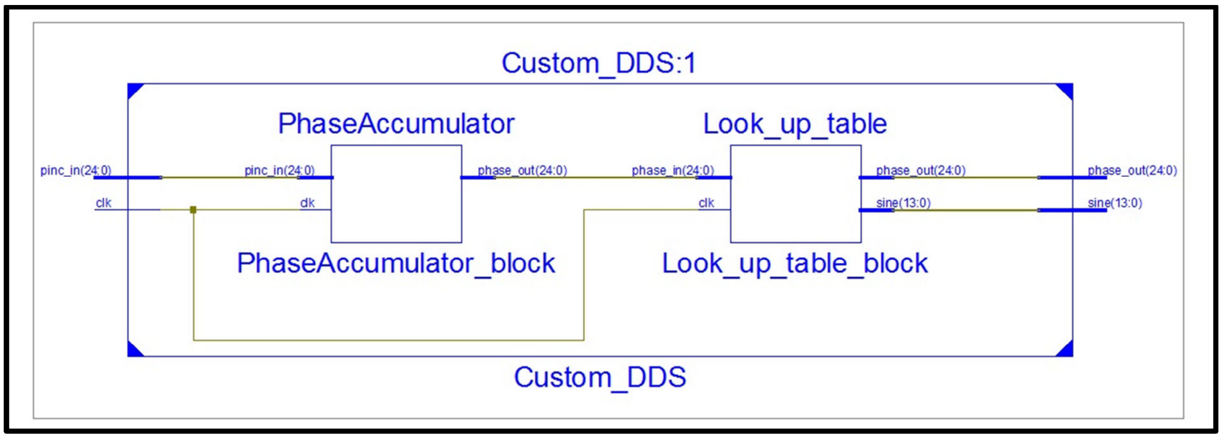
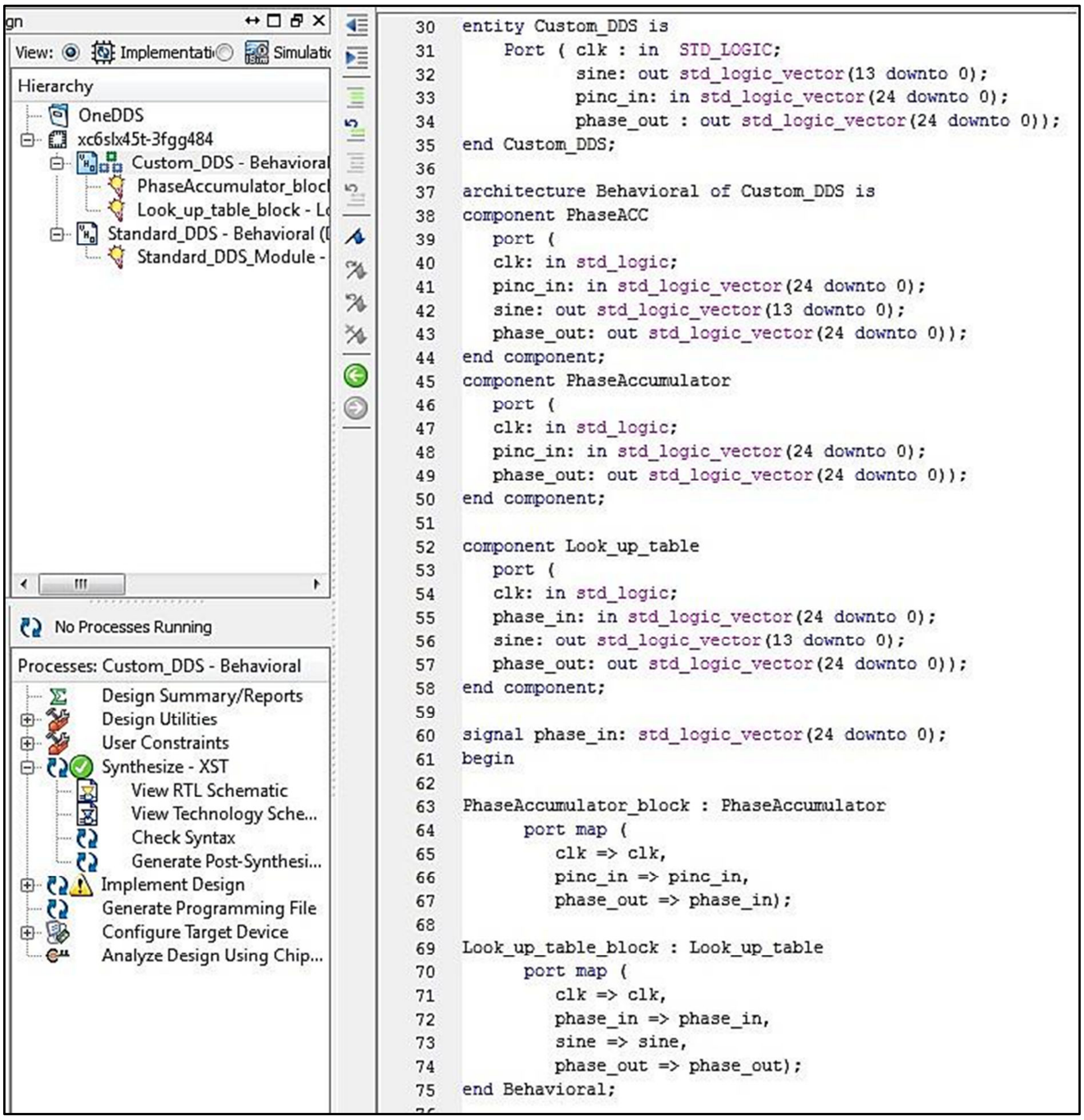


Figure 3.20: RTL schematic of a custom DDS module.

The model in Figure 3.20 is implemented using VHDL and simulated in ISim. In this model, two different modules were created by using the IP Core generator and then put together so that it can be modified according to the design requirements. The custom DDS uses a 25-bit phase accumulator and a 25-to-14 bit LUT block. In a Xilinx system, truncation is handled by the LUT module before the phase-to-amplitude conversion is implemented.



```vhdl
entity Custom_DDS is
    Port ( clk : in  STD_LOGIC;
           sine: out std_logic_vector(13 downto 0);
           pinc_in: in std_logic_vector(24 downto 0);
           phase_out : out std_logic_vector(24 downto 0));
end Custom_DDS;

architecture Behavioral of Custom_DDS is
component PhaseACC
   port (
   clk: in std_logic;
   pinc_in: in std_logic_vector(24 downto 0);
   sine: out std_logic_vector(13 downto 0);
   phase_out: out std_logic_vector(24 downto 0));
end component;
component PhaseAccumulator
   port (
   clk: in std_logic;
   pinc_in: in std_logic_vector(24 downto 0);
   phase_out: out std_logic_vector(24 downto 0));
end component;

component Look_up_table
   port (
   clk: in std_logic;
   phase_in: in std_logic_vector(24 downto 0);
   sine: out std_logic_vector(13 downto 0);
   phase_out: out std_logic_vector(24 downto 0));
end component;

signal phase_in: std_logic_vector(24 downto 0);
begin

PhaseAccumulator_block : PhaseAccumulator
      port map (
         clk => clk,
         pinc_in => pinc_in,
         phase_out => phase_in);

Look_up_table_block : Look_up_table
      port map (
         clk => clk,
         phase_in => phase_in,
         sine => sine,
         phase_out => phase_out);
end Behavioral;
```

Figure 3.21: VHDL code for a standard DDS module.

Figure 3.21 shows the implementation of a custom DDS module using the Xilinx VHDL editor. This module has a clock, a 25-bit frequency tuning word and a 14-bit LUT. The generation of the sample phases and the phase-to-amplitude conversion is clocked by the rising edge of a 250 MHz clock labelled as clk. These codes are synthesisable and can be compiled and generated as a programming file. This programming file is used to execute the 8-core DDS module on Spartan-6.

3.5.4.1 Simulation Results of a Single DDS Core Module

20 MHz is randomly chosen for the simulation. Based on the 20 MHz signal frequency, the frequency tuning word and the step size can be obtained by using equations (2.0) and (3.5):

$$Tuning\ Word = \frac{20\ M\ x\ 2^{25}}{250\ M} = 3355443$$

$$Step\ size \quad = \frac{3355443}{1} \quad = 3355443$$

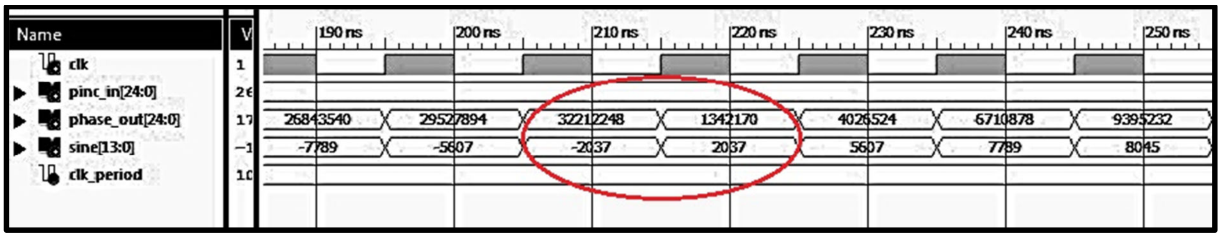


Figure 3.22: Simulation results of a DDS Core.

Figure 3.22 shows the simulation results generated by ISim. The waveform highlighted in red indicates the session where the overflow of the 25-bit phase accumulator has occurred. In the DDS module, the outputs from a phase accumulator have been converted to amplitudes by a LUT. The conversion is triggered on the same rising edge as the phase accumulator because both share the same clock.

No. Data	**Digital Amplitude**	**No. Data**	**Digital Amplitude**
1	0	**16**	7789
2	3945	**17**	8045
3	6915	**18**	6311
4	8174	**19**	3015
5	7411	**20**	-1026
6	4814	**21**	-4814
7	1026	**22**	-7410
8	-3015	**23**	-8174
9	-6311	**24**	-6915
10	-8045	**25**	-3946
11	-7789	**26**	0
12	-5607	**27**	3945
13	-2037	**28**	6915
14	2037	**29**	8174
15	5607	**30**	7411

Table 3.6: First 30 samples of 20 MHz sine-wave.

Table 3.6 lists the first 30 data points generated by the model in Figure 3.0. Samples highlighted in blue indicate that overflow have occurred in the phase accumulator. The accumulator overflows at 26 unit time. Hence, samples from 1st to 25th unit time formed a sine-wave cycle. The data were plotted by using Microsoft Excel and the result is shown in Figure 3.23. The samples are in a repeated pattern after the 25th clock pulse, which forms the subsequent sine-wave. Based on the first 30 clock cycles, the system generates 2.25 cycles of sine-wave.

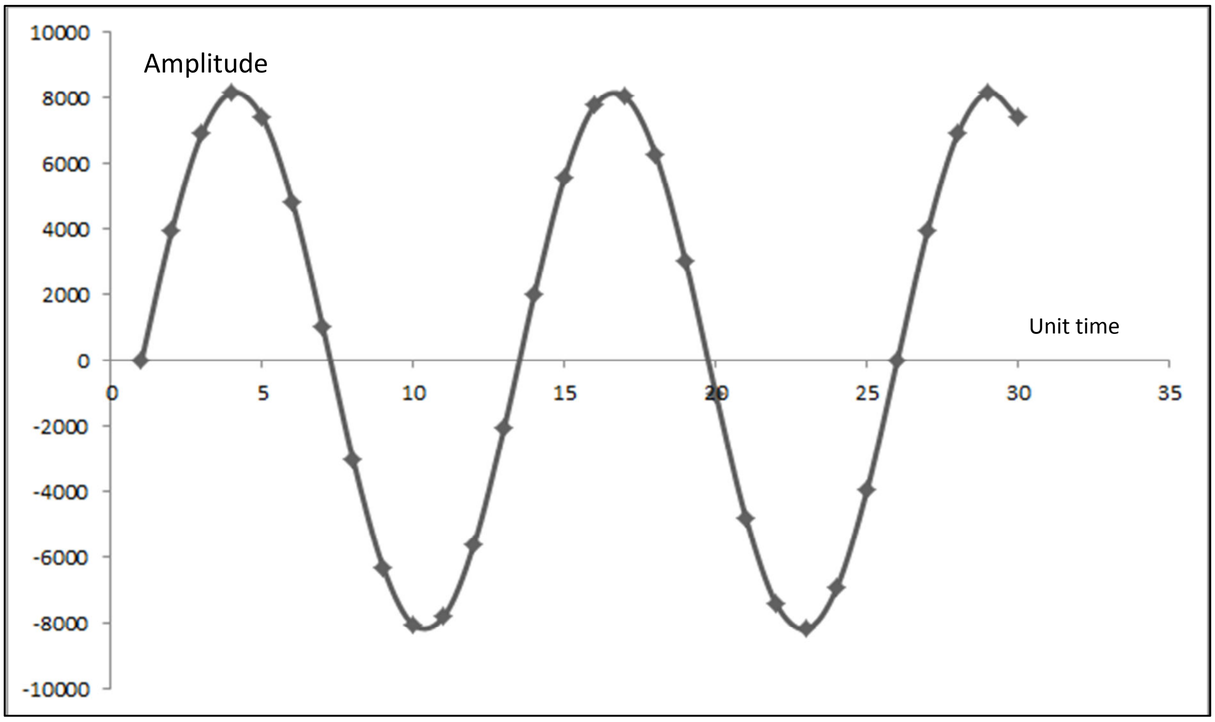


Figure 3.23: Sine-wave with frequency = 20 MHz.

Although a complete sine cycle has approximately 13 samples, it involves rounded-off errors. As is illustrated in Figure 3.23, the first two sine cycles require 25 sample amplitudes. Then a sine-wave cycle has 25 samples/ 2= 12.5 samples. The sum of the total samples counted should not be a fractional number. Therefore, it should be rounded to 13 samples.

Considering the equation (3.7):

For 12.5 samples,

$$\frac{250\ MHz}{12.5\ samples} = 20\ MHz$$

If 12.5 is rounded to 13 samples,

$$\frac{250\ MHz}{13\ samples} = 19.23\ MHz$$

The percentage of the round-off error is

$$\frac{(20 - 19.23)MHz}{20\ MHz}\ X\ 100\% = 3.85\%$$

The calculation of the total samples above is based on a unit cycle of sine-wave. When it comes to a continuous signal, the subsequent signals would blur the boundary between the first and the subsequent signals.

3.5.5 Development of an 8-core DDS *System*

The output generated by an 8-core DDS module is based on the concepts of under-sampling and over-sampling. For output up to 125 MHz, over-sampling is used to generate sample phases. The Nyquist frequency of a phase accumulator core is 125 MHz. For an output beyond 125 MHz, the signal is outside of the 1st Nyquist zone. Thus, an under-sampling technique is utilised to obtain a higher frequency signal. The under-sampling in an 8-core DDS module requires a basic knowledge of the Nyquist theorem especially regarding the Nyquist zones. An overview of the Nyquist zone is provided before going further in this thesis on the development of an 8-core DDS module.

The specifications of an RTL-level 8-core DDS system are as below:

System Clock, f_{system}	: 250 MHz
Frequency Resolution, Δf	: 0.93 Hz
Input Width of LUT	: 25-bit
Output Width of LUT	: 14-bit
Depth of sine LUT	: 16384 samples
Input width of the Phase Accumulator	: 25-bit
Depth of Phase Accumulator	: 33554432
Total Phase Offsets	: 8 sets with 25-bit of each

3.5.5.1 Overview of Nyquist Zones of an 8-core DDS System

Nyquist bandwidth is defined to be the frequency spectrum from DC to $f_{sampling}$ /2 ($f_{sampling}$ = sampling frequency). The frequency spectrum is divided into an infinite number of Nyquist zones, each having a width equal to $0.5f_{sampling}$. Theoretically, a Fast Fourier Transform (FFT) processor provides an output from DC to $f_{sampling}/2$. This frequency range is known as the 1st Nyquist zone. Any outputs from higher Nyquist zones are aliased to the 1st Nyquist zone.

An 8-core DDS system utilises the frequency width of Nyquist zones to calculate the tuning word for phase accumulation in an accumulator. In an 8-core DDS module, the phase accumulator generates accumulated phases every 250 MHz. Hence, the Nyquist frequency of the phase accumulator is 125 MHz that means the frequency band of the 1st Nyquist zone is measured from DC to 125 MHz. Let us assume that the entire 8-core DDS module has 2 GSPS in total, the Nyquist criterion allows the system to generate an output frequency at the maximum of 1 GHz. Initially, the development of an 8-core DDS module is based on the ideal signal frequency at 1 GHz. In practice, a 1 GHz of frequency could not be realised due to the impossibility to have an ideal low-pass filter. The 8-core DDS module supports a frequency up to 750 MHz during the testing of developed codes on the evaluation board. 750 MHz is the upper frequency just before the signal is distorted significantly.

There are two conditions, which might be encountered by the 8-core DDS system. The first case is to generate an output signal (fa) which is within the 1st Nyquist zone (Figure 3.25b). The second case is to generate an output signal (fa) which is outside of the 1st Nyquist zone (2nd to 6th Nyquist zones) (Figure 3.25a).

In Figure 3.24a, if the output frequency is in the 1st Nyquist zone, the over-sampling method is used to generate the signal. If the output signal is outside of the 1st Nyquist zone such as in Figure 3.24b, the image falls inside the 1st Nyquist zone. At this moment, the under-sampling technique can be used to get a higher frequency signal. This can be achieved by changing the sample phases of the aliased signal with obsolete offset values.

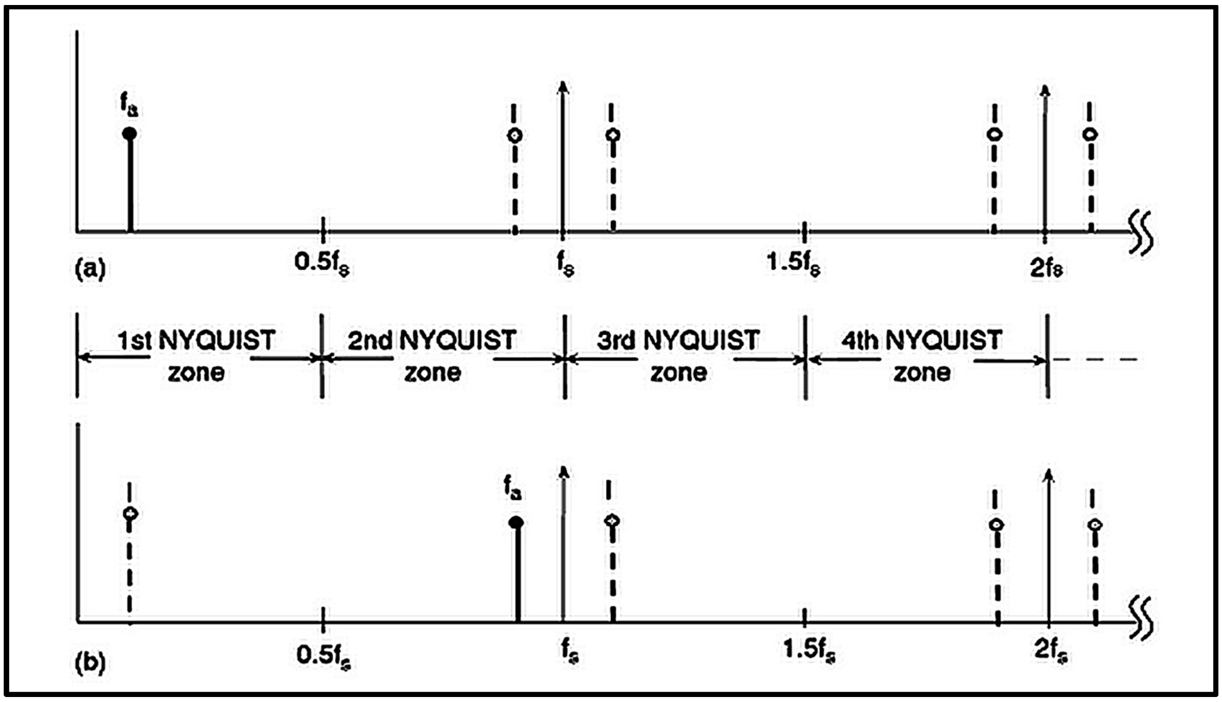


Figure 3.24: Diagram of output signal (fa) sampled at fs [51].

Nyquist Zone of an 8-core DDS System.

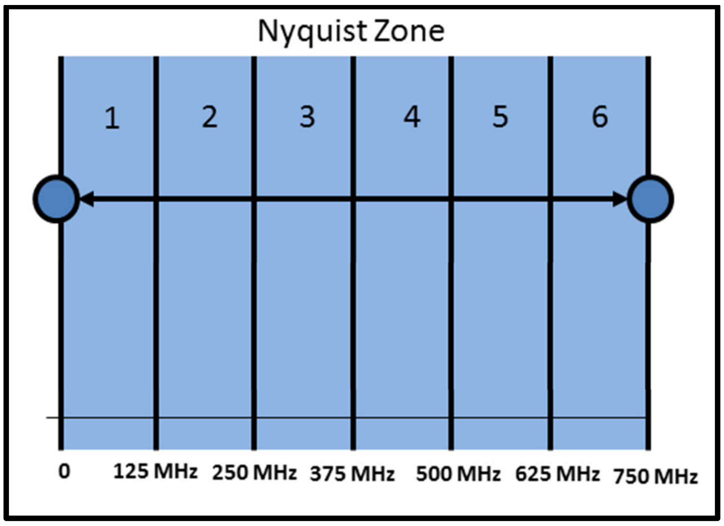


Figure 3.25: Nyquist zone.

Figure 3.25 shows the frequency bands of the 1st to the 6th Nyquist zone. Each Nyquist zone has a lower band-limit and an upper band-limit. During a signal sampling process, any signal higher than 125 MHz experiences a 'folding' effect. An aliased signal will be folded back in the 1st Nyquist zone [34].

For example, in Figure 3.26, a signal frequency of 200 MHz is equal to 125 MHz (lower band-limit) + 75 MHz (aliased frequency). 125 MHz is the lower band-limit of the 2nd Nyquist zone while 75 MHz is the aliased signal of 200 MHz signal.

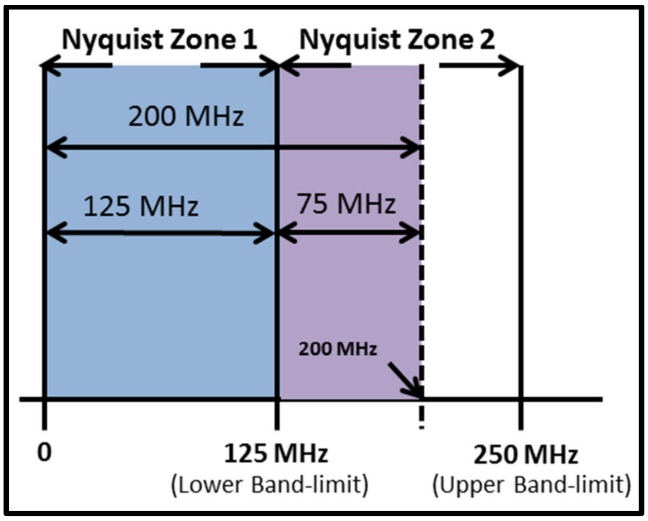


Figure 3.26: Diagram of under-sampling to obtain 200 MHz output.

In an 8-core DDS system with under-sampling, a VHDL program is developed to locate the aliased signal at first. Then, the aliased samples are shifted by eight phase offsets to reproduce eight more samples of the higher frequency signal. In this case, the phase offsets are not only used to shift an accumulated phase to eight consecutive samples, they also carry the information about lower band-limit frequency. The generation of a higher frequency signal can be generated based on the expression (3.12) below.

Signal Frequency = lower band-limit of Nyquist zone + Aliased Frequency (3.12)

Equation (3.12) is similar to mimicking the samples of the high-speed signal. A phase accumulator is used to generate accumulated phases of the aliased signal, and they are added with the tuning word, which represents the lower band-limit of the signal Nyquist zone.

Commonly, a signal outside of the 1st Nyquist zone cannot be generated by using a single DDS core. This is because a single DDS core cannot generate enough samples to maintain the shape as a sine-wave. When N-bit phase accumulator reaches '111…111", it represents a signal of 250 MHz because the 25-bit frequency tuning word is calculated based on the 250 MHz system frequency.

For example, the turning word of a 300 MHz frequency is

Output frequency = 300 MHz

Input width of phase accumulator, N = 25-bit

System frequency = 250 MHz

By applying the values above to the equation (2.0), the tuning word of 300 MHz frequency is

$M = \frac{300MHz\ x\ 2^{25}}{250\ MHz}$ = 10011001100110011001100110b

The result shows that a 300 MHz frequency (tuning word) overflows a 25-bit phase accumulator because 300 MHz requires a 26-bit phase accumulator. If the tuning word is calculated based on an N-bit phase accumulator and 250 MHz system frequency, any signal more than 250 MHz would request (n+1) -bit of tuning word as well as (n+1) -bit phase accumulator. By using the under-sampling technique, the tuning word of aliased frequency (lesser than 125 MHz) is used as a phase increment in an accumulator.

In an 8-core DDS module with under-sampling, some of the phase offsets may carry 26-bit or more. The results after adding between phase offsets and accumulated phases are truncated or cropped into lower 25 bits before being processed by the LUT module. In LUTs, these 25 bits are truncated to 14 bits again due to the standard process in LUT blocks.

3.5.5.2 *Block Diagram of an 8-core DDS Module*

Figure 3.27 shows the block diagram of an 8-core DDS module. The model consists of an offset divider, a phase accumulator, a LUT module (with eight LUTs) and an OSERDES module.

A 25-bit frequency tuning word provides the input of the offset divider and phase accumulator blocks. The phase accumulator generates an accumulator phase on every rising edge of 250 MHz clock.

The offset divider generates eight phase offsets that are placed on OFFSET_ [0:7] A. These phase offsets combine with accumulated phases from Phase_Out via adders. 25-bit outputs from each adder connect to 25-bit LUTs via Phase_in[0:7]. After the phase-to-amplitude conversion is completed, the converted amplitudes are placed on SINE_DB0 [A: D] and SINE_DB1 [A: D].

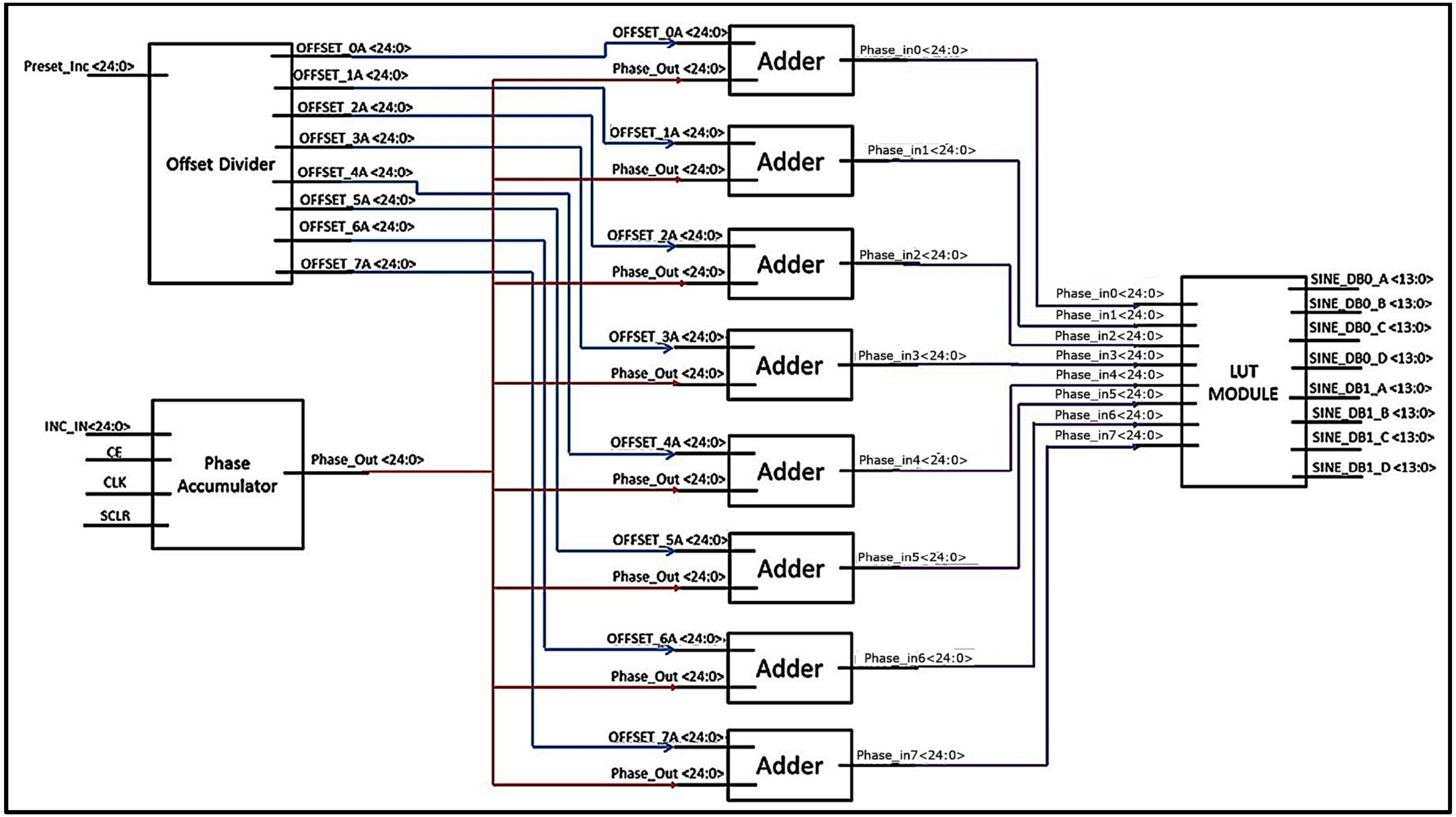


Figure 3.27: Block diagram of custom DDS with eight DDS cores.

3.5.5.3 *RTL Schematics of an 8-core DDS Module*

This section shows the main components to create an 8-core DDS module. The RTL schematics of the DDS system generated by Xilinx ISE are based on the developed VHDL codes for an 8-core DDS module. The RTL schematics give clear pictures of each component developed in the program. In Figure 3.28, the 8-core DDS system contains an offset divider module. This module divides the tuning word to support signal generation up to 1 GSPS, which will then become 2 GSPS in DAC. An offset divider module has six Symmetric Systolic Multiply-Add (MADD) processing elements [59]. These elements are DSP48 slices, which are used to achieve the addition and multiplication processes of digital data. In Figure 3.29, the outputs from a phase accumulator are combined with eight phase offsets in order to reproduce eight sample phases over a unit time. These eight sample phases will be converted to amplitude via a LUT module as shown in Figure 3.30.

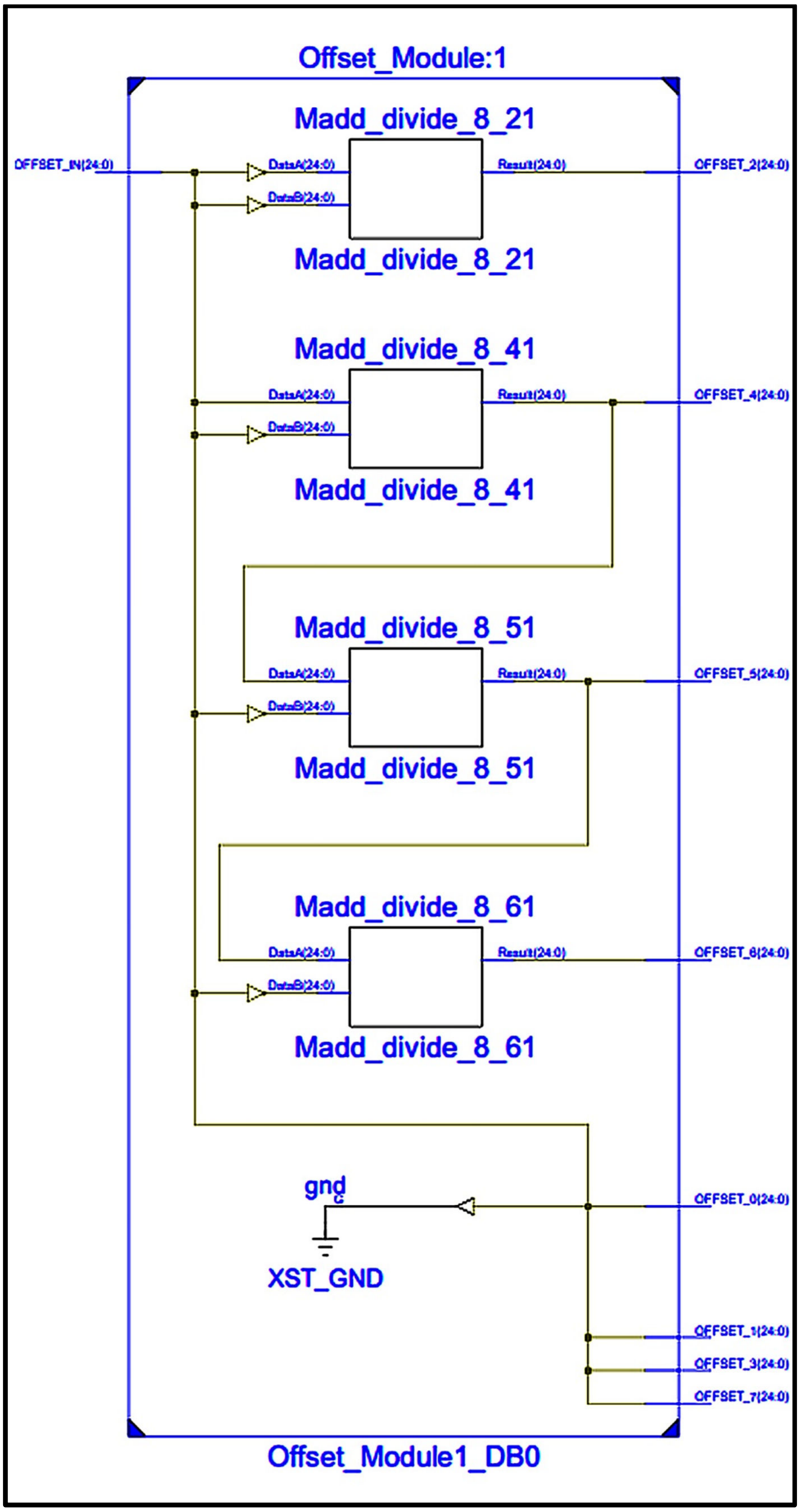


Figure 3.28: RTL schematic of the Offset Divider modules in an 8-core DDS system.

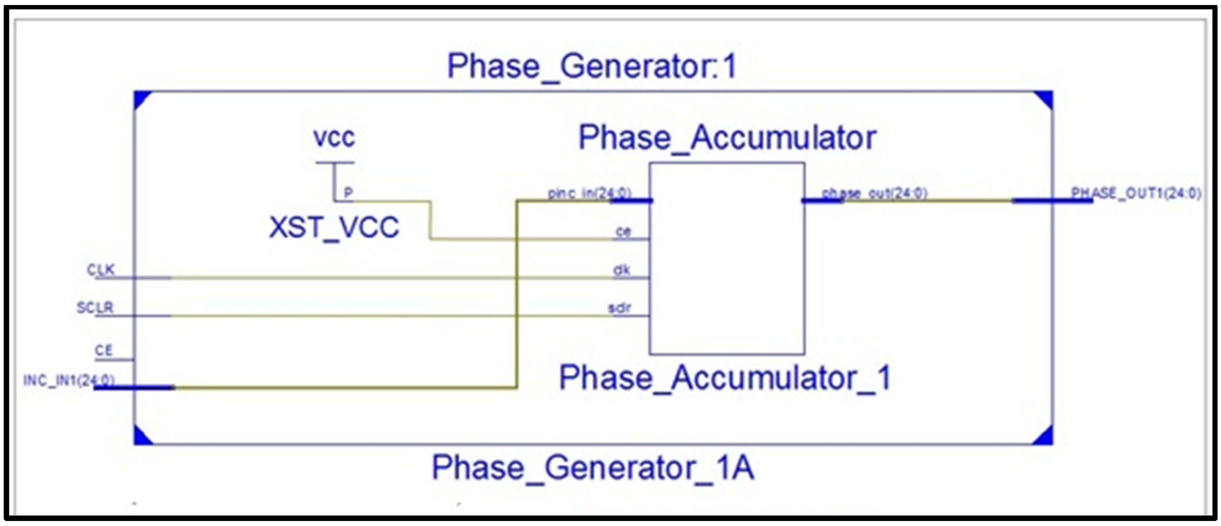


Figure 3.29: RTL schematic of a Phase Accumulators in an 8-core DDS System.

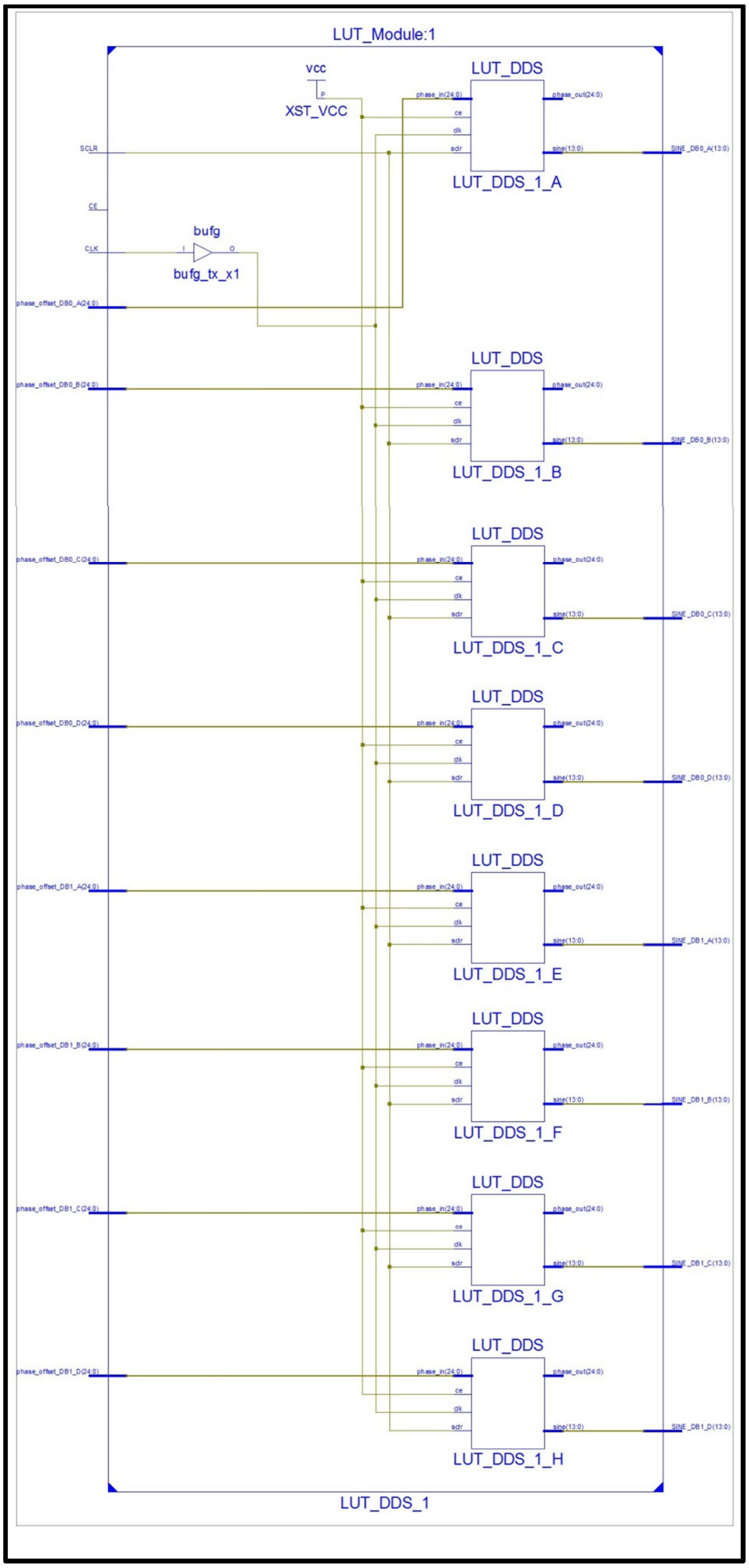
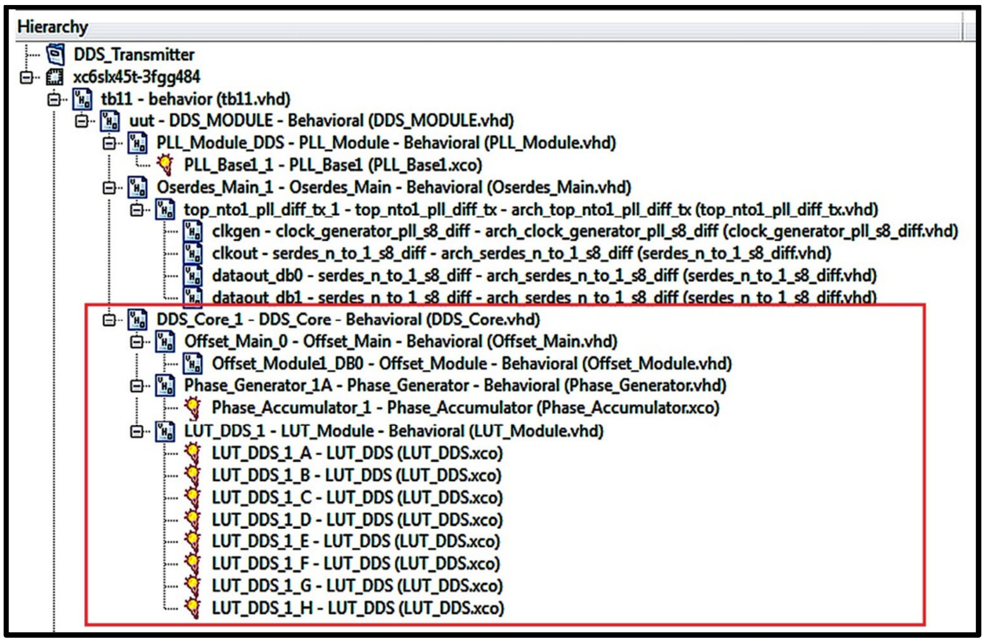


Figure 3.30: RTL Schematic of eight LUTs.

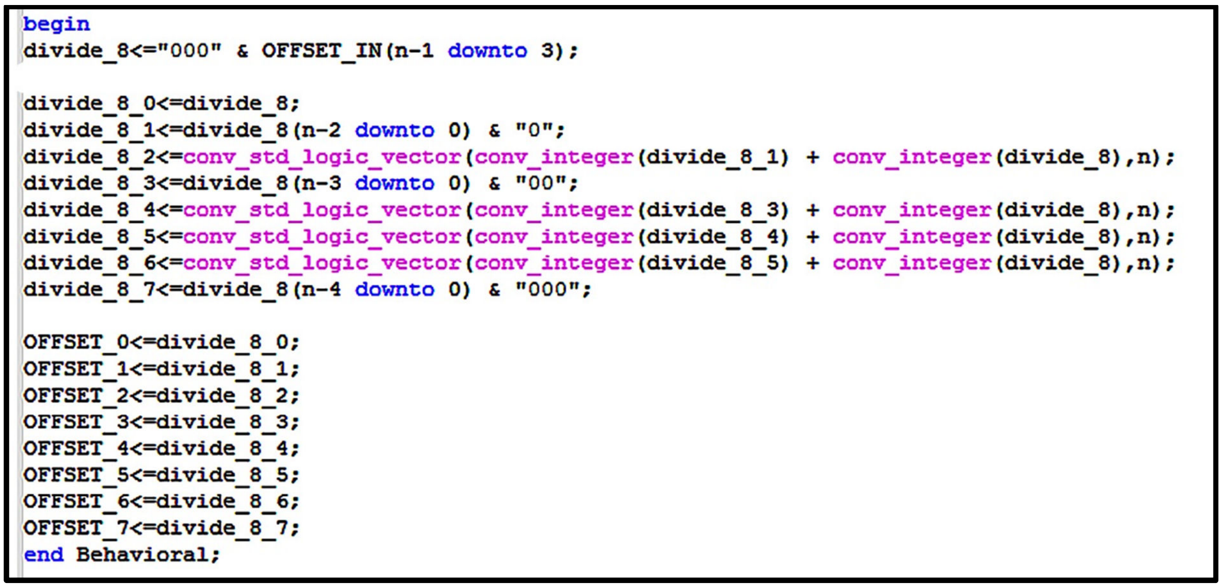
3.5.5.4 *VHDL Development of an 8-core DDS Module with Over-sampling Concept*



```
Hierarchy
DDS_Transmitter
xc6slx45t-3fgg484
  tb11 - behavior (tb11.vhd)
    uut - DDS_MODULE - Behavioral (DDS_MODULE.vhd)
      PLL_Module_DDS - PLL_Module - Behavioral (PLL_Module.vhd)
        PLL_Base1_1 - PLL_Base1 (PLL_Base1.xco)
      Oserdes_Main_1 - Oserdes_Main - Behavioral (Oserdes_Main.vhd)
        top_nto1_pll_diff_tx_1 - top_nto1_pll_diff_tx - arch_top_nto1_pll_diff_tx (top_nto1_pll_diff_tx.vhd)
          clkgen - clock_generator_pll_s8_diff - arch_clock_generator_pll_s8_diff (clock_generator_pll_s8_diff.vhd)
          clkout - serdes_n_to_1_s8_diff - arch_serdes_n_to_1_s8_diff (serdes_n_to_1_s8_diff.vhd)
          dataout_db0 - serdes_n_to_1_s8_diff - arch_serdes_n_to_1_s8_diff (serdes_n_to_1_s8_diff.vhd)
          dataout_db1 - serdes_n_to_1_s8_diff - arch_serdes_n_to_1_s8_diff (serdes_n_to_1_s8_diff.vhd)
      DDS_Core_1 - DDS_Core - Behavioral (DDS_Core.vhd)
        Offset_Main_0 - Offset_Main - Behavioral (Offset_Main.vhd)
          Offset_Module1_DB0 - Offset_Module - Behavioral (Offset_Module.vhd)
        Phase_Generator_1A - Phase_Generator - Behavioral (Phase_Generator.vhd)
          Phase_Accumulator_1 - Phase_Accumulator (Phase_Accumulator.xco)
        LUT_DDS_1 - LUT_Module - Behavioral (LUT_Module.vhd)
          LUT_DDS_1_A - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_B - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_C - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_D - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_E - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_F - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_G - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_H - LUT_DDS (LUT_DDS.xco)
```

Figure 3.31: Project hierarchy of an 8-core DDS module.

Figure 3.31 shows the hierarchy of a VHDL project belonging to an 8-core DDS module. The top module 'DDS_MODULE' is a top-level file, which is used to call all the sub-programs in the hierarchy. The files highlighted in red are the important part of the system. The 'DDS_Core.vhd' and its subsequence files contain the main programs, which are used to implement an 8-core DDS system. The other files in the hierarchy are developed to make the implementation of an evaluation board possible.



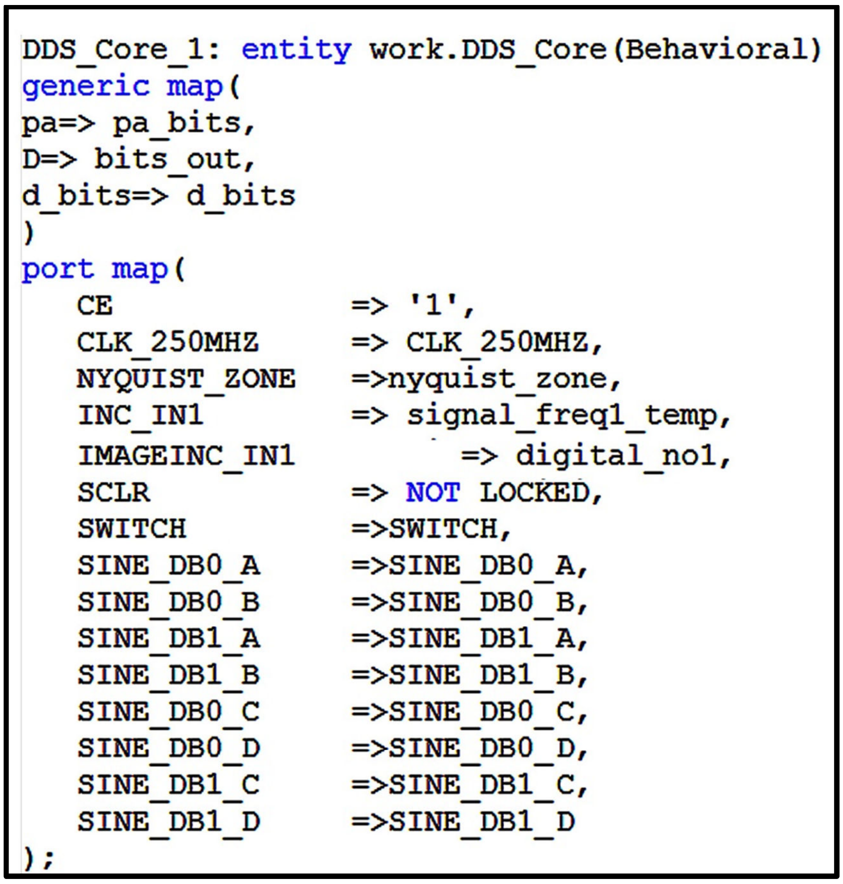
```vhdl
begin
divide_8<="000" & OFFSET_IN(n-1 downto 3);

divide_8_0<=divide_8;
divide_8_1<=divide_8(n-2 downto 0) & "0";
divide_8_2<=conv_std_logic_vector(conv_integer(divide_8_1) + conv_integer(divide_8),n);
divide_8_3<=divide_8(n-3 downto 0) & "00";
divide_8_4<=conv_std_logic_vector(conv_integer(divide_8_3) + conv_integer(divide_8),n);
divide_8_5<=conv_std_logic_vector(conv_integer(divide_8_4) + conv_integer(divide_8),n);
divide_8_6<=conv_std_logic_vector(conv_integer(divide_8_5) + conv_integer(divide_8),n);
divide_8_7<=divide_8(n-4 downto 0) & "000";

OFFSET_0<=divide_8_0;
OFFSET_1<=divide_8_1;
OFFSET_2<=divide_8_2;
OFFSET_3<=divide_8_3;
OFFSET_4<=divide_8_4;
OFFSET_5<=divide_8_5;
OFFSET_6<=divide_8_6;
OFFSET_7<=divide_8_7;
end Behavioral;
```

Figure 3.32: VHDL code for offset divider.

Figure 3.32 shows the VHDL programs of an offset divider. ‘divide_8_0’ to ‘divide_8_7’ are the temporary variables used to hold the calculation results of eight phase offsets. The results are placed on ‘OFFSET_0’ to ‘OFFSET_7’ and passed over to the next VHDL routine. The variable ‘divide_8_0’ to ‘divide_8_7' contain the respective values of 1^{st} to 8^{th} phase offsets.

In Figure 3.32, the line after ‘begin’ serves to divide the tuning word of the output signal by eight, which is similar to shifting the frequency tuning word right by three bits. The result obtained is the 1^{st} phase offset, which would be stored in ‘divide_8’. The operations of adding zeros to the 2^{nd} and 4^{th} phase offsets are similar to multiplying ‘divide_8’ by 2 and 4. This method can reduce the total adders needed as well as the propagation delays in the module. 3^{rd}, 5^{th}, $6^{th,}$ and 7^{th} phase offsets can be obtained by adding ‘divide_8’ to 2^{nd}, 4^{th}, $5^{th,}$ and 6^{th} phase offset values, respectively. The 8^{th} phase offset is always equal to the tuning word of the output signal.

The purpose of converting data type from 'std_logic_vector' to an integer then to 'std_logic_vector' is to maintain the 25-bit width of the phase offsets.



```
DDS_Core_1: entity work.DDS_Core(Behavioral)
generic map(
pa=> pa_bits,
D=> bits_out,
d_bits=> d_bits
)
port map(
   CE              => '1',
   CLK_250MHZ      => CLK_250MHZ,
   NYQUIST_ZONE    =>nyquist_zone,
   INC_IN1         => signal_freq1_temp,
   IMAGEINC_IN1          => digital_no1,
   SCLR            => NOT LOCKED,
   SWITCH          =>SWITCH,
   SINE_DB0_A      =>SINE_DB0_A,
   SINE_DB0_B      =>SINE_DB0_B,
   SINE_DB1_A      =>SINE_DB1_A,
   SINE_DB1_B      =>SINE_DB1_B,
   SINE_DB0_C      =>SINE_DB0_C,
   SINE_DB0_D      =>SINE_DB0_D,
   SINE_DB1_C      =>SINE_DB1_C,
   SINE_DB1_D      =>SINE_DB1_D
);
```

Figure 3.33: Entity declaration of DDS_Core.

In Figure 3.33, the entity declaration is used to inherit the parameters from the top module to sub-programs. The Chip Select (CE) bit is set to '1' so that the phase accumulator and LUTs are always enabled. The outputs of the accumulator and LUTs are controlled by the Synchronise Clear (SCLR) signal. The SCLR signal is retrieved from the LOCKED signal of a PLL. 'signal_freq1_temp' is a temporary register, which is used to pass the tuning word of the output frequency to phase register for calculating the phase offset. The variable 'digital_no1' is used to pass the image frequency of the output signal to the phase register for phase increment.

3.5.5.5 *Simulation Results of an 8-core DDS System with Over-sampling Concept*

Random frequency 20 MHz has been chosen as a frequency under test for simulating an 8-core DDS module. The tuning word and the step size of 20 MHz can be calculated by considering the equations (2.0) and (3.5).

$$Tuning\ Word = \frac{20\ M\ x\ 2^{25}}{250\ M} = 2684354$$

$$Step\ size \quad = \frac{2682354}{8} \quad = 3355443$$

Table 3.7 shows the amplitudes of a 20 MHz signal which are generated by an 8-core DDS module and captured by a test-bench file.

Clock cycle	Eight sequential samples generated over a unit time (Amplitude)							
	1st	2nd	3rd	4th	5th	6th	7th	8th
1	**0**	**514**	**1026**	**1535**	**2037**	**2531**	**3015**	**3487**
2	**3945**	**4388**	**4814**	**5221**	**5607**	**5971**	**6311**	**6626**
3	**6915**	**7177**	**7410**	**7614**	**7789**	**7932**	**8045**	**8126**
4	**8174**	**8190**	**8174**	**8126**	**8045**	**7932**	**7789**	**7615**
5	**7411**	**7177**	**6915**	**6626**	**6311**	**5971**	**5607**	**5221**
6	**4814**	**4388**	**3946**	**3487**	**3015**	**2531**	**2037**	**1535**
7	**1026**	**515**	**0**	**-514**	**-1026**	**-1535**	**-2037**	**-2531**
8	**-3015**	**-3487**	**-3945**	**-4388**	**-4814**	**-5221**	**-5607**	**-5971**
9	**-6311**	**-6626**	**-6915**	**-7177**	**-7410**	**-7614**	**-7789**	**-7932**
10	**-8045**	**-8126**	**-8174**	**-8190**	**-8174**	**-8126**	**-8045**	**-7932**
11	**-7789**	**-7615**	**-7411**	**-7177**	**-6915**	**-6626**	**-6311**	**-5971**
12	**-5607**	**-5221**	**-4814**	**-4388**	**-3946**	**-3487**	**-3015**	**-2531**
13	**-2037**	**-1535**	**-1026**	**-515**	**0**	**514**	**1026**	**1534**
14	**2037**	**2531**	**3015**	**3487**	**3945**	**4388**	**4814**	**5221**
15	**5607**	**5971**	**6311**	**6626**	**6915**	**7177**	**7410**	**7614**
16	**7789**	**7932**	**8045**	**8126**	**8174**	**8190**	**8174**	**8126**
17	**8045**	**7932**	**7789**	**7615**	**7411**	**7177**	**6915**	**6626**
18	**6311**	**5971**	**5607**	**5221**	**4814**	**4388**	**3946**	**3487**
19	**3015**	**2531**	**2037**	**1535**	**1026**	**515**	**0**	**-514**
20	**-1026**	**-1534**	**-2037**	**-2531**	**-3015**	**-3487**	**-3945**	**-4388**
21	**-4814**	**-5221**	**-5607**	**-5971**	**-6311**	**-6626**	**-6915**	**-7177**
22	**-7410**	**-7614**	**-7789**	**-7932**	**-8045**	**-8126**	**-8174**	**-8190**
23	**-8174**	**-8126**	**-8045**	**-7932**	**-7789**	**-7615**	**-7411**	**-7177**
24	**-6915**	**-6626**	**-6311**	**-5971**	**-5607**	**-5221**	**-4814**	**-4388**
25	**-3946**	**-3487**	**-3015**	**-2531**	**-2037**	**-1535**	**-1027**	**-515**
26	**0**	**514**	**1026**	**1534**	**2037**	**2531**	**3015**	**3487**

Table 3.7: Results of 8-core DDS system.

Eight sequential sample phases that are generated simultaneously over a unit clock cycle can be collected and used to plot eight separated sine-wave graphs as shown in Figure 3.34 to Figure 3.41. In Table 3.7, the amplitudes indicated by column '1st' are the 1st sample phase collected during the first 26 clock cycles. The 1st sample phases in Table 3.7 are similar to the amplitudes generated by a single DDS core in Table 3.6.

By using equation (2.0), (3.5) and (3.6), the frequency tuning word of a 20 MHz is 2684354, which is located in the 1st Nyquist zone. The step size of a 20 MHz signal is 335544 and so the phase offsets are:

- Phase Offset for 1st sample phase = step size = 335544
- Phase Offset for 2nd sample phase = 2 x 335544 = 671088
- Phase Offset for 3rd sample phase = 3 x 335544 = 1006632
- Phase Offset for 4th sample phase = 4 x 335544 = 1342176
- Phase Offset for 5th sample phase = 5 x 335544 = 1677720
- Phase Offset for 6th sample phase = 6 x 335544 = 2013264
- Phase Offset for 7th sample phase = 7 x 335544 = 348808
- Phase Offset for 8th sample phase = 8 x 335544 = 2684352

The phases from the accumulator are added to each of the phase offsets as listed above. On the rising edge of 250 MHz clock, an accumulated phase is generated. The step size, 335544 is added to the accumulated phase to produce the 1st sample phase. The second phase offset, 671088, would be added to the accumulated phase to produce the 2nd sample phase. The 3rd to 8th sample phases are produced in the same way as calculating the 1st and 2nd sample phases. It could be done by adding its respective phase offset value to the accumulated phase. The addition between an accumulated phase with eight phase offsets creates eight sample phases simultaneously.

The data from each column in Table 3.7 are used to create eight sine-wave graphs as shown in Figure 3.34 to Figure 3.41.

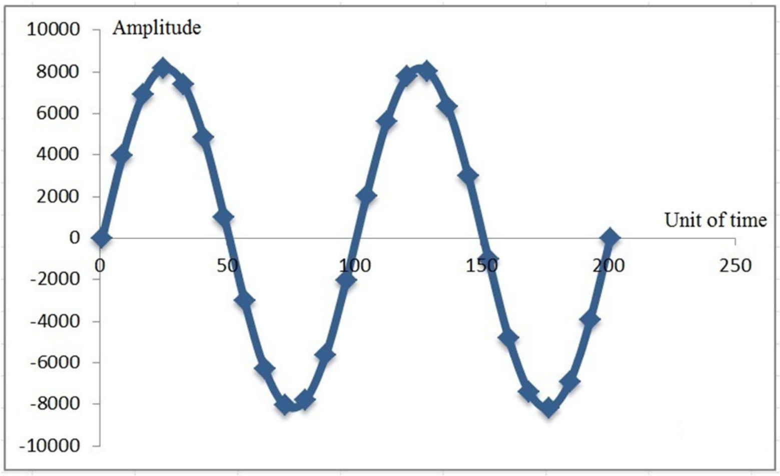


Figure 3.34: Plot of 1st sample phase.

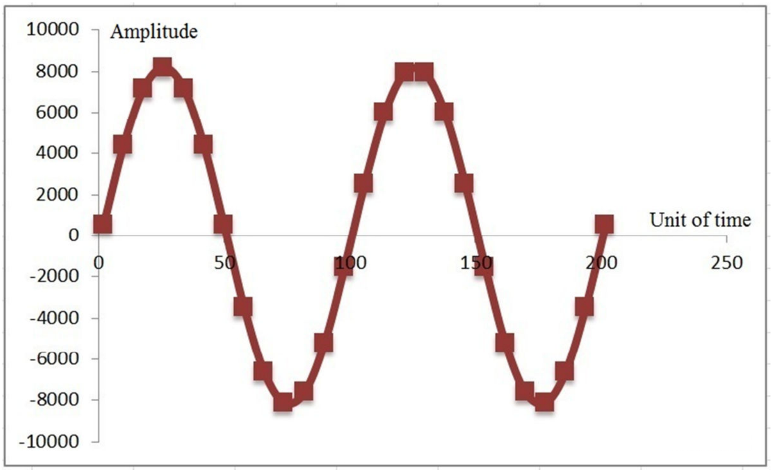


Figure 3.35: Plot of 2nd sample phase.

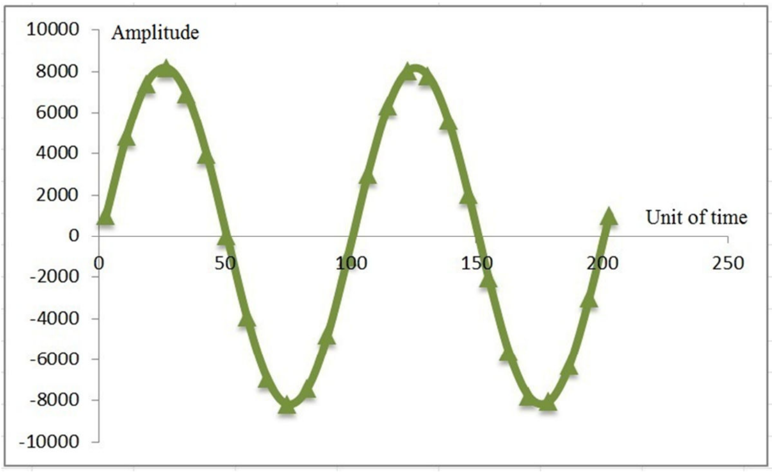


Figure 3.36: Plot of 3rd sample phase.

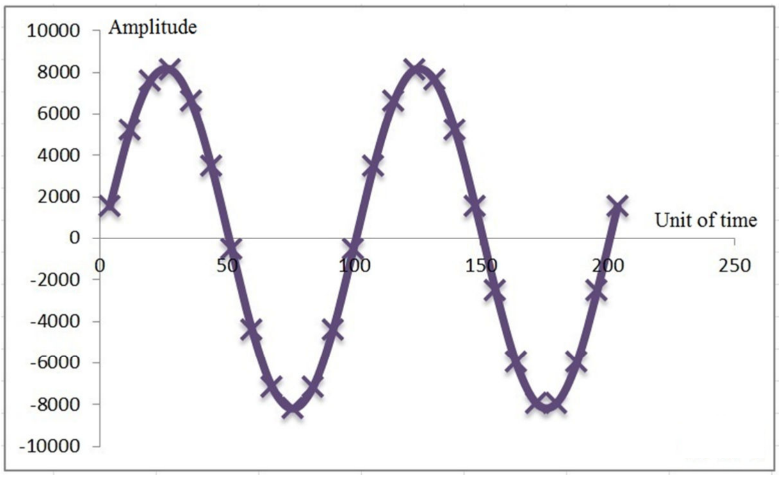


Figure 3.37: Plot of 4th sample phase.

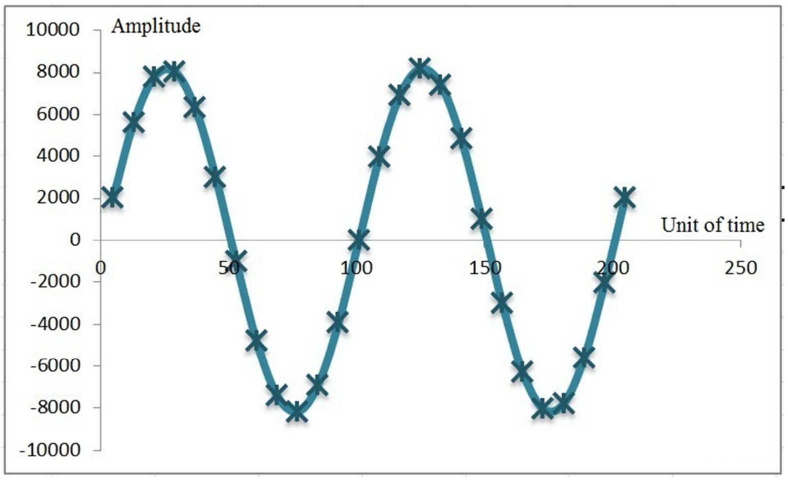


Figure 3.38: Plot of 5th sample phase.

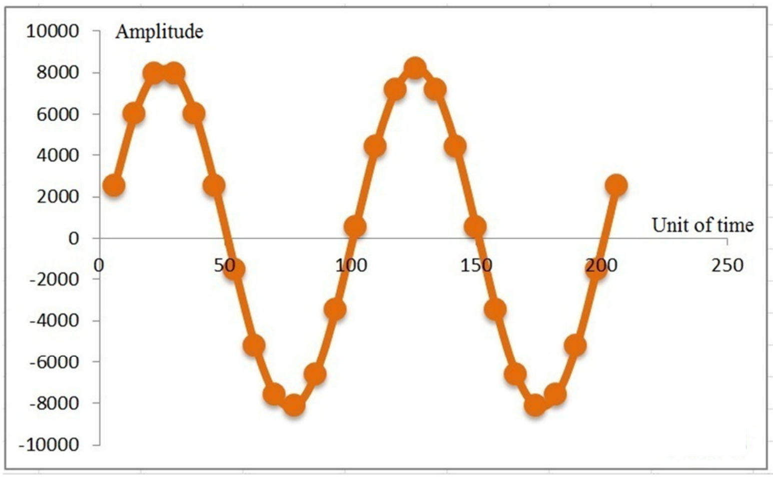


Figure 3.39: Plot of 6th sample phase.

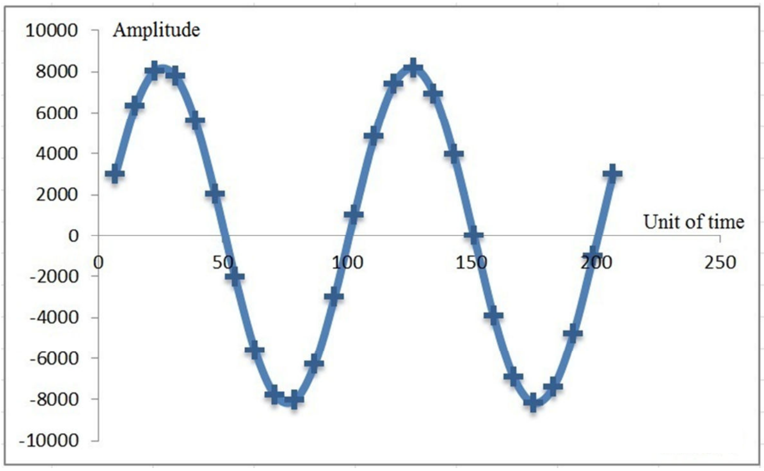


Figure 3.40: Plot of 7th sample phase.

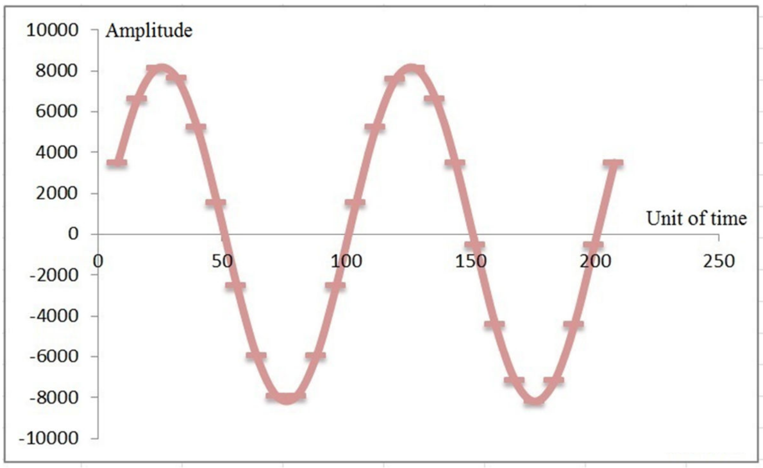


Figure 3.41: Plot of 8th sample phase.

Figure 3.34 to Figure 3.41 illustrate the samples generated by an 8-core DDS module containing the amplitude of eight sine-waves. These eight sine-waves have unity step size of 268435. By overlaying all the figures above, Figure 3.42 is formed which has a step size reduced from 268435 to 3355443.

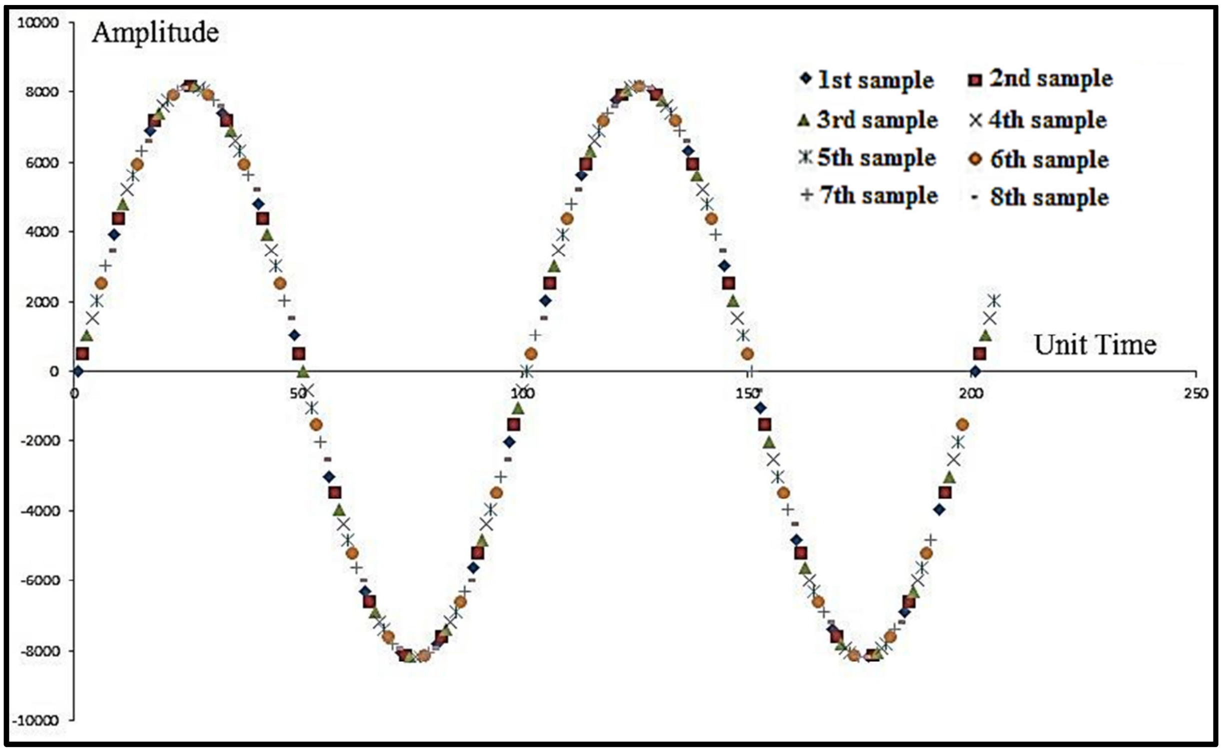


Figure 3.42: *Final output after overlaying Figure 3.34 to Figure 3.41.*

The overlaid signal in Figure 3.42 above has the same frequency as the eight sine-waves because these sine-waves have the same phase increment value. The signal frequency can be traced by multiplying the tuning word of 20 MHz with the frequency resolution of the system.

2684354 X 7.45 Hz = ~20 MHz

There is an alternative calculation to trace the output frequency by using equation (3.8). A unit cycle of sine in Figure 3.42 requires 100 samples. Therefore,

Output Frequency $= \frac{8 \; x \; 250 \; MHz}{100} = 20 \; MHz$

The overlaid output is more precise than the individual sine-waves because it has achieved 8 times more samples than the signals formed by the amplitudes from each column in Table 3.7.

Figure 3.44 below is a sine-waveform, which is formed by connecting a virtual line across each of the sample points in Figure 3.42.

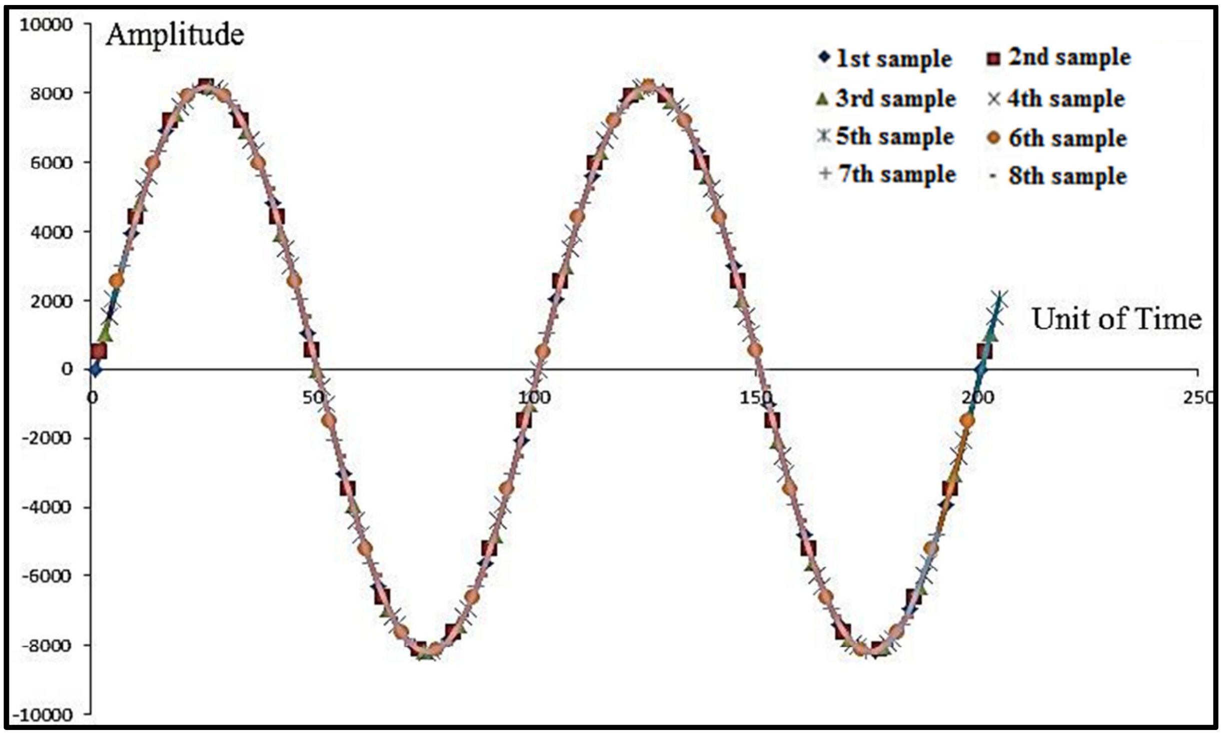


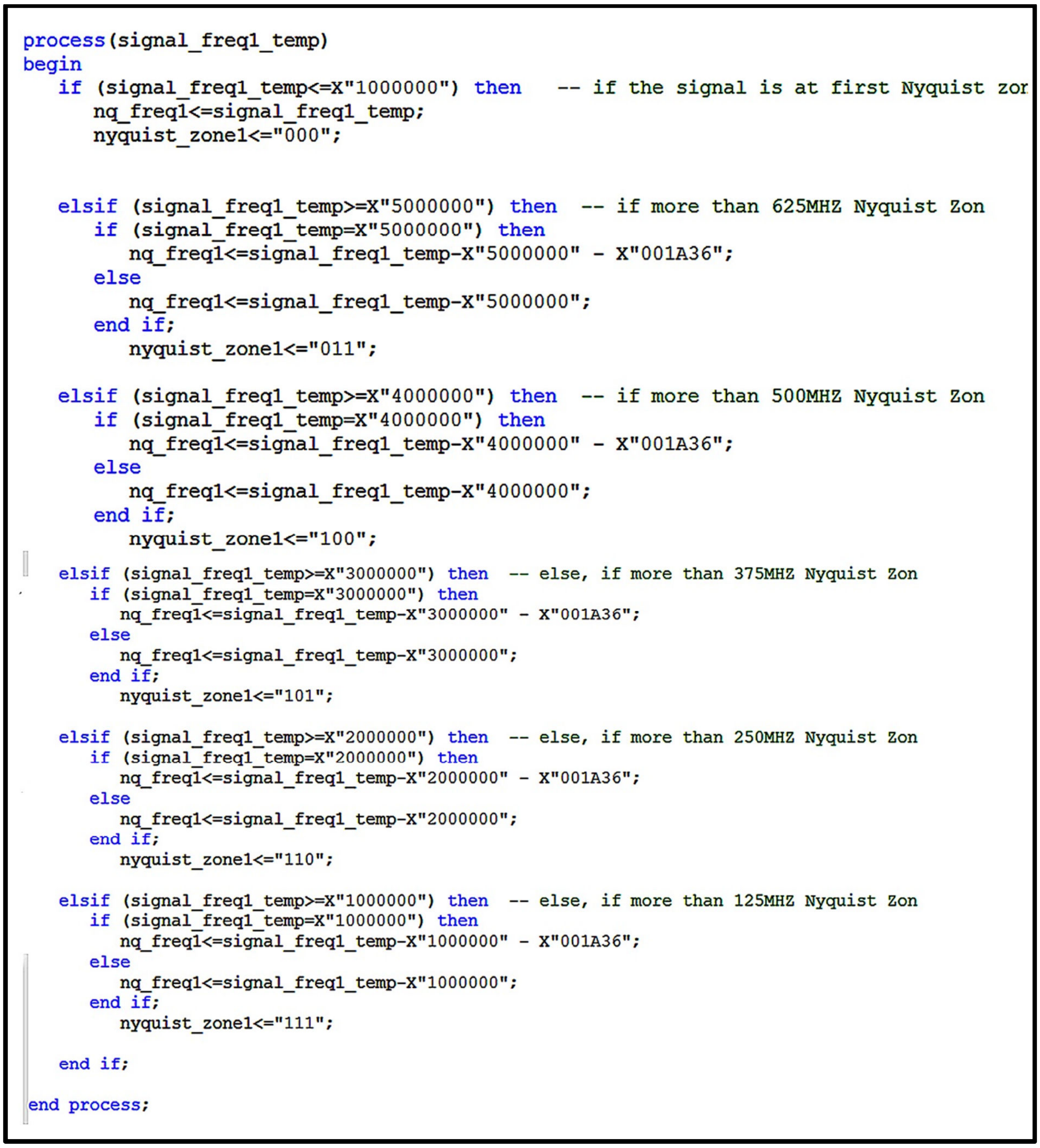
Figure 3.43: Visualisation of complete sine-wave.

3.5.5.6 *Development of an 8-core DDS System with Under-sampling*

This section describes the reconstruction to a higher frequency signal based on the under-sampling and aliasing concepts. Such a reconstruction is necessary because a signal frequency greater than 125 MHz does not meet the Nyquist criterion. The under-sampling is a method to synthesise an output signal by using its aliased frequency. The tuning word of the aliased frequency is the phase increment value in a phase accumulator. The accumulated phases generated by the accumulator are shifted by phase offset values in order to reproduce higher frequency samples.

3.5.5.7 *VHDL Implementation of an 8-core DDS System with Under-sampling*

Figure 3.44 shows the VHDL routine for tracking the Nyquist zone of an output signal. This routine is developed by referring to the equation (3.12).



```
process(signal_freq1_temp)
begin
   if (signal_freq1_temp<=X"1000000") then   -- if the signal is at first Nyquist zor
      nq_freq1<=signal_freq1_temp;
      nyquist_zone1<="000";


   elsif (signal_freq1_temp>=X"5000000") then  -- if more than 625MHZ Nyquist Zon
      if (signal_freq1_temp=X"5000000") then
         nq_freq1<=signal_freq1_temp-X"5000000" - X"001A36";
      else
         nq_freq1<=signal_freq1_temp-X"5000000";
      end if;
         nyquist_zone1<="011";

   elsif (signal_freq1_temp>=X"4000000") then  -- if more than 500MHZ Nyquist Zon
      if (signal_freq1_temp=X"4000000") then
         nq_freq1<=signal_freq1_temp-X"4000000" - X"001A36";
      else
         nq_freq1<=signal_freq1_temp-X"4000000";
      end if;
         nyquist_zone1<="100";

   elsif (signal_freq1_temp>=X"3000000") then  -- else, if more than 375MHZ Nyquist Zon
      if (signal_freq1_temp=X"3000000") then
         nq_freq1<=signal_freq1_temp-X"3000000" - X"001A36";
      else
         nq_freq1<=signal_freq1_temp-X"3000000";
      end if;
         nyquist_zone1<="101";

   elsif (signal_freq1_temp>=X"2000000") then  -- else, if more than 250MHZ Nyquist Zon
      if (signal_freq1_temp=X"2000000") then
         nq_freq1<=signal_freq1_temp-X"2000000" - X"001A36";
      else
         nq_freq1<=signal_freq1_temp-X"2000000";
      end if;
         nyquist_zone1<="110";

   elsif (signal_freq1_temp>=X"1000000") then  -- else, if more than 125MHZ Nyquist Zon
      if (signal_freq1_temp=X"1000000") then
         nq_freq1<=signal_freq1_temp-X"1000000" - X"001A36";
      else
         nq_freq1<=signal_freq1_temp-X"1000000";
      end if;
         nyquist_zone1<="111";

   end if;

end process;
```

Figure 3.44: VHDL program to search for image frequency.

In Figure 3.44, 'signal_freq1_temp' is a temporary variable, which contains a tuning word of an output frequency. If ‘signal_freq1_temp’ contains the tuning word, which is found to be in the 1st Nyquist zone, it is passed over to the phase accumulator via ‘nq_freq1’. Just in case, the ‘signal_freq1_temp’ has a frequency tuning word which is outside of the 1st Nyquist zone, the VHDL routine tracks the image frequency of the output signal by comparing the 'Signal_freq1_temp' with the lower band-limit of every Nyquist zone (in hex). Once the Nyquist zone is tracked, the logic status of a 3-bit variable, ‘nyquist_zone1’ is set according to the combinational logics in Table 3.8. The contents of ‘nyquist_zone1’ are passed over to the variable ‘NYQUIST_ZONE’ in the next VHDL routine.

3-bit ‘nyquist_zone1’	***Nyquist Zone***
000	1
111	2
110	3
101	4
100	5
011	6

Table 3.8: 3-bit ‘nyquist_zone1’ vs. Nyquist zone.

In practice, the 8-core DDS system can generate a signal up to the 6th Nyquist zone. Any signals beyond this have insufficient samples to maintain the shape as a sinusoidal signal.

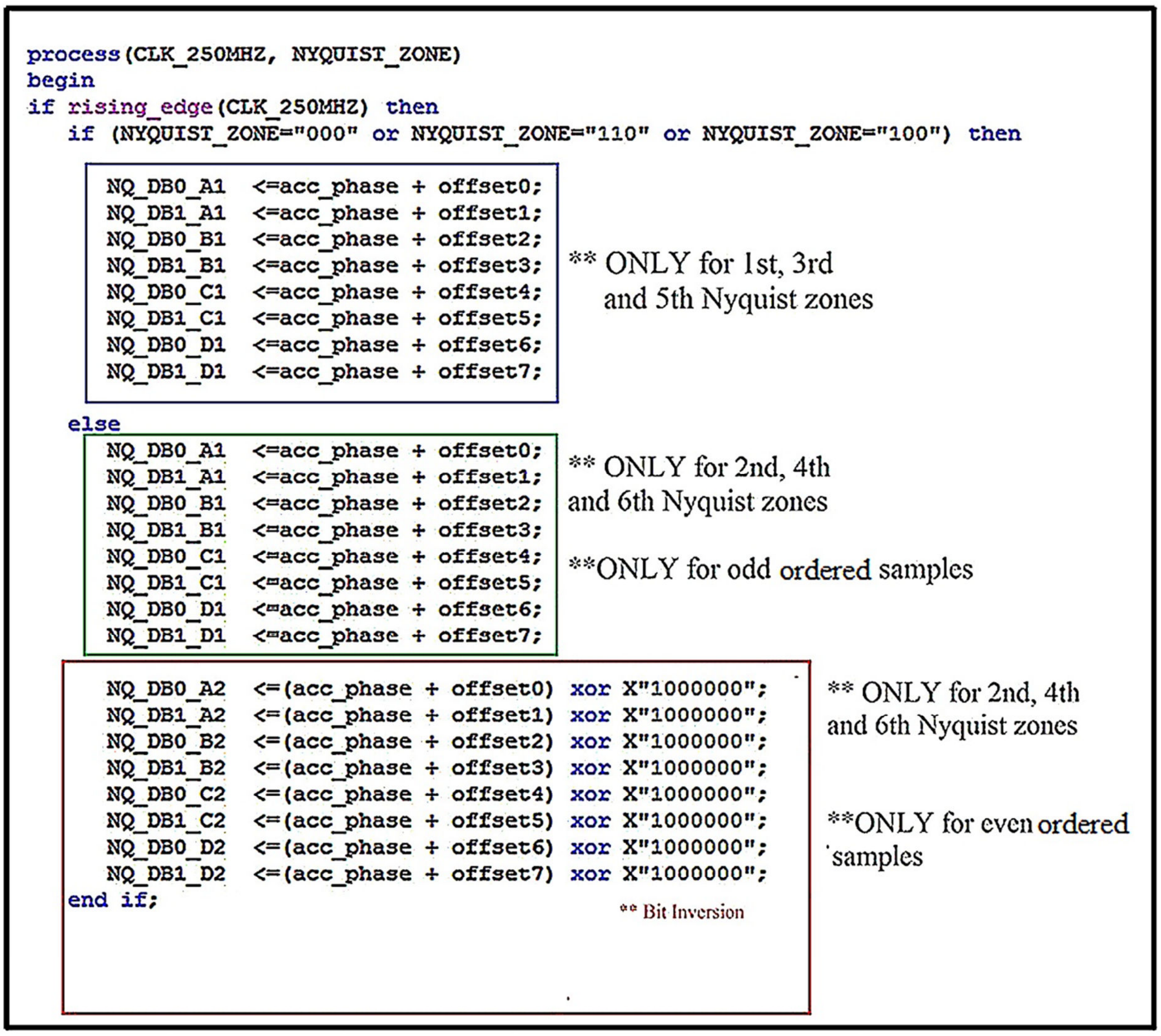
There are two conditions in synthesising a signal using an 8-core DDS module. The first condition is that the signal frequency might be in the 'blind spots' of the Nyquist zones. If the output frequency resided exactly at the multiple of the Nyquist frequency, the output frequency would equal to the image frequency. This is because the 8-core DDS module would not be able to distinguish the output frequency with the lower band-limit of the Nyquist zone. It would end up that no phase offsets are created to shift the samples of aliased frequency. Therefore, samples of aliased frequency would appear as an output signal.

In this case, a frequency tolerance would be inserted into the sample phases. Several frequency tolerances have been tried out and it was found that 50 kHz (1A36 hex) is the best option to shift the output frequency by 50 kHz lower than the actual output. If the frequency tolerance is less than 50 kHz, it could not shift the signal frequency away from the blind spot.

The second condition concerns the fact that some of the amplitudes from certain Nyquist zones encounter polarity errors because of the addition of the phase offsets with accumulated phases. These polarity errors will be solved in the next VHDL routine before transferring to the DAC chip.

Polarity Errors of Amplitude.

The MSB of the accumulated phases in a DDS represents the polarity of an output sample. In an 8-core DDS system, the addition of Nyquist offsets and the accumulated phases leads to polarity errors due to the overflowing of 25-bit sample phases. However, the polarity errors do not happen to every sample generated by the system. They only happen to the even ordered samples from the 2nd, 4th and 6th Nyquist zones. To implement the bit inversion, the phases are 'XOR' gated by '1000000 hex'.



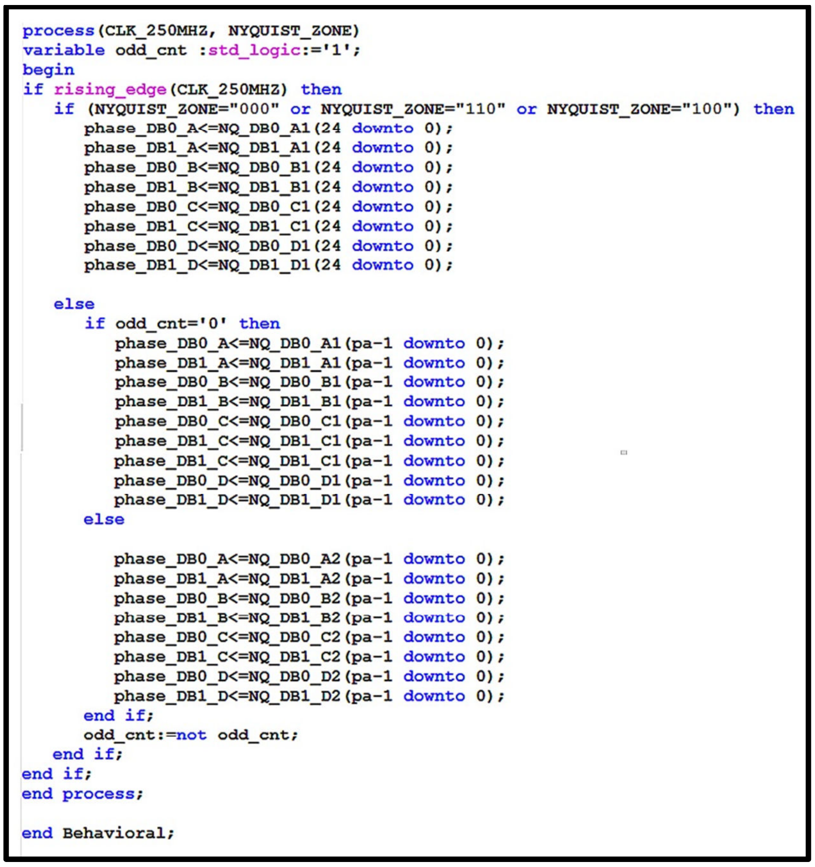
```
process(CLK_250MHZ, NYQUIST_ZONE)
begin
if rising_edge(CLK_250MHZ) then
   if (NYQUIST_ZONE="000" or NYQUIST_ZONE="110" or NYQUIST_ZONE="100") then
      NQ_DB0_A1   <=acc_phase + offset0;
      NQ_DB1_A1   <=acc_phase + offset1;
      NQ_DB0_B1   <=acc_phase + offset2;
      NQ_DB1_B1   <=acc_phase + offset3;
      NQ_DB0_C1   <=acc_phase + offset4;
      NQ_DB1_C1   <=acc_phase + offset5;
      NQ_DB0_D1   <=acc_phase + offset6;
      NQ_DB1_D1   <=acc_phase + offset7;
   else
      NQ_DB0_A1   <=acc_phase + offset0;
      NQ_DB1_A1   <=acc_phase + offset1;
      NQ_DB0_B1   <=acc_phase + offset2;
      NQ_DB1_B1   <=acc_phase + offset3;
      NQ_DB0_C1   <=acc_phase + offset4;
      NQ_DB1_C1   <=acc_phase + offset5;
      NQ_DB0_D1   <=acc_phase + offset6;
      NQ_DB1_D1   <=acc_phase + offset7;
      NQ_DB0_A2   <=(acc_phase + offset0) xor X"1000000";
      NQ_DB1_A2   <=(acc_phase + offset1) xor X"1000000";
      NQ_DB0_B2   <=(acc_phase + offset2) xor X"1000000";
      NQ_DB1_B2   <=(acc_phase + offset3) xor X"1000000";
      NQ_DB0_C2   <=(acc_phase + offset4) xor X"1000000";
      NQ_DB1_C2   <=(acc_phase + offset5) xor X"1000000";
      NQ_DB0_D2   <=(acc_phase + offset6) xor X"1000000";
      NQ_DB1_D2   <=(acc_phase + offset7) xor X"1000000";
   end if;
```

** ONLY for 1st, 3rd and 5th Nyquist zones
** ONLY for 2nd, 4th and 6th Nyquist zones
**ONLY for odd ordered samples
** ONLY for 2nd, 4th and 6th Nyquist zones
**ONLY for even ordered samples
** Bit Inversion

Figure 3.45: Bit inversion for the output phases.

In Figure 3.45, the polarity error of output samples is fixed by the VHDL routine highlighted in red. If the samples are of odd order, it does not require any inversion, just like the VHDL routine highlighted in green. If the samples are 1st, 3rd and 5th Nyquist zones, the inversion of MSB is unnecessary such as the VHDL routine shown highlighted in blue. A 3-bit 'NYQUST_ZONE' is a variable, which contains the information of 'nyquist_zone1' (Figure 3.44).

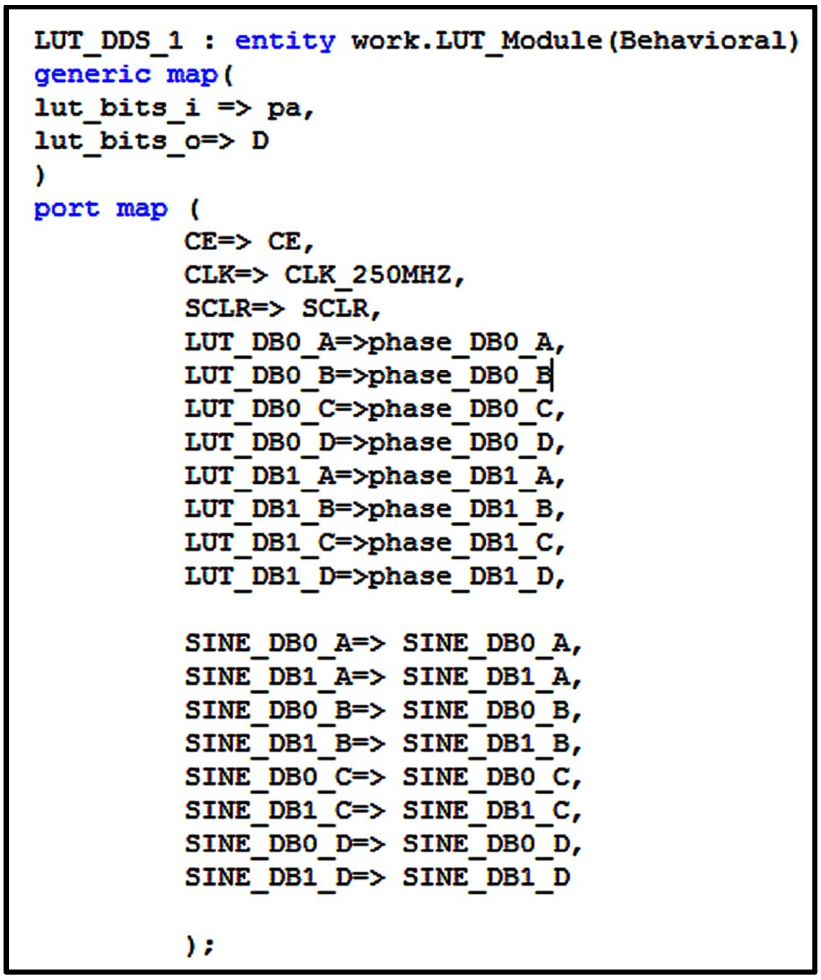
Besides fixing the polarity error, the VHDL routine in Figure 3.45 is also used to create eight sample phases. The 25-bit variable 'acc_phase' is used to store the accumulated phases generated by a phase accumulator while 'offset0' to 'offset7' are variables, which contain the phase offset values. The eight sequential samples which are created after adding the accumulated phases with the phase offsets are stored in NQ_DB0[A1: D2].



```
process(CLK_250MHZ, NYQUIST_ZONE)
variable odd_cnt :std_logic:='1';
begin
if rising_edge(CLK_250MHZ) then
   if (NYQUIST_ZONE="000" or NYQUIST_ZONE="110" or NYQUIST_ZONE="100") then
      phase_DB0_A<=NQ_DB0_A1(24 downto 0);
      phase_DB1_A<=NQ_DB1_A1(24 downto 0);
      phase_DB0_B<=NQ_DB0_B1(24 downto 0);
      phase_DB1_B<=NQ_DB1_B1(24 downto 0);
      phase_DB0_C<=NQ_DB0_C1(24 downto 0);
      phase_DB1_C<=NQ_DB1_C1(24 downto 0);
      phase_DB0_D<=NQ_DB0_D1(24 downto 0);
      phase_DB1_D<=NQ_DB1_D1(24 downto 0);

   else
      if odd_cnt='0' then
         phase_DB0_A<=NQ_DB0_A1(pa-1 downto 0);
         phase_DB1_A<=NQ_DB1_A1(pa-1 downto 0);
         phase_DB0_B<=NQ_DB0_B1(pa-1 downto 0);
         phase_DB1_B<=NQ_DB1_B1(pa-1 downto 0);
         phase_DB0_C<=NQ_DB0_C1(pa-1 downto 0);
         phase_DB1_C<=NQ_DB1_C1(pa-1 downto 0);
         phase_DB1_C<=NQ_DB1_C1(pa-1 downto 0);
         phase_DB0_D<=NQ_DB0_D1(pa-1 downto 0);
         phase_DB1_D<=NQ_DB1_D1(pa-1 downto 0);
      else

         phase_DB0_A<=NQ_DB0_A2(pa-1 downto 0);
         phase_DB1_A<=NQ_DB1_A2(pa-1 downto 0);
         phase_DB0_B<=NQ_DB0_B2(pa-1 downto 0);
         phase_DB1_B<=NQ_DB1_B2(pa-1 downto 0);
         phase_DB0_C<=NQ_DB0_C2(pa-1 downto 0);
         phase_DB1_C<=NQ_DB1_C2(pa-1 downto 0);
         phase_DB0_D<=NQ_DB0_D2(pa-1 downto 0);
         phase_DB1_D<=NQ_DB1_D2(pa-1 downto 0);
      end if;
      odd_cnt:=not odd_cnt;
   end if;
end if;
end process;

end Behavioral;
```

Figure 3.46: VHDL code to transfer the phases from the even Nyquist zone to LUTs.

In Figure 3.46, 25-bit variables, 'phase_DB0_ [A: D]' and 'phase_DB1_ [A: D]' are used to pass the sample phases to LUTs. ‘NYQUIST_ZONE’ is a variable used to determine the Nyquist zone of the output signal so that correct samples could be sent to the LUTs. ‘odd_cnt’ is a variable used to specify the order of the samples so that samples with or without MSB inverted can be clarified.

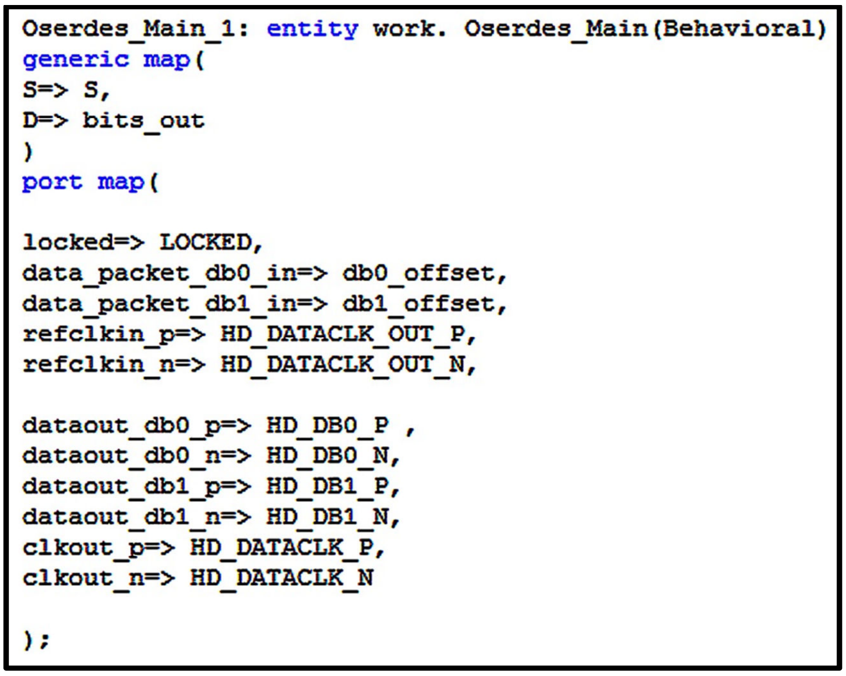


```
LUT_DDS_1 : entity work.LUT_Module(Behavioral)
generic map(
lut_bits_i => pa,
lut_bits_o=> D
)
port map (
         CE=> CE,
         CLK=> CLK_250MHZ,
         SCLR=> SCLR,
         LUT_DB0_A=>phase_DB0_A,
         LUT_DB0_B=>phase_DB0_B
         LUT_DB0_C=>phase_DB0_C,
         LUT_DB0_D=>phase_DB0_D,
         LUT_DB1_A=>phase_DB1_A,
         LUT_DB1_B=>phase_DB1_B,
         LUT_DB1_C=>phase_DB1_C,
         LUT_DB1_D=>phase_DB1_D,

         SINE_DB0_A=> SINE_DB0_A,
         SINE_DB1_A=> SINE_DB1_A,
         SINE_DB0_B=> SINE_DB0_B,
         SINE_DB1_B=> SINE_DB1_B,
         SINE_DB0_C=> SINE_DB0_C,
         SINE_DB1_C=> SINE_DB1_C,
         SINE_DB0_D=> SINE_DB0_D,
         SINE_DB1_D=> SINE_DB1_D

         );
```

Figure 3.47: Declaration of LUT module.

In Figure 3.47, the variables 'phase_DB0_ [A: D]' and 'phase_DB1_ [A: D]' pass the eight sample phases to the LUT module. Eight LUTs are synchronised by a 250 MHz clock signal, which is the same source as the phase accumulator. After the conversion, the amplitudes are stored in 25-bit 'SINE_DB0_ [A: D]' and 'SINE_DB1_ [A: D]'. These variables pass the amplitudes to the OSERDES module for transferring to DAC.

Arrangement of the Output Data



```
Oserdes_Main_1: entity work. Oserdes_Main(Behavioral)
generic map(
S=> S,
D=> bits_out
)
port map(

locked=> LOCKED,
data_packet_db0_in=> db0_offset,
data_packet_db1_in=> db1_offset,
refclkin_p=> HD_DATACLK_OUT_P,
refclkin_n=> HD_DATACLK_OUT_N,

dataout_db0_p=> HD_DB0_P ,
dataout_db0_n=> HD_DB0_N,
dataout_db1_p=> HD_DB1_P,
dataout_db1_n=> HD_DB1_N,
clkout_p=> HD_DATACLK_P,
clkout_n=> HD_DATACLK_N

);
```

Figure 3.48: VHDL function to transfer phase values to a LUT module.

The 8-core DDS system generates eight sample phases with 14-bit of each 8 X 14 = 112-bit on every 250 MHz. These 112-bit samples must be sent out from Spartan-6 before the accumulator triggers on the next clock cycle again. Since the DAC chip receives two sets of data in parallel (each set has 4 samples), then the Spartan-6 must transfer two sets of data at 1 GSPS of each to DAC (4 samples X 250 MSPS = 1 GSPS).

The transmission of the data is driven by a 500 MHz differential clock signal (HD_DATACLK_P and HD_DATACLK_N) from Spartan-6 to DAC. Meanwhile, a 500 MHz differential signal (‘HD_DATACLK_OUT_P’ and ‘HD_DATACLK_OUT_P’) are provided by the DAC chip for synchronising the transmission from Spartan-6 to DAC.

Sequence of Data	Variables
1	SINE_DB0_A
2	SINE_DB1_A
3	SINE_DB0_B
4	SINE_DB1_B
5	SINE_DB0_C
6	SINE_DB1_C
7	SINE_DB0_D
8	SINE_DB1_D

Table 3.9: Sequences to transfer digital amplitudes to DAC.

In Table 3.9, ‘SINE_DB0_ [A: D]’ and ‘SINE_DB1_ [A: D]’ are the eight samples of a signal. These samples are sequential and their orders in an output signal are stated in Table 3.9. During the OSERDES process, these samples are placed on 14-bit ‘HD_DB0_P/N’ and ‘HD_DB1_P/N’ differential signals. ‘HD_DB0_P/N’ sends the odd ordered samples to DAC while ‘HD_DB1_P/N’ would send the even ordered samples to DAC.

3.5.5.8 Simulation Results

Two random signal frequencies, 190 MHz and 290 MHz, were simulated using the VHDL compiler. The simulation results can be found in Figures 3.49 to 3.52. Table 3.10 and Table 3.11 list the amplitudes of the first 33 samples.

<u>190 MHz Sinusoidal Signal</u>

Based on equation (3.12),

190 MHz = 65 MHz + 125 MHz

The 190 MHz signal is from the 2nd Nyquist zone so the lower band-limit is 125 MHz. The system subtracts 125 MHz from 190 MHz and the remainder is the 65 MHz aliased signal. The frequency tuning words of 190 MHz and 65 MHz can be obtained by using equation (2.0).

$$M_{125MHz} = \frac{190\ MHz\ x\ 2^{25}}{250\ MHz} = 25501368$$

$$M_{65MHz} = \frac{65\ MHz\ x\ 2^{25}}{250\ MHz} = 8724152$$

The step size of the output signal can be obtained by using equation (3.5).

$$Step\ size = \frac{25501368}{8} = 3187671$$

After the step size is calculated, the eight phase offsets can be obtained by using equation (3.6).

- Phase Offset for 1st sample = step size = 3187671
- Phase Offset for 2nd sample = 2 x 3187671 = 6375342
- Phase Offset for 3rd sample = 3 x 3187671 = 9563013
- Phase Offset for 4th sample = 4 x 3187671 = 12750684
- Phase Offset for 5th sample = 5 x 3187671 = 15938355
- Phase Offset for 6th sample = 6 x 3187671 = 19126026
- Phase Offset for 7th sample = 7 x 3187671 = 22313697
- Phase Offset for 8th sample = 8 x 3187671 = 25501368

The tuning word of 65 MHz, 8724152, was sent to a phase accumulator for the accumulation process. Meanwhile, the tuning word of 190 MHz, 25501368, is passed over to the offset divider for the offset calculation. The addition of accumulated phases (image frequency) with the phase offsets creates the samples of a higher frequency signal.

The simulation waveform generated using ISim is shown in Figure 3.49. Results from the waveform were captured using a test-bench and are listed in Table 3.10. The 190 MHz signal is plotted in Figure 3.50 by using the data recorded in Table 3.10.

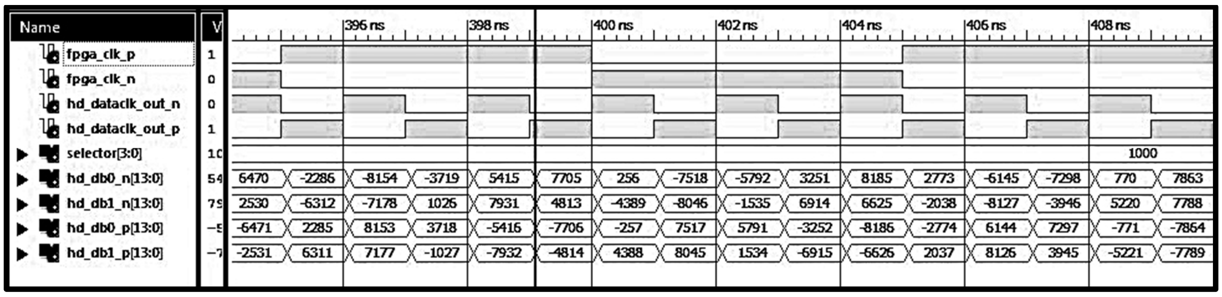


Figure 3.49: Simulation results of a 190 MHz signal using Isim.

Unit time	**Amplitude**	**Unit time**	**Amplitude**	**Unit time**	**Amplitude**
1	-6471	**12**	-4814	**23**	-2774
2	-2531	**13**	-257	**24**	2037
3	2285	**14**	4388	**25**	6144
4	6311	**15**	7517	**26**	8126
5	8153	**16**	8045	**27**	7297
6	7177	**17**	5791	**28**	3945
7	3718	**18**	1534	**29**	-771
8	-1027	**19**	-3252	**30**	-5221
9	-5416	**20**	-6915	**31**	-7864
10	-7932	**21**	-8186	**32**	-7789
11	-7706	**22**	-6626	**33**	-5020

Table 3.10: First 33 data points of a 190 MHz sine-wave.

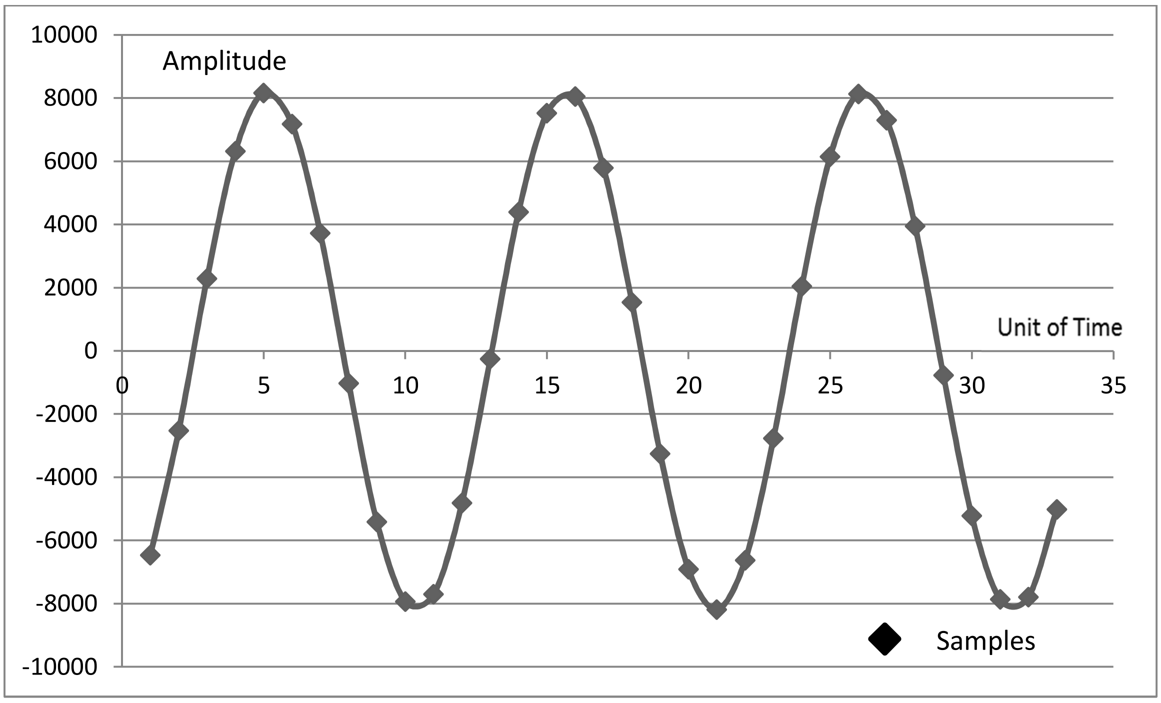


Figure 3.50: 190MHz sine-wave.

In Figure 3.50, the phase offsets shift the accumulated phases of 65 MHz to the position, which is equivalent to the samples of a 190 MHz signal. The plot does not show the actual frequency of the signal but an approximation on the total samples needed by a unit cycle of a 190 MHz sine-wave can be calculated by using the method below:

The ratio of output frequency to aliased frequency, $R_{ratio} = \frac{f_{out}}{f_{alias}}$. So,

$$f_{out} = f_{alias}\ x\ R_{ratio} \tag{3.13}$$

Referring to equation (3.8),

Total samples of a unit cycle of aliased signal, $T_{alias} = \frac{8\ x\ f_s}{f_{alias}}$ (3.14)

The equation (3.13) shows the output frequency is R_{ratio} times faster than f_{alias} . Then,

$$t_{alias} = R_{ratio}\ x\ t_{out} \tag{3.15}$$

Based on equation (3.15), the total samples in a unit t_{alias} is the same as the total samples in a unit t_{out}. Therefore, Total samples of a unit cycle of output signal,$T_{samples} = \frac{T_{alias}}{R_{ratio}}$ (3.16)

By substituting equation (3.13) with equation (3.14),

$$T_{alias} = \frac{8 \; x \; f_S \; x \; R_{ratio}}{f_{out}} \tag{3.17}$$

By substituting equation (3.17) with equation (3.16),

$$T_{samples} = \frac{8 \; x \; f_S}{f_{out}} \tag{3.18}$$

Equation (3.18) can be used to calculate the total samples of a unit sine-wave when the output frequency is outside of the 1st Nyquist zone.

Total samples of a unit cycle of 190 MHz sine-wave, $T_{samples} = \frac{8 \; x \; 250 \; MHz}{190 \; MHz}$

$T_{samples} \approx \mathbf{11 \; samples}$

By referring to Table 3.10 and Figure 3.50, the total samples in a unit sine-wave are 10 or 11 samples, which are similar to the result calculated by using the approximation method.

Another experiment is carried out on a 290 MHz output frequency. Figure 3.52 shows the test results plotted based on the data in Table 3.11 and Figure 3.51.

290 MHz Sinusoidal Signal

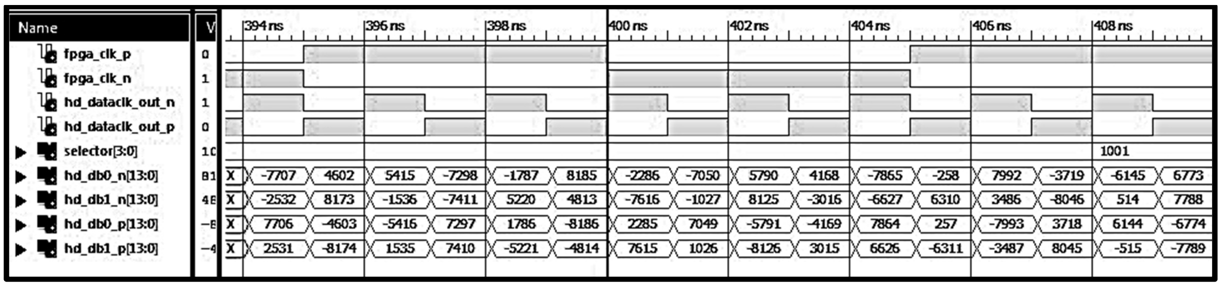


Figure 3.51: Simulation results of a 290 MHz signal using ISim.

Unit time	Amplitude	Unit time	Amplitude	Unit time	Amplitude
1	7706	**12**	-4814	**23**	257
2	2531	**13**	2285	**24**	-6311
3	-4603	**14**	7615	**25**	-7993
4	-8174	**15**	7049	**26**	-3487
5	-5416	**16**	1026	**27**	3718
6	1535	**17**	-5791	**28**	8045
7	7297	**18**	-8126	**29**	6144
8	7410	**19**	-4169	**30**	-515
9	1786	**20**	3015	**31**	-6774
10	-5221	**21**	7864	**32**	-7789
11	-8186	**22**	6626	**33**	-2774

Table 3.11: First 33 data points of a 290 MHz Sine-Wave (3rd Nyquist Zone).

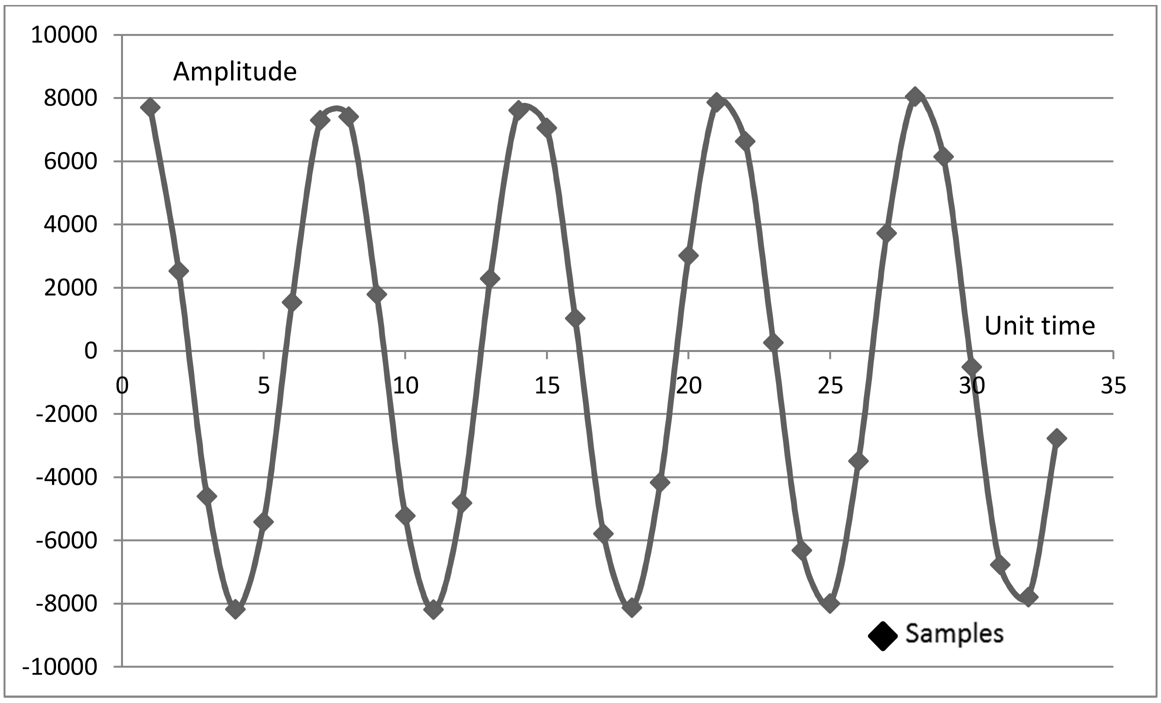


Figure 3.52: 290MHz sine-wave.

Based on equation (3.12),

290 MHz = 40 MHz + 250 MHz

40 MHz is the aliased frequency of 290 MHz and 250 MHz is the lower band-limit frequency of the Nyquist zone where 290 MHz is located.

The frequency tuning words of 190 MHz and 65 MHz can be obtained by using equation (2.0).

$$M_{290MHz} = \frac{290\ MHz\ x\ 2^{25}}{250\ MHz} = 38923141$$

$$M_{40MHz} = \frac{40\ MHz\ x\ 2^{25}}{250\ MHz} = 5368709$$

The step size of the output signal can be obtained by using equation (3.5).

$$Step\ size = \frac{38923141}{8} = 4865392$$

After the step size is calculated, the eight phase offsets can be obtained by using equation (3.6).

• Phase Offset for 1st sample	= step size	= 4865392
• Phase Offset for 2nd sample	= 2 x 4865392	= 9730784
• Phase Offset for 3rd sample	= 3 x 4865392	= 14596176
• Phase Offset for 4th sample	= 4 x 4865392	= 19461568
• Phase Offset for 5th sample	= 5 x 4865392	= 24326960
• Phase Offset for 6th sample	= 6 x 4865392	= 29192352
• Phase Offset for 7th sample	= 7 x 4865392	= 34057744
• Phase Offset for 8th sample	= 8 x 4865392	= 38923136

The approximation on the samples needed by a unit cycle of 290 MHz sine-wave can be calculated as below.

Referring to equation (3.18):

Total samples of a unit cycle of 190 MHz sine-wave, $T_{samples} = \frac{8\ x\ 250\ MHz}{290\ MHz}$

$T_{samples} \approx \mathbf{7\ samples}$

Table 3.11 and Figure 3.52 indicate that a 290 MHz sine-wave has 6 or 7 samples per unit cycle, which is similar to the result calculated by using the approximation method.

Discussion of Simulation Results

The simulation demonstrates how a sinusoidal signal can be generated by using the under-sampling concept. The simulation results do not display the actual frequency of the signal but they do indicate the total samples distributed in a signal. An approximation method can be used to predict the total sample in a unit sine-wave. The calculated result can be used to compare with the total samples counted from a sine-wave plot to verify the simulated results.

By comparing Figure 3.50 and Figure 3.52, it can be seen that 290 MHz has more sine cycles than 190 MHz. Within the first 33 samples, 290 MHz output signal has 4 cycles while 190 MHz has only 2.5 cycles. In 190 MHz output signal, a unit sine-wave requires about 11 samples. Since eight samples are generated over a unit time, which means some of the 25-bit sample phases overflow within two clock cycles even though the phase accumulator has yet to overflow. In 290 MHz output signal, the sine-wave requires approximately 7 samples to form a unit of sine-wave. This means some of the 25-bit sample phases overflow within the first clock cycle.

The step sizes of the 190 MHz and 290 MHz output signals are 3187671 and 4865392. The results show that 190 MHz has higher precision than 290 MHz. This is because the total samples required by a 290 MHz signal are less than 190 MHz signal. The total samples required to form a unit sine-wave are inversely proportional to the output frequency. Higher signal frequency has lesser samples distributed in the signal and vice versa. This is because the 25-bit sample phases of higher frequency signal overflow quicker than the sample phases of lower frequency signal.

The impression of distortion in Figure 3.50 and 3.52 is due to the X-Y (Scatter) function of Microsoft Excel. The interpolation filter used by Excel is not a sinc interpolation. Therefore, a perfect sine-wave could not be expected using Microsoft Excel.

3.5.6 Development of Multi-Frequency RF Generator

The multi-frequency RF generator contains four sets of 8-core DDS modules to support frequency switching up to four frequencies. The primary intention of this feature is to enable phase coherence among these four signs. However, not every module in the 8-core DDS system is duplicated to support this feature. Some modules such as LUTs and OSERDES are reusable. Three more offset dividers and phase accumulators are added to the system, which means the entire system contains four sets of offset dividers and phase accumulators. Although the samples of four signals are generated simultaneously, only one signal is selected to the output. The specifications of a multi-frequency RF generator are as below:

System Clock, f_{system}	: 250 MHz
Frequency Resolution, Δf	: 0.93 Hz
Total Input Tuning Words	: 4
Total Output Frequency	: 1
Total LUTs	: 8
Input Width of LUT	: 25-bit
Output Width of LUT	: 14-bit
Depth of sine LUT	: 16384 samples
Total Phase Accumulators	: 4
Input width of Phase Accumulator	: 25-bit
Depth of Phase Accumulator	: 33554432
Total Phase Offsets	: 8 sets with 25-bit of each

3.5.6.1 RTL Schematics of a Multi-Frequency RF Generator

The RTL schematics of a multi-frequency RF generator are similar to the 8-core DDS module except the multi-frequency RF generator has four sets of offset dividers and four sets of phase accumulators to support the switching between four frequencies. These can be seen in Figure 3.53 and Figure 3.54.

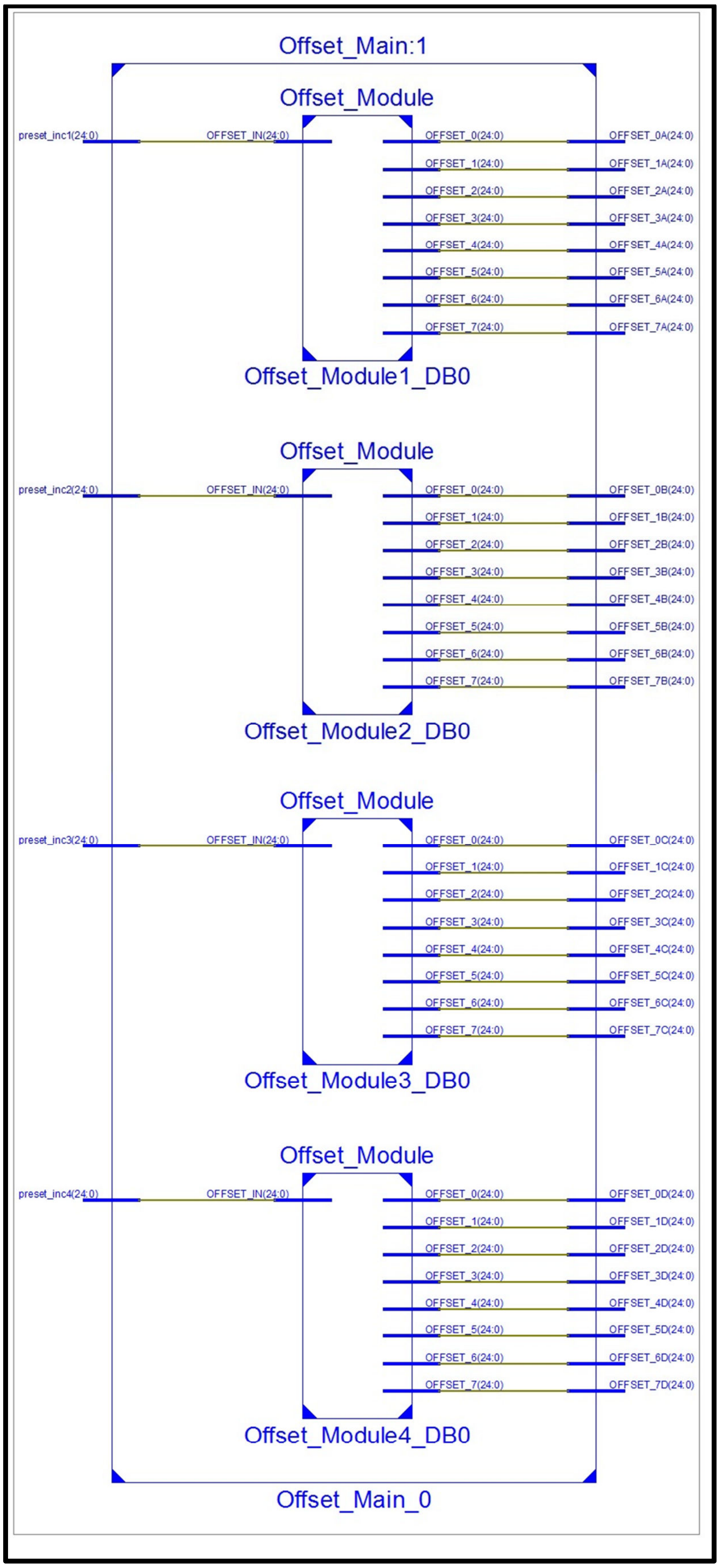


Figure 3.53: Four offset module.

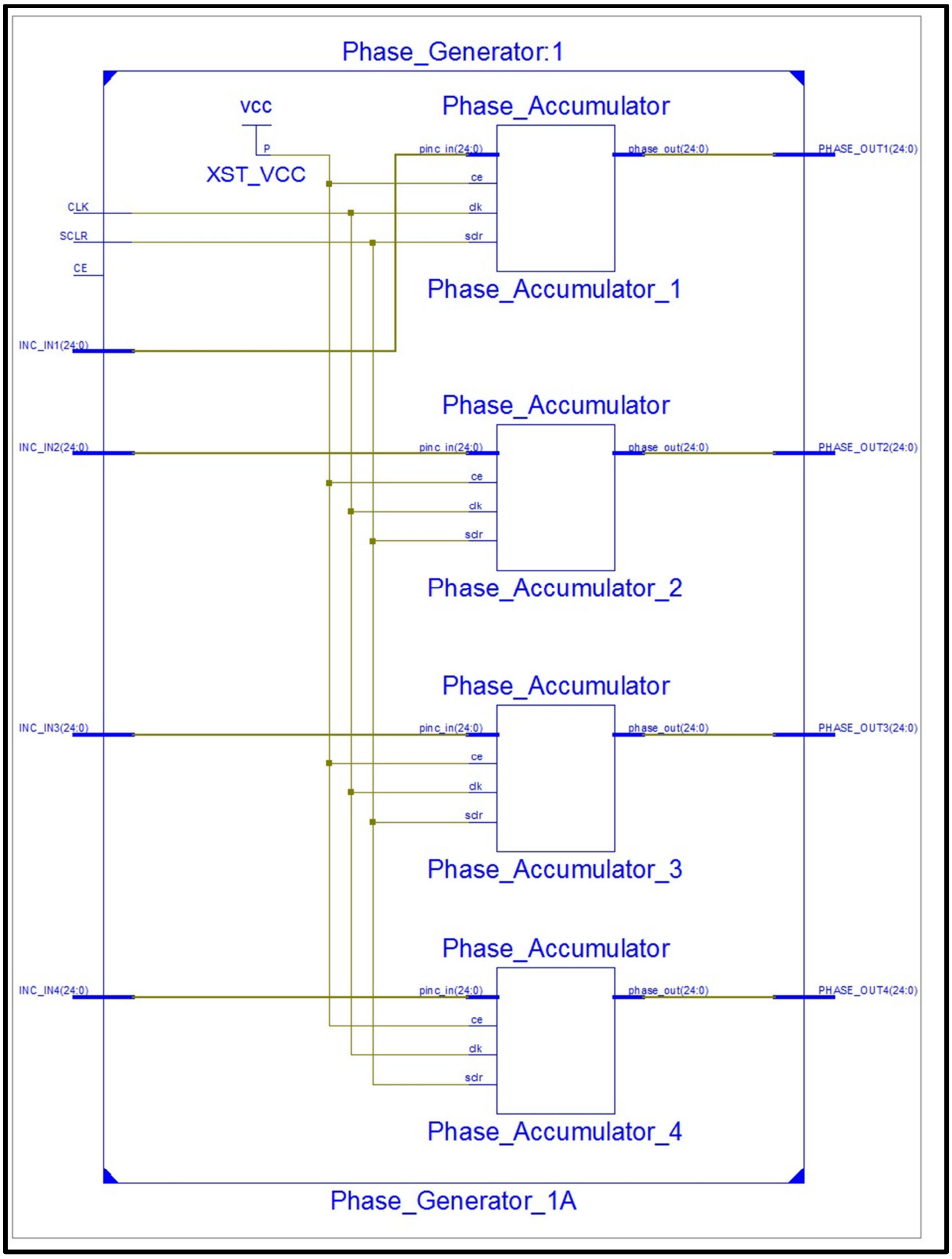
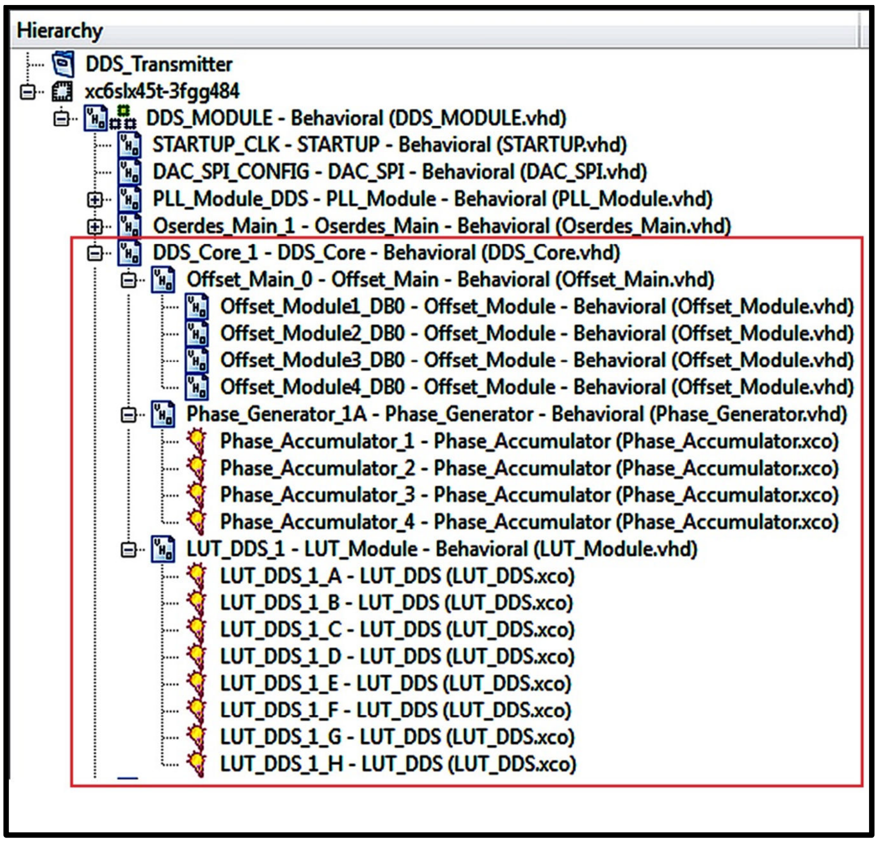


Figure 3.54: Four phase accumulators.

In Figure 3.54, each of the phase accumulators will be connected to a frequency tuning word. The phases generated by each accumulator will be added to eight phase offsets. This feature leads the Spartan-6 to process four signals in parallel. Hence, the entire operation is similar to four sets of 8-core DDS modules, which function simultaneously.

3.5.6.2 Implementation of Multi-Frequency RF Generator in VHDL

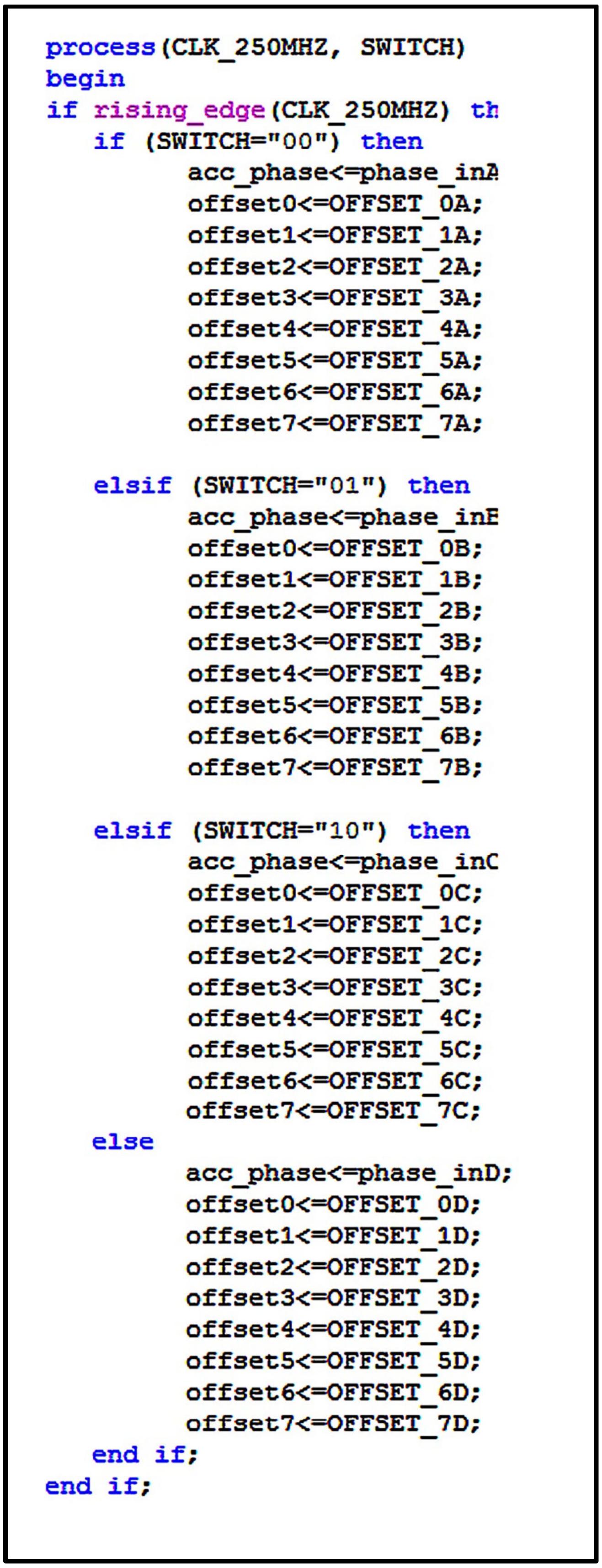


```
Hierarchy
  DDS_Transmitter
  xc6slx45t-3fgg484
    DDS_MODULE - Behavioral (DDS_MODULE.vhd)
      STARTUP_CLK - STARTUP - Behavioral (STARTUP.vhd)
      DAC_SPI_CONFIG - DAC_SPI - Behavioral (DAC_SPI.vhd)
      PLL_Module_DDS - PLL_Module - Behavioral (PLL_Module.vhd)
      Oserdes_Main_1 - Oserdes_Main - Behavioral (Oserdes_Main.vhd)
      DDS_Core_1 - DDS_Core - Behavioral (DDS_Core.vhd)
        Offset_Main_0 - Offset_Main - Behavioral (Offset_Main.vhd)
          Offset_Module1_DB0 - Offset_Module - Behavioral (Offset_Module.vhd)
          Offset_Module2_DB0 - Offset_Module - Behavioral (Offset_Module.vhd)
          Offset_Module3_DB0 - Offset_Module - Behavioral (Offset_Module.vhd)
          Offset_Module4_DB0 - Offset_Module - Behavioral (Offset_Module.vhd)
        Phase_Generator_1A - Phase_Generator - Behavioral (Phase_Generator.vhd)
          Phase_Accumulator_1 - Phase_Accumulator (Phase_Accumulator.xco)
          Phase_Accumulator_2 - Phase_Accumulator (Phase_Accumulator.xco)
          Phase_Accumulator_3 - Phase_Accumulator (Phase_Accumulator.xco)
          Phase_Accumulator_4 - Phase_Accumulator (Phase_Accumulator.xco)
        LUT_DDS_1 - LUT_Module - Behavioral (LUT_Module.vhd)
          LUT_DDS_1_A - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_B - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_C - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_D - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_E - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_F - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_G - LUT_DDS (LUT_DDS.xco)
          LUT_DDS_1_H - LUT_DDS (LUT_DDS.xco)
```

Figure 3.55: Project hierarchy of multi-frequency DDS system.

Figure 3.55 shows the VHDL project hierarchy of a multi-frequency DDS module. The VHDL file highlighted in red indicates that the phase accumulator and the offset divider from an 8-core DDS module have been duplicated into four sets of phase accumulators and four sets of offset dividers.

In a multi-frequency generator, the four phase accumulators generate sample phases in parallel. A switch is used to select the accumulated phases of only one phase accumulator and send it to the next routine for adding with its phase offsets. Therefore, a 'SWITCH' is used to control the switching between phase accumulators as shown in Figure 3.56.



```
process(CLK_250MHZ, SWITCH)
begin
if rising_edge(CLK_250MHZ) th
   if (SWITCH="00") then
         acc_phase<=phase_inA
         offset0<=OFFSET_0A;
         offset1<=OFFSET_1A;
         offset2<=OFFSET_2A;
         offset3<=OFFSET_3A;
         offset4<=OFFSET_4A;
         offset5<=OFFSET_5A;
         offset6<=OFFSET_6A;
         offset7<=OFFSET_7A;

   elsif (SWITCH="01") then
         acc_phase<=phase_inB
         offset0<=OFFSET_0B;
         offset1<=OFFSET_1B;
         offset2<=OFFSET_2B;
         offset3<=OFFSET_3B;
         offset4<=OFFSET_4B;
         offset5<=OFFSET_5B;
         offset6<=OFFSET_6B;
         offset7<=OFFSET_7B;

   elsif (SWITCH="10") then
         acc_phase<=phase_inC
         offset0<=OFFSET_0C;
         offset1<=OFFSET_1C;
         offset2<=OFFSET_2C;
         offset3<=OFFSET_3C;
         offset4<=OFFSET_4C;
         offset5<=OFFSET_5C;
         offset6<=OFFSET_6C;
         offset7<=OFFSET_7C;
   else
         acc_phase<=phase_inD;
         offset0<=OFFSET_0D;
         offset1<=OFFSET_1D;
         offset2<=OFFSET_2D;
         offset3<=OFFSET_3D;
         offset4<=OFFSET_4D;
         offset5<=OFFSET_5D;
         offset6<=OFFSET_6D;
         offset7<=OFFSET_7D;
   end if;
end if;
```

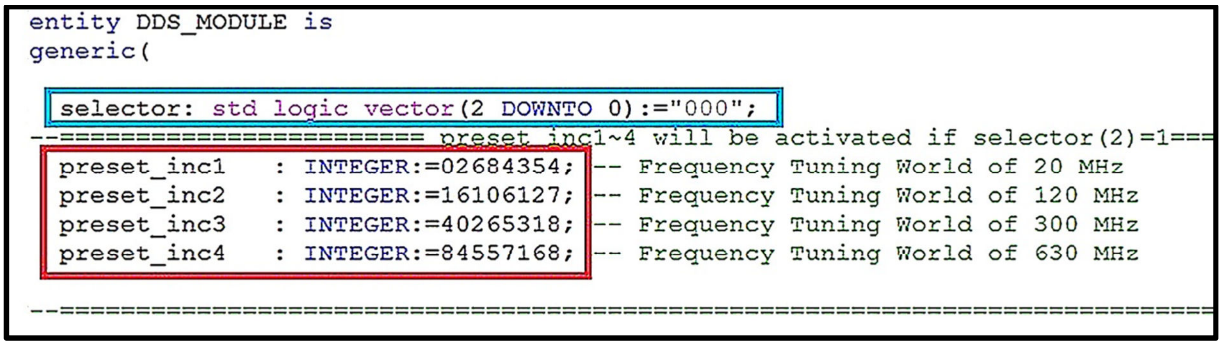
Figure 3.56: VHDL coding of signal switching routine.

In Figure 3.56, the 25-bit variables 'acc_phase' and 'offset0' to 'offset7' are similar to the variables, which have been explained previously in the 8-core DDS module. The variable 'phase_inA' to 'phase_D' are the accumulated phases of four phase accumulators. 'OFFSET_0A' to 'OFFSET_0D', 'OFFSET_1A' to 'OFFSET_1D', 'OFFSET_2A' to 'OFFSET_2D' and 'OFFSET_3A' to 'OFFSET_3D' are the phase offsets of four output signals. Depending on the logic status of 'SWICTH', the accumulated phases will be transferred to 'acc_phase' and their relevant phase offsets will be transferred to 'offset [0:7]'. Consequently, the samples of the selected signal would be generated and converted to an analog signal in DAC.

Selector	**SWITCH**	**Tuning Word**
0**00**	**00**	preset_inc1
0**01**	**01**	preset_inc2
0**10**	**10**	preset_inc3
0**11**	**11**	preset_inc4
1**XX**	XX	Signal generation by Distributed Memory block

Table 3.12: Logic status of SWITCH.

Table 3.12 shows the combinational logic of SWITCH. 'preset_inc1' to 'preset_in4' are four variables, which have the predefined tuning words of four output signals. The variable 'Selector' is used to select the signal synthesis method either by the DDS module or by the distributed memory block. The logic status of 'SWITCH' is retrieved from the 2-LSB of 'Select'. If the MSB of 'Selector' is set to '1', no matter the logics of bit-1 and bit-0 of 'Selector', a signal synthesis by using distributed memory block is triggered. The benefit of using memory block is that the output signal can be fully customised by users. It can be used to generate many forms of signal such as random noise, square-waves, sinc functions and other signals easily. Figure 3.57 shows the VHDL declaration of the variable 'selector' (highlighted in blue) and four sets of tuning words (highlighted in red).



```
entity DDS_MODULE is
generic(

 selector: std logic vector(2 DOWNTO 0):="000";
--======================= preset_inc1~4 will be activated if selector(2)=1===
 preset_inc1   : INTEGER:=02684354; -- Frequency Tuning World of 20 MHz
 preset_inc2   : INTEGER:=16106127; -- Frequency Tuning World of 120 MHz
 preset_inc3   : INTEGER:=40265318; -- Frequency Tuning World of 300 MHz
 preset_inc4   : INTEGER:=84557168; -- Frequency Tuning World of 630 MHz

--=========================================================================
```

Figure 3.57: Predefined tuning word.

The multi-frequency RF generator will be tested on an evaluation board. The test results will be further discussed in Chapter 4.

3.5.7 Development of a Custom Data Distributor by Using Distributed Memory Block

The specifications of a custom data distributor:

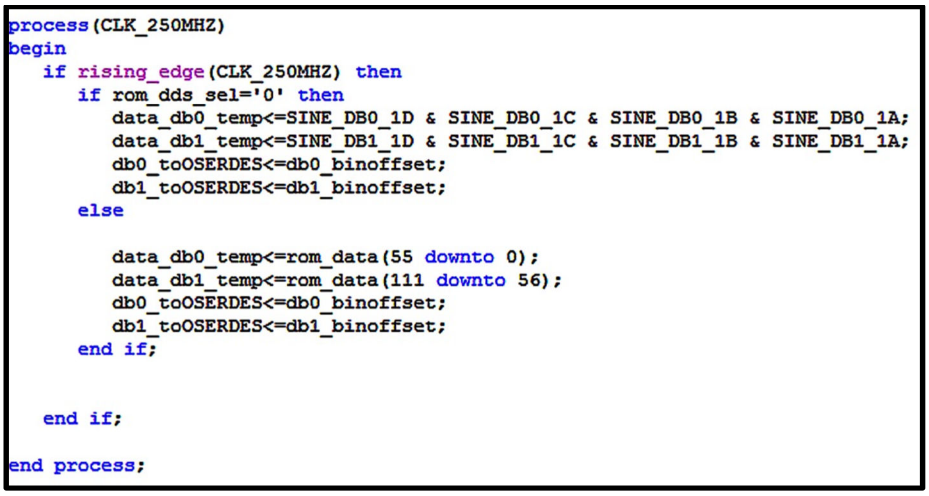
System Clock, f_{system}	: 250 MHz
Depth	: variable (16 to 65536)
Data Width	: 112
Radix	: 2, 10, 16

An addressable distributed memory block is very useful in generating a customised signal which is 100% DDS free. All data from the memory block can be fully defined by the users. Signals such as square-waves, pseudo noise, sinc functions and many more can be generated by using this method. These signals are achievable by loading the relevant coefficient file of the signal into the memory block. The memory block has data depth and data width ranges from 16 to 65536 bits and 1 to 1024 bits, respectively. These ratings are the upper limits for a distributed memory block to function properly.

In this thesis, the data width is fixed at 112-bit so that sample amplitudes can match with the OSERDES setting of an 8-core DDS. 112-bit means eight sets of 14-bit data would be transmitted on every 250 MHz clock so that a 2 GHz sampling period can be achieved. However, the depth and radix are variables. Radix refers to the base of a number system (binary, decimal or hexadecimal) which is used to represent the custom data in a coefficient file. The depth setting depends on the total entries in the coefficient file.

3.5.7.1 *Implementation of Addressable Distributed Memory Block in VHDL*

Figure 3.58 shows the VHDL routine for switching the signal synthesis method by either a multi-frequency RF generator or a distributed memory block.

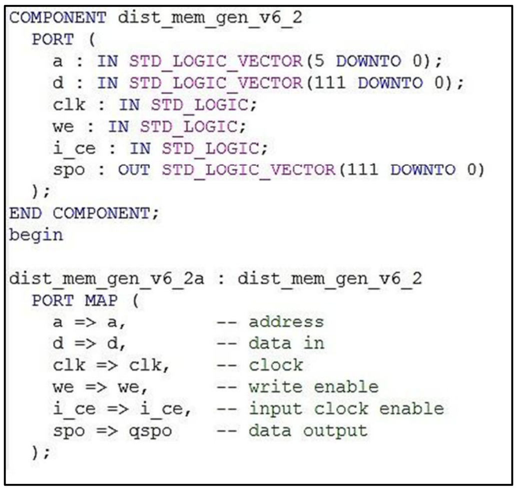


```
process(CLK_250MHZ)
begin
   if rising_edge(CLK_250MHZ) then
      if rom_dds_sel='0' then
         data_db0_temp<=SINE_DB0_1D & SINE_DB0_1C & SINE_DB0_1B & SINE_DB0_1A;
         data_db1_temp<=SINE_DB1_1D & SINE_DB1_1C & SINE_DB1_1B & SINE_DB1_1A;
         db0_toOSERDES<=db0_binoffset;
         db1_toOSERDES<=db1_binoffset;
      else

         data_db0_temp<=rom_data(55 downto 0);
         data_db1_temp<=rom_data(111 downto 56);
         db0_toOSERDES<=db0_binoffset;
         db1_toOSERDES<=db1_binoffset;
      end if;


   end if;

end process;
```

Figure 3.58: Switching between an 8-core DDS module and a distributed memory block.

The signal synthesis method depends on the logic level of ‘rom_dds_sel’. The logic level of ‘rom_dds_sel’ is retrieved from the MSB of ‘Selector’ (Table 3.12). If ‘rom_dds_sel’ is high, samples from the distributed memory clock are sent to the OSERDES module. Otherwise, the samples from the 8-core DDS module are sent to the OSERDES module.

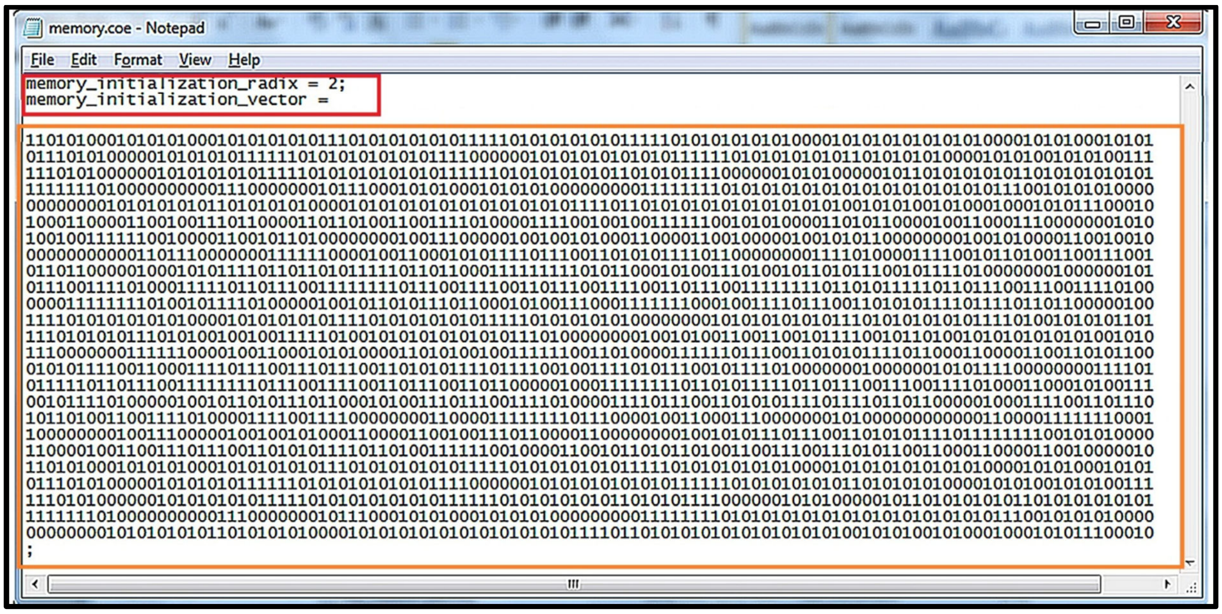
Samples from ‘data_db0’ and ‘data_b1’ undergo offset binary conversion. The results after conversion would be placed in ‘db0_binoffset’ and ‘db1_binoffset’. The samples in‘db0_binoffset’ and ‘db1_binoffset’ are transferred to the OSERDES module through ‘db0_to OSERDES’ and ‘db1_OSERDES’ on every rising edge of 250 MHz.



```
COMPONENT dist_mem_gen_v6_2
  PORT (
    a : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
    d : IN STD_LOGIC_VECTOR(111 DOWNTO 0);
    clk : IN STD_LOGIC;
    we : IN STD_LOGIC;
    i_ce : IN STD_LOGIC;
    spo : OUT STD_LOGIC_VECTOR(111 DOWNTO 0)
  );
END COMPONENT;
begin

dist_mem_gen_v6_2a : dist_mem_gen_v6_2
  PORT MAP (
    a => a,          -- address
    d => d,          -- data in
    clk => clk,      -- clock
    we => we,        -- write enable
    i_ce => i_ce,    -- input clock enable
    spo => qspo      -- data output
  );
```

Figure 3.59: Component declaration of distributed memory block.

In this research, the distributed memory block does not require any 'write' cycle to the memory block. It simply needs to read the customised samples of a signal from the memory block. In Figure 3.59, the read process can be executed by setting the 'Input Clock Enable' (i_ce) to be high. Based on the input address 'a', the contents of the memory block are retrievable via 'gspo <111:0>' lines. To store the custom data in the memory block, a coefficient file is necessary. The coefficient file is created by using a notepad from Windows. The headers (highlighted in red) and the desired output data (highlighted in orange), as shown in Figure 3.60, are typed into the notepad manually.



```
memory_initialization_radix = 2;
memory_initialization_vector =

1101010001010101000101010101011101010101010101111101010101010111110101010101010000101010101010100001010100010101
0111010100000101010101111110101010101010111100000010101010101010111111010101010101101010101000010101001010100111
1110101000000101010101011111010101010101011111101010101010110101011110000001010100000101101010101011010101010101
1111111010000000000011100000001011100010101000101010000000000111111101010101010101010101010101011100101010100000
0000000010101010101101010101000010101010101010101010111101101010101010101010100101010010100010001010111000010
1000110000110010011101100001101101001100111010000111001001111110010101000011010110000100110001110000001010
1001001111110010000110010110100000000100111000001001001010001100001100100000100101011000000001001010000110010010
0000000000110111000000011111100001001100010101111011100110101011110110000000011110100001111001011010011001110001
0110110000010001010111101101101011111011011000111111111010110001010011101001011010111001011110100000000100000101
0111001111010001111101101110011111111011100111100110111001111001101110011111111101101011111011011100111001111010 0
0000111111110100101111010000010010110101110110001010011100011111110001001111011100110101011110111101101100000100
1111010101010101000010101010101111010101010101111101010101010000000010101010101011101010101010111110100101010101
1110101010111010100100100111110100101010101010101110100000000100101001100110010111100101101001010101010101001010
1110000000111111000010011000101010000110101001001110110100001111110011010000111111010011100101111010000000000101101100
0101011110011000111101110011101110011010101111011110010011110101110010111101000000010000000101011110000000001111
0111110110111001111111011100111100111011100110100000100011111111011010111011011100111001110011110100011000100111
0010111101000001001011010111011000101001110111001111010000111101110010101011110111101101100000100011110011011101
1011010011001111010000111100111100000000110000111111111011100001001100011100000001010000000000001100001111111110001
1000000001001110000010010010100011000011001001110110000110000000010010101101111001011010110100100111111111100101010000
1100001001100111011100110101011110110100111111001000011001011010110100110011100111010110011000110000110010000010
1101010001010101000101010101011101010101010101111101010101010111110101010101010000101010101010100001010100010101
0111010100000101010101111110101010101010111100000010101010101010111111010101010101101010101000010101001010100111
1110101000000101010101011111010101010101011111101010101010110101011110000001010100000101101010101011010101010101
1111111010000000000011100000001011100010101000101010000000000111111101010101010101010101010101011100101010100000
0000000010101010101101010101000010101010101010101010111101101010101010101010100101010010100010001010111000010
;
```

Figure 3.60: Coefficient file.

Figure 3.60 shows the coefficient file loaded into the distributed memory block. The ‘memory_initialisation_radix’ (highlighted in red) configures the number system in the coefficient file. The number system can be either 2 (binary), 10 (decimal) or 16 (hexadecimal) which is used to symbolise the number system of the custom data. In this thesis, binary number system has been selected.

The custom data must be written in such a manner, as the inputs of the DAC chip are deinterleaved. The dual data paths to the DAC chip are DB0 and DB1. The data on DB0 are the odd ordered samples while those in DB1 are the even ordered samples. This can be shown by the expression below:

The first eight 14-bit samples of a custom signal (in sequence) are:

[…, sample 7, sample 6, sample 5, sample 4, sample 3, sample 2, sample 1, sample 0]

In the coefficient file, the data of each row must be written in the sequence as below:

[sample 7, sample 5, sample 3, sample 1, sample 6, sample 4, sample 2, sample 0]

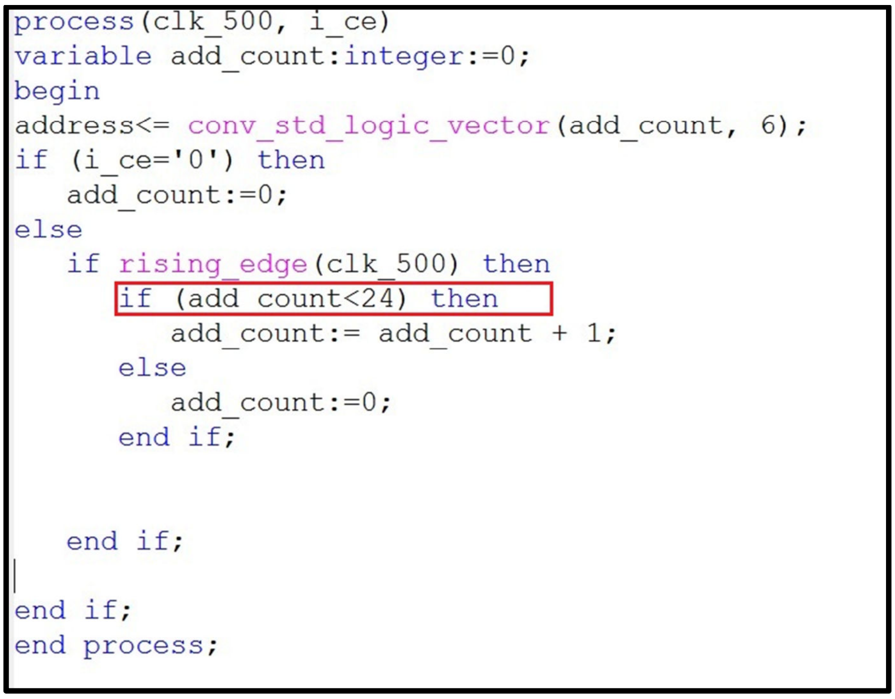
When these samples are retrieved by a VHDL routine, the routine splits ‘sample 7’ until ‘sample 0’ into two data sets. The OSERDES will transmit these two data sets to DAC by the following method:

DB0: [sample 6, sample 4, sample 2, sample 0]

DB1: [sample 7, sample 5, sample 3, sample 1]

In Figure 3.60, the custom data has 25-rows, and each row contains 112-bit (eight sets of 14-bit data). The content in the coefficient file is retrievable on every rising edge of a 250 MHz clock. The content can be read by inserting the relevant addresses to the memory block via 'a<5:0>' lines.

Once the 112-bit data have been retrieved, it is split into two data sets and shifted to OSERDES. From the OSERDES module, two sets of 14-bit data will be transmitted from the Spartan-6 on every rising edge and falling edge of a 500 MHz clock (DDR mode).



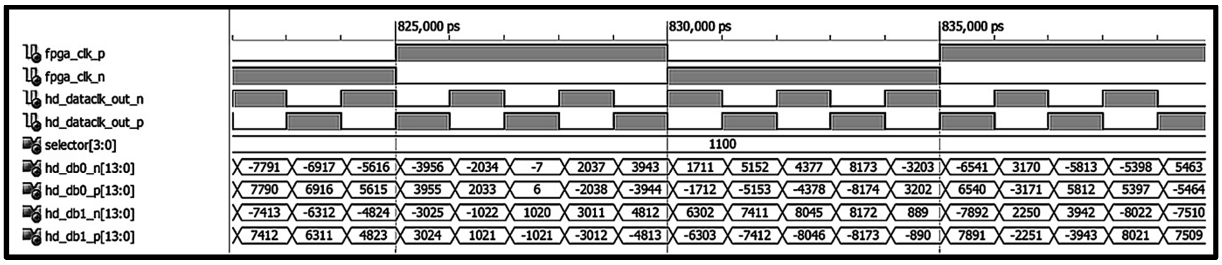
```
process(clk_500, i_ce)
variable add_count:integer:=0;
begin
address<= conv_std_logic_vector(add_count, 6);
if (i_ce='0') then
   add_count:=0;
else
   if rising_edge(clk_500) then
      if (add_count<24) then
         add_count:= add_count + 1;
      else
         add_count:=0;
      end if;


   end if;

end if;
end process;
```

Figure 3.61: VHDL coding to retrieve the content of a coefficient file.

In Figure 3.61, the VHDL program serves to retrieve the content of a distributed memory block. The <If-statement> (highlighted in red) shows that a counter 'add_count' is used to generate memory addresses for accessing into the memory block. The upper limit of the 'add_count' is calculated by <total row -1> of the custom data. The illustration in Figure 3.60 has 25 rows, so the ‘add_count’ counts to 24.

3.5.7.2 Simulation Results

In this simulation, binary numbers are inserted into the coefficient file randomly. Figure 3.62 shows the ISim simulation results of a distributed memory block. The results are recorded in Table 3.13. By comparing the amplitudes in Table 3.13 with the content of the coefficient file (Figure 3.60), it is found that the results of both are the same. Based on the first 140 data points captured by using the test-bench, the data are plotted using Microsoft Excel as shown in Figure 3.63. The data points on the graph can be traced by placing cursor on one of the data points on the graph. Excel will then display the x and y values of that point.



825,000 ps
830,000 ps
835,000 ps
fpga_clk_p
fpga_clk_n
hd_dataclk_out_n
hd_dataclk_out_p
selector[3:0] 1100
hd_db0_n[13:0] -7791 -6917 -5616 -3956 -2034 -7 2037 3943 1711 5152 4377 8173 -3203 -6541 3170 -5813 -5398 5463
hd_db0_p[13:0] 7790 6916 5615 3955 2033 6 -2038 -3944 -1712 -5153 -4378 -8174 3202 6540 -3171 5812 5397 -5464
hd_db1_n[13:0] -7413 -6312 -4824 -3025 -1022 1020 3011 4812 6302 7411 8045 8172 889 -7892 2250 3942 -8022 -7510
hd_db1_p[13:0] 7412 6311 4823 3024 1021 -1021 -3012 -4813 -6303 -7412 -8046 -8173 -890 7891 -2251 -3943 8021 7509

Figure 3.62: Simulation results of distributed memory block using Isim.

Unit	Data	Unit	Data	Unit	Data	Unit	Data	Unit	Data	Unit	Data	Unit	Data
1	-7790	**21**	-5781	**41**	4781	**61**	7381	**81**	7790	**101**	-4378	**121**	4775
2	-8046	**22**	5613	**42**	6101	**62**	-3944	**82**	7412	**102**	-8046	**122**	-4054
3	-8174	**23**	-5037	**43**	-5457	**63**	-3009	**83**	6916	**103**	-8174	**123**	-5499
4	-8173	**24**	6916	**44**	-4267	**64**	-2033	**84**	6311	**104**	-8173	**124**	-5461
5	-8043	**25**	7412	**45**	-5458	**65**	61	**85**	5615	**105**	3202	**125**	5462
6	-7786	**26**	7790	**46**	4266	**66**	-540	**86**	4823	**106**	-890	**126**	5502
7	-7412	**27**	8046	**47**	5122	**67**	175	**87**	3955	**107**	6540	**127**	-5441
8	-6916	**28**	8174	**48**	-683	**68**	-4502	**88**	3024	**108**	7891	**128**	7489
9	4921	**29**	8173	**49**	5450	**69**	-767	**89**	2033	**109**	-3171	**129**	-2731
10	-5156	**30**	8046	**50**	-5400	**70**	3955	**90**	1021	**110**	-2251	**130**	-342
11	971	**31**	7790	**51**	5781	**71**	-654	**91**	6	**111**	5812	**131**	5802
12	-3944	**32**	7412	**52**	-2902	**72**	5606	**92**	-1021	**112**	-3943	**132**	5461
13	61	**33**	6916	**53**	-6562	**73**	6311	**93**	-2038	**113**	5397	**133**	336
14	-2033	**34**	6311	**54**	-5849	**74**	6916	**94**	-3012	**114**	8021	**134**	5471
15	-2693	**35**	5615	**55**	297	**75**	7412	**95**	-3944	**115**	-5464	**135**	-4770
16	6	**36**	4823	**56**	-1362	**76**	7790	**96**	-4813	**116**	7509	**136**	-1407
17	517	**37**	3955	**57**	6572	**77**	8046	**97**	-1712	**117**	5386	137	-6832
18	-7170	**38**	3024	**58**	-6916	**78**	8174	**98**	-6303	**118**	4437	138	1360
19	-6680	**39**	2033	**59**	-1256	**79**	8173	**99**	-5153	**119**	6101	139	-5461
20	-5139	**40**	1021	**60**	-5606	**80**	8046	**100**	-7412	**120**	-2795	140	5909

Table 3.13: First 136 data points captured by the test-bench.

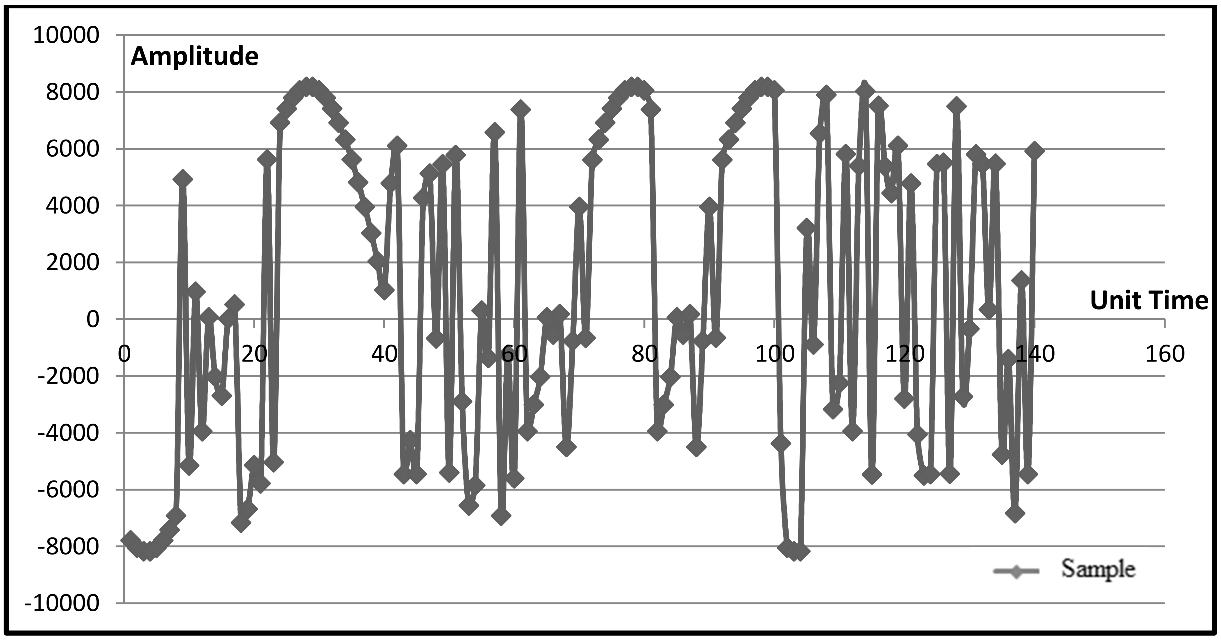


Figure 3.63: Simulated results of a random signal.

Chapter 4

Hardware Implementation, Testing and Results

This chapter discusses the implementation of a high-speed RF transmitter by transferring the design in Chapter 3 to a T-Rex v4.0 board. To accomplish this, the existing VHDL files are modified to activate the on-board clock chip and DAC through serial programming. In this chapter, the programming file, which is used to configure the clock chip and DAC are loaded into the Spartan-6. These two chips are then examined with an embedded, software-based logic analyser called Chipscope. Chipscope is software developed by Xilinx that allows the user to view any internal signal or node, including embedded hard or soft processors of a design system. Signals are captured in the system at the speed of operation and brought out to a computer screen via the Chipscope Pro Analyser tool.

After the clock chip and DAC are found to perform properly, the ability of an 8-core DDS module and multi-frequency RF generator are tested on the prototype board. The phase coherence is tested by switching an output signal from one frequency to another. Finally, a distributed memory block of Spartan-6 is configured to generate a custom signal by loading a coefficient file into the memory block. A Tektronix MSO 4104 Mixed Signal Oscilloscope is used to evaluate the results from all experiments.

4.1 Overview of T-Rex v4.0

Figure 4.0 shows the block diagram of a T-Rex v4.0 board. The components highlighted in red are available on the evaluation board, but they are reserved for future development in an NMR RF receiver, which is not part of the research in this thesis. In contrast, components highlighted in green are the modules, which will be used to implement the NMR RF transmitter.

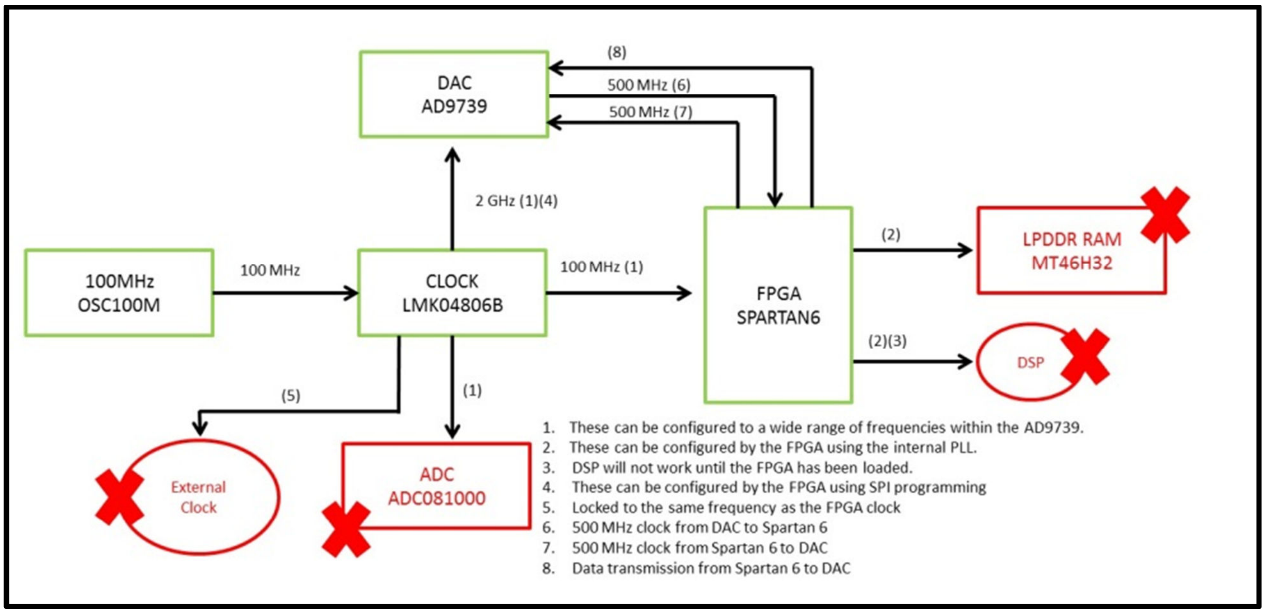

DAC
AD9739
(8)
500 MHz (6)
500 MHz (7)
2 GHz (1)(4)
100MHz
OSC100M
100 MHz
CLOCK
LMK04806B
100 MHz (1)
FPGA
SPARTAN6
(2)
LPDDR RAM
MT46H32
(2)(3)
DSP
(5)
(1)
External
Clock
ADC
ADC081000
1. These can be configured to a wide range of frequencies within the AD9739.
2. These can be configured by the FPGA using the internal PLL.
3. DSP will not work until the FPGA has been loaded.
4. These can be configured by the FPGA using SPI programming
5. Locked to the same frequency as the FPGA clock
6. 500 MHz clock from DAC to Spartan 6
7. 500 MHz clock from Spartan 6 to DAC
8. Data transmission from Spartan 6 to DAC

Figure 4.0: Block diagram of T-Rex v4.0 [46].

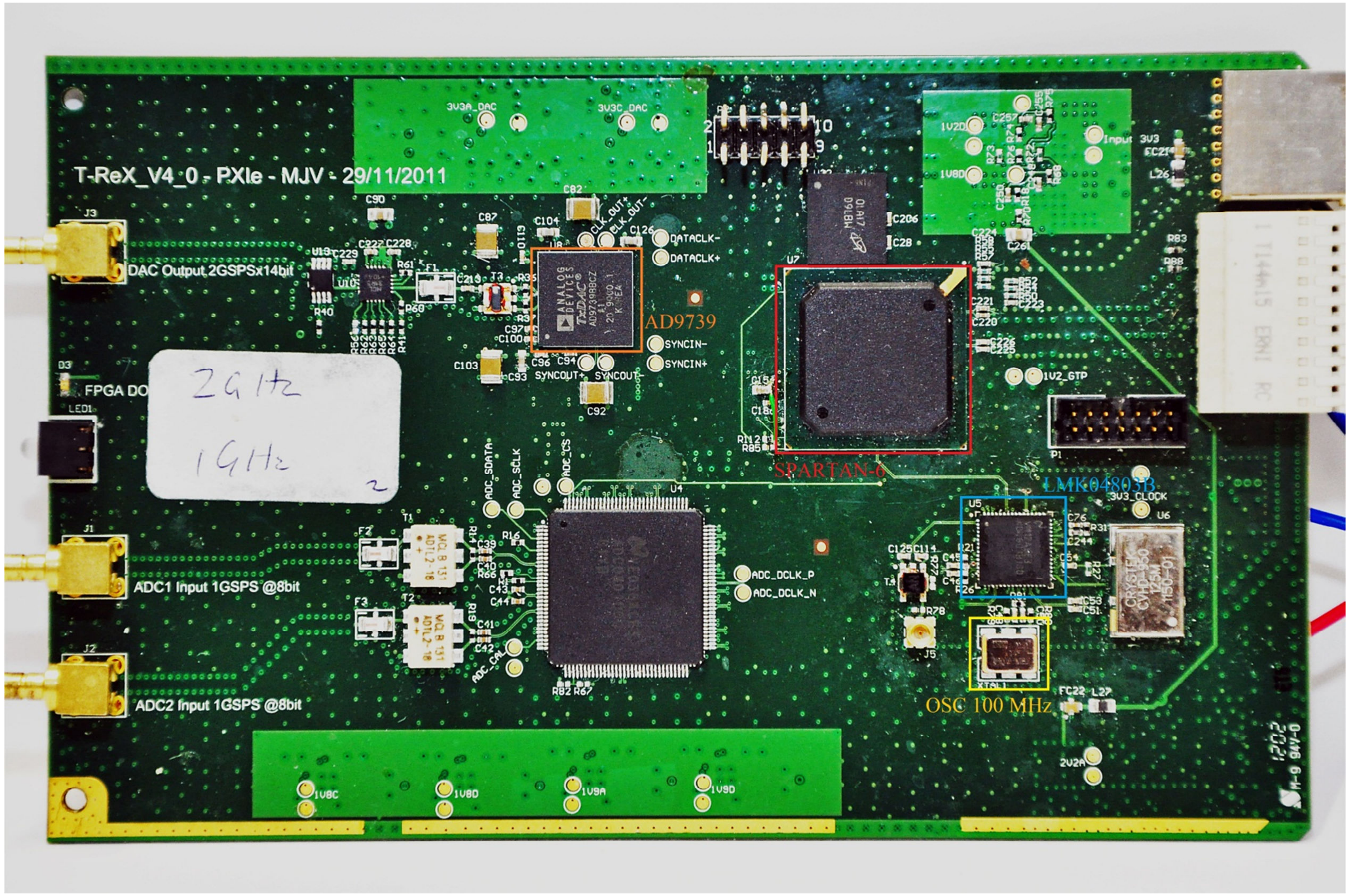


Figure 4.1: Top view of T-Rex v4.0.

Figure 4.1 shows the top view of the T-Rex v4.0 experimental board. The crystal oscillator (highlighted in yellow) supplies a 100 MHz clock signal to the clock chip (highlighted in blue). This clock chip regenerates a 2 GHz clock signal to the DAC chip (highlighted in orange) and a 100 MHz clock signal to the Spartan-6 chip (highlighted in red). Clock chip LMK04806B is manufactured by Texas Instruments. It is a high performance clock conditioner with superior clock jitter cleaning, generation and distribution with advanced features such as dual loop architecture, which consists of two high performance PLLs, a low-noise crystal oscillator circuit, and an effective VCO. The first PLL in the clock chip provides a jitter cleaner while the second PLL performs the clock generation to the system [42].

The 14-bit DAC chip (AD9739) is a dual-port chip operating at 2.5 GSPS. It has an embedded input clock pin (DCI) and an output clock pin (DCO). The DCI clock is a low-voltage differential signal (LVDS) from the Spartan-6 chip for driving data transmission in the DDR format to AD9739. This DCI clock is essentially an alternating 010101...01010 bit pattern with a frequency equal to 1/4 of the DAC system clock. To simplify the synchronisation with the host processor in DAC, the DAC chip passes an LVDS clock signal via its DCO port to the Spartan-6 chip, which is also equal to the DCI frequency [8].

4.2 Serial Programming on DAC and Clock Chips

Programmable registers of both DAC and clock chip must be configured to monitor various internal parameters of both chips. The registers pertaining to similar functions are grouped together and assigned adjacent address. There are some precautions that should be taken while carrying out the serial programming. Firstly, the bits of a register should be assigned '0' if it is undefined. Meanwhile, registers which are unrelated should not be written. Hardware reset is recommended upon power-up to place the registers in a known state. The initialisation is necessary before starting the programming of the chip.

In this project, the LMK04803B clock chip is the first component to undergo a configuration process. It needs to generate two respective clock signals, 2 GHz to DAC and 100 MHz to Spartan-6. 100 MHz will be redistributed as a 10 MHz signal from the FPGA device to drive the SPI configuration of DAC. Both DAC and clock chips have a unique serial programming protocol. The DAC chip uses a standard SPI configuration which consists of three or four SPI ports to program the SPI registers. This method is a common SPI standard which is widely used in most electronic components available in the market. In contrast, the clock chip utilises a Serial MICROWIRE method which consists of three sequential ports to program the chip registers. Details on both configurations are explained in the following section.

4.2.1 Clock Chip Configuration: LMK04803B

The main purpose of the clock chip configuration is to generate a 100 MHz clock to Spartan-6 and a 2 GHz clock to DAC chip.

The LMK04803B chip is programmed using 32-bit registers. Each register consists of a 5-bit (bit-0 to bit-4) address field and a 27-bit data field (bit-5 to bit-31). The contents of each register are clocked in bit-31 being the first, and bit-0 being the last.

Programming ports of LMK04803B are 'DATAuWire', 'CLKuWire', and 'LEuWire' as shown in Figure 4.2. The 'LEuWire' line works as an acknowledge signal to notify the clock chip on the incoming register. The 'DATAuWire' pin of the LMK04803B chip is clocked into a shift register on each rising edge of the 'CLKuWire' signal. A register is sent from the shift register to the register addressed on the positive edge of the 'LEuWire' signal.

During the configuration of the LMK04803B chip, an FPGA device controls the 'LEuWire' signal from high to low. Then, the FPGA device starts to oscillate the 'CLKuWire' signal. The serial data of a register are clocked on the falling edge of the 'CLKuWire' signal so that the clock chip will capture the incoming register in the middle of the data valid window. After the last bit is transmitted, the FPGA device toggles the 'LEuWire' signal from low-to-high-to-low to latch the contents into the register selected by the address field. As recommended by Texas Instruments, the registers should be programmed in numerical order, from R0 to R16, and R24 to R31 so that the device can operate properly.

After the configuration, the VCO in the clock chip should be locked, and the 'DATAuWire', 'CLKuWire' and 'LEuWire' lines will be returned to '0'. If the 'CLKuWire' and 'DATAuWire' signals are toggled while the VCO is locked, in some cases if these lines are shared by other parts, phase noise may be degraded during the programming process [42]. Figure 4.2 and Table 4.0 show the minimum timing required at each programming port.

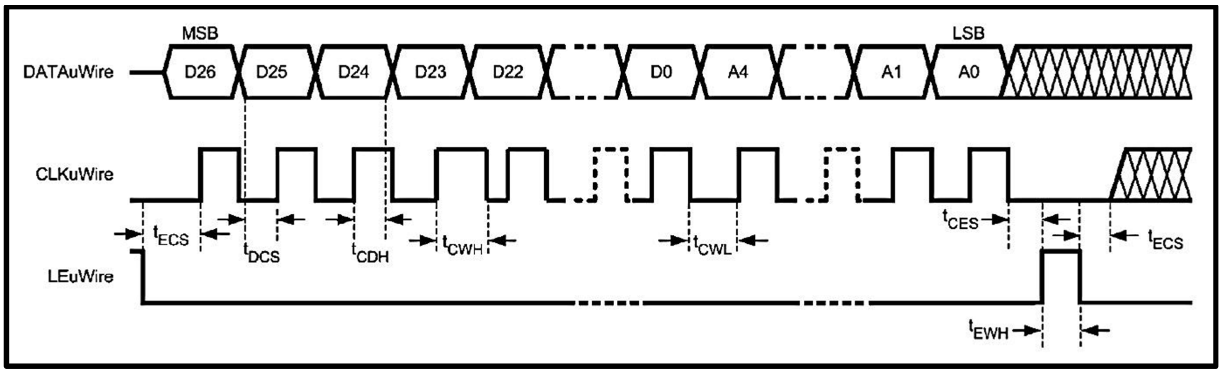
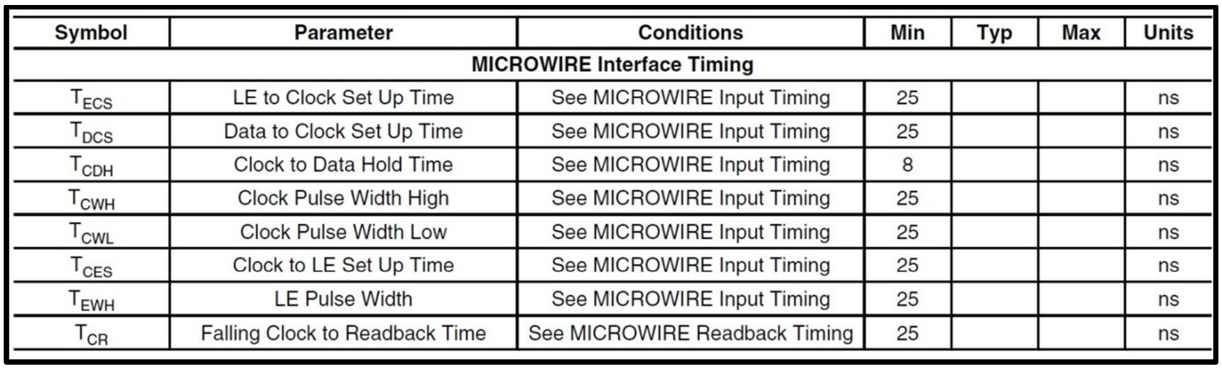


Figure 4.2: MICROWIRE timing diagram [42].

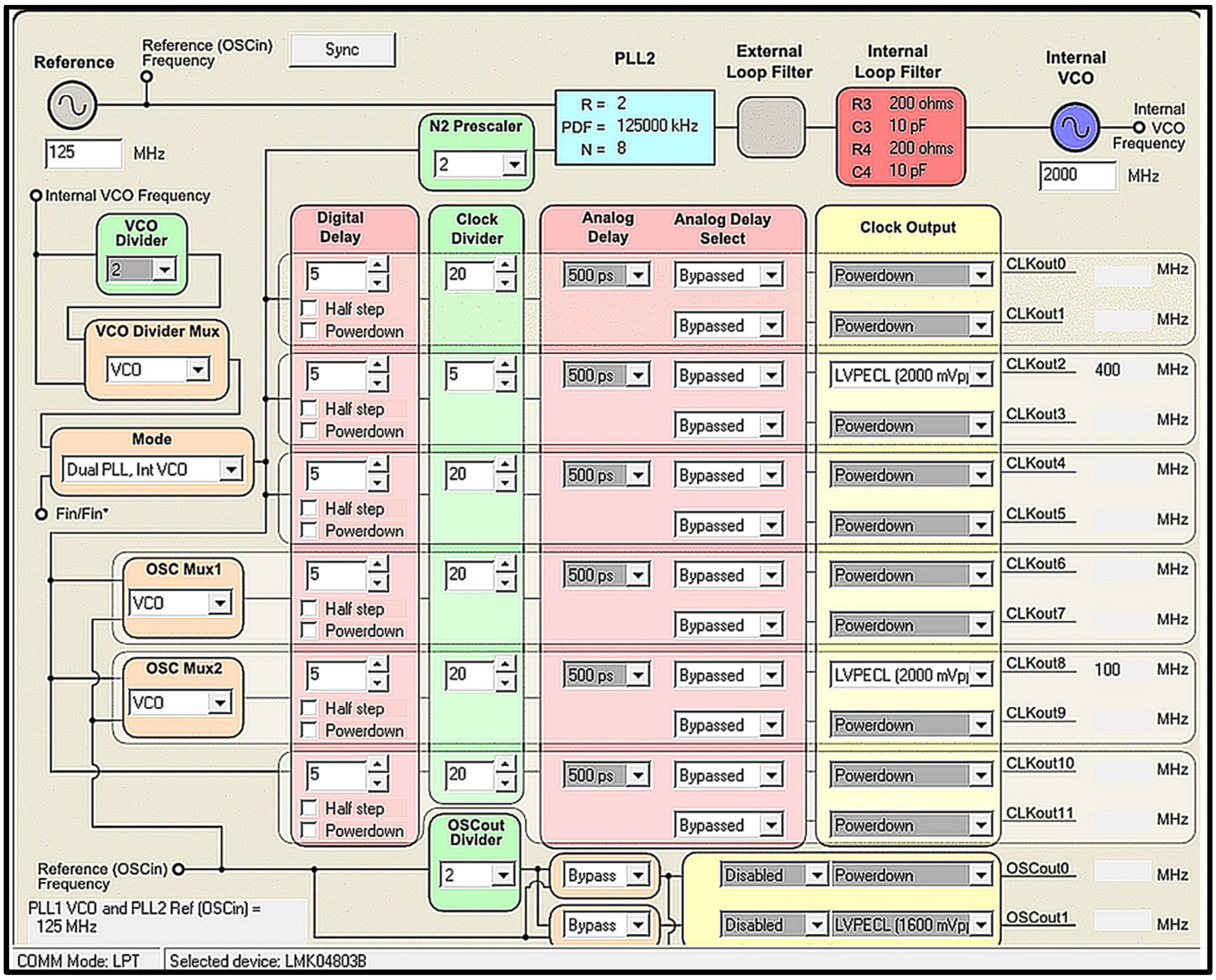


Symbol	Parameter	Conditions	Min	Typ	Max	Units
MICROWIRE Interface Timing						
T_{ECS}	LE to Clock Set Up Time	See MICROWIRE Input Timing	25			ns
T_{DCS}	Data to Clock Set Up Time	See MICROWIRE Input Timing	25			ns
T_{CDH}	Clock to Data Hold Time	See MICROWIRE Input Timing	8			ns
T_{CWH}	Clock Pulse Width High	See MICROWIRE Input Timing	25			ns
T_{CWL}	Clock Pulse Width Low	See MICROWIRE Input Timing	25			ns
T_{CES}	Clock to LE Set Up Time	See MICROWIRE Input Timing	25			ns
T_{EWH}	LE Pulse Width	See MICROWIRE Input Timing	25			ns
T_{CR}	Falling Clock to Readback Time	See MICROWIRE Readback Timing	25			ns

Table 4.0: MICROWIRE timing diagram [42].

4.2.1.1 Development of MICROWIRE for an LMK04803B

Parameter settings of LMK0403B are quite complicated as plenty of registers that must be configured to meet the design requirements. However, the complexity of setting the programmable registers can be avoided by using software ‘CodeLoader 4’. This software is developed by Texas Instruments, which provides an interface for configuring the features of LMK0403B. The main interface of ‘CodeLoader4’can be seen in Figure 4.3.



Reference (OSCin) Frequency: 125 MHz

PLL2: R = 2, PDF = 125000 kHz, N = 8

N2 Prescaler: 2

Internal Loop Filter: R3 200 ohms, C3 10 pF, R4 200 ohms, C4 10 pF

Internal VCO Frequency: 2000 MHz

VCO Divider: 2

VCO Divider Mux: VCO

Mode: Dual PLL, Int VCO

OSC Mux1: VCO

OSC Mux2: VCO

OSCout Divider: 2

Digital Delay	Half step	Powerdown	Clock Divider	Analog Delay	Analog Delay Select	Clock Output	Output	MHz
5	☐	☐	20	500 ps	Bypassed	Powerdown	CLKout0	
					Bypassed	Powerdown	CLKout1	
5	☐	☐	5	500 ps	Bypassed	LVPECL (2000 mVp	CLKout2	400
					Bypassed	Powerdown	CLKout3	
5	☐	☐	20	500 ps	Bypassed	Powerdown	CLKout4	
					Bypassed	Powerdown	CLKout5	
5	☐	☐	20	500 ps	Bypassed	Powerdown	CLKout6	
					Bypassed	Powerdown	CLKout7	
5	☐	☐	20	500 ps	Bypassed	LVPECL (2000 mVp	CLKout8	100
					Bypassed	Powerdown	CLKout9	
5	☐	☐	20	500 ps	Bypassed	Powerdown	CLKout10	
					Bypassed	Powerdown	CLKout11	

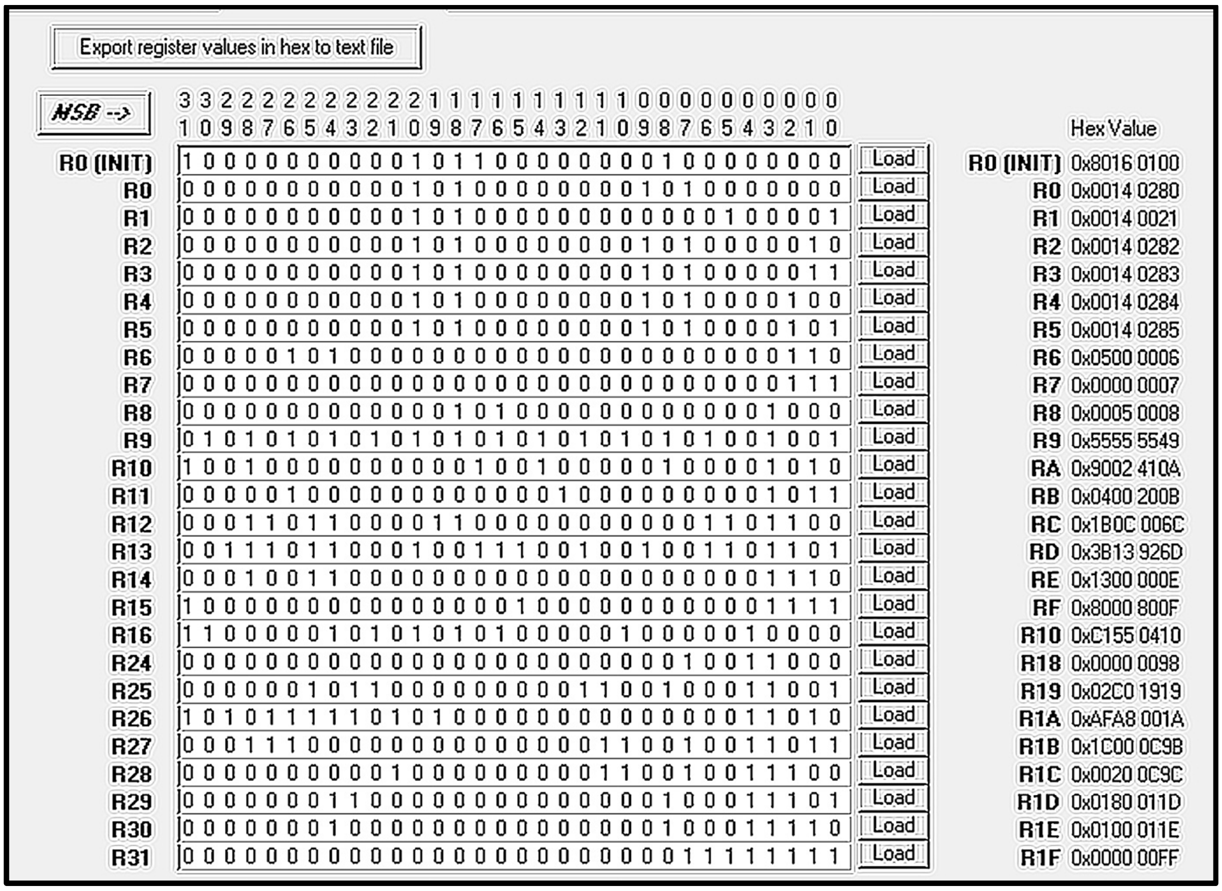
Bypass	Disabled	Powerdown	OSCout0	MHz
Bypass	Disabled	LVPECL (1600 mVp	OSCout1	MHz

PLL1 VCO and PLL2 Ref (OSCin) = 125 MHz

COMM Mode: LPT | Selected device: LMK04803B

Figure 4.3: Interface of CodeLoader4.

Bit selection of the registers can be done simply using 'CodeLoader4'. By configuring each component of the clock chip via the interface as shown in Figure 4.3, a summary of the bit setting is generated as shown in Figure 4.4. Definitions of every bit of each register can be found on the data sheet provided by Texas Instruments [42]. Hence, it will not be further explained in this thesis.



Export register values in hex to text file

MSB -->	3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0			
	1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0			Hex Value
R0 (INIT)	1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	Load	R0 (INIT)	0x8016 0100
R0	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0	Load	R0	0x0014 0280
R1	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	Load	R1	0x0014 0021
R2	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0	Load	R2	0x0014 0282
R3	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1	Load	R3	0x0014 0283
R4	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0	Load	R4	0x0014 0284
R5	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1	Load	R5	0x0014 0285
R6	0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0	Load	R6	0x0500 0006
R7	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1	Load	R7	0x0000 0007
R8	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	Load	R8	0x0005 0008
R9	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 0 1	Load	R9	0x5555 5549
R10	1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0	Load	RA	0x9002 410A
R11	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1	Load	RB	0x0400 200B
R12	0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0	Load	RC	0x1B0C 006C
R13	0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1	Load	RD	0x3B13 926D
R14	0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0	Load	RE	0x1300 000E
R15	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	Load	RF	0x8000 800F
R16	1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0	Load	R10	0xC155 0410
R24	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0	Load	R18	0x0000 0098
R25	0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 1	Load	R19	0x02C0 1919
R26	1 0 1 0 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0	Load	R1A	0xAFA8 001A
R27	0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 1	Load	R1B	0x1C00 0C9B
R28	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 1 0 0	Load	R1C	0x0020 0C9C
R29	0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1	Load	R1D	0x0180 011D
R30	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 0	Load	R1E	0x0100 011E
R31	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	Load	R1F	0x0000 00FF

Figure 4.4: Binary numbers of R0 to R31 generated by CodeLoader4.

Two clock outputs, CLKout2 and CLKout8 are selected because these two clocks are routed to Spartan-6 on T-Rex v4.0. During the experiment on the serial programming of LMK04803B, these CLKout2 and CLKout8 are set to 400 and 100 MHz, respectively. However, 400 MHz will be replaced by 2 GHz during the experiment on signal synthesis. The changing of 400 MHz to 2 GHz can be done by replacing R1 with 0x001400A1 while the rest of the registers remain unchanged. Setting the R1 to 0x001400A1 is equivalent to changing the clock divider of CLKout2 from 5 to 1 through the interface of CodeLoader4.

Table 4.1 lists all of the register settings required to generate a 400 and a 100 MHz clock.

Register	Write Value	Description
R0 (INIT)	0x80160100	Reset the clock chip
R0	0x00140280	Clear the Reset bit at R0, bit-17. Program as necessary to configure the clock outputs, CLKout0 to CLKout11, as desired. These registers configure clock output controls such as power down, digital delay and divider value, analog delay select and clock source select.
R1	0x00140021	
R2	0x00140282	
R3	0x00140283	
R4	0x00140284	
R5	0x00140285	
R6	0x05000006	Program as necessary to configure the clock outputs, CLKout0 to CLKout11, as desired. These registers configure the output format for each clock output and the analog delay for the clock output groups.
R7	0x00000007	
R8	0x00010008	
R9	0x55555549	Program this register as shown in the register map for proper operation.
R10	0x9002410A	For setting the OSCouts, VCO divider, and 0-delay. • Enable and configure clock outputs OSCout0/1. • Set and select a VCO divider (VCO bypass is recommended). • Set 0-delay feedback source if used.
R11	0x0400200B	For setting the Part mode, Synchronise (SYNC), and XTAL. • Program to configure the mode of the part, to configure SYNC functionality and pin, and to enable crystal mode.
R12	0x1BCC006C	For setting the Pins, SYNC, and holdover mode. • Status_LD pin, more SYNC options to generate a SYNC upon PLL1 and/or PLL2 lock detect. • Enable clock features such as a holdover.
R13	0x3B13926D	For setting the Pins, holdover mode, and CLKins. • Status_HOLDOVER, Status_CLKin0, and Status_CLKin1 pin controls. • Enable clock inputs for use in specific part modes.
R14	0x1300000E	For setting the Pins, LOS, CLKins, and DAC. • Status_CLKin1 pin control. • Loss of signal detection, CLKin type, DAC rail detect enable and high and low trip points.
R15	0x8000800F	For setting the DAC and holdover mode. • Program to enable and set the manual DAC value. • HOLDOVER mode options.
R16	0xC1550410	For setting the Crystal amplitude. • Increasing XTAL_LVL can improve tunable crystal phase noise performance.
R24	0x00000098	For setting the PLL1 and PLL2. • PLL1 N and R delay and PLL1 digital lock delay value. • PLL2 integrated LF.
R25	0x02C01919	Program to configure DAC update clock divider and PLL1 digital lock detect count.
R26	0xAFA8001A	Program to configure PLL2 options.
R27	0x1C000A1B	For setting the CLKins and PLL1. • Clock input pre-dividers. • Program to configure PLL1 options.
R28	0x00200C9C	Program to configure PLL2 R and PLL1 N.
R29	0x0180011D	Program to configure the oscillator input frequency, PLL2 fast

		phase detector frequency mode, and PLL2 N calibration value.
R30	0x0200011E	Program to configure PLL2 prescaler and PLL2 N value.
R31	0x000000FF	Program to set the uWire_LOCK bit.

Table 4.1: Register values of 400 MHz and 100 MHz clock signals [42].

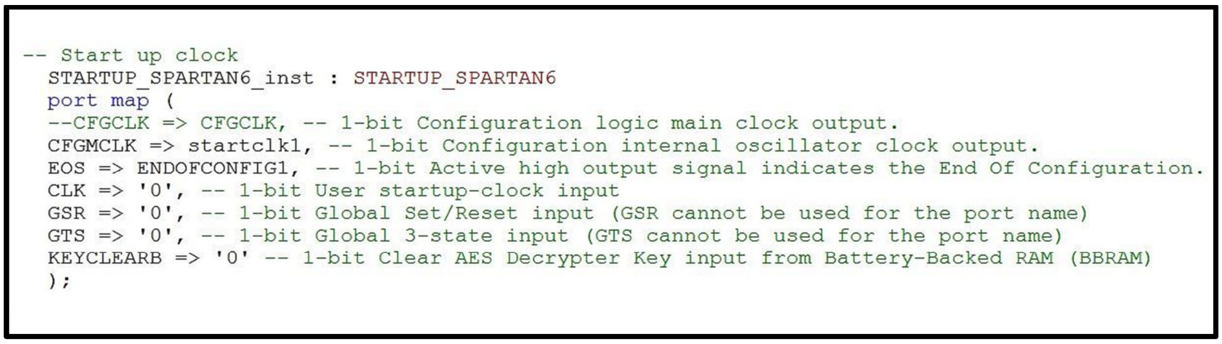
During the serial programming, registers are programmed, with R0 being the first and R31 the last as shown in Table 4.1. Programming register R0 involves initiating the chip to the default state by setting the reset bit (b17) to 1. After that, register R0 is programmed again to clear the reset bit. Then, R0 to R16 and R24 to R32 are programmed in numerical order to the clock chip by using the serial programming method as shown in Figure 4.2.

After all the registers have been programmed, the PLLs should be locked. The clock chip starts to generate the 400 MHz clock on CLKout2 and the 100 MHz clock on CLKout8. These clocks are generated using a VCO frequency of 2000 MHz. 400 MHz and 100 MHz are generated by dividing the VCO frequency by 5 and 20, respectively.

The process of obtaining 400 MHz can be done through the programming R1 [15:5] with ‘00000000101’. This is similar to setting CLKout2_3_DIV = 5. For a 100 MHz clock, the programming is done on R4 [15:5] by inserting ‘00000010100’ to R4 [15:5]. This would set CLKout8_9_DIV=20. The OSCin port must be driven with a valid signal before programming R30 in order to achieve proper frequency calibration. Any changes to the ‘PLL2 R divider’ or the ‘OSCin’ frequency require R30 to be reloaded so that the frequency calibration process can be triggered. Refer to reference [42], page 61 to page 99 for the functions of other registers.

4.2.1.2 Implementation of Serial Programming for an LMK04803B

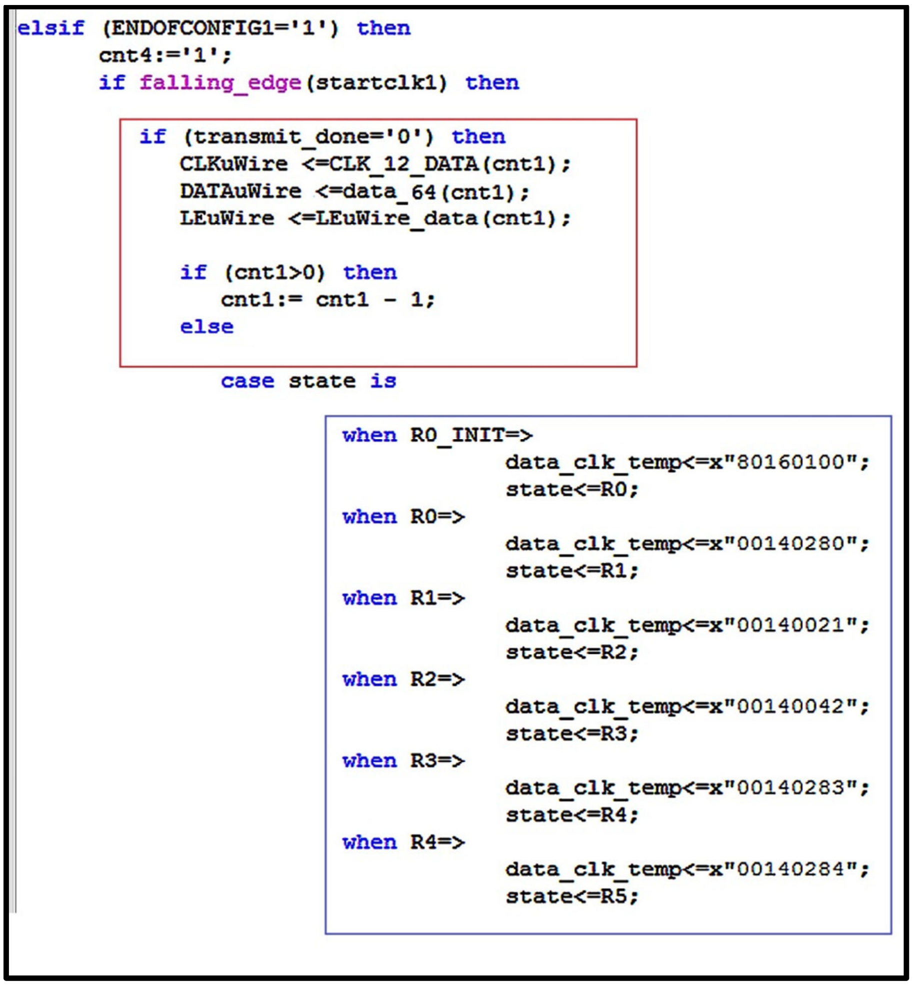
A start-up clock from Spartan-6 must be activated before starting the sequential programming of LMK04803B. The internal oscillator of the Spartan-6 device generates a 50 MHz signal. The 50 MHz clock is stepped down to a lower clock speed to clock the serial programming. Figure 4.5 shows the initialisation of the start-up clock in VHDL.



```
-- Start up clock
  STARTUP_SPARTAN6_inst : STARTUP_SPARTAN6
  port map (
  --CFGCLK => CFGCLK, -- 1-bit Configuration logic main clock output.
  CFGMCLK => startclk1, -- 1-bit Configuration internal oscillator clock output.
  EOS => ENDOFCONFIG1, -- 1-bit Active high output signal indicates the End Of Configuration.
  CLK => '0', -- 1-bit User startup-clock input
  GSR => '0', -- 1-bit Global Set/Reset input (GSR cannot be used for the port name)
  GTS => '0', -- 1-bit Global 3-state input (GTS cannot be used for the port name)
  KEYCLEARB => '0' -- 1-bit Clear AES Decrypter Key input from Battery-Backed RAM (BBRAM)
  );
```

Figure 4.5: Initialisation of a Spartan 6 start-up clock.

In Figure 4.5, 'EOS' indicates the configuration status of the start-up clock in Spartan-6. ‘EOS’ is ‘1’ means that the built-in oscillator has started to generate a 50 MHz internal clock.



```
elsif (ENDOFCONFIG1='1') then
      cnt4:='1';
      if falling_edge(startclk1) then

         if (transmit_done='0') then
            CLKuWire <=CLK_12_DATA(cnt1);
            DATAuWire <=data_64(cnt1);
            LEuWire <=LEuWire_data(cnt1);

            if (cnt1>0) then
               cnt1:= cnt1 - 1;
            else

               case state is

                  when R0_INIT=>
                                 data_clk_temp<=x"80160100";
                                 state<=R0;
                  when R0=>
                                 data_clk_temp<=x"00140280";
                                 state<=R1;
                  when R1=>
                                 data_clk_temp<=x"00140021";
                                 state<=R2;
                  when R2=>
                                 data_clk_temp<=x"00140042";
                                 state<=R3;
                  when R3=>
                                 data_clk_temp<=x"00140283";
                                 state<=R4;
                  when R4=>
                                 data_clk_temp<=x"00140284";
                                 state<=R5;
```

Figure 4.6: VHDL coding of serial programming of LMK04803B.

In Figure 4.6, the VHDL program (highlighted in red) is used to shift the register bits from ‘data_64’ to the clock chip via DATAuWire. A 50 MHz start-up clock is stepped down to 12.5 MHz by sending ‘001100110011...’ bit pattern to the clock chip via the CLKuWire line. After transmitting the 32-bit serial data to the clock chip, the same process is repeated on the next register. ‘cnt1’ is a counter used to synchronise the serial data (DATAuWire) and the serial clock (CLKuWire).

The program highlighted in blue is the example of the state machine, which is used to point a register and place it in a buffer for transmitting to the clock chip. In fact, the VHDL program of the state machine covers R0 to R31. R0_INIT is a reset cycle to drive the clock chip back to its default state.

Serial programming of LMK04803B only requires a ‘write’ process to configure the chip. It does not need any ‘read-back’ operation to read the returned status from the clock chip. The results of the configuration can be verified by measuring the clock signals on the T-Rex v4.0 prototype board using an oscilloscope.

4.2.1.3 Tests and Results

The parameters from Table 4.1 will be programmed into the LMK04803B chip. Register R0 (INIT) resets the hardware and initialises the programmable registers of the clock chip. The purpose is to place all the registers in a known state. A register map table from the data sheet of LMK04803B can be referred while setting the register R0 to R31.

The registers in Table 4.1 managed to generate 400 MHz and 100 MHz clocks on the CLKout2 and CLKout8 outputs. 2 GHz is too fast and cannot be captured by the MSO4104 oscilloscope. Therefore, it was replaced by a 400 MHz clock to test that the serial programming is functioning properly. As required by the DAC chip, the 2 GHz clock will be used in the final stage to run the entire RF system. 400 MHz can be changed to 2 GHz by reconfiguring the register R1 while other registers are kept unchanged.

Figure 4.7 shows the real-time simulation results of registers (R9 and R10) using Chipscope. Chipscope is used to capture the activities of the internal logics of the FPGA device. After loading the programming file into Spartan-6, 32-bit data are transmitted to the clock chip on the falling edge of the CLKuWire line. At the receiving end, it will be processed by the clock chip on the rising edge of the CLKuWire line.

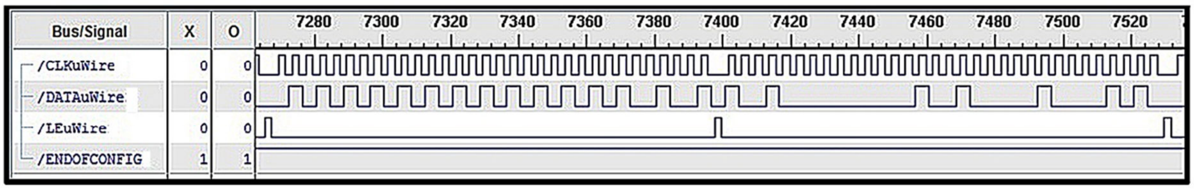


Figure 4.7: Real-time simulation results.

In Figure 4.7, the Chipscope analyser shows that the clock transition of CLKuWire occurs in the middle of the data valid window of DATAuWire. This situation can be simplified by Figure 4.8.

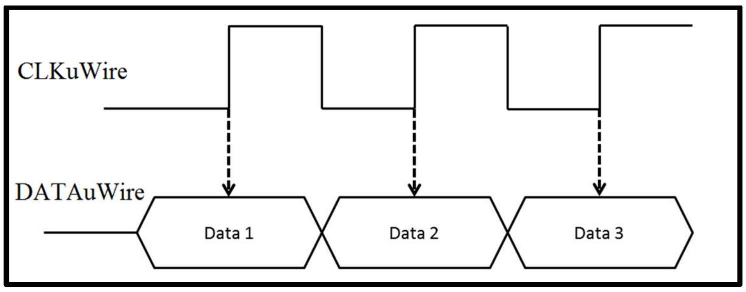
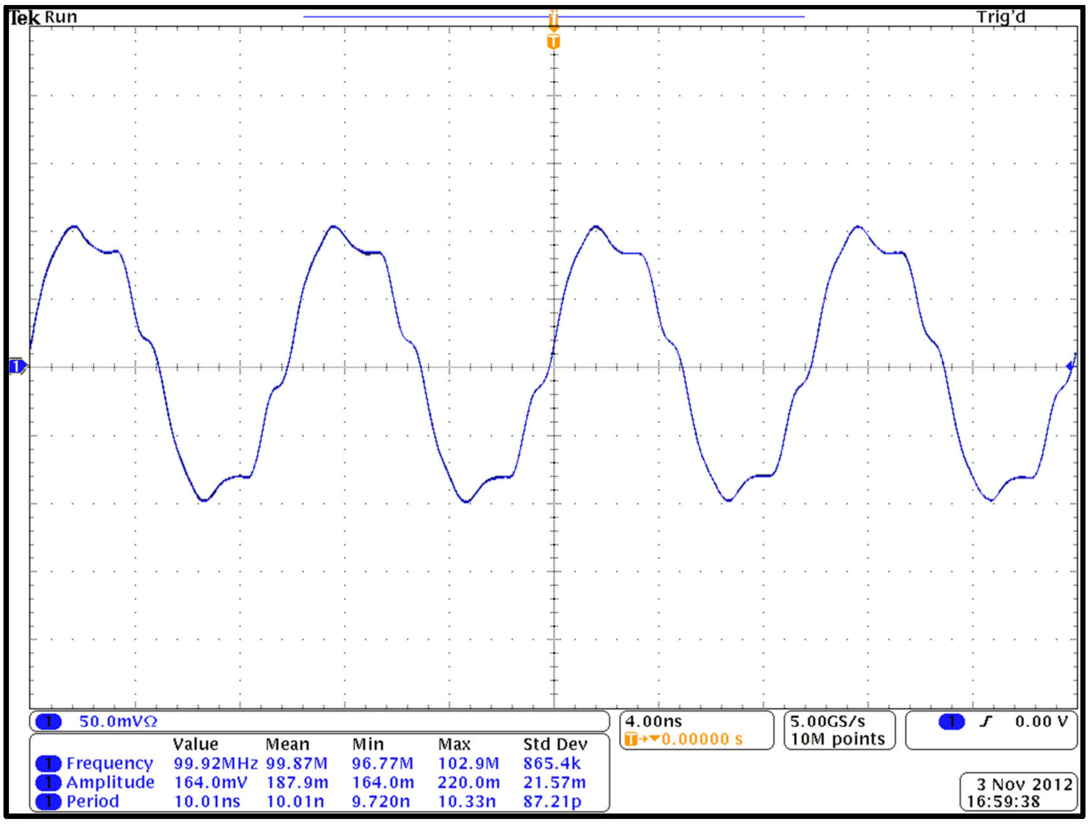


Figure 4.8: Simplified diagram of clock transition on data valid window.

If a 32-bit data is transmitted to the chip on the rising edge of CLKuWire, the clock signal may trigger the chip at the wrong time if there is a slight jitter that disturbs the clock edges. This may lead the chip to process incorrect data.

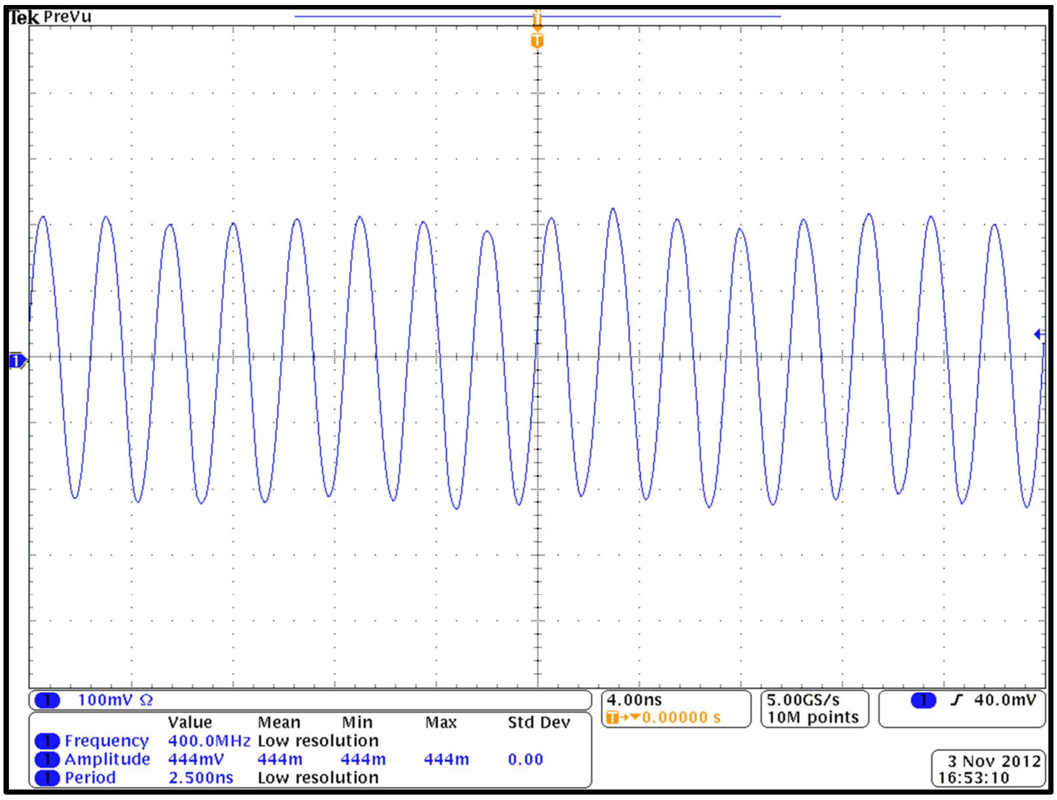
After the transmission of the 32-bit data is over, the LEuWire line toggles again to repeat the transmission process by inserting the data of other registers. Sequential programming is completed after the clock chip generates two clock signals to the DAC chip and Spartan-6.

An oscilloscope is used to measure the generated clock signals from the T-Rex v4.0 board. The results are shown in Figure 4.9 and Figure 4.10.



	Value	Mean	Min	Max	Std Dev
Frequency	99.92MHz	99.87M	96.77M	102.9M	865.4k
Amplitude	164.0mV	187.9m	164.0m	220.0m	21.57m
Period	10.01ns	10.01n	9.720n	10.33n	87.21p

Figure 4.9: 100 MHz clock signal at CLKout8.



	Value	Mean	Min	Max	Std Dev
Frequency	400.0MHz	Low resolution			
Amplitude	444mV	444m	444m	444m	0.00
Period	2.500ns	Low resolution			

Figure 4.10: 400 MHz clock signal at CLKout2.

The output clocks look like sine-waves because the oscilloscope has insufficient bandwidth to include enough harmonic content that sharpens the transition times and flattens the top of the signals. An ideal square-wave is impractical because it requires infinite bandwidth to achieve the transitions between high and low signal levels. In practice, finite bandwidth is the limitation of the oscilloscope to capture a perfect square-wave. The Tektronix MSO 4104 oscilloscope has 1 GHz bandwidth. By comparing bandwidth to the 100 MHz and 400 MHz clock signals, the bandwidth of the oscilloscope might not be large enough to capture adequate harmonics to maintain the 'clean' square-wave.

4.2.2 SPI Configuration for DAC Chip: AD9739

The purpose of the DAC serial configuration is to enable the DAC chip to detect a 2 GHz clock from the clock chip, to generate and to capture a data driven 500 MHz clock from Spartan-6 and to convert two sets of digital data to an analog signal.

An AD9739 chip applies a standard SPI configuration method for programming the SPI registers. The default SPI interface contains four SPI ports such as SDO (Serial Data Output), SDIO (Serial Data Input Output), serial clock (SCLK) and Active Low CE (*CS). The 3- or 4-wire SPI allows read/write access to all the registers that configure the chip internal parameters. An SPI register contains 16-bit data. The first 8-bit indicates the address of a register while the rest of the bits represent the content of the register. The maximum frequency of an SCLK is 20 MHz but only 10 MHz is used because it is a typical clock speed recommended by Analog Devices. The 10 MHz clock signal is regenerated by the built-in PLL of Spartan-6, which is based on the 100 MHz clock signal from the LMK04803B chip.

The SDO pin becomes active when an 8-bit data is ready to be transmitted to the host processor. Otherwise, it remains tri-state all the time. The SPI configuration of the DAC chip provides a flexible, synchronous serial communication port with 3.3 V series I/O. The 3.3 V voltage level is compatible with many industry-standard microcontrollers and microprocessors and with most transfer formats [8].

The SPI configuration of the DAC chip can be implemented in 3- or 4- wire SPI interfaces. If the 3-wire SPI interface is selected, the SDO pin will remain tri-state during the whole configuration process while the SDIO pin becomes bidirectional. This means the SDIO pin reads register bits from the FPGA device during a write operation. The SDIO pin will change to an input pin when the output data from the DAC chip appears on the SDIO pin. If the 4-wire SPI interface is selected, the SDIO pin becomes an input pin to receive 8-bit data from the FPGA device while SDO pin will remain as an output pin to return register bits to the FPGA device.

4.2.2.1 Development of SPI Configuration for an AD9739

The first step in the SPI configuration is to issue a hardware or software reset to the SPI register at 0x00. This step places all the AD9739 SPI registers in a known state. After the reset procedure, FPGA begins to configure the SPI registers by writing relevant bits to the DAC chip.

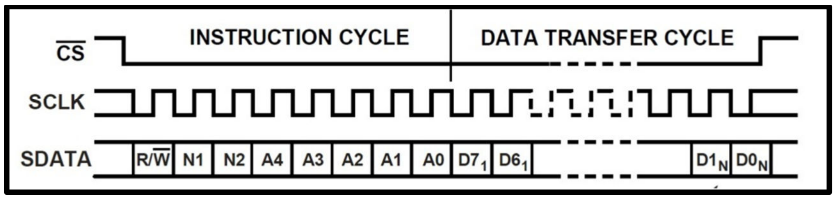


Figure 4.11: SPI timing diagram of AD9739 [8].

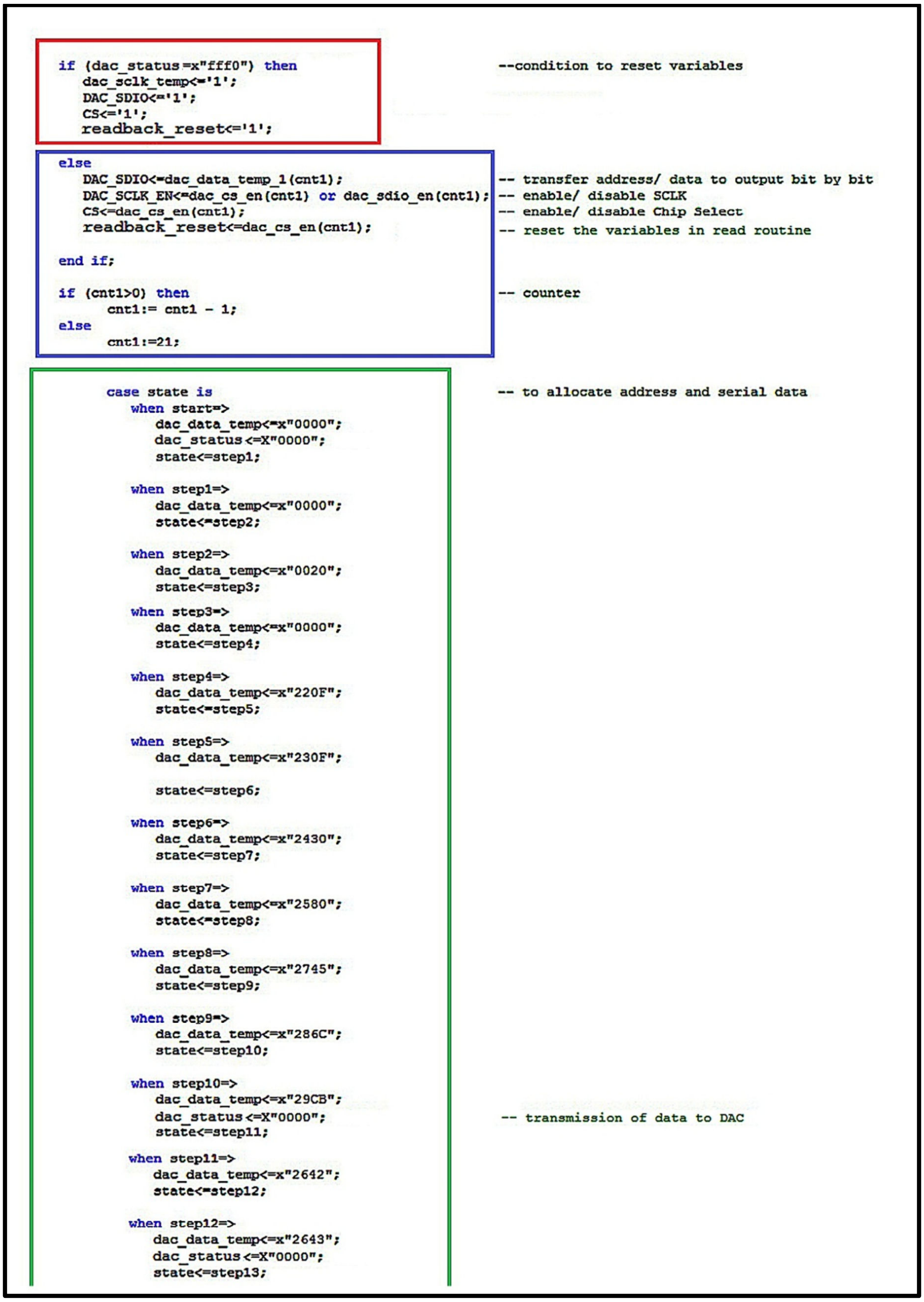
Figure 4.11 shows that two stages are needed to complete a configuration of a register. The first stage is an instruction cycle, which has an MSB bit to represent the indicator bit of 'Read/Write'. The 'Read/Write' bit = '1' indicates that the 7-bit address is sent to the DAC chip to initiate a read-back routine. When 'Read/Write' equals to '0', a specific register in the DAC chip will be programmed (write routine). The '*CS' line goes low to start the configuration of AD9739. The SCLK clock will drive serial data on the SDIO line on every falling edge of the SCLK signal so that the data can be read by the DAC chip on every rising edge of the SCLK clock [8]. This process makes sure that the rising edge of the SCLK clock is located at the middle of the valid data window so that the DAC chip does not capture the wrong information if a minor phase shift exists in the clock signal.

To initiate the write routine, the 'R /*W' bit is set to '0'. After the '*CS' line goes low, the 7-bit address is placed on the SDIO line. The address is then shifted to the DAC chip on the falling edge of the SCLK clock over the next eight clock cycles. During the data transfer cycle, 8-bit data will be transmitted to the DAC chip via SDIO on the falling edge of the SCLK clock. Sending serial data on the falling edge from FPGA to DAC assures that the 8-bit data will be read by the DAC chip on the rising edge of the SCLK clock.

After eight clock cycles, the '*CS' line goes high again after the last bit of the register is processed by the DAC chip. During the write operation, SDO line is maintained tri-state because the read-back procedure is not required at this moment [8].

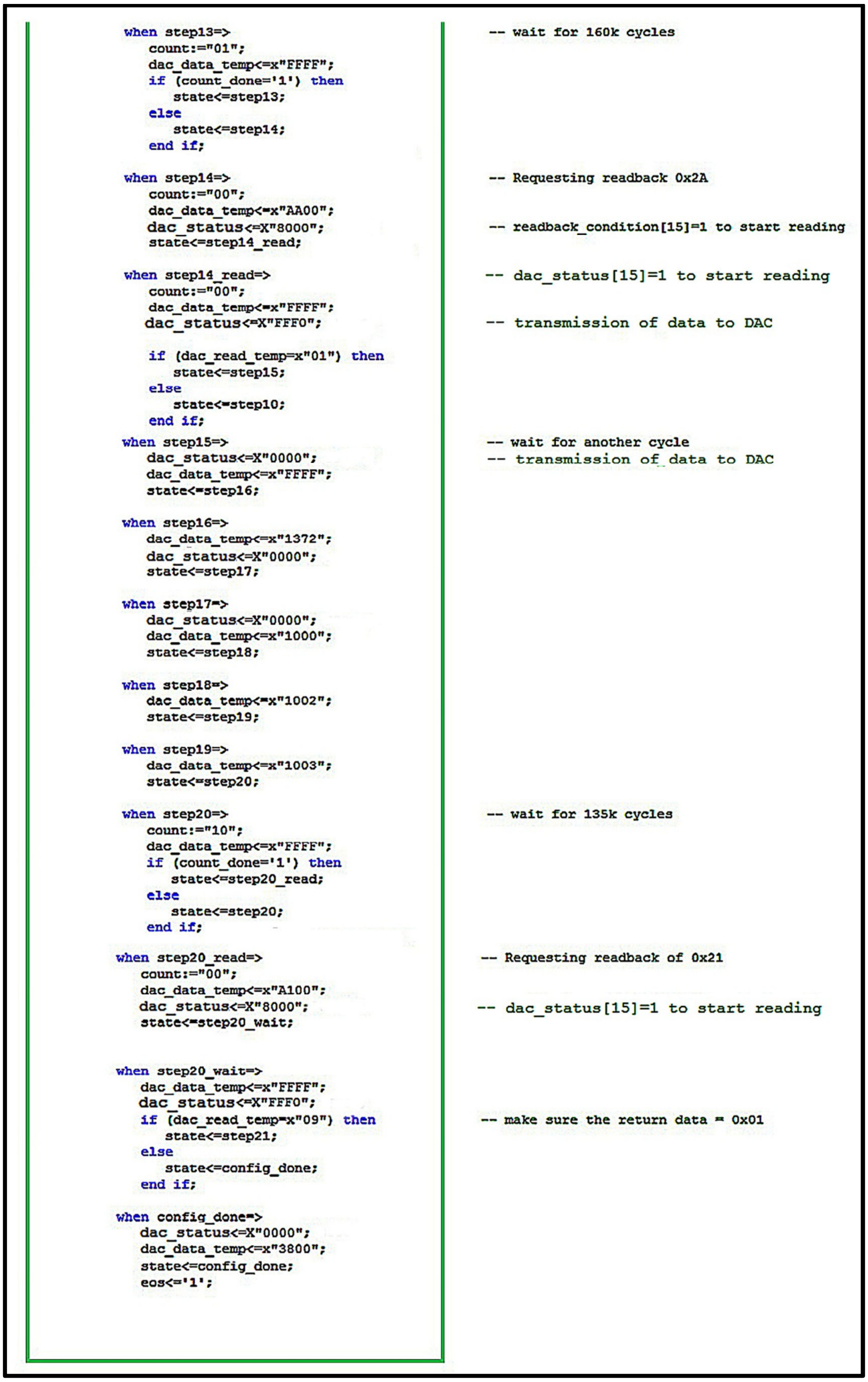
To initiate the read-back routine, the 'R / *W' bit is set to '1'. The '*CS' line goes low to enable the SCLK clock so that the 7-bit address can be shifted to DAC on the falling edge of the SCLK clock. After eight clock cycles, the SDIO pin remains at high impedance throughout the data transfer cycle. At this moment, the SDO pin becomes active, and it returns 8-bit data to the Spartan-6 device on the rising edge of the SCLK clock. The Spartan-6 device processes the 8-bit data on the falling edge of the SCLK clock. The *CS line goes high after the 8-bit data are received by the DAC chip.

4.2.2.2 Implementation of SPI Configuration for an AD9739



```
if (dac_status=x"fff0") then                                --condition to reset variables
   dac_sclk_temp<='1';
   DAC_SDIO<='1';
   CS<='1';
   readback_reset<='1';

else
   DAC_SDIO<=dac_data_temp_1(cnt1);                         -- transfer address/ data to output bit by bit
   DAC_SCLK_EN<=dac_cs_en(cnt1) or dac_sdio_en(cnt1);      -- enable/ disable SCLK
   CS<=dac_cs_en(cnt1);                                     -- enable/ disable Chip Select
   readback_reset<=dac_cs_en(cnt1);                         -- reset the variables in read routine

end if;

if (cnt1>0) then                                            -- counter
     cnt1:= cnt1 - 1;
else
     cnt1:=21;

        case state is                                       -- to allocate address and serial data
           when start=>
              dac_data_temp<=x"0000";
              dac_status<=X"0000";
              state<=step1;

           when step1=>
              dac_data_temp<=x"0000";
              state<=step2;

           when step2=>
              dac_data_temp<=x"0020";
              state<=step3;
           when step3=>
              dac_data_temp<=x"0000";
              state<=step4;

           when step4=>
              dac_data_temp<=x"220F";
              state<=step5;

           when step5=>
              dac_data_temp<=x"230F";

              state<=step6;

           when step6=>
              dac_data_temp<=x"2430";
              state<=step7;

           when step7=>
              dac_data_temp<=x"2580";
              state<=step8;

           when step8=>
              dac_data_temp<=x"2745";
              state<=step9;

           when step9=>
              dac_data_temp<=x"286C";
              state<=step10;

           when step10=>
              dac_data_temp<=x"29CB";
              dac_status<=X"0000";                          -- transmission of data to DAC
              state<=step11;
           when step11=>
              dac_data_temp<=x"2642";
              state<=step12;

           when step12=>
              dac_data_temp<=x"2643";
              dac_status<=X"0000";
              state<=step13;
```

Figure 4.12(A): Write routine of SPI configuration (Not a full VHDL program).



```
    when step13=>                                   -- wait for 160k cycles
       count:="01";
       dac_data_temp<=x"FFFF";
       if (count_done='1') then
          state<=step13;
       else
          state<=step14;
       end if;

    when step14=>                                   -- Requesting readback 0x2A
       count:="00";
       dac_data_temp<=x"AA00";
       dac_status<=X"8000";                         -- readback_condition[15]=1 to start reading
       state<=step14_read;

    when step14_read=>                              -- dac_status[15]=1 to start reading
       count:="00";
       dac_data_temp<=x"FFFF";
       dac_status<=X"FFF0";                         -- transmission of data to DAC

       if (dac_read_temp=x"01") then
          state<=step15;
       else
          state<=step10;
       end if;
    when step15=>                                   -- wait for another cycle
       dac_status<=X"0000";                         -- transmission of data to DAC
       dac_data_temp<=x"FFFF";
       state<=step16;

    when step16=>
       dac_data_temp<=x"1372";
       dac_status<=X"0000";
       state<=step17;

    when step17=>
       dac_status<=X"0000";
       dac_data_temp<=x"1000";
       state<=step18;

    when step18=>
       dac_data_temp<=x"1002";
       state<=step19;

    when step19=>
       dac_data_temp<=x"1003";
       state<=step20;

    when step20=>                                   -- wait for 135k cycles
       count:="10";
       dac_data_temp<=x"FFFF";
       if (count_done='1') then
          state<=step20_read;
       else
          state<=step20;
       end if;

    when step20_read=>                              -- Requesting readback of 0x21
       count:="00";
       dac_data_temp<=x"A100";
       dac_status<=X"8000";                         -- dac_status[15]=1 to start reading
       state<=step20_wait;


    when step20_wait=>
       dac_data_temp<=x"FFFF";
       dac_status<=X"FFF0";
       if (dac_read_temp=x"09") then                -- make sure the return data = 0x01
          state<=step21;
       else
          state<=config_done;
       end if;

    when config_done=>
       dac_status<=X"0000";
       dac_data_temp<=x"3800";
       state<=config_done;
       eos<='1';
```

Figure 4.12 (B): Write routine of SPI configuration (Not a full VHDL program).

Figure 4.12 (A and B) shows the main VHDL routine of SPI configuration of AD9739. In the figures above, the VHDL codes highlighted in green are the state machine of the entire configuration. The state machine transfers the serial data to a buffer for transmitting to the DAC chip via a 'transmit routine' (highlighted in blue). Two variables, ‘dac_sdio_en’ and ‘dac_cs_en’, are used to control the SCLK clock and *CS signals. These two variables contain pre-set values which are used to control SCLK and *CS to start and to end a programming process on a register.

```
dac_sdio_en [21:0]  <= “1110000000000000000111";
dac_cs_en [21:0]     <= “1100000000000000000011”;
```

The pre-set values of these two arrays are retrievable by a counter, cnt1, on every 10 MHz SCLK clock. The content of ‘dac_cs_en’ informs the *CS line to enable or disable the serial programming process. ‘dac_cs_en’, read as ‘0’, enables the *CS line and vice versa. When *CS is valid, the SCLK clock is enabled by the combinational logics of both ‘dac_sdio_en’ and ‘dac_cs_en’. The SCLK line starts to oscillate a clock signal to the DAC chip when ‘dac_sdio_en’ and ‘dsac_cs_en’ are read as ‘0’. Else, the SCLK oscillation will be ceased.

After a register is programmed, the *CS and SCLK lines will remain in ‘high’ for six cycles due to the logic high in ‘dac_sdio_en’ and ‘dac_cs_en’. Then, the serial programming is enabled again to program the next register. The six cycles are used to break down the programming session of each register so that the waveforms of SCLK, SDIO and *CE can be clearly observed on the computer screen.

During the SPI configuration, not all the registers have to be programmed. Some of the registers such as 0x2A and 0x21 request the read-back routine. The purpose of the read-back routine is to acknowledge the Spartan-6 device with the DAC configuration outcomes. Hence, a variable, 'dac_status' is used to switch the programming modes. Three possible modes are assigned to the ‘dac_status’ variable as listed in Table 4.2.

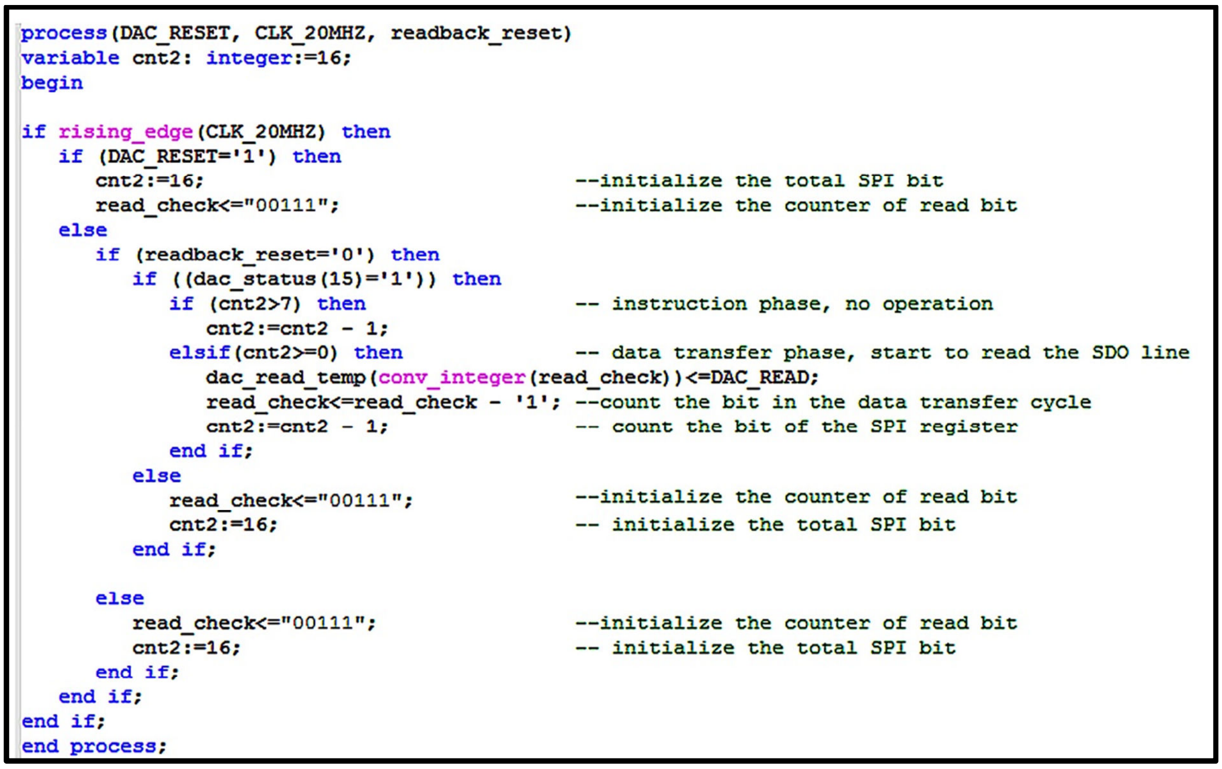
dac_status (hex)	Mode
‘0000’	Initiate transmission of data to the DAC chip
‘8000’	Enable Read-back process
‘fff0’	Reset temporary variables

Table 4.2: Modes in SPI configuration.

'dac_status' is "0000" by default initiates the transmission of serial data to the DAC chip. If the state machine points to Register 0x2A and Register 0x21, the Spartan-6 device requests read-back the configuration status of the DAC chip. After transferring the addresses to these registers during the instruction cycle, the DAC chip informs Spartan-6 on the configuration status by returning an 8-bit data to the Spartan-6 device. If the 8-bit data shows correct answer, the entire SPI configuration is considered successful and the configuration process will be terminated. Otherwise, the DAC chip needs to be reconfigured again. In order to read from the DAC chip, the 'dac_status' variable should be set to "8000" to trigger the 'read routine' so that the related VHDL program will capture the results from the DAC chip. After receiving the read-back data from the DAC chip, the 'dac_status' variable is set to 'fff0' to reset all the temporary variables as highlighted in red in Figure 4.12 (A).

The SPI configuration can be started by initiating the R/*W bit as '0' to the DAC chip. Then, the 7-bit address and the 8-bit data of a specific register is shifted to the DAC chip after R/*W. In 'transmission routine', the 'cnt1' counter counts in descending order in the 'DAC_SDIO' array (Figure 4.12A). The contents of the array will be transmitted to the DAC chip in a bit-by-bit sequence. When 'cnt1' reaches zero, the state machine will allocate data of another register to 'DAC_SDIO'.

Figure 4.13 shows the VHDL program that is developed to read the 8-bit data returned from AD9739 via the SDO pin. In order to execute the read-back process, the R/*W bit is set to 'high' to initiate a read routine. Once the DAC chip received R/*W and the 7-bit address of a register, the 8-bit data of the register will be returned to Spartan-6.



```vhdl
process(DAC_RESET, CLK_20MHZ, readback_reset)
variable cnt2: integer:=16;
begin

if rising_edge(CLK_20MHZ) then
   if (DAC_RESET='1') then
      cnt2:=16;                           --initialize the total SPI bit
      read_check<="00111";                --initialize the counter of read bit
   else
      if (readback_reset='0') then
         if ((dac_status(15)='1')) then
            if (cnt2>7) then              -- instruction phase, no operation
               cnt2:=cnt2 - 1;
            elsif(cnt2>=0) then           -- data transfer phase, start to read the SDO line
               dac_read_temp(conv_integer(read_check))<=DAC_READ;
               read_check<=read_check - '1'; --count the bit in the data transfer cycle
               cnt2:=cnt2 - 1;            -- count the bit of the SPI register
            end if;
         else
            read_check<="00111";          --initialize the counter of read bit
            cnt2:=16;                     -- initialize the total SPI bit
         end if;

      else
         read_check<="00111";             --initialize the counter of read bit
         cnt2:=16;                        -- initialize the total SPI bit
      end if;
   end if;
end if;
end process;
```

Figure 4.13: Read process of SPI configuration.

During the read routine, the 'cnt2' counter will start to count in descending order from 16 to 7. As long as 'cnt2' counts between 16 and 7, the read-back process will be idled because it needs to wait for the address to be transferred to the DAC chip. Once the 'cnt2' counts below 7, the SDO line becomes active. It will start to capture the 8-bit data returned from the DAC chip and store them in 'DAC_READ'. The results from the 'DAC_READ' array will be transferred to the ‘dac_read_temp’ array for examination. If the results do not show correct answers, the DAC chip will be reconfigured. Otherwise, the configuration carries on to the next registers.

All SPI programming is mainly to configure the mu controller and data receiver controller (Rx controller) in the DAC chip. A mu controller is used to adjust the timing relationship between internal digital and analog domains of AD9739 such that the data are successfully transferred into a TxDAC core at rates of up to 2.5

GSPS [8]. The transfer rate is monitored by the Delay Lock Loops (DLL) in the mu controller. The Rx controller is used to search the closest rising edge of the input clock (500 MHz data driven clock from Spartan-6) by simultaneously adjusting the delays in the clock used to register the input clocks and data inputs [8].

For the DAC chip, 0x2A and 0x21 are the only registers that request a read-back process. Based on Analog Devices, these registers should return 0x01 and 0x09 to Spartan-6. Register 0x2A indicates the DLL in a mu controller is configured properly and 0x09 indicates the data receiver controller is enabled.

4.2.2.3 Tests and Results

In Table 4.3, the columns 'Address' and 'Write Value' are the sequential data used to program the DAC chip. The data listed in 'Write Value' are the parameters used to set the SPI registers.

Step	Address (Hex)	Write Value	Comment
1	0x00	0x00	Configure for the 4-wire SPI mode with MSB. Note that Bits [7:5] must be mirrored onto Bits [2:0] because the MSB/LSB format can be unknown at power-up.
2	0x00	0x20	Software reset to default SPI values.
3	0x00	0X00	Clear the reset bit.
4	0x22	0X0F	Set the common-mode voltage of differential DAC Master Clock (DACCLK_P and DACCLK_N) inputs.
5	0x23	0X0F	
6	0x24	0X30	Configure the mu controller. Refer to Table 28 for recommended target Mu slope and phase settings vs. clock rate.
7	0x25	0X80	
8	0x27	0X45	
9	0x28	0X6C	
10	0x29	0XCB	
11	0x26	0X42	
12	0x26	0X43	Enable the mu controller search and track mode.
13	Not applicable	Not applicable	Wait for 160 K × 1/f_{DATA} cycles.
14	0x2A		Read-back Register 0x2A and confirm that it is equal to 0x01 to ensure that the DLL loop is locked. If it is not locked, proceed to step 10 and repeat. Limit attempts to three before breaking out of the loop and reporting a mu lock failure.
15	Not applicable	Not applicable	Ensure that AD9739 is fed with DCI clock input from the data source.
16	0X13	0X72	Set FINE_DEL_SKEW to 2.
17	0X10	0X00	Disable the data Rx controller before enabling it.
18	0X10	0X02	Enable the data Rx controller for loop and IRQ.
19	0X10	0X03	Enable the data Rx controller to search and track mode.
20	Not applicable	Not applicable	Wait for 135 K × 1/f_{DATA} cycles.
21	0X21		Read-back Register 0x21 and confirm that it is equal to 0x09 to ensure that the DLL loop is locked and tracking. If it is not locked and tracked, proceed to step 17 to repeat the attempt. Limit attempts to three before breaking out of the loop and reporting an Rx data lock failure.
22	0X06, 0X07	0X00, 0X02	Optional: modify the TxDAC IOUTFS setting (the default is 20 mA).
23	0X08	0X00	Optional: modify the TxDAC operation mode (the default is normal mode).

Table 4.3: SPI initialisation with SYNC controller disabled [8].

In Table 4.3, step 13 and step 20 are two 'wait cycles’, which must be inserted into the SPI configuration. The total ‘wait cycles’ are recommended by Analog Devices to allow the DAC chip to adjust its internal settings based on the serial data received. After the first wait cycle (at step 13), 0xAA is sent to the DAC chip to initiate a read process on the Register 0x2A. 0xAA is sent instead of 0x2A because the R/*W bit of Register 0x2A is set to 'high'. The byte returned from the DAC chip reveals the status of the mu controller in the DAC chip. If the returning byte equals 0x01, it indicates that DLL in the mu controller is locked. Otherwise, step 10 needs to be repeated to lock the DLL loop.

Step 20 from Table 4.3 is the second 'wait cycle' in the SPI configuration. Once the 'wait cycle' is over, the Register 0x21 would be accessed by placing 0xA1 on the SDIO line. 0xA1 is selected instead of 0x21 because the R/*W bit of Register 0x21 is set to 'high'. If the retuning byte equals 0x09, it indicates that the tracking of the data receiver controller is established and the DLL loop of the controller has been locked and has not been lost. Otherwise, the configuration proceeds to step 17 to repeat the attempt. The attempts will be limited to three before breaking out of the loop and reporting an Rx data lock failure.

Chipscope: output waveform of Write process

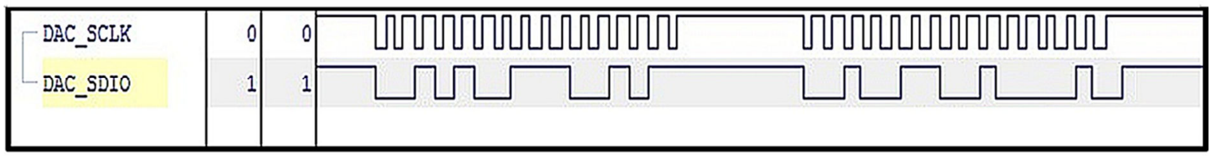


Figure 4.14: Write routine to SPI interface.

Figure 4.14 shows the real-time simulation results captured using Chipscope. A few cycles delays are inserted to break down the transmission of each register so that the waveform of these registers can be seen clearly on the Chipscope analyser. The extra idle cycles between each register do not affect the configuration of the DAC chip.

The results show that 0x29CB and 0x2642 have been written to the AD9739 SPI interface. Serial data of 0x29CB is shifted to the DAC chip via the SDIO pin on every falling edge of the 10 MHz clock. After the instruction cycle, data is received and captured by DAC on every rising edge of the 10 MHz clock. SCLK and SDIO are maintained as 'high' after the last bit is processed. After 6 cycles, the writing process is repeated on the register 0x2642.

<u>Chipscope: output waveform of Read-back procedure</u>

In Figure 4.15, the DAC chip has been accessed by Register 0x2A. The serial data, which are sent over to the DAC chip is 0xAA instead of 0x2A. The reason is that the R/*W bit of the register is set to '1'. It changes 0x2A to 0xAA. The serial transmission begins once the SCLK clock is toggled from '1' to '0'. The transmission of the serial data from Spartan-6 to DAC is registered on the falling edge of the SCLK clock. In DAC, the reception of the data is registered on the rising edge of the SCLK clock as seen in Figure 4.15. After the address is processed by DAC, 8-bit data is returned to the FPGA device on the falling edge of the SCLK clock. In the Spartan-6, the 8-bit data is registered on the rising edge of the SCLK clock.

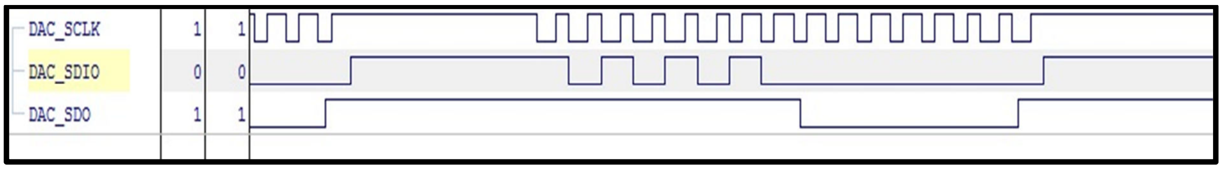


Figure 4.15: Read routine to SPI interface at Register 0x2A.

Based on the 'DAC_SDO' line shown in Figure 4.15, the 8-bit data returned from the DAC chip is 0x01. This indicates that the DLL of the mu controller in DAC is locked. This means a 500 MHz clock ($f_{DAC}/4$ where $f_{DAC} = 2$ GHz) has been generated by the clock distributor in the AD9739 chip. This 500 MHz clock is fed to the Spartan-6 via the DCO pin of AD9739 while another 500 MHz clock is regenerated from the FPGA device by shifting the alternating "101010…" bit pattern to the DCI pin of AD9739. The alternating bit pattern serves to ensure the coincident transitions with the sine-wave data (samples for digital-to-analog conversion). This is because the DCI clock edges must be coincident with the data bit transitions with minimum skew, jitter, and intersymbol interference [8].

After the mu controller is locked, the Rx controller is enabled by programming Register 0x10 in the DAC chip. After 135k wait cycles, the status of the Rx controller is retrievable by accessing Register 0x21 (Figure 4.16). This process could be done by sending 0xA1 to the DAC chip. 0xA1 is sent instead of 0x21 because the R/*W bit of the register is set to '1'. It changes 0x21 to 0xA1.

After 0xA1 is processed by the DAC chip, the status returned by Register 0x21 is 0x09. 0x09 indicates the Rx controller in AD9739 has been successfully locked and tracked.

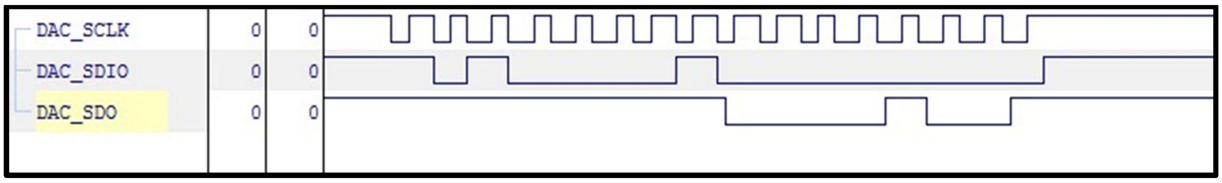
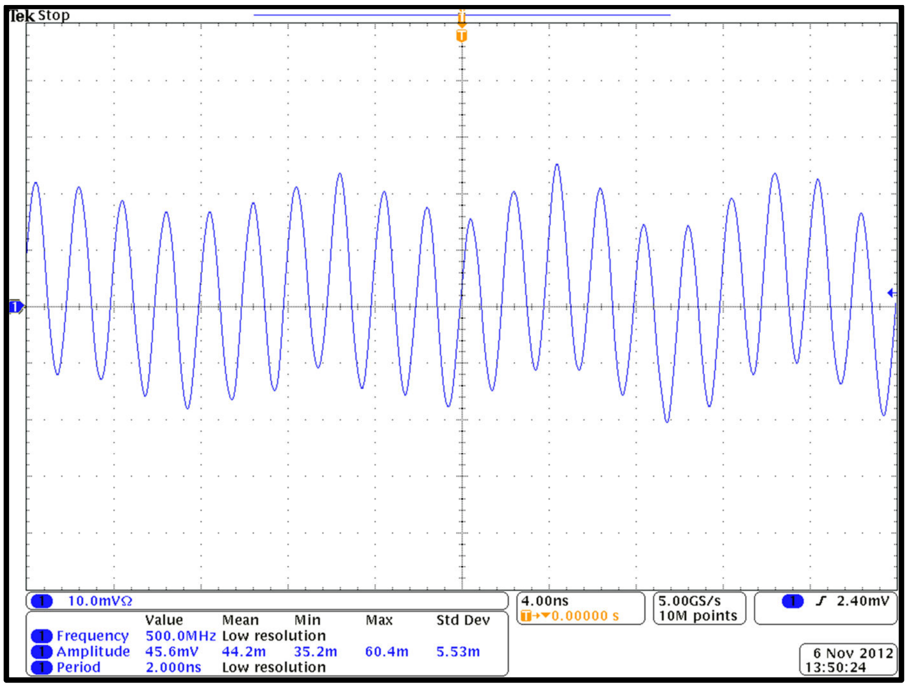


Figure 4.16: Read routine to SPI interface at Register 0x21.

Results measured on T-Rex v4.0 after DAC configuration

Before measuring DCO form AD9739, the clock chip is configured to generate a 2 GHz signal to the DAC chip so that DCO can generate a 500 MHz data driven clock to Spartan-6.

The signal shown in Figure 4.17 is a 500 MHz clock signal measured on the DCO pin of AD9739. An oscilloscope shows that the 500 MHz signal has 45.6 mV amplitude with 2 ns period.



	Value	Mean	Min	Max	Std Dev
1 Frequency	500.0MHz	Low resolution			
1 Amplitude	45.6mV	44.2m	35.2m	60.4m	5.53m
1 Period	2.000ns	Low resolution			

Figure 4.17: 500 MHz clock generated by AD9739 DCO pin to Spartan-6.

The 500 MHz clock is formed by dividing the 2 GHz external clock from the DACCLK pin. In the DAC chip, the 2 GHz signal is locked and tracked in the DAC chip by a mu controller. After this, it is divided by four in a clock distributor to produce 500 MHz. This 500 MHz is then placed on the DCO line and sent back to the FPGA device.

The amplitude jitter of the 500 MHz clock is due to the quad switch DAC architecture which causes switching event in both edges of f_{DAC} (DACCLK, a master clock of AD9739) [8]. An AD9739 chip is built with external clocking mode and it does not contain an internal oscillator to generate a ‘cleaner’ clock signal to Spartan-6. This is because the external clocking mode requires an internal clock distributor to regenerate its internal clocks (500 MHz) from an external clock (2 GHz) [8]. As a result, jitter may couple into the DCO pin [8]. Besides this, there are other factors contributed to jitters such as the internal circuitry of PLL and the nearby resonating devices in the system. Furthermore, the T-Rex v4.0 is a prototype and yet to be tested thoroughly. Undeniably, termination dependency, cross talk, reflection, or Electromagnetic Interference (EMI) from nearby components may also contribute to jitter. An effective way to reduce jitter problems is by having a good circuit design to maintain performance as close to theoretical as possible.

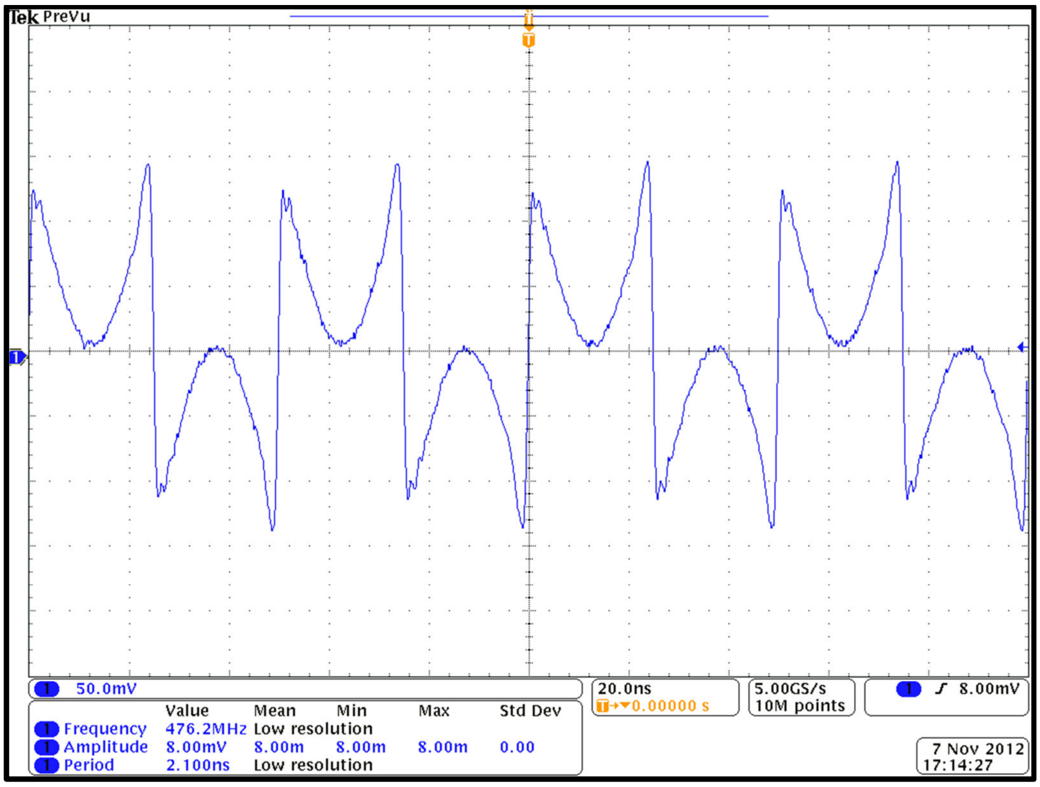
After the serial configuration, the developed 8-core DDS module and multi-frequency RF module are tested on the T-Rex 4.0 evaluation board. The experimental results will be further discussed in section 4.3.

4.3 Hardware Implementation of High Speed RF Transmitter

The specifications of a high-speed RF transmitter are as below:

System Clock, f_{system}	: 250 MHz
Frequency Resolution, DF	: 0.93 Hz
Input Width of LUT	: 25-bit
Output Width of LUT	: 14-bit
Depth of sine LUT	: 16384 samples
Input width of Phase Accumulators	: 25-bit
Depth of Phase Accumulator	: 33554432
Total Phase Offsets	: 8 sets with 25-bit of each

The output from LUTs is in 2's complement. It cannot be applied to the input of the AD9739 chip directly because the DAC chip only recognises input data in offset binary format. Therefore, all the output of a multi-frequency RF generator must undergo data conversion prior to being processed by the DAC. Figure 4.18 shows the results without offset binary conversion and they are measured at the output port of the DAC chip. The waveform shows that the positive and negative cycles of a full sine-wave are upside down. In order to restore the waveform, the polarity of digital data must be corrected by inverting the MSB of every sample.



	Value	Mean	Min	Max	Std Dev
Frequency	476.2MHz	Low resolution			
Amplitude	8.00mV	8.00m	8.00m	8.00m	0.00
Period	2.100ns	Low resolution			

Figure 4.18: Analog output without offset binary conversion.

The dual data port of the AD9739 chip receives two data streams simultaneously. Hence, these two data streams should be sent to the AD9739 chip from the Spartan-6 device by using an OSERDES module, which contains two OSERDES blocks. The host processor on the DAC chip provides AD9739 with a deinterleaved data stream so that DB0 and DB1 data ports receive alternating samples that are odd/ even data streams. When the AD9739 chip receives the deinterleaved data, the internal data assembler of the chip multiplexes the alternating samples into their original order before signal reconstruction takes place.

4.3.1 Experimental Results of 8-core DDS System

In this section, a T-Rex v4.0 board is used to test the VHDL program. Six random frequencies, 100 kHz, 500 kHz, 210 MHz, 450 MHz, 590 MHz and 750 MHz are used to test the T-Rex v4.0 board. While the evaluation board is powered up, a frequency tuning word is loaded into a phase register. The tuning word is then processed by an offset divider module before going to a phase accumulator. The output of the phase accumulator is added to each of the eight offset values to synthesise eight sequential sample phases. These sample phases are converted to amplitudes by a LUT module and then converted into an analog signal by AD9739.

The frequency tuning word of each signal can be calculated using equation (2.0):

For 100 kHz signal,

$$M = \frac{100\ kHz\ X\ 2^{25}}{250\ MHz} = 13422$$

For 500 kHz signal,

$$M = \frac{500\ kHz\ X\ 2^{25}}{250\ MHz} = 67108$$

For 210 MHz signal,

$$M = \frac{210\ MHz\ X\ 2^{25}}{250\ MHz} = 28185722$$

For 450 MHz signal,

$$M = \frac{450\ MHz\ X\ 2^{25}}{250\ MHz} = 60397977$$

For 590 MHz signal,

$$M = \frac{590\ MHz\ X\ 2^{25}}{250\ MHz} = 79188459$$

For 750 MHz signal,

$$M = \frac{750\ MHz\ X\ 2^{25}}{250\ MHz} = 100663296$$

Table 4.4 shows the phase offsets of the six random frequencies. The step size, which is the 1st phase offset of each signal, can be calculated by using equation (3.5).

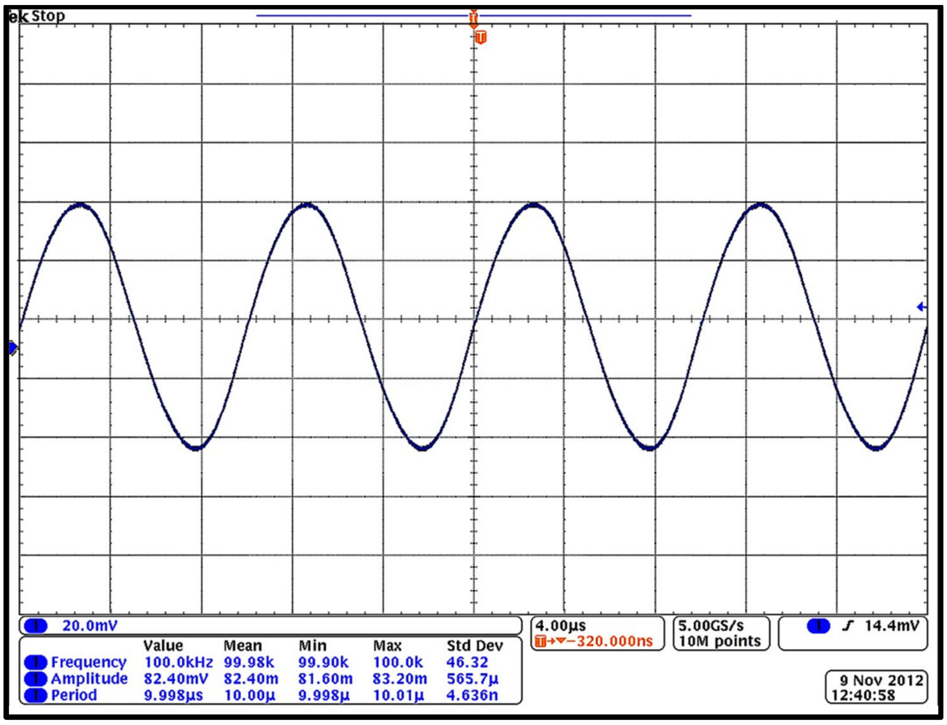
From the table it can be seen that the step size is proportional to the signal frequency. When the frequency increases, the step size decreases. This implies that the total number of samples required forming a cycle of sine-wave decreases at higher frequencies.

Every time the phase accumulator generates an output, it is to be added to 1st – 8th phase offset values as shown in Table 4.4 below. This results in eight samples being generated on every rising edge of a 250 MHz clock. With the phase offsets, the system can overflow quickly to reach the output frequency.

Sequential	**100 kHz**	**500 kHz**	**210 MHz**	**450 MHz**	**590 MHz**	**750 MHz**
1st (Step Size)	1677	8388	3523215	7549747	9898557	12582912
2nd	3355	16777	7046430	15099494	19797114	25165824
3nd	5033	25165	10569646	22649241	29695672	37748736
4th	6711	33554	14092861	30198988	39594229	50331648
5th	8388	41943	17616076	37748736	49492787	62914560
6th	10066	50331	21139292	45298483	59391344	75497472
7th	11744	58720	24662507	52848230	69289902	88080384
8th	13422	67108	28185722	60397977	79188459	100663296

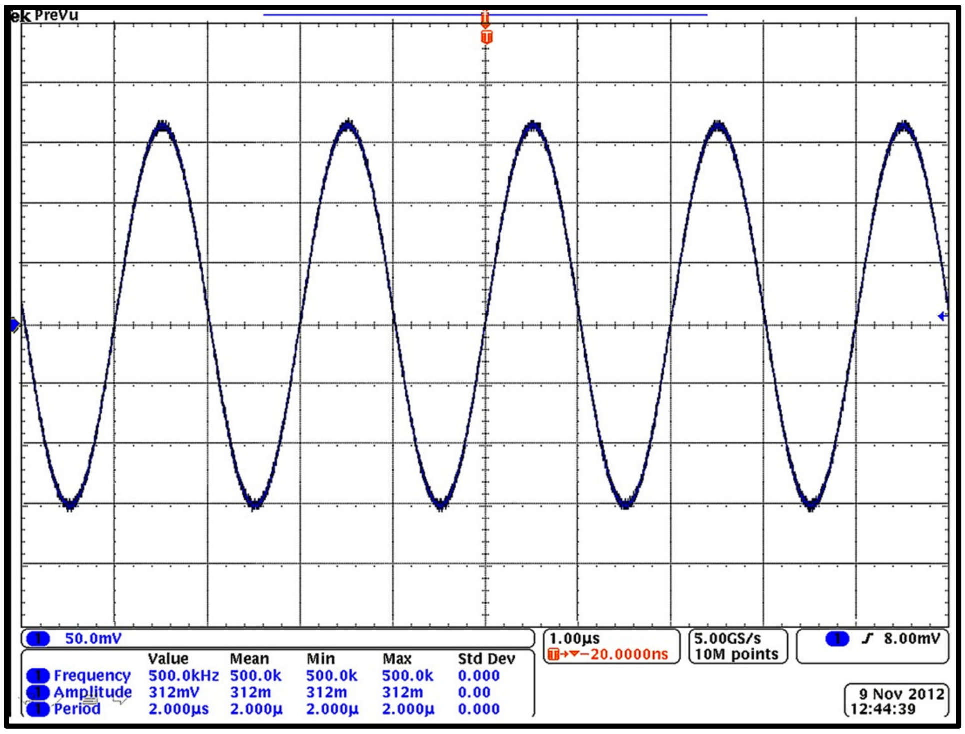
Table 4.4: Eight Phase offsets of six signals.

After loading the programming files into the Spartan-6 device, the output signals of each frequency are measured and recorded. The screen shots of 100 kHz, 500 kHz, 210 MHz, 450 MHz, 590 MHz and 750 MHz output signals are shown in Figures 4.19 - 4.24.



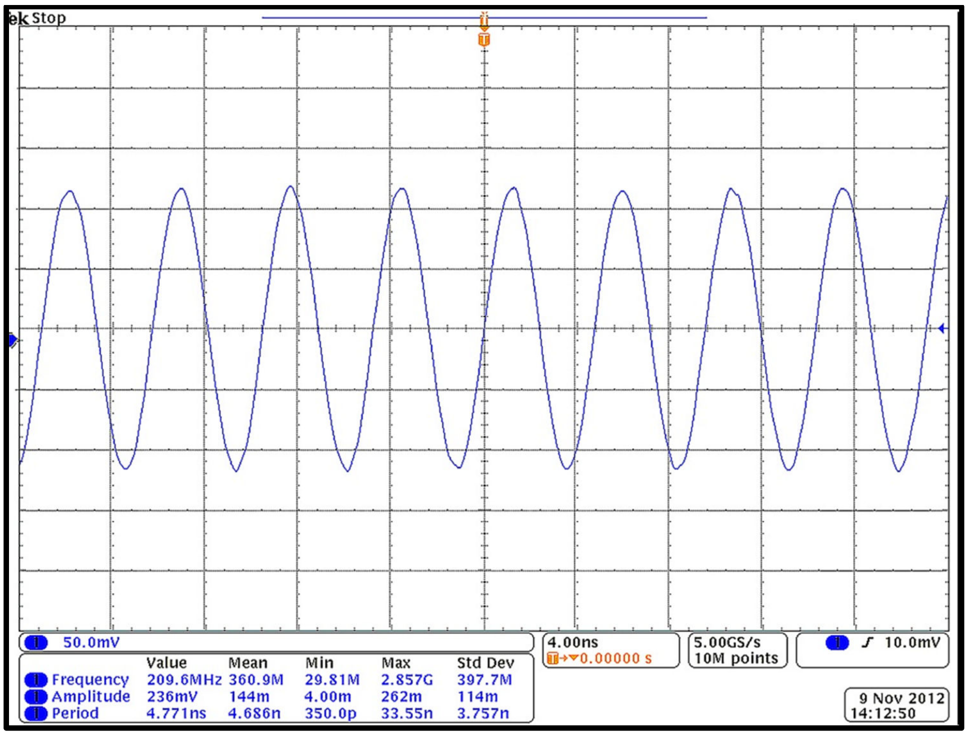
```
1  20.0mV                                          4.00µs            5.00GS/s     1  ʃ 14.4mV
                                                   T→▾−320.000ns     10M points
             Value     Mean     Min      Max      Std Dev
1 Frequency  100.0kHz  99.98k   99.90k   100.0k   46.32
1 Amplitude  82.40mV   82.40m   81.60m   83.20m   565.7µ                          9 Nov 2012
1 Period     9.998µs   10.00µ   9.998µ   10.01µ   4.636n                          12:40:58
```

Figure 4.19: 100 kHz sine-wave.



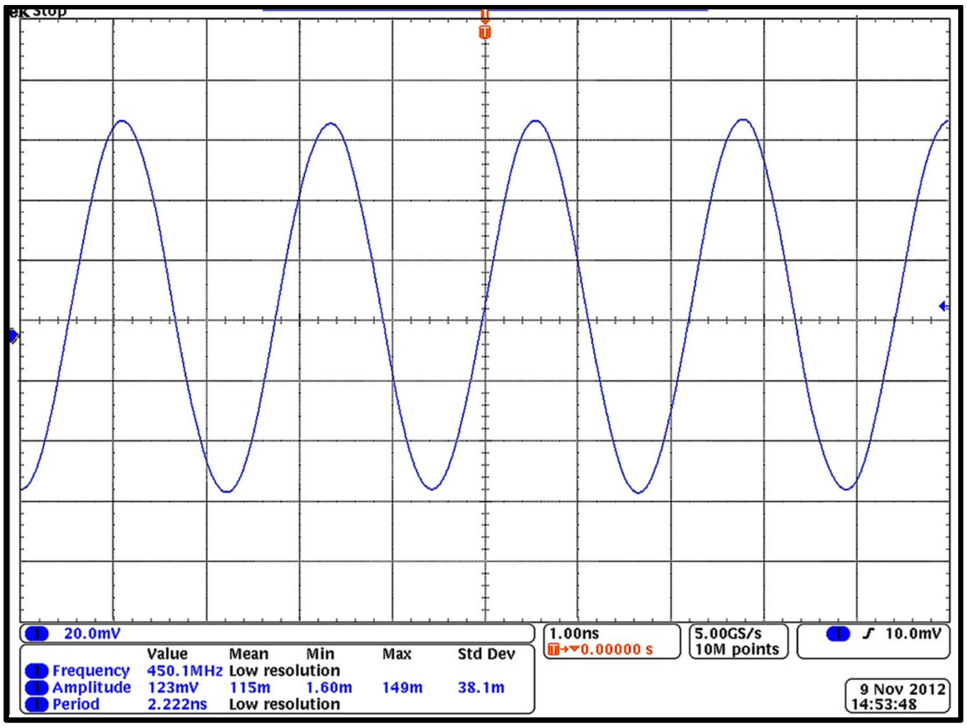
```
1  50.0mV                                          1.00µs            5.00GS/s     1  ʃ 8.00mV
                                                   T→▾−20.0000ns     10M points
             Value     Mean     Min      Max      Std Dev
1 Frequency  500.0kHz  500.0k   500.0k   500.0k   0.000
1 Amplitude  312mV     312m     312m     312m     0.00                            9 Nov 2012
1 Period     2.000µs   2.000µ   2.000µ   2.000µ   0.000                           12:44:39
```

Figure 4.20: 500 kHz sine-wave.



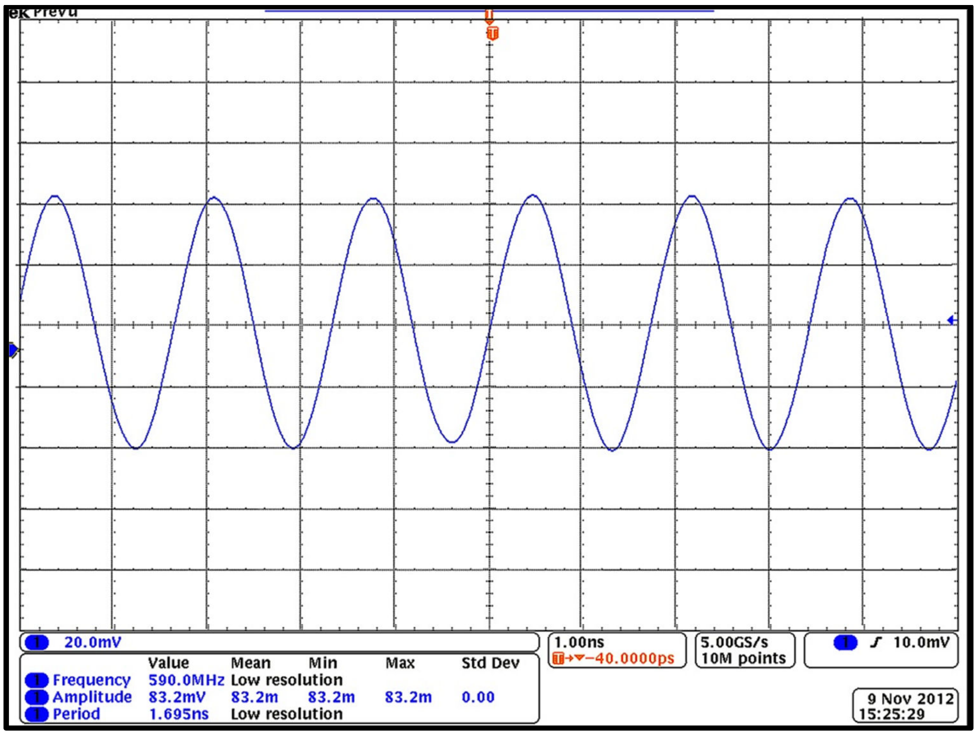
```
            Value     Mean    Min     Max     Std Dev
Frequency   209.6MHz  360.9M  29.81M  2.857G  397.7M
Amplitude   236mV     144m    4.00m   262m    114m
Period      4.771ns   4.686n  350.0p  33.55n  3.757n
```

Figure 4.21: 210 MHz sine-wave.



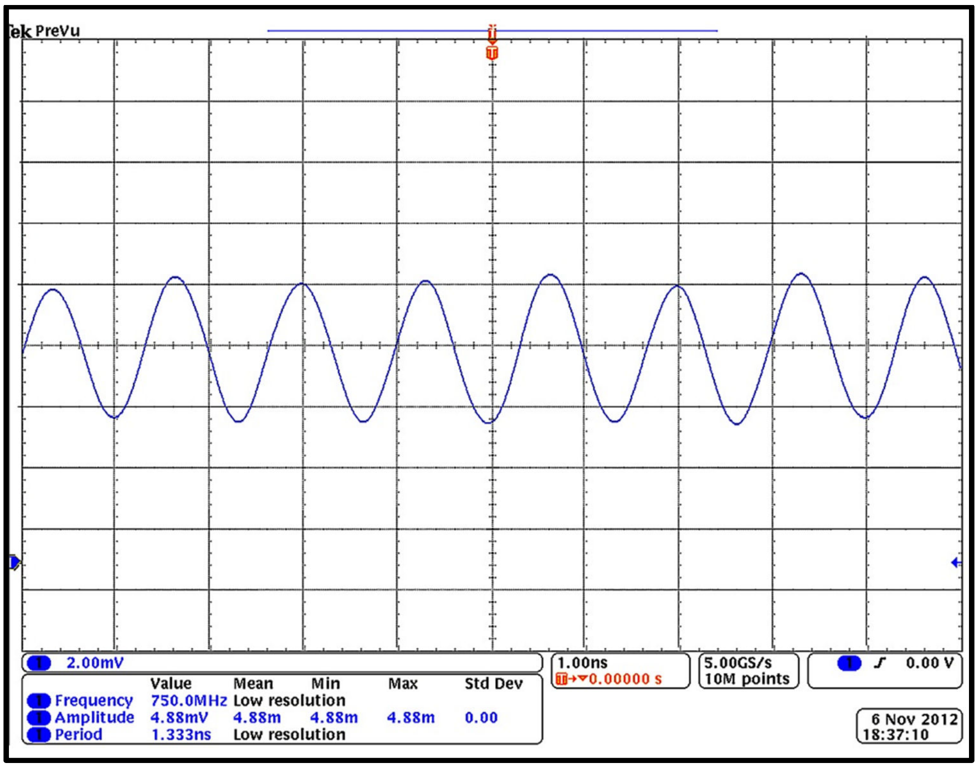
```
            Value     Mean            Min     Max     Std Dev
Frequency   450.1MHz  Low resolution
Amplitude   123mV     115m            1.60m   149m    38.1m
Period      2.222ns   Low resolution
```

Figure 4.22: 450 MHz sine-wave.



```
            Value      Mean            Min     Max     Std Dev
Frequency   590.0MHz   Low resolution
Amplitude   83.2mV     83.2m           83.2m   83.2m   0.00
Period      1.695ns    Low resolution
```

Figure 4.23: 590 MHz sine-wave.



```
            Value      Mean            Min     Max     Std Dev
Frequency   750.0MHz   Low resolution
Amplitude   4.88mV     4.88m           4.88m   4.88m   0.00
Period      1.333ns    Low resolution
```

Figure 4.24: 750 MHz sine-wave.

4.3.1.1 Discussion

Figures 4.19 - 4.24 show that the sinusoidal signals have been successfully generated by the 8-core DDS system. The 100 kHz and 500 kHz signals are generated based on over-sampling while other frequencies are generated based on under-sampling.

During signal synthesis, the step size that is also the 1st phase offset of a signal is calculated by dividing the frequency tuning word of the signal by eight. 2nd to 8th phase offsets are created based on the multiple of the step size. The eight phase offsets are created in parallel and they are added with accumulated phases simultaneously.

Adding an accumulated phase with each of the phase offsets can improve the total sample phases in a signal by 8 times. It also forces the 25-bit phases to overflow quicker so that the amount of revolution is identical to the frequency of an output signal. After this, eight consecutive sample phases are converted to amplitudes by a LUT module. Digital amplitudes are then converted to an analog signal by AD9739.

Despite the fact that the offset values used in under-sampling and over-sampling are known as phase offsets, both contain different information. For any signal within the 1st Nyquist zone, the eight phase offsets contain the data to reproduce eight sample phases, which deliver the same output frequency as the accumulated phases generated by a phase accumulator. In other words, the phase offsets increase the total number of samples in a sine-wave over a unit time without changing the output frequency. Regarding under-sampling, the phase offsets contain information to shift an accumulated phase to eight sample phases of higher frequency signal. The purpose is to force the eight sample phases to overflow quicker so that a higher frequency signal can be achieved. This concept is to generate a high-speed signal by using a low speed signal. A high-speed signal normally refers to the output signal while a low speed signal refers to the image frequency of the output signal. The entire procedure is akin to imitating the samples of the output signal by offsetting the samples of aliased signal.

From Figure 4.19-4.24, the total samples used to form a high-speed sine-wave are fewer than for a lower speed signal. When the frequency of a sinusoidal signal is higher, the sine-wave has a shorter cycle period. A higher speed signal requires the

25-bit sample phase to overflow quicker so that revolution can be achieved faster. When the overflowing becomes faster, the total amount of phase hopping becomes less and therefore, fewer samples are allocated for a signal. Consequently, the sample distribution makes a signal harder to maintain a sinusoidal shape. As a result, the amplitude might be distorted. The 8-core DDS module is not desirable to generate a signal beyond 750 MHz because any signal beyond that frequency is unstable. The distortion gets significant if the output frequency is even higher. As a result, the upper frequency, which can be generated by the 8-core DDS module, is 750 MHz.

Figure 4.19 and Figure 4.20 show that the amplitude voltage levels have increased from 82.4 mV to 312 mV when the frequencies increase from 100 kHz to 500 kHz. The 100 kHz and 500 kHz frequencies have lower amplitude voltage levels compared to other frequencies, which are due to the effect of the on-board RF transformer. The transformer has 75 ohms impedance and 5 MHz - 3000 MHz frequency range. 100 kHz and 500 kHz are not within the acceptable range of the transformer [67]. The attenuation of the signals is due to the insertion loss and the input return loss of the transformer. The 500 kHz signal has higher amplitude than a 100 kHz signal because 500 MHz is closer to the minimum frequency of the RF transformer. Therefore, the attenuation at 500 kHz is less than 100 kHz. Any signal at 1 MHz or less could have insertion loss and input return loss as high as 1.85 dB and 10.86 dB, respectively [67].

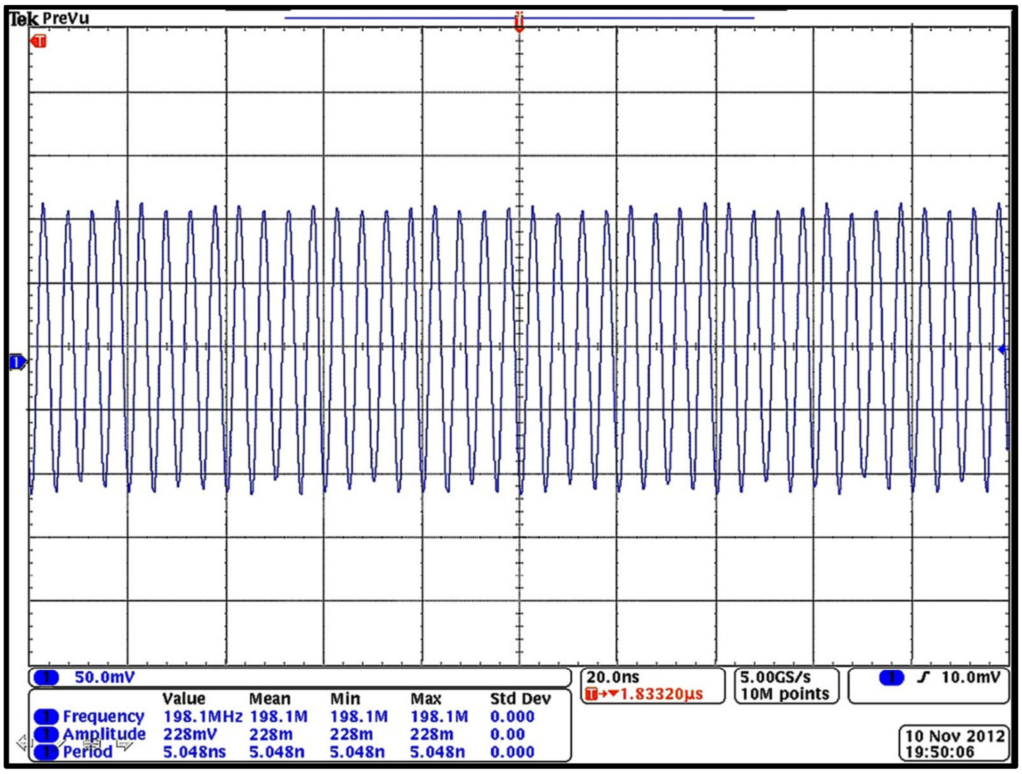
Figure 4.21 to Figure 4.24 show that the amplitude voltage levels are inversely proportional to the signal frequencies. When the signal frequencies increase, the amplitude voltage levels are decreased. More attenuation at higher frequencies is due to the effect of the low-pass filter used on the evaluation board. The low-pass filter has frequency ranges from DC to 490 MHz with 1.2 dB at the pass-band and up to 40 dB at the stop-band [68].

In summary, the combination of the RF transformer and low-pass filter provides more attenuation at higher frequencies than at lower frequencies. The attenuation makes the amplitude decrease when the signal frequency increases. Nonetheless, for signals below 5 MHz the signal amplitude decreases at lower frequency.

4.3.2 Experimental Results of Phase Coherence

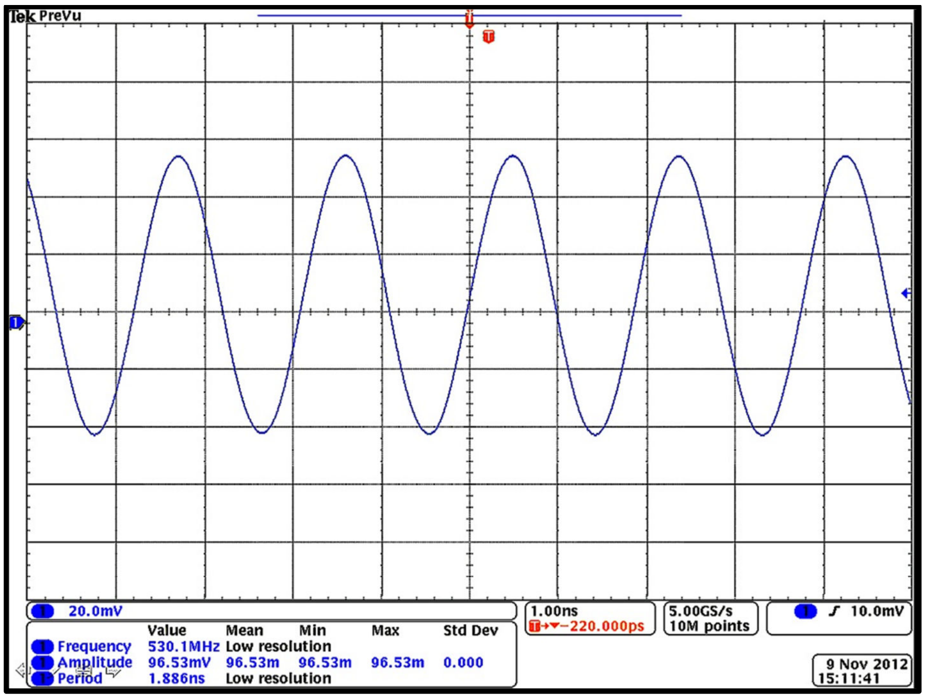
The experiment is carried out by placing the tuning word of 200 MHz and 530 MHz signals to 'preset_inc1' and 'presetn_inc2'. These two frequencies are used because the differences between them are large enough to be viewed on an oscilloscope.

Before starting the switching process between these two signals, separated waveforms of 200 MHz and 530 MHz are recorded in Figure 4.25 and Figure 4.26. Note that the variance in the amplitude is due to the finite resolution of the oscilloscope.



```
           Value     Mean    Min     Max     Std Dev
Frequency  198.1MHz  198.1M  198.1M  198.1M  0.000
Amplitude  228mV     228m    228m    228m    0.00
Period     5.048ns   5.048n  5.048n  5.048n  0.000
```

Figure 4.25: 200 MHz sine-wave.



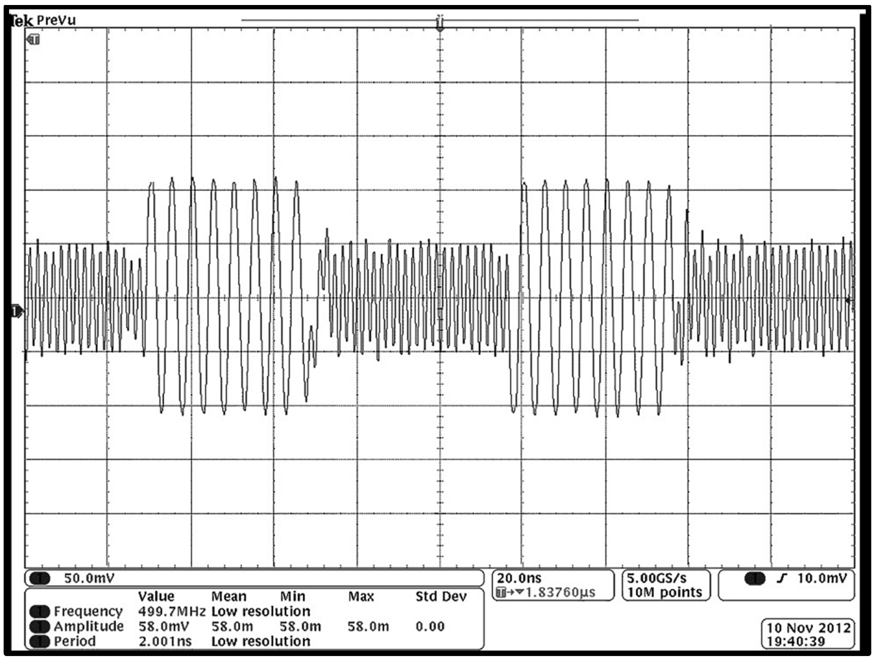
```
1  20.0mV                              1.00ns            5.00GS/s     1  ∫  10.0mV
                                       T→▾−220.000ps     10M points
             Value      Mean            Min       Max       Std Dev
1 Frequency  530.1MHz   Low resolution
1 Amplitude  96.53mV    96.53m          96.53m    96.53m    0.000
1 Period     1.886ns    Low resolution
                                                          9 Nov 2012
                                                          15:11:41
```

Figure 4.26: 530 MHz sine-wave.

In Figure 4.25, the peak-to-peak of the 200 MHz sine-wave has about 4.6 divisions along the vertical axis. Each division is 50 mV so a 200 MHz sine-wave has ≈ 230 mV of amplitude. Figure 4.26 shows that the peak-to-peak of the 530 MHz sine-wave has about 5 divisions along the vertical axis. Each division is 20 mV so the 530 MHz signal is approximately ≈ 100 mV.

The 200 MHz sine-wave in Figure 4.25 has 4 cycles of 200 MHz sine-wave per division on the horizontal axis. This means each division is equal to 4 X 5.048 ≈ 20.192 ns (5.048 ns is the period of the signal measured using an oscilloscope). In contrast, a single sine cycle of a 530 MHz signal occupies about 1.8 divisions along the horizontal axis. This means each square on the x-axis of a 530 MHz signal is equal to 1.886 ns/ 1.08 ≈ 1.048 ns per division (1.886 ns is the period of the signal measured using an oscilloscope). During the switching process between two signals, it is difficult to capture both signals by using an oscilloscope because the entire process is fast. In order to make sure a screen shot contains both signals, the switching process was repeated on every 11th cycle of the 250 MHz clock.

A 200 MHz sine-wave was oscillated when the T-Rex 4.0 board was turned on. After 40 ns, 'Selector' switched the output signal to 530 MHz. 'Selector' switched again after another 40 ns. Then, the 530 MHz sine-wave was replaced by a 200 MHz sine-wave. The entire switching process was repeated so that the results, as shown in Figure 4.27, can be captured from the oscilloscope.



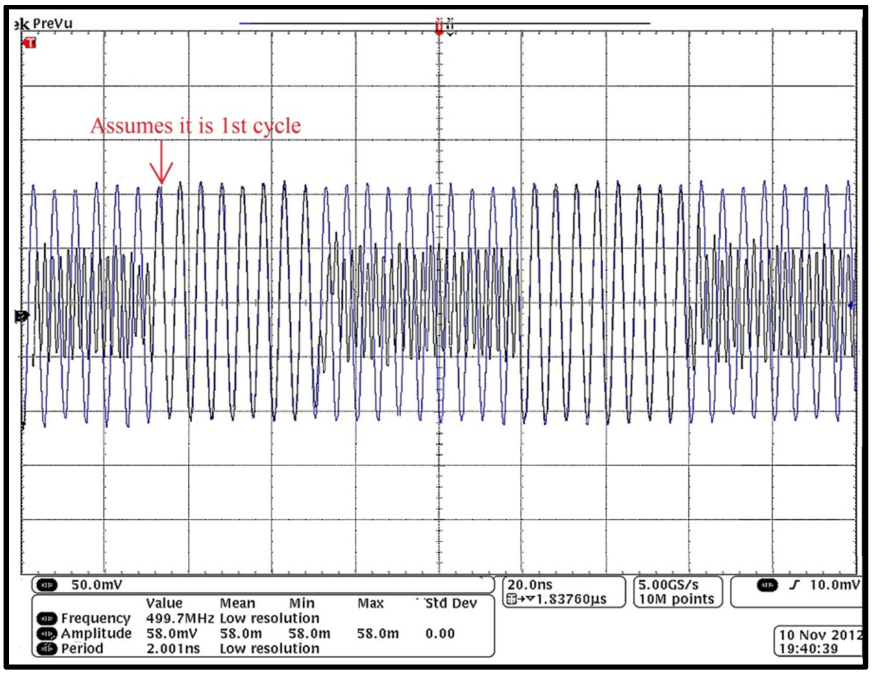
```
1  50.0mV                                              20.0ns            5.00GS/s      1  ƒ 10.0mV
              Value     Mean          Min      Max      Std Dev     T→▼1.83760µs      10M points
1 Frequency   499.7MHz  Low resolution
1 Amplitude   58.0mV    58.0m         58.0m    58.0m    0.00                                   10 Nov 2012
1 Period      2.001ns   Low resolution                                                         19:40:39
```

Figure 4.27: Switching between 200 MHz and 530 MHz.

Figure 4.27 shows the switching between 200 MHz and 530 MHz sine-waves. The 200 MHz sine signal after switching from 530 MHz is in-phase with the 200 MHz sine signal before switching to 530 MHz.

In Figure 4.27, the amplitude and frequency measurement during the switching process is inaccurate because the oscilloscope is unable to detect the fast changing between two signals. The amplitudes of 200 MHz and 530 MHz signals can be estimated by referring to the total divisions of these signals on the vertical axis. The amplitudes of 530 MHz and 200 MHz sine-waves have approximately 2 and 4.4 divisions. Each division on the vertical axis is 50 mV. Hence, the amplitude level of 530 MHz is 2 X 50 mV $\approx$ 100 mV and the amplitude level of 200 MHz is 4.6 X 50 mV $\approx$ 230 mV. From the calculation, the 530 MHz and 200 MHz signals after the switching process are still able to maintain the amplitude voltage levels, the same as the separated sine-waves shown in Figure 4.25 and Figure 4.26.

The phase coherence of 200 MHz in Figure 4.27 can be verified by comparing the individual 200 MHz signal in Figure 4.25. By overlaying Figure 4.25 with Figure 4.27, the overlaid waveform in Figure 4.28 can be obtained.



```
            Value      Mean            Min     Max     Std Dev
Frequency   499.7MHz   Low resolution
Amplitude   58.0mV     58.0m           58.0m   58.0m   0.00
Period      2.001ns    Low resolution
```

Figure 4.28: Phase coherence of 200 MHz sine-wave.

In Figure 4.28, the 200 MHz signal before and after switching from 530 MHz (highlighted in black) managed to maintain a fixed phase relationship. Let us assume the 1st cycle of a 200 MHz signal is indicated by the arrow in Figure 4.28. The 200 MHz signal has oscillated from the 1st to 8th cycle. After that, it was replaced by a 530 MHz signal on the 9th cycles. The 530 MHz signal then oscillates between the 9th and 18th cycle. Again, 530 MHz output was replaced by a 200 MHz signal on the 19th cycle. Throughout the whole switching process, the 200 MHz signal before and after switching, is in-phase. It can be verified by comparing the result highlighted in black with the individual 200 MHz reference signal highlighted in blue. In Figure 4.28, both signals match each other in terms of their phase and amplitude. This is because the 8-core DDS module kept generating the 200 MHz samples after they were replaced by 530 MHz signals. Therefore, the switching process does not affect the synthesis of other signals in the DDS module.

4.4 Hardware Implementation of Distributed Memory Block

The specification of the distributed memory block:

System Clock, f_{system}	: 250 MHz
Depth	: variable (16 to 65536)
Data Width	: 112
Radix	: 2, 10, 16

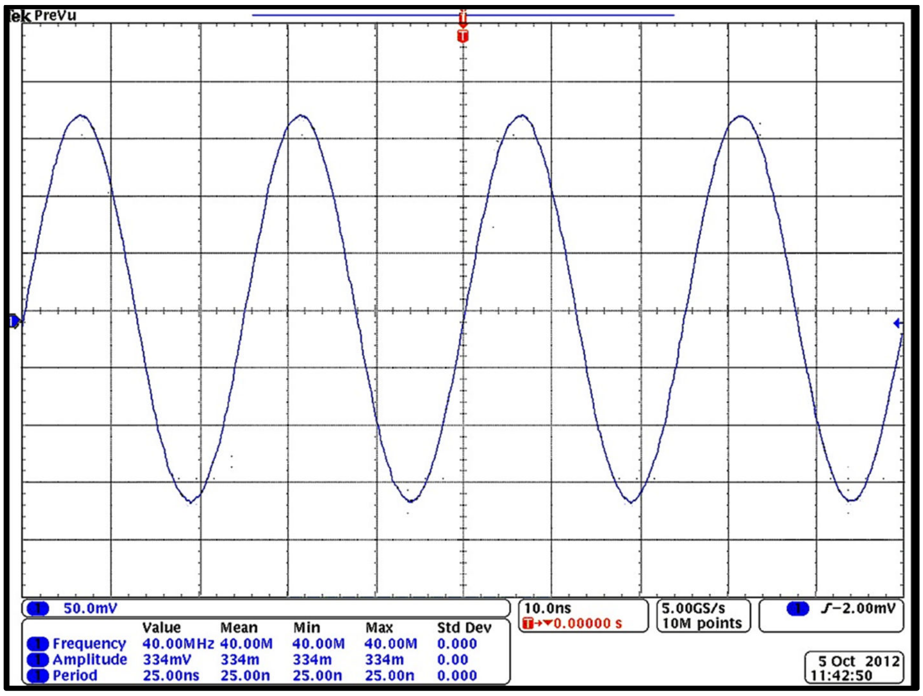
The distributed memory block can be activated by setting the MSB of 'Selector' to '1XX'; 'X' could be any logics. Once the system reads 'Selector' is equal to '1XX', the content of the coefficient file is retrieved on every rising edge of the 250 MHz signal.

The sample amplitudes of the 40 MHz sinusoidal signal (Figure 4.29) are used to run the experiment. The purpose of choosing a 40 MHz sine-wave instead of other signals is because they are generated by using a distributed memory block that can be easily compared with the 40 MHz sine-wave generated by using 8-core DDS module. If random data are used, it is difficult to assess whether the signal generated is correct or incorrect.

4.4.1 Test Results of Distributed Memory Block

In this section, a 40 MHz sine-wave was generated by using a distributed memory block. It is used to compare with a 40 MHz sine-wave generated by using an 8-core DDS module.

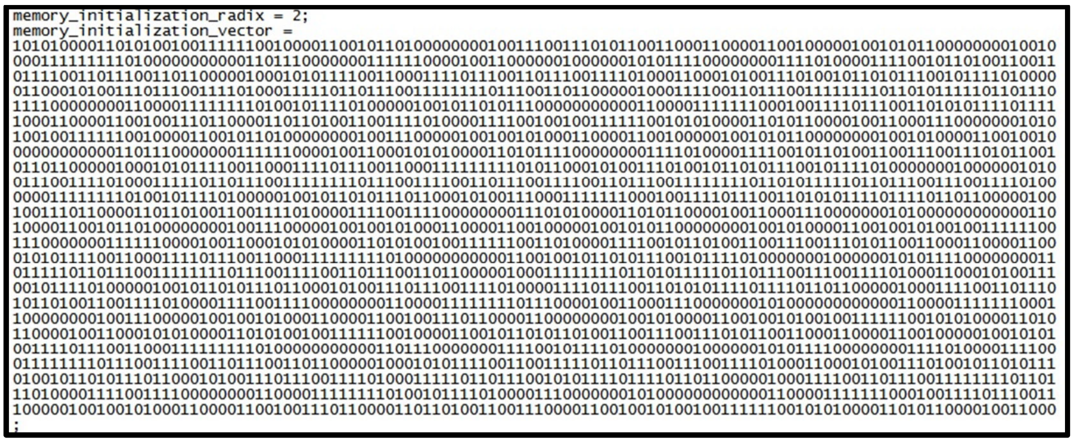
Figure 4.29 shows the 40 MHz sine-wave generated by an 8-core DDS module.



	Value	Mean	Min	Max	Std Dev
Frequency	40.00MHz	40.00M	40.00M	40.00M	0.000
Amplitude	334mV	334m	334m	334m	0.00
Period	25.00ns	25.00n	25.00n	25.00n	0.000

Figure 4.29: 40 MHz signal generated by 8-core DDS module.

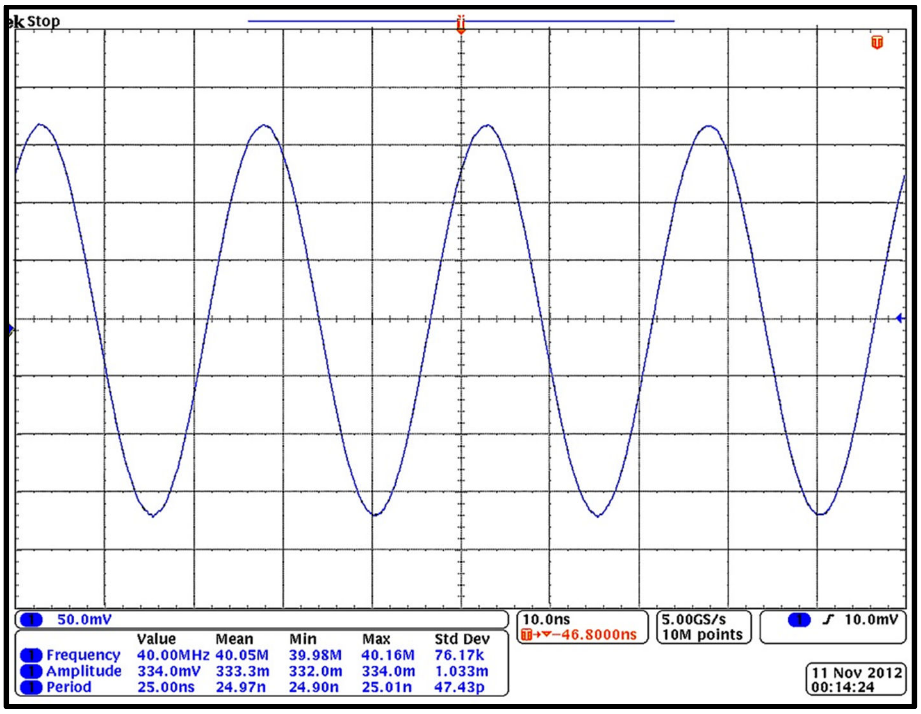
The sample amplitudes generated by the 8-core DDS module (Figure 4.29) were captured by using a test-bench file. The amplitudes were then copied to the coefficient file as shown in Figure 4.30. The entire project was then compiled to generate a programming file and loaded into the Spartan-6 device. Theoretically, the waveform generated by the memory block must show the same result as generated by the 8-core DDS module.



```
memory_initialization_radix = 2;
memory_initialization_vector =
1010100001101010010011111100100001100101101000000000100111001110101100110001100001100100000010010101100000000010010
0001111111110100000000000011011100000000111111000010011000000010000001010111100000000111101000011110010110100110011
0111100110111001101100000100010101111001100011110111001101110011110100011000101001110100101101011100101111010000
01100010100111011100111101000111110110111001111111101110011011000001000111100110111001111111101101011111101101110
1111000000000110000111111110100101111010000010010110101110000000000001100001111111000100111101110011010101111101111
1000110000110010011101100001101101001100111101000011110010010011111100101010000110101100001001100011100000001010
100100111111001000011001011010000000010011100000010010010100011000011001000000100101011000000001001010000110010010
00000000001101110000000011111100001001100010101000011010111100000000011110100001111001011010011001110011101011001
0110110000010001010111100110001111011100110001111111110101100010100111010010110101110010111101000000010000001010
0111001111010001111101101110011111111011100111100110111001111001101110011111111011010111110110111001110011110100
00001111111101001011110100000100101101011101100010100111000111111100010011110111001101010111101111101101100000100
1001110110000110110100110011110100001111001111000000000111010100001101011000010011000111000000010100000000000110
1000011001011010000000010011100000010010010100011000011001000000100101011000000001001010000110010010100100111111100
11100000000111111000010011000101010000110101001001111110011010000111100101101001100111001110101100110001100001100
010101111001100011110111001100011111111101000000000000110010010110101110010111101000000010000001010111100000000011
0111110110111001111111101110011110011011100110110000010001111111101101011111011011100111001111010001100010100111
0010111101000001001011010111011000101001110111001111010000111101110011010101111011110110110000010001111001101110
1011010011001111010000111100111100000000110000111111110111000010011000111000000010100000000000011000011111110001
1000000001001110000010010010100011000011001001110110000110000000010010100001100100101001001111110010101000011010
11000010011000101010000110101001001111110010000110010110101101001100111001110101100110001100001100100000010010101
001111011100110001111111110100000000000110111000000011110010111101000000001000000010101111000000000111101000011110
0111111110111001111001101110011011000001000101011110011001111101101110011100111101000110001010011101001011010111
0100101101011101100010100111011100111101000111110110111001010111101111011011000001000111100110111001111111101101
11010000111100111100000000110000111111110100101111010000111000000010100000000000001100001111111000100111101110011
10000001001001010001100001100100111011000011011010011001110000110010010100100111111001010100001101011000010011000
;
```

Figure 4.30: Custom data for 40 MHz sine-wave.

When the system was powered on, the system retrieved the data from the distributed memory block and the DAC chip converted them to an analog signal. The analog signal was then measured by using an oscilloscope. The waveform is shown in Figure 4.31.

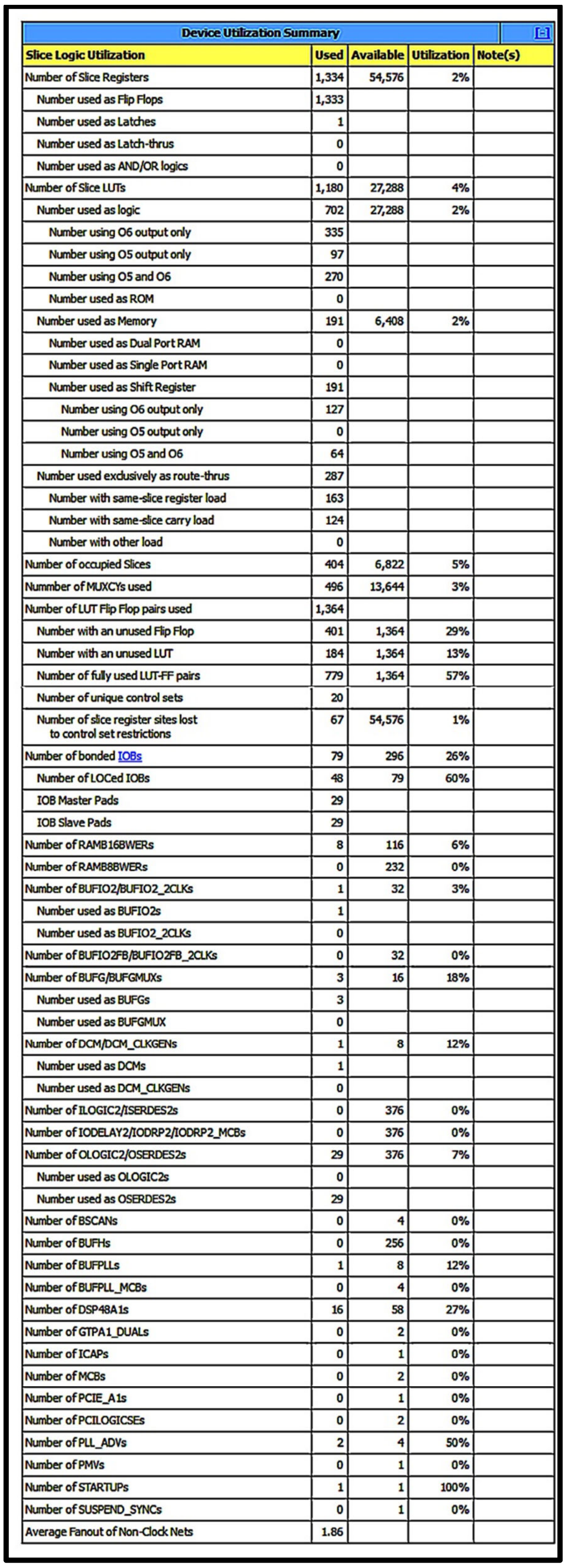


```
Stop
1  50.0mV                       10.0ns          5.00GS/s     1  ⨍  10.0mV
                                T→▼−46.8000ns   10M points
            Value     Mean     Min      Max      Std Dev
1 Frequency 40.00MHz  40.05M   39.98M   40.16M   76.17k
1 Amplitude 334.0mV   333.3m   332.0m   334.0m   1.033m
1 Period    25.00ns   24.97n   24.90n   25.01n   47.43p
                                                  11 Nov 2012
                                                  00:14:24
```

Figure 4.31: 40 MHz signal generated by distributed memory block.

By comparing Figure 4.29 and Figure 4.31, we see that both are the same in terms of signal shape, frequency, and amplitude. This implies that the data are successfully retrieved from the distributed memory block to DAC. The sine-wave generated from the distributed memory block is 40 MHz with 25 ns periods and 334 mV amplitude voltage level, which are the same as the 40 MHz sine-wave generated by the 8-core DDS module.

4.4.2 FPGA Device Utilisation Summary



Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	1,334	54,576	2%	
Number used as Flip Flops	1,333			
Number used as Latches	1			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	1,180	27,288	4%	
Number used as logic	702	27,288	2%	
Number using O6 output only	335			
Number using O5 output only	97			
Number using O5 and O6	270			
Number used as ROM	0			
Number used as Memory	191	6,408	2%	
Number used as Dual Port RAM	0			
Number used as Single Port RAM	0			
Number used as Shift Register	191			
Number using O6 output only	127			
Number using O5 output only	0			
Number using O5 and O6	64			
Number used exclusively as route-thrus	287			
Number with same-slice register load	163			
Number with same-slice carry load	124			
Number with other load	0			
Number of occupied Slices	404	6,822	5%	
Nummber of MUXCYs used	496	13,644	3%	
Number of LUT Flip Flop pairs used	1,364			
Number with an unused Flip Flop	401	1,364	29%	
Number with an unused LUT	184	1,364	13%	
Number of fully used LUT-FF pairs	779	1,364	57%	
Number of unique control sets	20			
Number of slice register sites lost to control set restrictions	67	54,576	1%	
Number of bonded IOBs	79	296	26%	
Number of LOCed IOBs	48	79	60%	
IOB Master Pads	29			
IOB Slave Pads	29			
Number of RAMB16BWERs	8	116	6%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	1	32	3%	
Number used as BUFIO2s	1			
Number used as BUFIO2_2CLKs	0			
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	3	16	18%	
Number used as BUFGs	3			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	1	8	12%	
Number used as DCMs	1			
Number used as DCM_CLKGENs	0			
Number of ILOGIC2/ISERDES2s	0	376	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0%	
Number of OLOGIC2/OSERDES2s	29	376	7%	
Number used as OLOGIC2s	0			
Number used as OSERDES2s	29			
Number of BSCANs	0	4	0%	
Number of BUFHs	0	256	0%	
Number of BUFPLLs	1	8	12%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	16	58	27%	
Number of GTPA1_DUALs	0	2	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCIE_A1s	0	1	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	2	4	50%	
Number of PMVs	0	1	0%	
Number of STARTUPs	1	1	100%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	1.86			

Figure 4.32: Device utilisation summary.

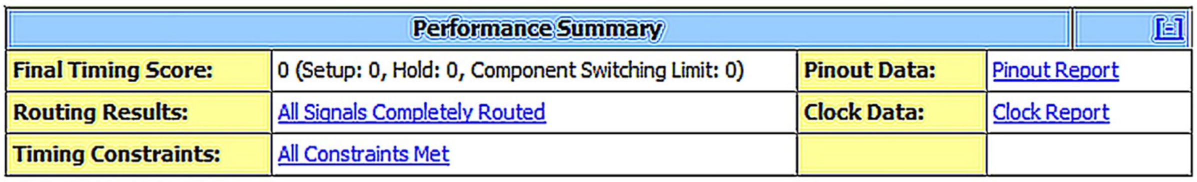
Figure 4.32 presents a summary of the resources that have been used to develop a multi-frequency generator in Spartan-6. From it, we can conclude that the multi-frequency generator only used part of the resources available to construct the system. However, a plenty of resources that remain unused in Spartan-6. These resources can be utilised for future improvement such as implementing an RF receiver. All common components needed by the NMR RF receiver such as band-pass filter, quadrant detector and so on can be replaced by a Spartan-6 device.

The summary shows most of the slice logic utilisation is less than 20% except for LUT flip-flop pairs, DSP48A1s, PLL_ADV, and IOB. 779 out of 1364 pairs of LUT flip-flops have been used to build the multi-frequency RF generator (approximately 57%). 13.5% (184 out of 1364) LUTs and about 29.4% (401 out of 1364) flip-flops remained unused.

In IOB, 48 out of 79 LOCed IOBs have been occupied. The number shows 60% of the clock-capable I/Os are used to interface with the clock chip and the DAC chip. In other words, 40% of the clock-capable I/Os blocks are still available for future enhancement.

The primary role of DSP48A1 is to perform arithmetic operations in Spartan-6. All the arithmetic calculations in the 8-core DDS module require 27% of the DSP48A1s. 2 of the 4 PLL_ADVs are utilised to monitor the internal clocking of the 8-core DDS module.

Figure 4.33 shows that the VHDL program, which is developed for the multi-frequency RF generator, has met the timing constraint of a Spartan-6 device. All the input and output signals are completely routed, which do not cause any redundancy in the I/O resources. The final timing score shows that the module does not encounter any timing issues because lower score means better performance.



Performance Summary			
Final Timing Score:	0 (Setup: 0, Hold: 0, Component Switching Limit: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Figure 4.33: Performance summary.

Chapter 5

Conclusion and Future Work

This chapter presents the conclusion of this Master's thesis as well as the future work to enhance and improve the NMR RF transmitter.

5.1 Conclusion

The 8-core DDS system is successfully implemented and is simulated using Matlab-Simulink and ISim simulator. The discussion of the conceptual designs of the NMR RF transmitter can be found in Chapter 2. The experimental results provide a better understanding of how a DDS system generates signals up to 750 MHz with various frequency tuning words.

The input to a DDS system is known as a frequency tuning word. It is a phase increment value used to accumulate 25-bit phases to generate an output signal. An 8-core DDS system consists of a 25-bit phase accumulator that generates 25-bit phases at 250 MHz. The 25-bit accumulated phases are truncated to 14-bit because this method can reduce the resources needed to form a LUT without changing the frequency resolution. Furthermore, a 14-bit phase truncation is selected because each input port of AD9739 has 14 pins.

In an 8-core DDS module, a single phase accumulator can generate phases of a signal up to 125 MHz. Nonetheless, the eight phase offsets manage to shift the accumulated phases to eight consecutive samples. It increases eight more samples in a sine-wave over a unit time. Therefore, the 8-core DDS module allows signal generation up to 8 x 125 = 1 GHz. A phase accumulator has 250 MSPS sample rates, with eight samples generated over a unit time causing the system to have 2 GSPS. The AD9739 DAC chip has dual data path with each sampling at 1 GSPS. Therefore, the two sample sets that are sent from Spartan-6 to DAC should be in 1 GSPS of each. With the DAC chip, the dual-input is multiplexed and is reconstructed as a single output at 2 GHz sampling rate.

The design of an 8-core DDS module is based on under-sampling and over-sampling. For any output signal that is lower than 125 MHz, the signal frequency is converted to a tuning word and the sample phases are generated by a phase accumulator. If the signal is beyond 125 MHz, the samples would be generated by using under-sampling. In this circumstance, the aliased frequency of the output signal is used for phase increment in the phase accumulator. The accumulated phases will be shifted to a higher frequency by eight phase offsets. These phase offsets are calculated by using the lower band-limit of the Nyquist zone where the output signal is located. The adding of the offset values with the accumulated phases overflows the 25-bit sample phases quicker and therefore an output signal with a desired frequency can be generated by the 8-core DDS module.

In a multi-frequency generator, the VHDL project of an 8-core DDS module is modified so that the system can generate four signals simultaneously. The output signals are switchable, but only one output can be selected at a time via a switch. The multi-frequency generator is designed to sustain the phase coherence of an output signal. With the phase coherence, the signal that is switching to another frequency then back to its original frequency is in-phase. This feature is very important because it makes the excitation of nuclei at different times as possible while the phase coherence is maintained. This would benefit physicists to examine the physical and chemical properties of molecules or atoms of the sample under examination in an NMR system.

A distributed memory block is an alternative solution for the generation of a custom signal. Any signal can be generated by using this method as long as the custom data of a desired signal are written into the coefficient file. During the RTL-level of simulation, a random signal is generated. The random signal is verified by comparing sample outputs retrieved from the distributed memory block with the random data written into the coefficient file. When it comes to the T-Rex v4.0 board, the output signal is hardly verified. This is because analog signal is a representation of time varying quantities in a continuous signal. It does not contain any discrete values and therefore it cannot be compared with the coefficient file. Due to this reason, a sine-wave is preferred for testing the distributed memory block. The sine-wave of the same frequency is generated by a distributed memory block and an 8-core DDS module. Both are compared and the results show the distributed memory block is functioning properly.

A T-Rex 4.0 evaluation board is used to test the developed DDS system. All the VHDL modules presented in this thesis can be readily synthesised except the test-bench code because it is a virtual environment used to verify the correctness of the design. Before generating an analog signal by using the T-Rex v4.0 board, both clock chip and DAC chip from T-Rex v4.0 board should be configured before synthesising the signal. The clock chip is enabled to generate clock signals to support the clocking in the entire system while the DAC chip is programmed to convert digital to analog signal at 2 GSPS.

Six random sinusoidal signals were examined during the hardware testing session. The results displayed in the thesis show the developed DDS system work as expected. The results indicate that the DDS system managed to generate signals ranging from 100 kHz to 750 MHz. However, any signal beyond 750 MHz causes signal distortion. This is because insufficient samples are generated by the system to maintain the signal shape as sinusoidal. From the results generated by the T-Rex v4.0 board, any signal below 5 MHz has amplitudes attenuated by the on-board RF transformer. The attenuation of the signal below 5 MHz is proportional to the signal frequency. When the signal is at 5 MHz and more, the amplitude attenuation is inversely proportional to the signal frequency, with attenuation being due to the low-pass filter on the board.

In conclusion, the results obtained from the simulations and experiments show the objectives of the research are fulfilled. The methodologies discussed in the thesis can be applied to any NMR RF transmitter. Any sinusoidal signals that are generated by using these methodologies must fall within the allowable frequency range from 100 kHz to 750 MHz. Any signal other than a sine-wave can be generated by using the built-in distributed memory block of Spartan-6. The only precaution of using this method is that the custom data must be written into the coefficient file by following the data format as described in Chapter 3.

5.2 Future Work

A new solution can be investigated to improve the total samples in a signal especially at higher frequency. The increment in total samples can extend the frequency range of an output signal to go beyond 750 MHz. When more samples are inserted, the signal at higher frequency can be formed because the signal shape can be maintained by the distribution of the samples in the signal.

The existing 8-core DDS module can be enhanced by using 1/4 of the data width from a phase accumulator to generate a signal. This solution can be achieved by mirroring the 1/4 phases of a signal to produce a full sine-wave. This method can reduce the size of a LUT but it requires an extra bit in the phase accumulator to determine which axes of the samples should be mirrored. Nevertheless, it is a good attempt to find out other possible solutions to compress the content of a LUT without affecting the output.

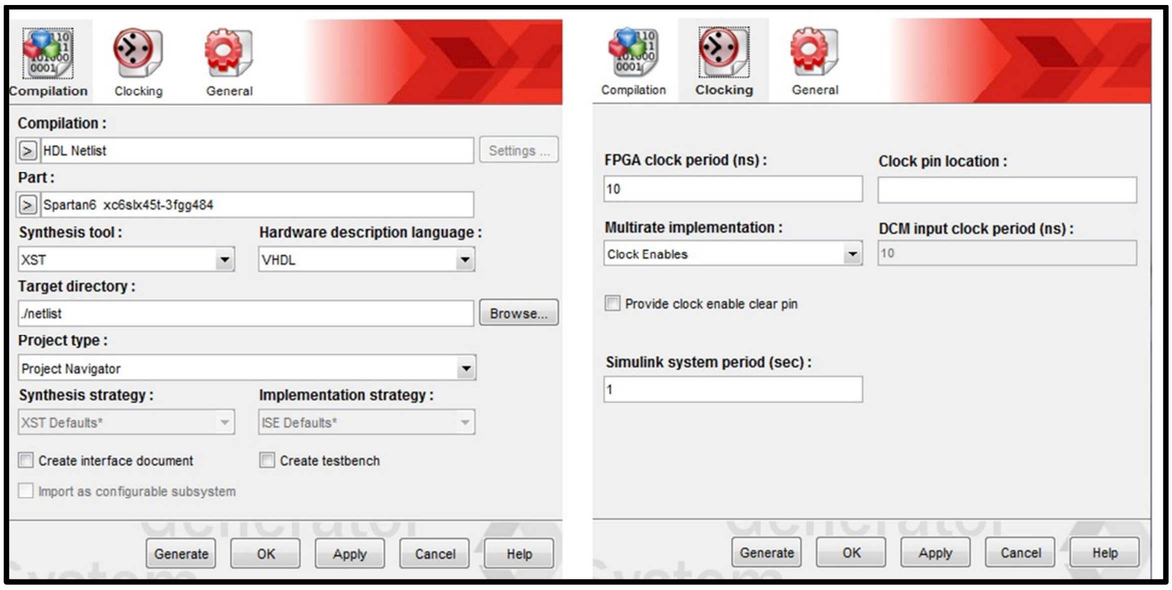
An NMR RF receiver can be developed in combination with an NMR RF transmitter. This combination could form a spectrometer for NMR application. A typical spectrometer consists of analog and digital circuits, which consume a lot of area on the board. An NMR RF receiver can be implemented by using a single FPGA device. Many external circuits such as a quadrant detector, an anti-aliasing pre-filter and so on can be integrated into a single FPGA device [6]. This method could ease the signal routing and it could reduce the size of the motherboard.

The main purpose of this thesis was to design, develop, and explore the potential of FPGA to implement a DDS system to generate samples at high speed. Therefore, it does not consider the impacts of SNR and other forms of noise coupling into the system. Hence, any future enhancement can include the analysis and study of noise factors so that a better DDS system with a cleaner output signal can be created.

Appendix A

Parameter Settings of MATLAB-Simulink Blocks

1. Parameter setting of System Generator Block



Compilation : HDL Netlist
Part : Spartan6 xc6slx45t-3fgg484
Synthesis tool : XST
Hardware description language : VHDL
Target directory : ./netlist
Project type : Project Navigator
Synthesis strategy : XST Defaults*
Implementation strategy : ISE Defaults*
☐ Create interface document
☐ Create testbench
☐ Import as configurable subsystem

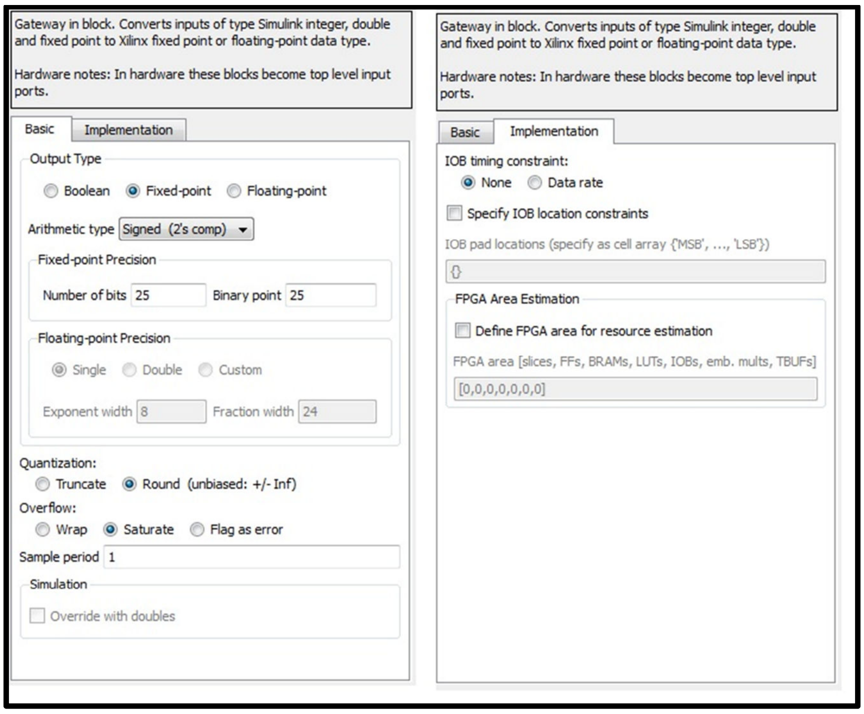
FPGA clock period (ns) : 10
Clock pin location :
Multirate implementation : Clock Enables
DCM input clock period (ns) : 10
☐ Provide clock enable clear pin
Simulink system period (sec) : 1

Figure A1: System Generator Block

Notes:

1. *This block is exclusively used for FPGA synthesis and implementation by using Matlab-Simulink.*
2. *The synthesis and implementation of a single DDS core, 8-core DDS module, and multi-frequency generator are completed by using the Xilinx VHDL compiler (ISE Project Navigator) instead of Matlab-Simulink.*

2. Setting of Gateway In Block



Gateway in block. Converts inputs of type Simulink integer, double and fixed point to Xilinx fixed point or floating-point data type.

Hardware notes: In hardware these blocks become top level input ports.

Basic | Implementation

Output Type
Boolean [✓ Fixed-point] Floating-point

Arithmetic type: Signed (2's comp)

Fixed-point Precision
Number of bits: 25 Binary point: 25

Floating-point Precision
[✓ Single] Double Custom
Exponent width: 8 Fraction width: 24

Quantization:
Truncate [✓ Round (unbiased: +/- Inf)]

Overflow:
Wrap [✓ Saturate] Flag as error

Sample period: 1

Simulation
☐ Override with doubles

Gateway in block. Converts inputs of type Simulink integer, double and fixed point to Xilinx fixed point or floating-point data type.

Hardware notes: In hardware these blocks become top level input ports.

Basic | Implementation

IOB timing constraint:
[✓ None] Data rate

☐ Specify IOB location constraints

IOB pad locations (specify as cell array {'MSB', ..., 'LSB'})
{}

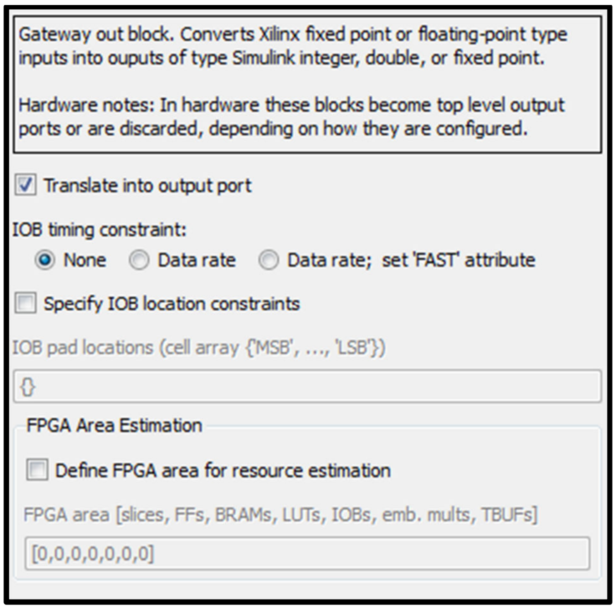
FPGA Area Estimation
☐ Define FPGA area for resource estimation

FPGA area [slices, FFs, BRAMs, LUTs, IOBs, emb. mults, TBUFs]
[0,0,0,0,0,0,0]

Figure A2: Gateway In Block

Notes: This block is applied to all the Matlab-Simulink simulations

3. Setting of Gateway Out Block



Gateway out block. Converts Xilinx fixed point or floating-point type inputs into ouputs of type Simulink integer, double, or fixed point.

Hardware notes: In hardware these blocks become top level output ports or are discarded, depending on how they are configured.

☑ Translate into output port

IOB timing constraint:
[✓ None] Data rate Data rate; set 'FAST' attribute

☐ Specify IOB location constraints

IOB pad locations (cell array {'MSB', ..., 'LSB'})
{}

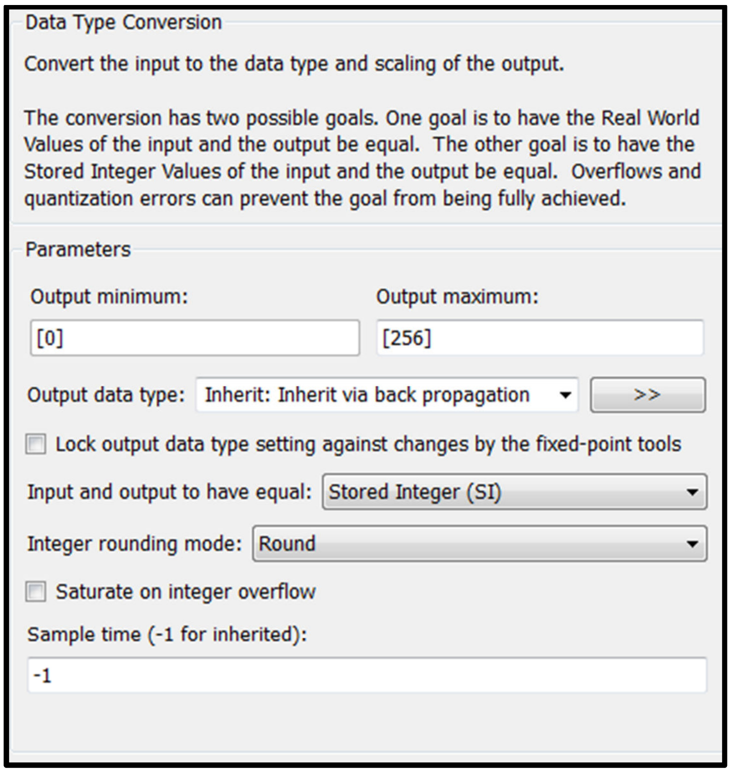
FPGA Area Estimation
☐ Define FPGA area for resource estimation

FPGA area [slices, FFs, BRAMs, LUTs, IOBs, emb. mults, TBUFs]
[0,0,0,0,0,0,0]

Figure A3: Gateway Out Block

Notes: This block is applied to all the Matlab-Simulink simulations

4. Setting of Data Type Conversion Block


Data Type Conversion

Convert the input to the data type and scaling of the output.

The conversion has two possible goals. One goal is to have the Real World Values of the input and the output be equal. The other goal is to have the Stored Integer Values of the input and the output be equal. Overflows and quantization errors can prevent the goal from being fully achieved.

Parameters

Output minimum: [0]

Output maximum: [256]

Output data type: Inherit: Inherit via back propagation

☐ Lock output data type setting against changes by the fixed-point tools

Input and output to have equal: Stored Integer (SI)

Integer rounding mode: Round

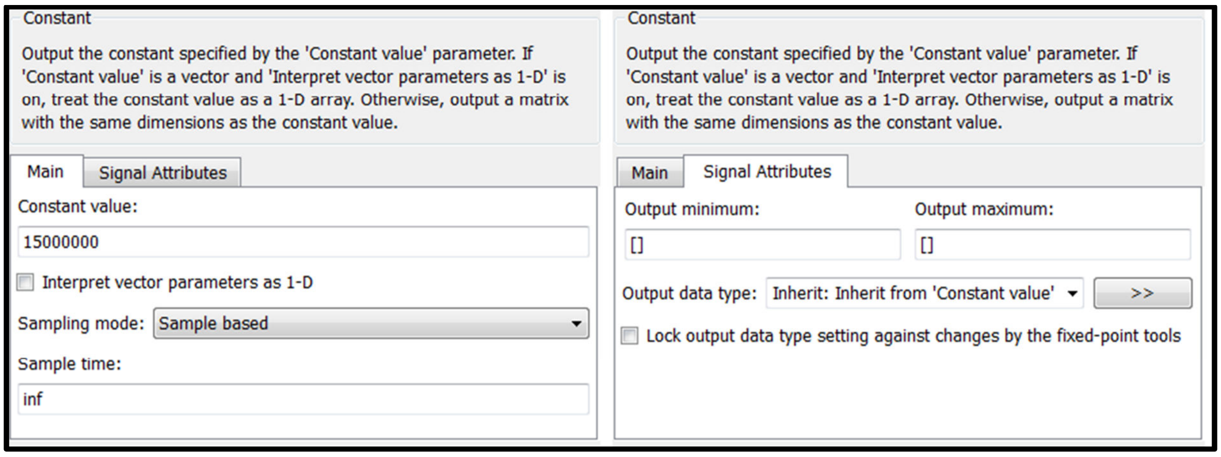
☐ Saturate on integer overflow

Sample time (-1 for inherited): -1

Figure A4: Data Type Conversion Block

Notes: This block is applied to all the Matlab-Simulink simulations

5. Constant Block


Constant

Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.

Main | Signal Attributes

Constant value: 15000000

☐ Interpret vector parameters as 1-D

Sampling mode: Sample based

Sample time: inf

Constant

Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.

Main | Signal Attributes

Output minimum: []

Output maximum: []

Output data type: Inherit: Inherit from 'Constant value'

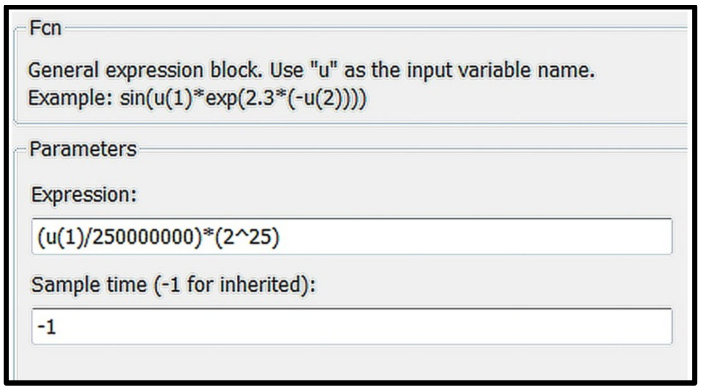
☐ Lock output data type setting against changes by the fixed-point tools

Figure A5: Constant Block (Constant 3)

Notes:

1. *The constant value is set to the desired value as shown in Matlab-Simulink of a single DDS core, 8-Core DDS module and Multi-frequency RF generator.*
2. *Other settings are remained default.*

6. Function Block


Fcn

General expression block. Use "u" as the input variable name.
Example: sin(u(1)*exp(2.3*(-u(2))))

Parameters

Expression:

(u(1)/250000000)*(2^25)

Sample time (-1 for inherited):

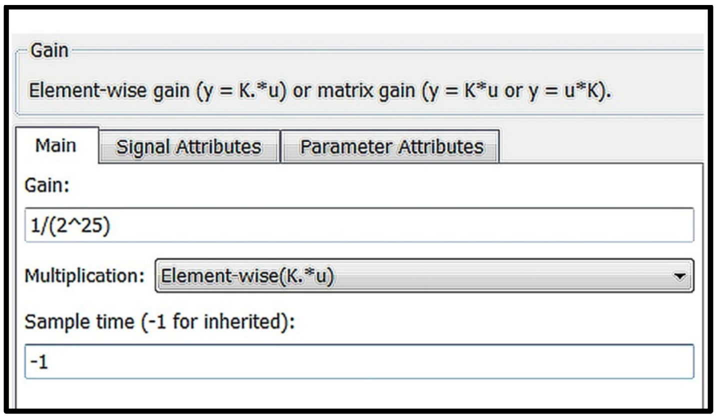
-1

Figure A6: Function Block (Fcn1)

Notes:

1. *The equation in the expression is used to calculate the frequency tuning word.*
2. *The setting of this block is applied to Matlab-Simulink of a single DDS core, 8-Core DDS module, and Multi-frequency RF generator.*

7. Gain Block


Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Gain:

1/(2^25)

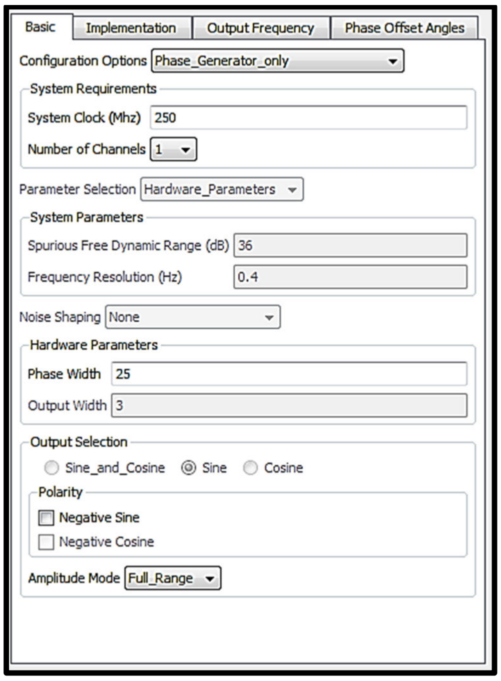
Multiplication: Element-wise(K.*u)

Sample time (-1 for inherited):

-1

Figure A7: Gain Block

8. Phase Accumulator



Basic | Implementation | Output Frequency | Phase Offset Angles

Configuration Options: Phase_Generator_only

System Requirements
System Clock (Mhz): 250
Number of Channels: 1

Parameter Selection: Hardware_Parameters

System Parameters
Spurious Free Dynamic Range (dB): 36
Frequency Resolution (Hz): 0.4

Noise Shaping: None

Hardware Parameters
Phase Width: 25
Output Width: 3

Output Selection
Sine_and_Cosine [✓ Sine] Cosine

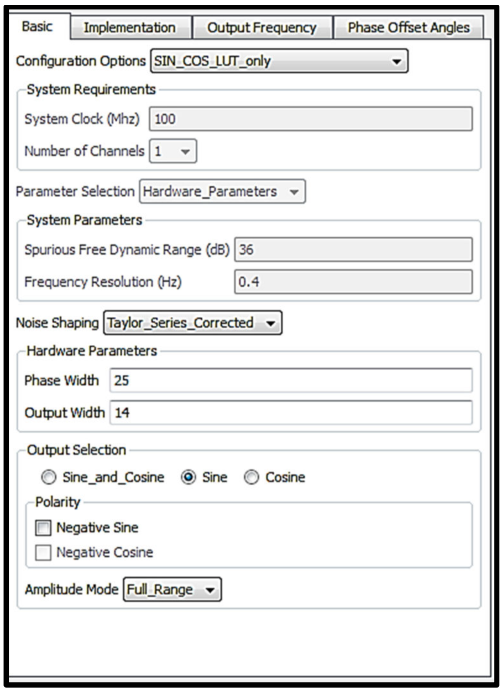
Polarity
Negative Sine
Negative Cosine

Amplitude Mode: Full_Range

Figure A8: Phase Accumulator

Note: 'Implementation', 'Output Frequency', and 'Phase Offset Angle' tabs are remained default.

9. Look-up Table



Basic | Implementation | Output Frequency | Phase Offset Angles

Configuration Options: SIN_COS_LUT_only

System Requirements
System Clock (Mhz): 100
Number of Channels: 1

Parameter Selection: Hardware_Parameters

System Parameters
Spurious Free Dynamic Range (dB): 36
Frequency Resolution (Hz): 0.4

Noise Shaping: Taylor_Series_Corrected

Hardware Parameters
Phase Width: 25
Output Width: 14

Output Selection
Sine_and_Cosine [✓ Sine] Cosine

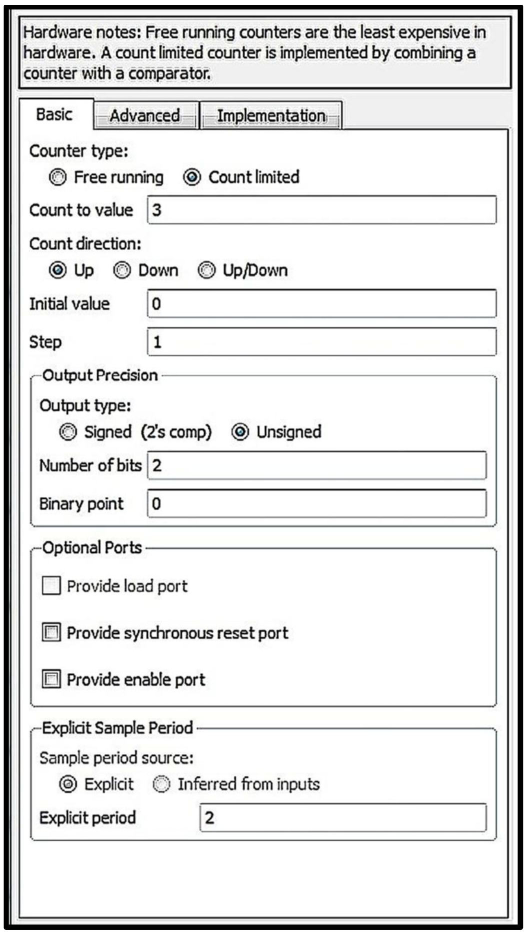
Polarity
Negative Sine
Negative Cosine

Amplitude Mode: Full_Range

Figure A9: Look Up Table

Note: 'Implementation', 'Output Frequency', and 'Phase Offset Angle' tabs are remained default.

10. Counter in 2 GHz Sine LUT block


Hardware notes: Free running counters are the least expensive in hardware. A count limited counter is implemented by combining a counter with a comparator.

Basic | Advanced | Implementation

Counter type:
Free running [✓ Count limited]

Count to value: 3

Count direction:
[✓ Up] Down Up/Down

Initial value: 0

Step: 1

Output Precision
Output type:
Signed (2's comp) [✓ Unsigned]
Number of bits: 2
Binary point: 0

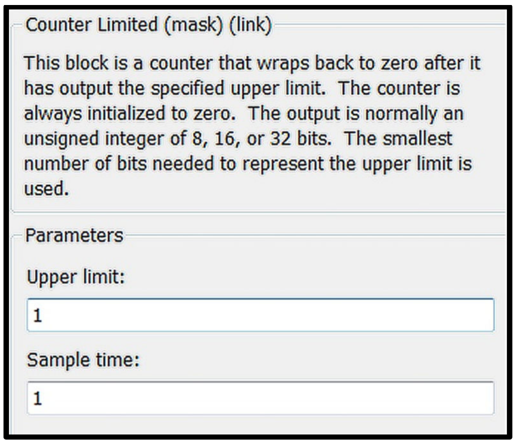
Optional Ports
☐ Provide load port
☐ Provide synchronous reset port
☐ Provide enable port

Explicit Sample Period
Sample period source:
[✓ Explicit] Inferred from inputs
Explicit period: 2

Figure A10: Counter

Notes: Only applicable to Sine LUTs of 8-core DDS module and Multi-frequency RF Generator

11. Counter in Mock DAC


Counter Limited (mask) (link)

This block is a counter that wraps back to zero after it has output the specified upper limit. The counter is always initialized to zero. The output is normally an unsigned integer of 8, 16, or 32 bits. The smallest number of bits needed to represent the upper limit is used.

Parameters

Upper limit: 1

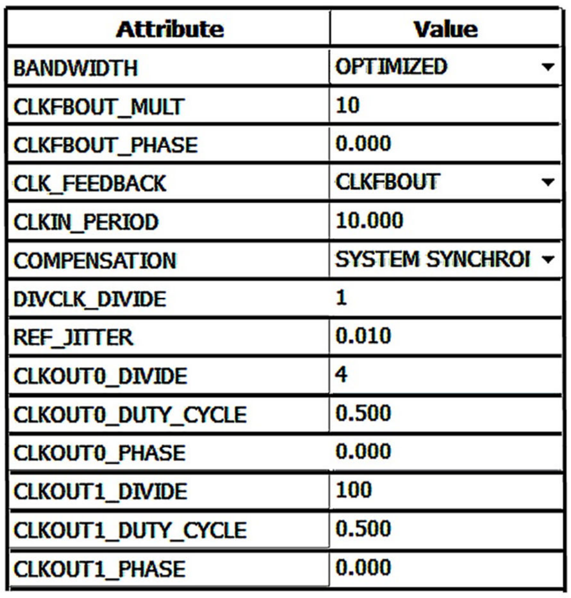
Sample time: 1

Figure A11: Counter

Appendix B:

Parameter Settings of Xilinx IP Cores

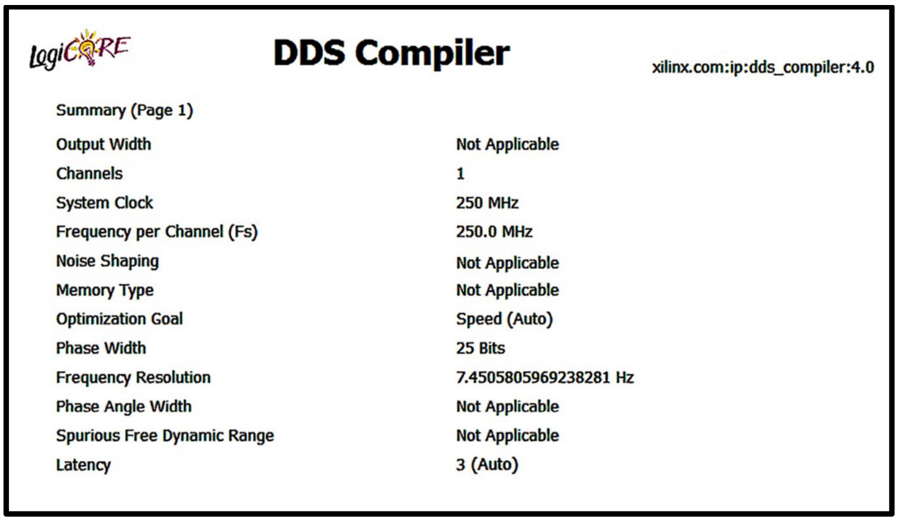
1. PLL_Base



Attribute	Value
BANDWIDTH	OPTIMIZED
CLKFBOUT_MULT	10
CLKFBOUT_PHASE	0.000
CLK_FEEDBACK	CLKFBOUT
CLKIN_PERIOD	10.000
COMPENSATION	SYSTEM SYNCHROI
DIVCLK_DIVIDE	1
REF_JITTER	0.010
CLKOUT0_DIVIDE	4
CLKOUT0_DUTY_CYCLE	0.500
CLKOUT0_PHASE	0.000
CLKOUT1_DIVIDE	100
CLKOUT1_DUTY_CYCLE	0.500
CLKOUT1_PHASE	0.000

Figure B1: Page 3 of Page 6

2. DDS Core - Phase Accumulator



DDS Compiler

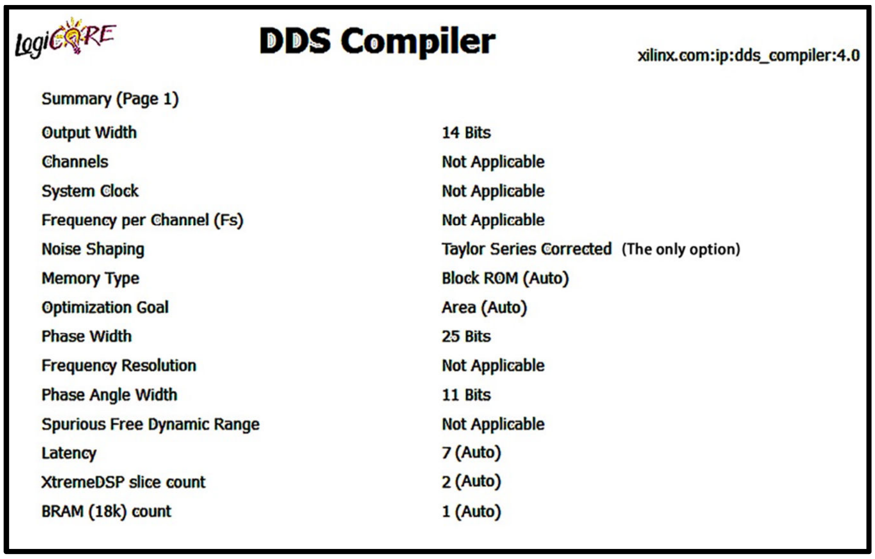
xilinx.com:ip:dds_compiler:4.0

Summary (Page 1)

Output Width	Not Applicable
Channels	1
System Clock	250 MHz
Frequency per Channel (Fs)	250.0 MHz
Noise Shaping	Not Applicable
Memory Type	Not Applicable
Optimization Goal	Speed (Auto)
Phase Width	25 Bits
Frequency Resolution	7.4505805969238281 Hz
Phase Angle Width	Not Applicable
Spurious Free Dynamic Range	Not Applicable
Latency	3 (Auto)

Figure B2: Phase Accumulator

3. DDS Core - LUT



DDS Compiler

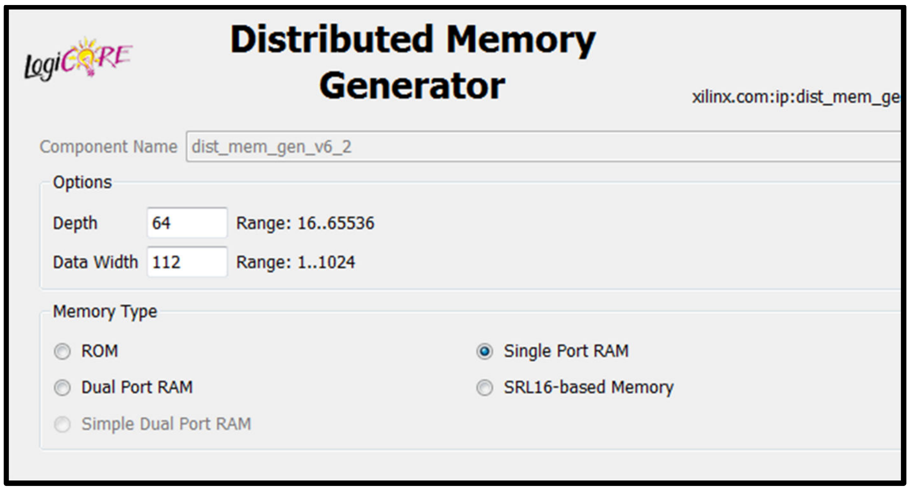
xilinx.com:ip:dds_compiler:4.0

Summary (Page 1)

Output Width	14 Bits
Channels	Not Applicable
System Clock	Not Applicable
Frequency per Channel (Fs)	Not Applicable
Noise Shaping	Taylor Series Corrected (The only option)
Memory Type	Block ROM (Auto)
Optimization Goal	Area (Auto)
Phase Width	25 Bits
Frequency Resolution	Not Applicable
Phase Angle Width	11 Bits
Spurious Free Dynamic Range	Not Applicable
Latency	7 (Auto)
XtremeDSP slice count	2 (Auto)
BRAM (18k) count	1 (Auto)

Figure B3: LUT

4. Distributed Memory Block



Distributed Memory Generator

xilinx.com:ip:dist_mem_ge

Component Name: dist_mem_gen_v6_2

Options

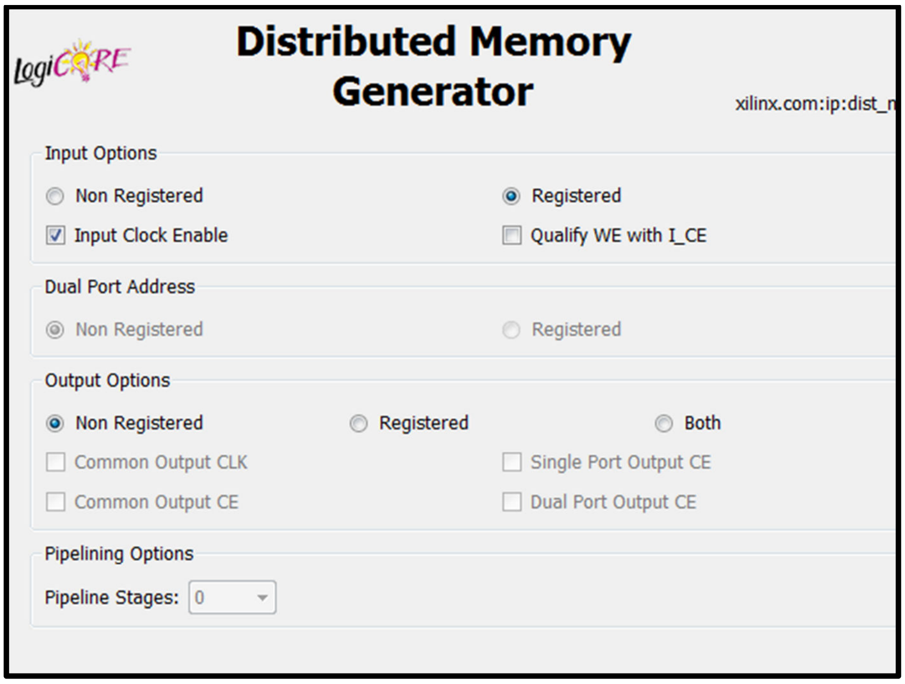
Depth: 64 Range: 16..65536

Data Width: 112 Range: 1..1024

Memory Type

- ○ ROM
- ● Single Port RAM
- ○ Dual Port RAM
- ○ SRL16-based Memory
- ○ Simple Dual Port RAM

Figure B4: Page 1 of Page 3



Distributed Memory Generator

xilinx.com:ip:dist_m

Input Options

- ( ) Non Registered
- (●) Registered
- [✓] Input Clock Enable
- [ ] Qualify WE with I_CE

Dual Port Address

- (●) Non Registered
- ( ) Registered

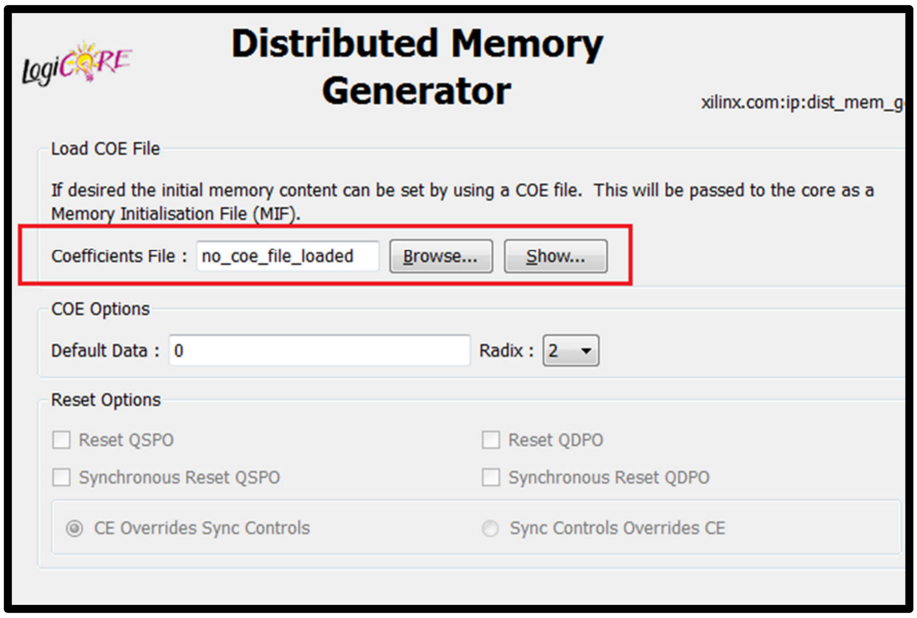
Output Options

- (●) Non Registered
- ( ) Registered
- ( ) Both
- [ ] Common Output CLK
- [ ] Single Port Output CE
- [ ] Common Output CE
- [ ] Dual Port Output CE

Pipelining Options

Pipeline Stages: 0

Figure B5: Page 2 of Page 3



Distributed Memory Generator

xilinx.com:ip:dist_mem_g

Load COE File

If desired the initial memory content can be set by using a COE file. This will be passed to the core as a Memory Initialisation File (MIF).

Coefficients File : no_coe_file_loaded

COE Options

Default Data : 0

Radix : 2

Reset Options

- [ ] Reset QSPO
- [ ] Reset QDPO
- [ ] Synchronous Reset QSPO
- [ ] Synchronous Reset QDPO
- (●) CE Overrides Sync Controls
- ( ) Sync Controls Overrides CE

Figure B6: Page 3 of Page 3

Note:

1. *The coefficient file is loaded to the highlighted section.*

References

[1] Nobelprize.org (1944), "Isidor Isaac Rabi- Biography", Retrieved 26/1/2012 from http://www.nobelprize.org/nobel_prizes/physics/laureates/1944/rabi-bio.html

[2] The Nobel Prize in Physics 1952 Felix Bloch, E. M. Purcell (1952), Retrieved 26/1/2012, from http://www.nobelprize.org/nobel_prizes/physics/laureates/1952/

[3] Agilent Technologies (April 11, 2011), "Development of 1-GHz NMR Spectrometer- Collaboration with Bridge12 Technologies on DNP Technology", Retrieved 20/8/2011 from http://www.agilent.com/about/newsroom/presrel/2011/11apr-ca11027.html

[4] Agilent Technologies, "Waveform Phase Continuity", Retrieved 1/3/2012 from http://wireless.agilent.com/wireless/helpfiles/n5106a/waveform_phase_continuity.htm

[5] Analog Devices (1999), "A Technical Tutorial on Digital Signal Synthesis", Retrieved 5/3/2010 from http://www.analog.com/static/imported-files/tutorials/450968421DDS_Tutorial_rev12-2-99.pdf

[6] Analog Devices (2001), "Datasheet: AD6620 67 MSPS Digital Receive Signal Processor", Retrieved 20/3/2011 from http://www.analog.com/static/imported-files/data_sheets/AD6620.pdf

[7] Analog Devices (2006), "Data Sheet: AD9736, 14 bit, 1200 MSPS DAC", Retrieved 26/5/2010 from http://www.analog.com/static/imported-files/data_sheets/AD9734_9735_9736.pdf

[8] Analog Devices (2009), "Datasheet: AD9739, 14 Bit, 2500 MSPS RF DAC", Retrieved 12/10/2011 from http://www.analog.com/static/imported-files/data_sheets/AD9739.pdf

[9] Analog Devices (2009), "MT-085: Fundamentals of Direct Digital Synthesis (DDS)", Retrieved 7/3/2013 from http://www.analog.com/static/imported-files/tutorials/MT-085.pdf

[10] Andraka, R. (1998), "A Survey of CORDIC algorithms for FPGA based Computers", In ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, pp. 1-10.

[11] Simoes, B., Cardoso, J.M. , Cruz, N. , Correia, and C.M.B.A. . (1999), "A Windows CE stand-alone digital spectrometer", In Nuclear Science Symposium, pp. 260 – 262.

[12] Cardells-Tormo.F., and Valls-Coquillat.J. (2002), "High Performance Quadrature Digital Mixers for FPGA", In 12th International Conference on Field-Programmable Logic and Applications", pp. 905-914.

[13] Cardells-Tormo.F., and Valls-Coquillat.J. (March, 2003), "Area-Optimized Implementation of Quadrature Direct Digital Frequency Synthesizers on LUT-Based FPGAs", In IEEE Transaction on Circuit and System: Analog and Digital Processing, Vol.50, No.3, pp. 135-138.

[14] Chang, Y.K., and Swartzlander, E.E. (May, 2006), "Digit- Pipelined Direct Digital Frequency Synthesis Based on Differential CORDIC", IEEE Transaction on Circuits and Systems, Vol. 53, No.5, pp. 1035-1044.

[15] Dipayan, M., and Kadami, G. R. (2012), "Method and Apparatus for Direct Digital Synthesis of Signals Using Taylor Series Expansion", PCT Patent Application PCT/IB2010/054721, 2010.

[16] Fukushima, E., and Roeder. S.B.W. (1981), Experimental Pulse NMR: A Nuts and Bolts Approach. Westview Press.

[17] Filler, A. (2009), "The History, Development and Impact of Computed Imaging in Neurological Diagnosis and Neurosurgery: CT, MRI, and DTI", Internet Journal of Neurosurgey", Vol. 7, No.1.

[18] Gao, S., Cao, S., and Zhang, Y. (May 22-23,2010), "Sinusoidal Pulse Width Modulation Design Based DDS", In International Workshop on Intelligent Systems and Applications (ISA), pp. 1-4.

[19] Herveille, R. (January 2, 2012), "CORDIC Core Specification", Retrieved 21/2/2012 from http:// www.opencores.org

[20] Hu, X., Harber, R.G., and Bass, S.C. (1988), "Expanding the Range of Convergence of the CORDIC Algorithm", In IEEE TRANSACTIONS ON COMPUTERS, Vol.40 No.1, pp. 13-21.

[21] Kallstrom, P. (2010, May 2), "Field-programmable gate array", Retrieved 9/4/2013, from http://en.wikipedia.org/wiki/File:FPGA_cell_example.png

[22] Kerr. R.J., Weaver,L.A (1988), "High Resolution Phase-to-Sine Amplitude Conversion", PCT Applicatin Number: PCT/US1988/004622

[23] Lauterbur, P.C. and Liang, Z. (2000), "Principles of Magnetic Resonance Imaging", IEEE Press Wiley Inter-Science.

[24] Lin, C., and Wu, A. (November, 2005), "Mixed-Scaling-Rotation CORDIC (MSR-CORDIC) Algorithm and Architecture for High-Performance Vector Rotational DSP Applications", IEEE Transactions on Circuits and Systems, Vol. 52, No.11, pp. 2385-2396.

[25] Liu, Y., and Li, G. (May 16-18, 2008), "A Microprocessor-Based Nuclear Magnetic Resonance Spectrometer", In International Conference on Bioinformatics and Biomedical Engineering, pp. 2264 – 2265.

[26] Liu, Z., Zhao, C., Zhou, H., and Feng, H. (August 30 - September 3, 2006), "A Novel Digital Magnetic Resonance Imaging Spectrometer", In IEEE conference on Engineering in Medicine and Biology Society, pp. 280 - 283.

[27] Magritek (2012), "Kea Specifications", Retrieved 17/8/2012 from Magritek: http://www.magritek.com/products-kea-specifications

[28] Maxfield, C. (2004), "The Design Warrior's Guide to FPGAs", Newnes.

[29] McClellan,J.H., Schafer, R.W, and Yoder, M.A. (2003), "Signal Processing First", Prentice Hall.

[30] McDonald, J.S. (2002), "Quick HDL User's and Reference Manual", Retrieved 8/9/2012 from http://paws.kettering.edu/~mcdonald/class/ce422/

[31] Noto, S., and Sarmento, H. (2008), "From Matlab/Simulink to ISE", Retrieved 27/5/2010 from http://prosys.inesc-id.pt/UWBR/Reports/UWBR-matlab-ise.pdf

[32] Rabi,I.I., Zacharias, J.R., Millman, S. and Kusch, P. (1938), "A New Method of Measuring Nuclear Magnetic Moment", Physical Review, Vol. 53, No. 4, pp. 318.

[33] Punk, O., Döhler, S., and Heuert, U. (May 12-16, 2010), "Modern device concepts – digital part of a NMR spectrometer", In International Seminor on Electronics Technology (ISSE), pp. 386-389.

[34] Qi, H. (2001), "ECE310 - Introduction to Signal and System", Department of Engineering and Computer Science - The University of Tennessee, Retrieved 9/10/2012, from http://web.eecs.utk.edu/~qi/teaching/ece310s01/

[35] Samuel Ginn College of Engineering, "ELEC4200 Lecturer Note: FPGA Overview", Auburn University, Retrieved 15/5/2013 from http://www.eng.auburn.edu/~strouce/class/elec4200/FPGAoverview.pdf

[36] Sawyer, N. (2010, June 3), "Application Note XAPP1064 (v1.1): Source-Synchronous Serialization and Deserialization", Retrieved 2/6/2010 from http://www.xilinx.com/support/documentation/application_notes/xapp1064.pdf

[37] Smith III, J.O., and Cook, P.R. (1992), "The Second-Order Digital Waveguide Oscillator", International Conference on Computer Music, pp. 1-6.

[38] SpinCore Technologies, "RadioProcessor: Complete RF Acquisition and Excitation System with Digital Detection, Real-Time Signal Processing Signal Averaging" Retrieved 11/4/2011 from http://www.spincore.com/products/RadioProcessor/RadioProcessor.shtml

[39] SpinCore Technologies, " PulseBlasterDDS: Programmable TTL and DIrect Digital Synthesis RF Pulse Generator", Retrieved 13/4/2011 from http://www.spincore.com/products/PulseBlasterDDS-300/PulseBlasterDDS-300.shtml

[40] Takeda, K. (March, 2007), "A highly integrated FPGA-based nuclear magnetic resonance spectrometer". Review of Scientific Instruments, Vol.78 , No. 3, pp. 033103-6.

[41] Takeda, K. (June, 2008), "OPENCORE NMR: Open-source core modules for implementing an integrated FPGA-based NMR spectrometer", Journal of Magnetic Resonance 192, Vol. 192 , No.2, pp. 218-229.

[42] Texas Intruments (March 29, 2012), "LMK04800 Family Low-Noise Clock Jitter Cleaner with Dual Loop PLLs", Retrieved 28/10/2012 from http://www.ti.com/lit/ds/symlink/lmk04800.pdf

[43] Turner, C. S. (May, 2003), "Recursive Discrete-Time Sinusoidal Oscillators", Journal of IEEE Signal Processing Magazine, Vol.20, No.3, pp. 103-111.

[44] Valls, J., Sansaloni, T., Perez-Pascual, A., Torres, V., and Almenar, V. (September, 2006), "The Use of CORDIC in Software Defined Radios: A tutorial", Journal of IEEE Communication Magazine, Vol.44, No.9, pp. 46-50.

[45] Vanderwerff, M. (2010), "Schematic of T-Rex V3.0", Beta Solution.

[46] Vanderwerff, M. (2011), "Shcematic of T-Rex V4.0", Beta Solution.

[47] Vanderwerff, M (2009), "T-ReX V2.0 Specifications (REV A)", Magritek/ Beta Solutions LTD.

[48] Vankka, J. (November 2000), "Direct Digital Synthesizers: Theory, Design and Applications", Doctorate Thesis, Helsinki University of Technology.

[49] Vladimirova, T., and Tiggeler, H. (1998), "FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm", In Military and Aerospace Applications of Programmable Devices and Technologies Conference, pp. 28-30.

[50] Volder, J.E. (September, 1959), "The CORDIC Trigonometric Computing Technique", IRE Transaction on Electronic Computers, Vol. 8, No.3, pp. 330-334.

[51] Williston, K. (2009), "Digital Signal Processing: World Class Designs", Newnes.

[52] Xilinx (April 28, 2005), "DS246: Logic Core IP DDS v5.0", Retrieved 13/9/2012 from http://www.xilinx.com/support/documentation/ip_documentation/dds.pdf

[53] Xilinx (April 28, 2005), "DS275: Sine/Cosine Look-Up Table v5.0", Retrieved 12/7/2012 from http://www.xilinx.com/support/documentation/ip_documentation/sincos.pdf

[54] Xilinx (June 24, 2009), "UG381(v1.0):Spartan-6 FPGA SelectIO Resources", Retrieved 12/9/2010 from http://www.xilinx.com/support/documentation/user_guides/ug381.pdf

[55] Xilinx. (February 23, 2010), "UG384:Spartan-6 FPGA Configurable Logic Block", Retrieved 12/5/2013 from http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

[56] Xilinx (October 5, 2011), "DS160:Spartan-6 Family Overview", Retrieved 12/4/2013 from http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[57] Xilinx (October 17, 2011), "DS162 (v3.0): Spartan-6 FPGA Data Sheet: DC and Switching Characteristics", Retrieved 21/5/2010 from http://www.xilinx.com/support/documentation/data_sheets/ds162.pdf

[58] Xilinx (March 1, 2011), "DS249: LogiCore IP CORDIC V4.0", Retrieved 27/6/2012 from http://www.xilinx.com/support/documentation/ip_documentation/cordic_ds249.pdf

[59] Xilinx (March 1, 2011), "DS558: LogiCORE IP DDS Compiler v4.0", Retrieved 12/8/2011 from http://www.xilinx.com/support/documentation/ip_documentation/dds_ds558.pdf

[60] Xilinx (October 19, 2011), "DS858: LogiCORE IP CORDIC v5.0" Retrieved 20/12/2011 from http://www.xilinx.com/support/documentation/ip_documentation/cordic/v5_0/ds858_cordic.pdf

[61] Xilinx (July 20, 2012), "UG382 (v1.7):Spartan-6 FPGA Clocking Resources", Retrieved 25/7/2012 from http://www.xilinx.com/support/documentation/user_guides/ug382.pdf

[62] Xilinx (November 5, 2012), "UG474 (v1.4):7 Series FPGAs Configurable Logic Block", Retrieved 7/5/2013 from http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[63] Zhang, X., Xiao, L., Anferova, S., Anferov, V., and Yu, H. (2011), "Development of a Portable Sectrometer Based on OPENCORE NMR",In International Conference on Magnetic Resonance Micoscopy Joint Topic Conference on NMR in Well Logging and Core Analysis, pp. P60.

[64] Zeidman, B. (March 21, 2006), "All About FPGAs", Programmable Logic Deisng Line, Retrieved 5/4/2013 from http://www.pldworld.com/html/technote/pldesignline/allaboutfpgas-bobz.htm

[65] Zhang. T, Zhao. J, and Chen, D. (September 9-11, 2011), "A Realization of WiMAX Digital Intermediate Frequency Based on FPGA", In International Conference on Electronics, Communications and Control (ICECC), pp. 1853-1856.

[66] Zhang, Z., and Dong, F. (May 16-28, 2010), "A harmonic signal generator based on DDS and SOPC", In Control and Decision Conference (CCDC), pp. 1542 – 1547.

[67] Mini-circuits (2013), “Datasheet: RF Transformer TC1-33-75G2+”, Retrieved 7/4/2013 from http://www.minicircuits.com/pdfs/TC1-33-75G2+.pdf

[68] Mini-circuits (2013), “Datasheet: Ceramic Low Pass Filter, 50 ohms, DC-490 MHz, LFCN-490+”, Retrieved 15/4/2013 from http://www.minicircuits.com/pdfs/LFCN-490.pdf