DESIGN AND DEVELOPMENT OF AN FPGA BASED DDFS SIGNAL GENERATOR

BY

WALTER MAINA MUTEITHIA

I56/72200/2008

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Science in Physics, University of Nairobi.

APRIL 2014

Declaration

This work is an original thesis submitted for the Degree of Master of Science (M.Sc) in the Department of Physics, University of Nairobi, and has not been submitted before for examination at any other institution/University.

Walter Maina Muteithia

I56/72200/2008

Department of Physics, University of Nairobi.

Signature……. Date…….

This thesis is submitted with our approval as University of Nairobi Supervisors

Mr. Mjomba A.C. Kale.

Department of Physics, University of Nairobi.

Signature……. Date…….

Dr. Kenneth. A. Kaduki

Department of Physics, University of Nairobi.

Signature……. Date…….

Dedicated to my family

ACKNOWLEDGEMENT

I am grateful to my supervisors Mr. Mjomba A.C. Kale and Dr. Kenneth Amiga Kaduki for their guidance and support throughout the research period. I am thankful to all my friends and colleagues who contributed to my thesis with their continuous encouragement. I would also like to express my profound appreciation to my family for their continuous support.

Abstract

Over time there has been an increase in speed and density of Field Programmable Gate Arrays (FPGAs), this has enabled more complex designs to be constructed within a short time frame. In addition, the flexibility of FPGA devices has also eased the integration of a design with a wide variety of components on a single chip. The aim of this study was to design and implement an FPGA based direct digital frequency synthesizer (DDFS) signal generator. The focus was on spectral purity improvement.

Since phase and amplitude samples in a DDFS are represented using a finite word length, the output signal of a DDFS is usually faced by a spectral purity challenge. The details of this challenge and how to deal with it is also covered in this thesis. The design flow used in this work entailed modeling and simulation at the software level using SystemC and prototyping in hardware using Actel's fusion field programmable gate array (FPGA).

The resulting FPGA prototype had spurious free dynamic range (SFDR) improved from 48 dBc to 85 dBc and a noise floor of -116 dBc. Four signal types could be generated: sine, square, saw tooth and triangle. The frequency resolution was 0.047 Hz and the maximum output frequency was 25 MHz. Therefore, due to its high frequency resolution and spectral purity the proposed signal generator design can be useful in performing a wide range of laboratory experiments.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
DAC	Digital to Analog Converter
dB	decibel
dBc	decibels with respect to the carrier
DDFS	Direct Digital Frequency Synthesizer
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IDE	Integrated Development Environment
IP	Intellectual Property
LC	Inductor Capacitor
LFSR PN	Linear Feedback Shift Register Pseudo-noise
LPF	Low Pass Filter
LUT	look up table
MHz	Megahertz
mHz	milihertz
NF	Noise Floor
OSCI	Open SystemC Initiative
FFT	Fast Fourier Transform
PLL	Phase Locked Loop
PSM	Process State Machine
RC	Resistor Capacitor
ROM	Read Only Memory
RTL	Register Transfer Level
SFDR	Spurious Free Dynamic Range
SNR	Signal to Noise Ratio
UART	Universal Asynchronous Receiver / Transmitter
VCD	Value Change Dump
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integrated Circuit

GPIO	General Purpose Input Output

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Preamble

A signal generator is an electronic instrument that generates repeating voltage waveforms. This important device finds a wide range of applications in any electronics laboratory, such as characterizing analogue and digital systems. In order for a signal generator to be suitable for a wide range of purposes, it should provide a wide frequency range, high frequency resolution, high spectral purity and the ability to generate different types of waveforms. A variety of methods can be employed for generating waveforms in signal generators, these include resistor capacitor (RC) oscillator, inductor capacitor (LC) oscillator, multivibrators, direct digital frequency synthesizers (DDFS), phase locked loops (PLL) and other variations of these methods. The DDFS has been chosen for use in this study because it provides many significant advantages over the other approaches. Some of these benefits are that DDFS designs are tunable to many different frequencies with the use of a constant operating frequency, it has a fast settling time, Sub-hertz frequency resolution, continuous phase-switching response, low phase noise and its implementation allows for a standalone precise, fast frequency changing device capable of generating different types of waveforms (Vankka, 2001).

The DDFS technique of waveform generation can be implemented using several technologies. This includes the use of a microcontroller (MCU) (Popa and Sorana, 2007), an application specific integrated circuit (ASIC) (Analog devices, 2003), a digital signal processor (DSP) (Sia et al., 2007) and a field programmable gate array (FPGA) (Sharma and Upadhyaya, 2010). An FPGA offers several advantages over the other technologies

such as: the ability to integrate a large part of the DDFS based signal generator in a single chip, reconfigurability and the ability to implement parallel circuits which operate at a high speed. In addition, it allows design decisions to be made when sufficient information is available. The purpose of this study was to design and implement an FPGA based DDFS signal generator that has optimized spectral purity. The optimization investigation was carried out through modeling and simulation techniques based on SystemC.

1.2 Problem Statement

The direct digital frequency synthesizer (DDFS) is a method of signal generation that has many advantages. DDFS designs are able to generate signals that have high frequency resolution, provide precise frequency control and fast frequency switching. However, the generation of high frequency resolution signals in a DDFS necessitates the use of a large phase accumulation register and subsequent truncation of the resulting phase information so that a smaller and fast memory can be used. Due to the truncation of phase information, the output signals of a DDFS usually contain spurs (unwanted frequency components). Additional spurs are introduced if truncation of amplitude information is done so that narrower data paths and coarse resolution digital to analog converters can be used. These spurs cause a reduction in spectral purity of the output signals. Accordingly there is a need for the inclusion of a mechanism to reduce the level of spurs in the design of a DDFS. One of the ways of reducing the spurs of a DDFS is by using the dithering technique of spur reduction; it involves the addition of a low-level random noise, or dither signal to the amplitude or the phase samples. Some of the methods of dithering available include phase information dithering, amplitude

information dithering and a combination of both phase and amplitude dithering. This work intended to design and implement a high frequency resolution DDFS based signal generator, whose output signal quality has been optimized using the most appropriate method of dithering for the proposed DDFS.

1.3 Aim

The aim of this work was to design and implement on FPGA a spectral purity optimized DDFS based signal generator.

1.4 Specific Objectives

(1) To investigate the effect of the following methods of dithering on the spurious free dynamic range and signal to noise ratio of the proposed DDFS signal generator:

- **a)** Phase dithering only
- **b)** Amplitude dithering only
- **c)** Combination of phase and amplitude dithering

(2) To compare the results obtained in (1) and select a dithering method that offers the optimal spurious free dynamic range and signal to noise ratio for the proposed DDFS.

(3) To implement on FPGA a DDFS based signal generator whose spectral purity has been optimized using the dithering method selected in (2).

1.5 Justification and Significance of the study

FPGAs enable complex designs to be implemented within a short time frame in addition to facilitating the integration of a wide variety of a design's components in a single chip. These benefits have the potential to result in a low cost and a fast to implement signal generator like the one proposed in this study.

1.6 Thesis organization

The organization of this thesis is as follows. In chapter 2 previous work related to DDFS implementation technologies, modeling and simulation tools and spectral purity optimization methods is reviewed. Chapter 3 covers some background on how a DDFS based signal generator operates, spectral purity of DDFS signals and the dithering spectral purity improvement method. Chapter 4 presents the design, modeling, simulation of the DDFS signal generator using SystemC and results of simulating the models. Chapter 5 presents FPGA implementations of the DDFS signal generator and results of testing the FPGA prototype. Chapter 6 presents the conclusion, summary and recommendation.

CHAPTER 2

LITERATURE REVIEW

This chapter reviews previous publications related to DDFS implementation technologies, sine wave DDFS spectral purity improvement methods and modeling and simulation tools.

2.1 DDFS implementation technologies

The DDFS technique of waveform generation can be implemented using several technologies. This section will examine previous implementations reported in the literature; microcontroller, an application specific integrated circuit (ASIC), a digital signal processor (DSP) and a field programmable gate array (FPGA).

(Popa and Sorana, 2007) describe a programmable signal generator implemented using the pulse width modulation (PWM) method on the advanced 32-bit microcontroller family. It is capable of generating sine, pulse, saw tooth and custom waveforms. Output frequency range is between 1 Hz and 600 Hz. One major disadvantage of using a microcontroller for the implementation of a DDFS based signal generator is that every additional instruction executed by the microprocessor reduces the maximum output frequency of the design; because of the sequential execution of instructions.

(Analog devices, 2003) is an Application Specific Integrated Circuit (ASIC) technology application note from Analog Devices; this document exhaustively discusses the AD9833 DDFS ASIC which is a low power, programmable waveform generator. Among other things it presents the features, possible applications and a detailed description of how to make use of the AD9833. The frequency range of this DDS IC is 0

MHz to 12.5 MHz. The drawback of ASICs is that they typically take months to fabricate and cost hundreds of thousands to millions of dollars to obtain the first device.

(Sia et al., 2007) present a digital-signal-processor-based waveform generator. The device can generate a sine wave up to 24 kHz, a square wave, up to 5 kHz and a triangular wave of up to 12 kHz. Due to the fact that a DSP is a highly specialized microprocessor, it can impose a low limit on the maximum signal generator output frequency because the processor must use shared resources like memory busses, or even the processor core which can be prevented from taking interrupts for some time.

Although signal generator designs utilizing the DDFS technique of waveform generation can be implemented using a microcontroller, digital signal processor and an application specific integrated circuit, this thesis proposes the use of an FPGA because it offers the following advantages over the other technologies (Dubey, 2009);

- **Reconfigurability**: Field programmable devices can be reconfigured at anytime. Designers can add modifications or do complete behavior changes.
- **Parallelism**: Circuits implemented in an FPGA can be designed in a totally parallel manner. This is analogous to using multi-path analogue circuits. A user can instantiate numerous hardware implementations on the same chip without cross-module interference or computation loading.
- **High speed**: Because an FPGA is a hardware implementation running with rapid clock rates, designers can achieve very high speeds. Coupled with parallelism, FPGA implementation can do better than processor-based systems.

2.2 Modeling and Simulation tools

From the examination of previous work it is evident that modeling and simulation of a DDFS can be done using the MATLAB/Simulink environment (Vankka, 2001), (Chimakurthy et al., 2006) and (Kamboj and Mehra, 2012) or hardware description languages (HDLs) such as VHDL (Xiaoqin and Yin, 2007). Although modeling and simulating designs using the MATLAB/Simulink environment or HDLs is a possible option, this work proposes a different tool; the SystemC language (Open SystemC initiative [online], 2012). The major reason for using SystemC is that SystemC is capable of offering an order of magnitude faster simulation for abstract models (Agostinelli et al, 2010), (Vachoux and Grimm, 2003). In addition MATLAB supports behavioral modeling and simulation, while HDLs tend to support architectural modeling and simulation. SystemC supports both.

2.3 Spectral purity optimization methods

The spectral purity of a signal generated by a DDFS can be measured using two quantities, which are the spurious free dynamic range (SFDR) and the noise floor (NF). The noise floor (NF) is commonly understood as the average (sometimes also maximal) power of random noise (i.e. the noise that is freed of any harmonic, spurious and DC components) in frequency spectrum. The value of the SNR determines the noise floor level, in decibels with respect to full-scale (dBfs); the relationship is as follows (Slepička, 2000):

$$NF\ (dBfs) = -\ SNR\ (dBfs) = -\ (6.02m + 1.76)\ dB \qquad 2.1$$

Where m is the word length of the output amplitude word in bits.

SFDR is the difference between the carrier amplitude level (which is the desired signal) and the maximum level of spurs in the output spectrum of a DDFS. SFDR of a DDFS can be calculated using the equation (Cordeses, 2004 (part 1)):

SFDR = 6.02k - 3.92 dB 2.2

Where k is the number of phase bits.

SFDR of a DDFS can be improved by one of the techniques covered in (Vankka, 2001), (Cordeses, 2004 (part 1)) and (Cordeses, 2004 (part 2)), which include:

a) Increasing the size in bits of the phase information.

b) Using the odd-number spur reduction technique.

c) Using the dithering spur reduction technique.

d) Using the noise shaping spur reduction.

2.3.1 Increasing the size in bits of the phase information

The SFDR of a DDFS is related to the number of phase bits as follows: SFDR = 6.02k dBc, where k is the number of phase bits, increasing the size in bits of the phase information is the easiest method of increasing the SFDR of a DDFS. It is stated in (Cordeses, 2004 (part 1)) that a value of 9 phase bits would be ideal. However, for larger values the memory requirements would become impractical at high frequency or for embedded system applications, because the size of a look up table in a DDFS depends exponentially on the number of phase bits and linearly on the number of amplitude bits (Flanagan and Zimmerman, 1995). This can be expressed by the equation:

$$w = m * 2^k \quad 2.3$$

Where:

w is the size in bits of the look up table

m is the size in bits of the amplitude word

k is the number of phase bits used by the look up table

2.3.2 The odd-number approach

The odd-number approach involves making the phase increment word (ΔP) an odd number; this improves SFDR by a maximum of 3.9 dB according to reference (Cordeses, 2004 (part 2)).

2.3.3 The noise shaping approach

The noise shaping approach improves the SFDR of a DDFS by filtering out the quantization noise. Noise shaping can be applied to phase or amplitude signals. The noise shaping approach is usually faced with a difficulty of implementing analog filters with variable pass bands (Vankka, 2001).

2.3.4 Dithering

Dithering is a technique that allows the decrease of phase or amplitude word length without escalating spur magnitudes by first adding a low-level random noise, or dither signal to the phase and/or the amplitude samples, which are at first expressed in a longer word length. The resultant sum, a dithered phase or amplitude value, is truncated or rounded to the smaller, preferred word length (Flanagan and Zimmerman, 1995).

Selection of a spectral purity optimization technique for this work

In this study the dithering technique was selected as the suitable SFDR optimization technique, the main reasons being that the dithering technique offers a larger spur reduction (12 dB per phase bit) than the odd number approach (fixed 3.9 dB). In addition dithering does not require the use of analog filters with changeable pass bands (they are difficult to implement) unlike the noise shaping approach.

Studies in (Vankka, 2001) on dithering indicate that the expense of phase dithering is an increased noise floor while the penalty of amplitude dithering is a reduced dynamic range and a loss of the amplitude information caused by the need to scale the amplitude information so that the original signal plus the dither will stay within the non-saturating region. In (Flanagan and Zimmerman, 1995) these drawbacks are mentioned to be a small price to pay for the large increase in SFDR offered by the dithering technique.

2.4 Summary

The DDFS was first proposed by J. Tierney in 1971. Since then there has been major developments in;

- DDFS implementation technologies.
- Modeling and simulation tools.
- Spectral purity improvement methods.

This work sought to design and implement a DDFS based signal generator that benefits from the advances in the above three areas. From the review of recent work FPGA has been identified as the most appropriate implementation technology because it offers the ability to integrate a large part of the design in one chip. SystemC was used for

modeling and simulation due to its relatively faster simulation time while dithering was selected for spur reduction because of its capability to offer a large improvement in spectral purity.

CHAPTER 3
THEORETICAL BACKGROUND

This chapter presents the basic aspects of a direct digital frequency synthesizer (DDFS), such as the DDFS technique of signal generation, waveform generation, frequency resolution and range, factors affecting the spectral purity of DDFS signals and the dithering spectral purity improvement method.

3.1 Direct Digital Frequency Synthesizer (DDFS) Theory

The DDFS was first proposed by J. Tierney in 1971 (Vankka, 2001). A typical DDFS system uses a fixed reference clock, a phase accumulator, phase to amplitude converter, digital to analog converter and smoothing filter to generate a constant frequency signal (Flanagan and Zimmerman, 1995). This work focused on the phase accumulator and phase to amplitude converter presented in Figure 3.1. In the figure ΔP is the phase increment word, j is the number of phase accumulator bits, f_{clk} is the clock frequency, k is the number of phase bits used as address for the phase to amplitude converter and m is the word length of the amplitude word.

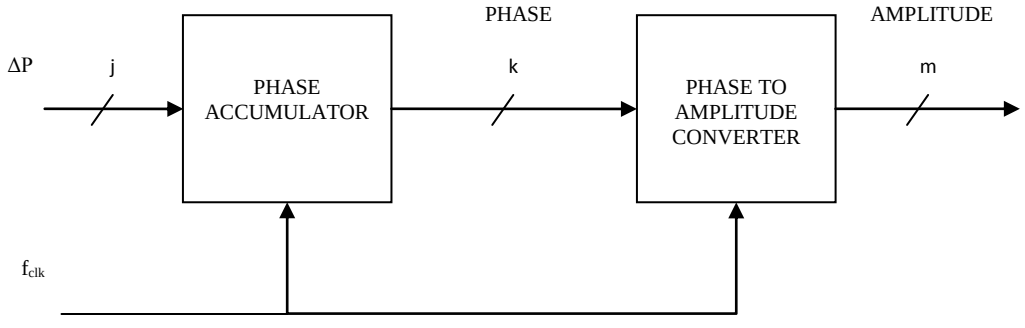


Figure 3.1: Basic Direct Digital Frequency Synthesizer

3.1.1 Phase Accumulator

The phase accumulator is made up of a *j*-bit frequency register that stores a digital phase increment word followed by a *j*-bit full adder and a phase register. The digital input phase increment word is entered in the frequency register. At each clock cycle the phase increment value is added to the data previously held in the phase register; this results in the production of a linearly rising digital value.

The frequency of the data generated by the phase accumulator depends on the reference clock frequency, the phase increment register value and length of phase accumulator as shown in equation 3.1. From the equation it can be inferred that increasing the phase increment for a constant clock frequency and size of phase accumulator results in an increase in output frequency.

$$f_{out} = \frac{\Delta P * f_{clk}}{2^{j}} \qquad 3.1$$

Where:

ΔP is the phase increment word

j is the number of phase accumulator bits

f_{clk} is the clock frequency

f_{out} is the output frequency

The frequency resolution, maximum output frequency and frequency range can also be obtained from equation 3.1 in the following way:

Frequency Resolution

Frequency resolution refers to the smallest step in frequency that a DDFS can achieve. It is a function of the reference clock frequency and number of bits employed in

phase accumulator. For a fixed reference clock and size of the phase accumulation register, the frequency resolution can be calculated using equation 3.2; this equation is derived from equation 3.1 by setting $\Delta P = 1$. Digit one is the minimum value that ΔP can assume because it is an integer. In order to have an improved frequency resolution for a fixed clock frequency, the number of bits employed in the phase accumulator can be increased.

$$\Delta f = \frac{f_{clk}}{2^{j}} \qquad 3.2$$

Where:

Δf is the frequency resolution

j is the number of phase accumulator bits

f_{clk} is the clock frequency

Maximum output frequency

The highest frequency that a DDFS can produce digitally is determined by its sampling frequency; increasing the sampling frequency increases the maximum output frequency. The Nyquist Theorem states that the highest frequency which can be generated accurately is less than half of the sampling rate. As a result, the highest frequency that can be generated by a DDFS module is:

$$F_{O_{\max}} = \frac{f_{clk}}{2} \qquad 3.3$$

Where:

F_{Omax} is the maximum output frequency

f_{clk} is the clock frequency

Equation 3.3 can also be obtained from equation 3.1 by setting $\Delta P = 2^{j-1}$. Using equation 3.3 the maximum output frequency can only be increased by increasing the clock frequency.

Frequency Range

The output frequency range of a DDFS based signal generator is determined by the frequency resolution of the DDFS and its maximum output frequency. The frequency resolution determines the lower limit of the frequency range while the maximum output frequency sets the upper limit. Thus from equations 3.2 and 3.3 used in calculating frequency resolution and maximum output frequency respectively, the frequency range of a DDFS can be increased by increasing the number of phase accumulator bits and or the clock frequency.

3.1.2 Phase to amplitude converter

The phase to amplitude converter is a periodic wave look up table (LUT) which converts the phase accumulator value to an amplitude value. This periodic wave lookup table is generally implemented using a read only memory (ROM) which stores the periodic waveform samples. The look up table maps the full scale of the phase value output by the phase accumulator to one cycle of a periodic wave. As the phase value increases from 0 to full scale, one periodic wave is created.

The phase to amplitude converter can also be implemented without a ROM; in this case the amplitude values are computed; some examples of these techniques include

sine-phase difference algorithm, Sunderland techniques, first order Taylor series expansion and higher order Taylor series expansion. The simplicity of the ROM circuit makes the ROM LUT easier to implement. In addition a sine ROM LUT has been shown in (Vankka, 2001) to provide a better SFDR than any ROMless architecture for same bit width.

3.2 Generation of different wave shapes

The common waveforms that a DDFS based signal generator can produce include: Sine, Square, Saw tooth and triangle. The following section describes how each of these waveforms is generated.

3.2.1 Sine Wave

The generation of a sine wave in a DDFS based signal generator can be done using the phase accumulator and a phase to amplitude converter that has a full sine wave samples stored in its look up table (LUT). The phase accumulator generates phase values for the sine wave while the phase to amplitude converter uses the phase values as address for the look up table. Every time the phase accumulator register overflows, sampled values of a full sine wave are generated. The samples are then converted to analog form using a digital to analog converter. Samples used to create this kind of waveform can be generated using the equation:

$$Amplitude(i) = (\sin\left(i * 2 * \frac{\pi}{2^k}\right)) * 2^{m-1} \qquad 3.4$$

Where:

Amplitude(i) is the corresponding amplitude value for the i^{th} phase value, i

assumes values between 0 and 2^k- 1.

sin is the sine function

k is the number of phase bits used in the phase to amplitude conversion

m is the size of the amplitude word in bits

From equation 3.4 it can be observed that increasing the value of k will cause an increase in the number of samples stored in the LUT while increasing m will result in an increase in bits of the amplitude resolution of the sine samples.

3.2.2 Square Wave

The generation of a square wave in a DDFS based signal generator does not entail a lot of effort because that waveform is already available as the most significant bit of the phase accumulator.

The generation of a square wave in a DDFS based signal generator can also be done using the phase accumulator and a phase to amplitude converter that has a full square wave stored in its look up table (LUT). The phase accumulator generates phase values for the square wave while the phase to amplitude converter uses the phase values as address for the look up table. Every time the phase accumulator register overflows, sampled values of a full square wave are generated. The samples are then converted to analog form using a digital to analog converter. Samples used to create this kind of waveform can be generated by setting the amplitude register at its maximum value if the phase accumulator is in the first half of the cycle and setting the amplitude register to its minimum value if the phase accumulator is in the second half of the cycle. Equations 3.5 and 3.6 summarize this relationship.

Square(i) is the maximum value of the 2^m amplitude register ;for i <= 2^{j-1} 3.5

Square(i) is the minimum value of the 2^m amplitude register ;for i > 2^{j-1} 3.6

Where:

Square(i) is the amplitude value of the square wave for the i^{th} phase value, i can only assume values between 0 and 2^{j-1}.

m is the number of amplitude bits

j is the number of phase bits used

From equation 3.5 and 3.6 it can be inferred that increasing j will cause an increase in the number of samples stored in the LUT while increasing m will cause an increase in bits of the square wave amplitude resolution.

3.2.3 Saw tooth Wave

The output of the phase accumulator in a DDFS is usually a linearly increasing digital value generated by using the modulo 2^j overflowing property of a *j*-bit phase accumulator. This property allows the output of the phase accumulator to be used for the generation of a saw tooth waveform in a DDFS based signal generator. The output sequence of the phase accumulator is usually given by:

$$P(n) = (P(n-1) + \Delta P) mod 2^j \quad 3.7$$

Where:

P(n) is the phase register value at the *n*th clock period (present value of the phase accumulation register).

P(n-1) is the phase register value at the *n-1* clock period (previous value of the phase accumulation register).

ΔP is the phase increment word.

The digital values generated by the phase accumulator can be converted to analog form using a digital to analog converter.

The generation of a saw tooth wave in a DDFS based signal generator can also be done using the phase accumulator and a phase to amplitude converter that has a full saw tooth wave stored in its look up table (LUT). The phase accumulator generates phase values for the saw tooth wave while the phase to amplitude converter uses the phase values as address for the look up table. Every time the phase accumulator register overflows, sampled values of a full saw tooth wave are generated. The samples are then converted to analog form using a digital to analog converter. Samples used to create this kind of waveform can be generated by using the equation of a straight line that has a slope of 1. Equations 3.8 summarizes this relationship.

$$Sawtooth(n) = p(n) \qquad 3.8$$

Where:

Sawtooth(n) is the amplitude value of the saw tooth wave for the n^{th} phase register value.

p(n) is the n^{th} output of the phase accumulation register, n can only assume values between 0 and 2^{j-1}. The value of p(n) at any time is given by equation 3.7.

j is the number of phase bits used

From equation 3.7 and 3.8 it can be inferred that increasing j will cause an increase in the number of samples stored in the LUT and an increase in bits of the amplitude resolution of the saw tooth wave.

3.2.4 Triangle Wave

Triangle waveform can be generated by performing logic operations on the output of the phase accumulator. This process involves inverting all values below one half of the full scale.

The generation of a triangle wave in a DDFS based signal generator can also be done using the phase accumulator and a phase to amplitude converter that has a full triangle wave stored in its look up table (LUT). The phase accumulator generates phase values for the triangle wave while the phase to amplitude converter uses the phase values as address for the look up table. Every time the phase accumulator register overflows, sampled values of a full triangle wave are generated. The samples are then converted to analog form using a digital to analog converter. For the two mentioned methods of creating a triangle wave, samples used to create the waveform can be generated by using the equations of two lines with opposite slope. Equations 3.9 and 3.10 summarize this relationship.

$$triangle(i) = g * p(i) + c \; ; for \; i \leq 2^{j-1} \qquad 3.9$$

$$triangle(i) = -g * p(i) + c \; ; for \; i > 2^{j-1} \qquad 3.10$$

Where:

triangle (i) is the amplitude value of the triangle wave for the i^{th} phase value.

p(i) is the i^{th} phase value, i can only assume values between 0 and 2^j-1.The value of p(i) at any time is given by equation 3.7.

g is the gradient of the line.

j is the number of phase bits used.

c is the y intercept.

From equation 3.9 and 3.10 it can be inferred that increasing j will cause an increase in the number of samples stored in the LUT and an increase in bits of the amplitude resolution of the triangle wave.

3.3 Spectral purity of DDFS signals

The quality of a signal generated by a DDFS is determined by the following (Vankka, 2001):

1) Truncation of the phase accumulator bits addressing the read only memory (ROM).
2) Distortion from compressing the ROM.
3) The finite precision of the binary word stored in the ROM.
4) Digital-to-analog conversion.
5) Post-filter error.
6) Phase noise of the clock frequency and the frequency error. The frequency error causes a frequency offset, but not noise and spurs.

In this thesis attention will be focused on truncation of the phase accumulator bits addressing the sine ROM and the finite precision of the sine samples stored in the ROM. The main reason for this course of action is that the spectral purity of a signal generated by a DDFS largely depends on digital errors (truncation and quantization) (Vankka, 2001).

3.3.1 Phase truncation

Phase truncation occurs when the phase information is reduced from j to k bits as

shown in Figure 3.2. The reason behind this truncation is to keep the memory requirements of the phase to amplitude converter low.

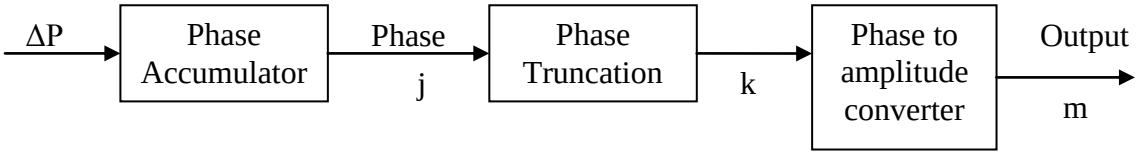


Figure 3.2: Simplified DDFS block diagram with phase quantization

Phase truncation is a vital feature of DDFS designs. Consider a DDFS with a 30-bit phase accumulator. To directly translate 30 bits of phase to matching amplitude would require 2^{30} entries in a lookup table. If each entry were stored with 16 bit precision, then 2-gigabytes of lookup table memory would be necessary. Such a huge lookup table would lead to high power utilization, lesser speed and significantly increased costs. The solution is to utilize a fraction of the most significant bits of the Phase accumulator output to offer phase information. For example, in a 30-bit DDFS design, only the upper most 8 bits might be used for phase information. The lower 22 bits would be truncated in this case.

Regrettably, the phase errors introduced by truncating the accumulator result in errors in amplitude during the phase-to-amplitude conversion process inbuilt in the DDFS. Since these amplitude errors are cyclic in the time domain, they emerge as line spectra (spurs) in the frequency domain and are what is known as phase truncation spurs (Xinguang et al., 2009). The maximum spur level power for a phase truncated DDFS is approximated by equation 3.11. A detailed derivation of this formula is covered in (Vankka, 2001):

$$S_{max} = -6.02k \text{ dB} \qquad 3.11$$

Where:

S_{max} is the maximum spur power level

k is the number of bits in the phase information after quantization

dB = decibels

The difference between the carrier power level (which is the desired signal) and the maximum power level of spurs is called spurious free dynamic range (SFDR) (Cordeses, 2004 (part 1)); SFDR should be large for spectrally pure sinusoids. Using this definition of SFDR and equation 3.12, SFDR can be calculated as follows when the carrier level is 0 dB

$$\text{SFDR} = 0 - (-6.02k)$$

$$= 6.02k \text{ dBc} \qquad 3.12$$

Where dBc means decibels with respect to the carrier (tuning word frequency)

3.3.2 Amplitude Quantization

Amplitude quantization occurs when the output of the phase to amplitude converter is represented using finite resolution for instance m bits, with m being the word length of the output amplitude word. Amplitude quantization results in a quantization error and gives rise to an effect known as quantization distortion. In the frequency domain, quantization distortion errors appear as discrete spurs in the DDFS signal output spectrum. The relationship between the amplitude resolution in bits and the carrier power (desired frequency) to spur power ratio as presented in (Vankka, 2001) is:

$$\frac{C}{S} = (1.76 + 6.02m) \text{ dBc} \qquad 3.13$$

Where:

m is the word length of the output amplitude word

C is the carrier power

S is the spur power

3.4 Phase dithering

Truncating phase information in a DDFS leads to an error in the phase information which in turn results in phase truncation spurs in the synthesizers' output spectrum. Phase dithering is capable of suppressing these spurs by breaking up the regularity of the phase error with an additive randomizing signal. The result is a higher spurious free dynamic range (SFDR). Phase dithering is accomplished by adding a dither signal to each phase value generated by the phase accumulator as shown in Figure 3.3. In the figure b is the number of bits of the dither signal, j is the size of the phase information in bits before truncation while k is the size of the phase information in bits after truncation.

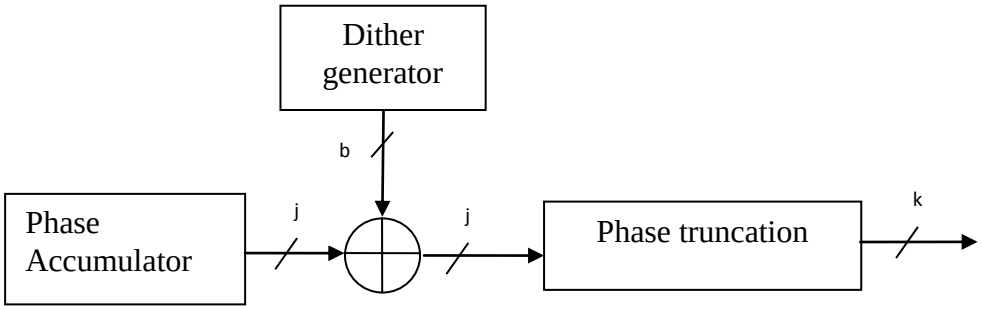


Figure 3.3: Addition of a dither signal to the phase information

The maximum spur power level relative to the desired signal after phase dithering as provided in (Vankka, 2001) is:

$$S_{max} = 7.84 - 12.04k \text{ dBc.} \qquad 3.14$$

Where:

S_{max} is the maximum spur power level

k is the number of bits remaining after truncating the phase accumulator word.

dBc is a unit representing decibels with respect to the carrier.

The constants 7.84 and 12.04 arise from the derivation of equation 3.14, which is presented in (Vankka, 2001)

From equation 3.14 it can be observed that phase dithering leads to spur attenuation because after dithering, the spurs follow a 12 dB per phase bit law instead of the 6 dB per phase bit of a DDFS without phase dithering given in equation 3.12.

The carrier-to-noise power spectral density after phase dithering as derived in (Vankka, 2001) is

$$\frac{C}{N} \cong (6.02k - 9.94 + 10\log_{10}(Pe))\,\text{dBc} \qquad 3.15$$

Where:

k is the number of bits remaining after truncating the phase accumulator word.

C is the carrier power

N is the noise power

dBc is a unit representing decibels with respect to the carrier.

Pe is the number of points in the output spectrum of the DDFS. It is equal to the numerical period of the phase accumulator; which is calculated using the equation.

$$Pe = \frac{2^j}{GCD(\Delta P, 2^j)} \qquad 3.16$$

Where:

ΔP is the phase increment word.

GCD (ΔP, 2^j) is the greatest common divisor of ΔP and 2^j.

j is the number of bits in the phase accumulator word before truncation.

The carrier-to-noise power spectral density in equation in (3.15) can be raised by increasing the number of the samples (*Pe*).

3.5 Amplitude dithering

Amplitude dithering involves the addition of a dither signal to the amplitude information as shown in Figure 3.4. This allows the amplitude information word length decrease without introducing additional spurs. Because after dithering the magnitude of the spurs will depend on the original (longer) word length and not the output (shorter) word length (Flanagan and Zimmerman, 1995). For amplitude dithering to be done the amplitude information stored in the look up table is usually reduced (scaled) so that the original signal plus the dither will stay within the non-saturating region. Scaling involves normalizing each of the b bit entries in the look up table so that the sinusoid amplitude equals $2^{(b-m)}$ b-bit quantization steps less than the full scale value (Flanagan and Zimmerman, 1995), where b is the number of bits representing the amplitude information before truncation and m is the number of bits used in the final output amplitude word after truncation. In Figure 3.4, x is the size of the dither signal in bits (x=b-m). The addition of a dither signal to the amplitude information does not lead to an increase in the number of amplitude bits as shown in Figure 3.4, this is possible because the amplitude information is scaled in order to make sure that the sum of the dither signal and the amplitude information does not exceed the maximum value that can be represented by the amplitude information register.

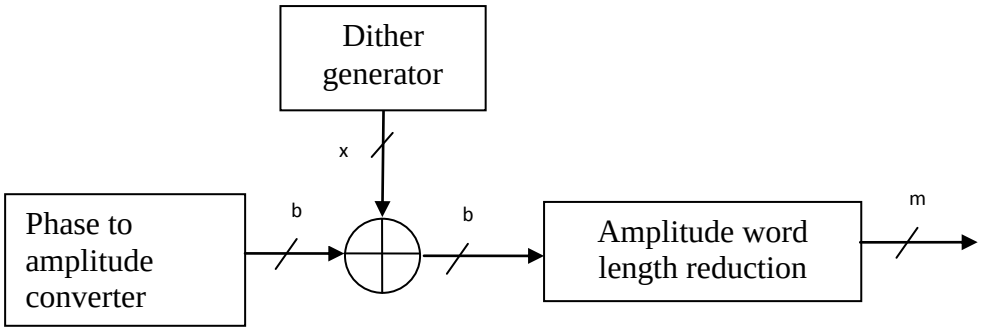


Figure 3.4: Addition of a dither signal to the amplitude information

This dithering technique works by spreading the spurs throughout the available bandwidth. The carrier-to-noise power spectral density after amplitude dithering is given as follows in (Vankka, 2001).

$$\frac{C}{N} \cong (1.76 + 6.02m + 10\log_{10}(\frac{Pe}{4}))\,\text{dBc} \qquad 3.17$$

Where:

m is the word length of the Output amplitude word.

Pe is the numerical period of the phase accumulator output sequence, it can be calculated using equation 3.16.

C is the carrier power.

N is the noise power.

3.6 Phase and amplitude dithering

The phase and amplitude method of dithering involves the addition of a dither signal to the phase and amplitude samples before truncation. The dither signals are added to the phase and amplitude samples in the manner described in section 3.4 and 3.5. The phase and amplitude method of dithering can be useful in reducing phase and amplitude quantization spurs.

3.7 Dither signal generation

Dither signal can be generated using a linear feedback shift register pseudo random number generator. A linear feedback shift register is a shift register whose input bit is a linear function of its previous state; it is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the shift register. The sequence of values generated by the register is completely determined by its current (or previous) state. Since a shift register has a finite number of possible states, the output sequence eventually repeats itself. In spite of this, an LFSR with a well selected feedback function can generate a sequence of bits which looks random and which has a very long cycle.

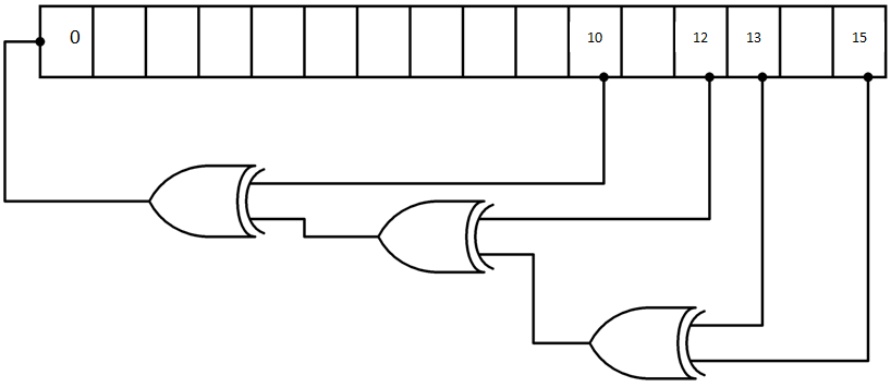


Figure 3.5: LFSR dither generator

The bit positions that affect the subsequent state are called the taps. In Figure 3.5 the taps are 10, 12, 13 and 15. The rightmost bit of the LFSR is called the output bit. The taps are XOR'd sequentially with the output bit and then fed back into the leftmost bit. The sequence of bits in the rightmost position is called the output stream. An output bus can also be formed by connecting the outputs of the entire register chain.

A maximum-length LFSR produces a y-sequence (i.e. it cycles through all possible $2^y - 1$ states within the shift register excluding the state where all bits are zero), if it contains all zeros, the output sequence will never change.

The arrangement of taps for feedback in an LFSR can be expressed in finite field arithmetic as a polynomial mod 2. This means that the coefficients of the polynomial must be 1's or 0's. This is called the feedback polynomial or characteristic polynomial. For example, if the taps are at the 15th, 13th, 12th and 10th bits (as shown), the feedback polynomial is

$$x^{15} + x^{13} + x^{12} + x^{10} + 1 \qquad 3.18$$

The 'one' in the polynomial does not correspond to a tap; it corresponds to the input to the first bit (i.e. x^0, which is equivalent to 1). The powers of the terms represent the tapped bits, counting from the left. The first and last bits are always connected as an input and tap respectively. A table of primitive polynomials from which maximum-length LFSRs can be constructed is given in (Xilinx, 2007).

The size in bits of a dither signal is usually given by the equation:

$$b = j - k \qquad 3.19$$

Where:

b is the size in bits of the dither signal

j is the number of bits in the phase accumulator or amplitude word before truncation.

k is the number of bits remaining after truncating the phase accumulator or amplitude word.

CHAPTER 4

SYSTEM DESIGN

This chapter covers design flow: specifications, modeling and simulation (verification) of the DDFS based signal generator using SystemC. Implementation and testing will be covered in chapter 5.

Modeling and simulation is an important part of the design process in electronic systems because it provides an early understanding of the design in addition to assisting in verifying that the design will function the way it was intended. In this work, modeling and simulation has also been used to examine the performance of a DDFS that employs the dithering technique of spur reduction. The objective was to identify a method of dithering that would result in the largest spur reduction. The models which were created and used for this investigation include: a model without dither signal, a model with phase dithering only, a model with amplitude dithering only and a model that had a combination of both phase and amplitude dithering. The performance figures used in comparing the four models are spurious free dynamic range (SFDR) and noise floor (NF); SFDR is used to indicate the level of spurs while NF indicates the level of noise of a system.

4.1 SystemC design flow

SystemC is a set of C++ classes and macros with event-driven simulation kernel in C++. These facilities enable a designer to simulate concurrent processes, each described using plain C++ syntax. SystemC processes can communicate in a simulated

real-time environment, using signals of all the data types offered by C++, some additional ones offered by the SystemC library, as well as user defined. SystemC is applied to:

- System-level modeling
- Architectural exploration
- Performance modeling
- Software development
- Functional verification
- High-level synthesis

Modules are the principal building blocks of a SystemC design hierarchy. A SystemC model usually consists of several modules which communicate through ports. Processes are the principal computation elements which fulfill necessary sequential behavior. They run concurrently with other processes. Events allow the synchronization between processes.

Ports of a module are the external interfaces that pass information to and from a module. They trigger actions within the module. Signals create the connections between the module ports allowing the modules to communicate. Channels are the communication elements of SystemC. They are generalized form of signals. Complex communication structures can be modeled using channels.

The major steps followed when carrying out modeling and simulation using SystemC include developing specifications for the models, creating the models and verifying them using simulation. Figure 4.1 shows the design flow that was followed when carrying out modeling and simulation using SystemC in this work.

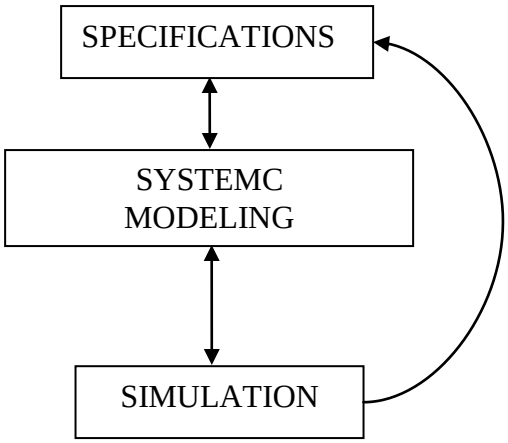


Figure 4.1: Design flow

4.1.1 Specifications

The first step in designing a SystemC model involves determining the specifications of the design. The design specifications of the proposed signal generator are given in section 4.3. The defined specifications contain necessary information about the types of waveforms, frequency resolution, frequency range and the size of the Look-up Table.

4.1.2 Modeling

After determination of the specifications, the structural and behavioral models of the signal generator are formed. In the structural model, the hierarchy of the modules and their interfaces are specified using a high level of abstraction. In the functional model, the processes and variables within the modules are specified.

To make a SystemC model, the designer writes the model in C++ using functions and data types defined in the SystemC class library following the methodology of describing a design in SystemC. The model, which is the executable specification, can then be compiled and linked to the SystemC simulation kernel and SystemC library to

create an executable (Bhasker, 2002). SystemC DDFS signal generator models used in this study are presented in section 4.6 to 4.9.

4.1.3 Verification

Once the behavioral models have been made, the next step involves verification of the models. Verification of a SystemC model is usually done to make sure that the model reflects the original intent of the design and that it performs efficiently, safely and successfully. Verification of models can be done using simulation based methods or formal methods (Gajski et al., 2009). Simulation based approach has been used in this study because SystemC only supports simulation. Simulation in this work was done by the use of a test bench. A test bench is a model that is used to exercise and verify the correctness of a design under test. A test bench has three main purposes.

a) To generate stimulus for simulation.

b) To apply the stimulus to the design under test and collect output responses.

c) To compare output responses with expected results.

In this study the test bench generates a frequency value for the signal generator module and receives output signals for the sine wave, square wave, triangle and saw tooth wave generated by the signal generator module. The generated and received signals are observed using GTK viewer. If the received signals have frequency corresponding to the value of the phase increment and expected shape, the signal generator module is considered capable of generating correct signals.

The test bench for this project was written using separate modules as shown in Figure 4.2.

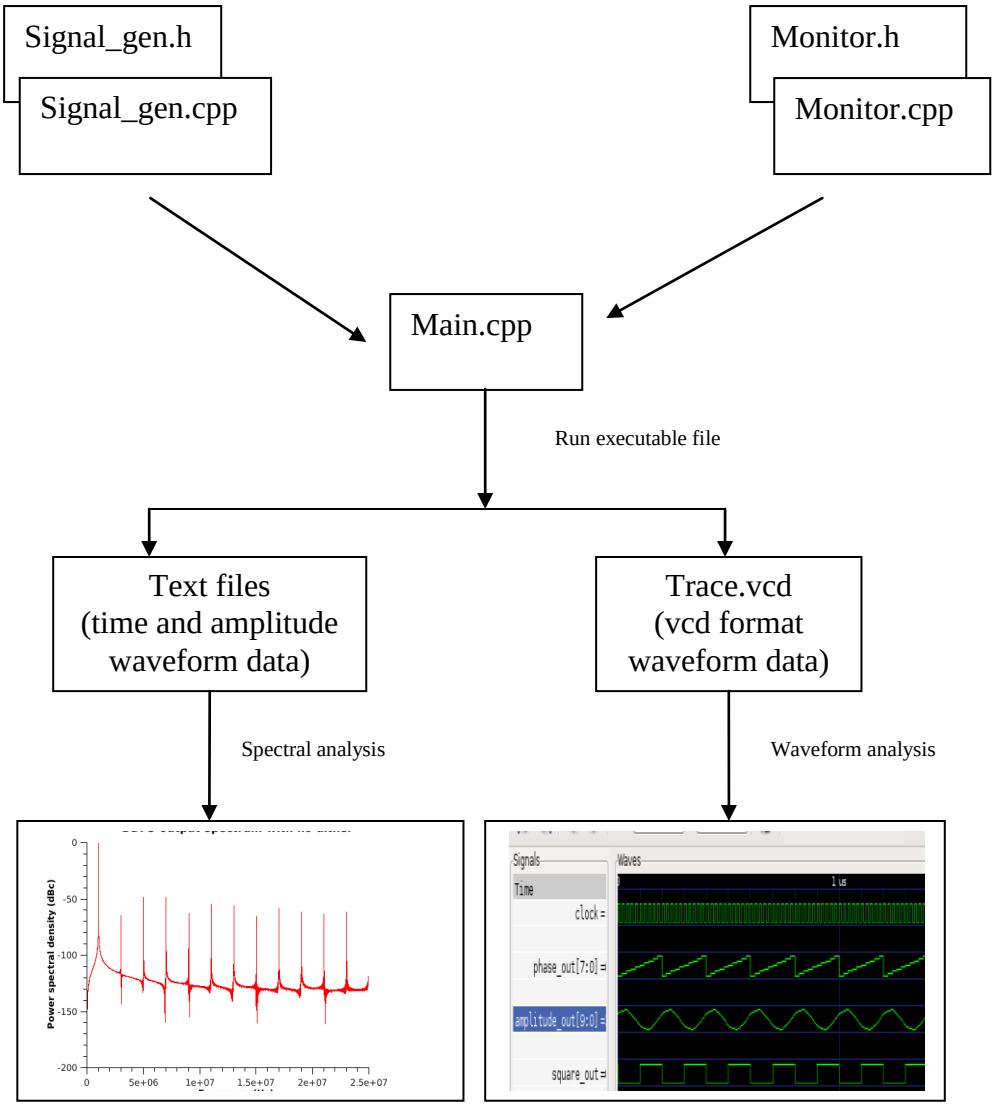


Figure 4.2: Verification Flow used for the SystemC Designs

The stimulus generation and the design under test were written in the module signal_gen.h and signal_gen.cpp. This is because the stimulus generation i.e. the driver serves the same purpose as the interface. Output monitoring and comparison was done in another module monitor.h and monitor.cpp. A main program main.cpp linked the various modules and interconnected them to form the testbench.

The sequence of events in the verification flow shown in Figure 4.2 is as follows: the main file was compiled and if there was no compilation error, the design project was

built. The design environment then generated an executable file. The executable file was run, resulting in the generation of a Value Change Dump (VCD) file and text files. The VCD file was used to hold the samples of simulated waveforms while the text files had sine, square and saw tooth wave form data. GTKWave Wave Analyzer V1.3.19 software was used to open the VCD file in order to observe simulation waveform. The results of this analysis are presented in section 4.11 for the signal generator module i.e. without dither signal, with phase dithering only, with amplitude dithering only and with a combination phase and amplitude dithering. QtiPlot data analysis tool was used to perform a Fast Fourier Transform (FFT) on the sine wave form data so that spectral information could be obtained. The results from this analysis are presented in section 4.11 for the four signal generator designs.

4.2 Software resources used in modeling and simulation

For this project SystemC version 2.2.0 was used together with Eclipse version 3.2.2 to carry out the modeling task. Eclipse is a multi-language software development platform. It consists of an Integrated Development Environment (IDE) with a flexible plug-in system. The IDE provides a source code editor with a rich set of source annotation and browsing capabilities, integrates a compiler, a source code debugger and many more facilities to aid the software development process. Eclipse's primary focus is the Java language, however with various plug-ins it addresses many other languages as well, such as C/C++, Cobol, Python, Perl and PHP. Eclipse's well defined plug in system makes it very attractive for customized extensions (Gajski et al., 2009).

4.3 DDFS signal generator specifications

The signal generator specifications that were used to create models of the DDFS based signal generator are presented in section 4.3.1 and 4.3.2. The specifications were arrived at after considering the need to have a device that can generate various waveforms of high resolution, high spectral purity, wide frequency range and optimal cost of resources. The specifications were placed under two categories i.e. functional and non-functional requirements. The functional requirements define what the signal generator is supposed to do while the non-functional requirements define the constraints of the signal generator.

4.3.1 Functional requirements

The proposed DDFS based signal generator is expected to meet the following requirements:

1. Receiving of signal parameters such as frequency, wave shape and amplitude.
2. Calculation of a phase increment value.
3. Continuous addition of the phase increment value to the phase accumulation register.
4. Generation of a dither signal to be used in phase dithering.
5. Addition of a dither signal to the phase information.
6. Conversion of phase information to amplitude information.
7. Generation of a dither signal to be used in amplitude dithering.
8. Addition of a dither signal to the amplitude information.

9. Generating common waveforms such as Sine Wave, Square Wave, Triangular Wave and Saw tooth Wave.
10. Facilitate the selection of output waveform.
11. Conversion of digital waveform samples into analog form.

4.3.2 Non-functional requirements

In order to meet the functional requirements stated in section 4.3.1, the constraints provided in table 4-1 were arrived at for the DDFS based signal generator. The specifications were made after identifying the maximum clock frequency of the used FPGA and the specifications of previous DDFS implementations such as a commercially available DDFS integrated circuit (Analog devices, 2003) and a microcontroller based signal generator (Silicon laboratories, 2003).

The FPGA that was used imposed three major constraints on the signal generator design: the size of lookup table, the clock frequency and the size of the phase accumulation register. Once these were set, they controlled most of the other constraints since:

- For a given lookup table size, there can be a tradeoff between phase quantization and amplitude quantization.
- For a given clock frequency there is an upper limit on the output frequency.
- The clock frequency and phase accumulation register size set the frequency resolution limit.

A more detailed explanation of the constraints is provided in section 4.3.3.

Table 4-1: DDFS signal Generator non-functional requirements

Frequency range	0.047 Hz to 25 MHz
Clock frequency	50 MHz
Number of Output Channel	1
Frequency Resolution	0.047 HZ
Phase word size after truncation	8-bit
Look-up Table Size	256 x 16 bit
Amplitude	0 to 1 V_{PP}
Spurious Free Dynamic Range (SFDR)	88 dBc
Output impedance	50 Ω

4.3.3 Explanation of DDFS signal generator specifications

The following section discusses the specifications presented in table 4-1.

4.3.3.1 Clock frequency and Upper frequency Limit

From theory,

$$F_{O_{\max}} = \frac{f_{clk}}{2} \qquad 4.1$$

Where:

$F_{O\,max}$ is the maximum output frequency

f_{clk} is the clock frequency

The maximum clock frequency limit imposed by the used FPGA was 50 MHz, hence using equation 4.1 the maximum output frequency expected therefore would be 25 MHz.

4.3.3.2 Types of waveforms and dithering

The proposed signal generator is expected to have a wide range of use hence four types of wave forms were selected for generation; sine, square, triangular and saw tooth. The square, triangular and saw tooth wave form were generated directly from the phase information as described in section 3.2. Direct generation of these waveforms was preferred because it would not impose additional memory requirements. The sine wave was generated from a look up table as described in section 3.2.1. The generation of a sine wave using a lookup table was preferred because it has been shown in (Vankka, 2001) to offer the best spectral purity over other techniques. The parameters of the used look up table and their justification is provided in section 4.3.3.4. Dithering was only applied to the sine wave due to the fact that the technique requires a phase to amplitude converter to be present.

4.3.3.3 Frequency resolution

Frequency resolution is influenced by the size of the phase accumulation register and the clock frequency as can be seen in the equation:

$$\Delta f = \frac{f_{clk}}{2^j} \qquad 4.2$$

where:

Δf is the frequency resolution

j is the number of phase accumulator bits

fclk is the clock frequency

For a standard 32 bit register, 2 bits were reserved for waveform selection and the remaining 30 bits were used for the phase accumulation. Since the maximum clock frequency of the used FPGA is 50MHz, the frequency resolution is:

$$\Delta f = \frac{50000000}{2^{30}} = 0.047Hz \qquad 4.3$$

4.3.3.4 Size of the Look up Table

The size of a look up table in a DDFS depends exponentially on the number of phase bits and linearly on the number of amplitude bits. This can be summarized by the equation

$$w = m * 2^k \qquad 4.4$$

Where:

w is the size in bits of the look up table

m is the size in bits of the amplitude word

k is the number of phase bits used by the look up table

The 30-bit phase accumulator used in this work would require 2^{30} entries in a lookup table in order to directly convert 30 bits of phase to corresponding amplitude. If each entry were stored with 16-bit accuracy, then using equation 4.4, two gigabytes of lookup table memory would be required. Such a large lookup table would have resulted in high power consumption, lower speed and greatly increased costs. Therefore a smaller memory that has been shown to be capable of fitting in a commercially available microcontroller (Silicon laboratories, 2003) was preferred. The look up table had 8 phase bits (2^8 = 256 entries) and each entry was 16 bits in size. Since the look up had 8 phase bits, only the 8 most significant bits of the 30 bit phase accumulator output were used to

provide phase information; truncation of the phase information was expected to result in spurs. Therefore, dithering was used to reduce these spurs.

4.3.3.5 Number of output channels

The number of output channels for the proposed signal generator was fixed at one because one channel was adequate for verifying the performance of the signal generator.

4.3.3.6 Output signal resolution and amplitude

The resolution in bits of the signal generator output signal was set at ten bits. This figure was found to be acceptable because it has been previously used in the popular and commercially available DDFS integrated circuit AD9833. In addition 10 bit DACs are readily available.

The maximum amplitude that the signal generator can produce is 1 volt peak to peak. The value was arrived at after considering the DAC transfer function (Texas Instruments, 2009):

$$V_{OUT} = I_{OUT} \times R_{LOAD} \qquad 4.5$$

Where:

V_{OUT} is the output voltage

I_{OUT} is the output current

R_{LOAD} is the load resistance

The maximum value for I_{OUT} is 20 mA while a typical value for R_{LOAD} is 50Ω, applying these two values in the DAC transfer function 4.5 implies that the maximum output voltage would be: 50 x 20/1000 = 1 volt.

4.3.3.7 Spurious Free Dynamic Range (SFDR)

The SFDR of 88 dBc was determined using the equation for calculating the SFDR for a DDFS that uses the dithering technique of spur reduction:

$$SFDR = -(7.84 - 12.04k)\,\text{dBc}. \qquad 4.6$$

Where k is the number of bits remaining after truncating the phase accumulator word. Using 8 as the value of k in equation 4.6 (see section 4.3.3.4) for an explanation of this value of k) the expected value of SFDR would be: - (7.84 -12.04 x 8) = 88 dBc. Besides dithering the resulting SFDR was also a result of a tradeoff between the size of the lookup table and spectral purity; when one is fixed the other follows. This relationship is summarized by equation 4.6; where for a given size of the phase address (k) the SDFR is fixed.

4.3.3.7 Output impedance

The output impedance of 50 Ω was used because it is the value of a load resistance that would allow the DAC to provide the maximum 1 volt peak to peak output voltage as specified in the DAC's datasheet (Texas Instruments, 2009).

4.4 Design model of DDFS signal generator

Figure 4.3 shows a block diagram of the designed DDFS based signal generator. The design was arrived at using the theory of a DDFS provided in section 3.1. The description of how dithering can be done is provided in section 3.4 and 3.5 while the signal generator's functional requirements are presented in section 4.3.1.

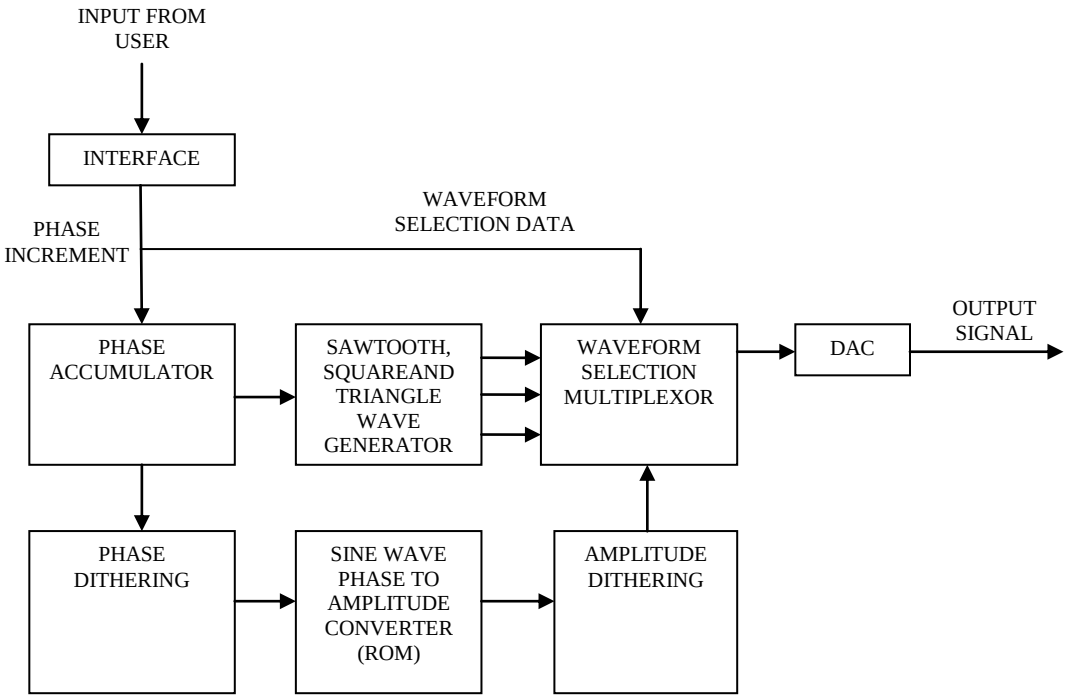


Figure 4.3: DDFS signal generator block diagram

As shown in Figure 4.3 the systems' interface receives waveform configuration parameters from the user, it then generates a phase increment value and waveform selection data. The phase increment is used by the phase accumulator to generate a continuously increasing phase value that can be used by the phase to amplitude converter to generate sine wave samples or by the saw tooth, square and triangle wave generator to generate the mentioned waveforms. The waveform selection data is used by the waveform selection multiplexor as a control signal for determining the type of waveform that should constitute its current output. The phase dithering and amplitude dithering modules generate and add a dither signal to each of the phase and amplitude samples respectively. The DAC converts the digital samples of the generated waveforms to an analog format.

4.5 SystemC implementation of the signal generator

This section describes how the DDFS based signal generator design was implemented using SystemC. Each of the blocks shown in Figure 4.3 was implemented using SystemC modules while communication between the modules was achieved using SystemC signals. The Specifications provided in table 4-1 and the functional requirements of the proposed DDFS signal generator guided the implementation of the modules.

Sections 4.5.1 to 4.5.8 present a detailed description of the signal generators' modules. GTK Wave viewer simulation results of the modules have also been presented. Four models of the signal generator design presented in Figure 4.3 were implemented; a model with no dithering, with phase dithering, with amplitude dithering and with phase and amplitude dithering. Sample code for the SystemC phase accumulator module is provided in appendix A.

Objective

The objective of creating the signal generator models was:

1) Architecture exploration – i.e. to get an early understanding of the signal generator design and also to verify that the design will function the way it was intended.
2) To examine the effects of phase and amplitude bits truncation on the output spectrum of the signal generator.
3) To examine the effect of phase dithering, amplitude dithering and combination of phase and amplitude dithering on the output spectrum of the signal generator.

4) To identify the dithering method that would result in the optimal reduction in spurs for the proposed signal generator.

4.5.1 Interface module

This module is responsible for receiving a frequency and waveform selection value from the user. It then calculates and outputs the phase increment value. The waveform selection value is passed on without any further processing. As shown in Figure 4.4 it has two output ports identified with sc_out. An input port is not required for this module because the frequency and wave selection values are entered by the user directly through the keyboard. The module has one process which is of kind SC_METHOD. Equation 4.7 is used in calculating the phase increment, this equation is obtained by making ΔP in equation 3.1 the subject of the equation.

$$\Delta P = f_{out} \frac{2^j}{f_{clk}} \qquad 4.7$$

Where:

ΔP is the phase increment word (30 bits) and it is an integer

j is the number of phase accumulator bits (30)

f_{clk} is the clock frequency (50MHz)

f_{out} is the required output frequency

Table 4-2 shows a mapping of the waveforms that can be generated by the signal generator to a unique number; which the user supplies in order to determine the current output waveform.

Table 4-2 Waveform selection codes

Waveform selection value	**Waveform**
1	Square
2	Sine
3	Saw tooth
4	Triangle

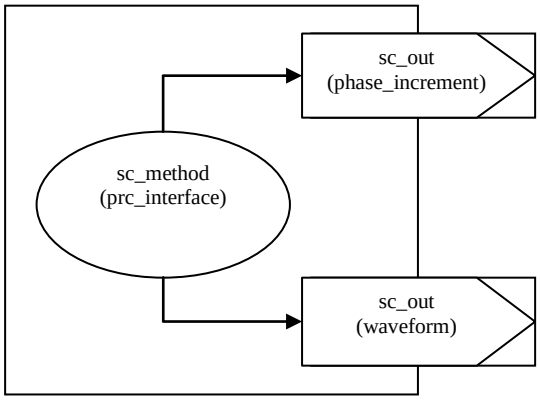


Figure 4.4: sc_module (interface)

Using equation 4.7 the expected phase increment rounded to the nearest integer for a frequency of 1 MHz is:

$$\Delta P = \frac{2^{30} * 1000000}{50000000} = 21474836$$

Note that the phase increment does not have units because the denominator and numerator have the same units (Hz). Rounding off the phase increment results in a frequency error that is less than the smallest step in frequency which in this case is 0.047

Hz. Hence the percentage error in the generated frequency decreases as the output frequency value gets larger.

After simulating the interface module it was observed that the module can generate the correct phase increment value for any frequency and also receive the wave selection value as shown in Figure 4.5, where the module generated an expected phase increment of 21474836 for a frequency of 1 MHz and recorded a wave selection value of 2 which corresponds to a sine wave.

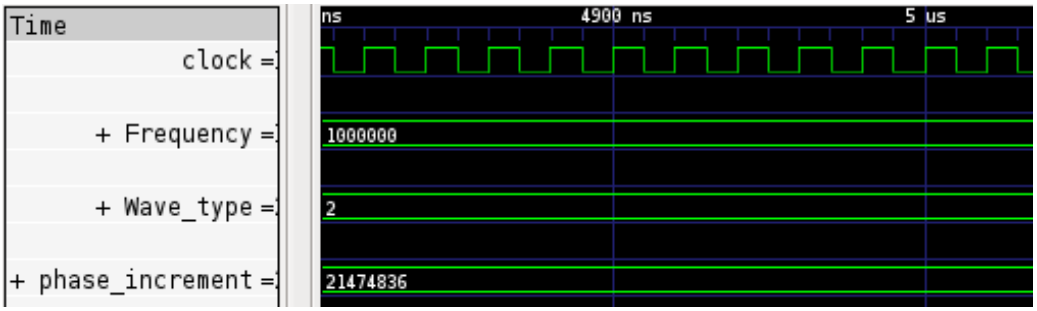


Figure 4.5: sc_module (interface) simulation for 1 MHz output frequency

4.5.2 phase accumulator module

This module is an implementation of the phase accumulator; it adds the phase increment value to the phase accumulation register on every positive clock edge. If the resulting sum exceeds the maximum value of the phase accumulation register, it overflows and the process begins all over again. As shown in Figure 4.6 it has two input ports identified with sc_in and one output port identified with sc_out. The module has one process which is of kind SC_METHOD.

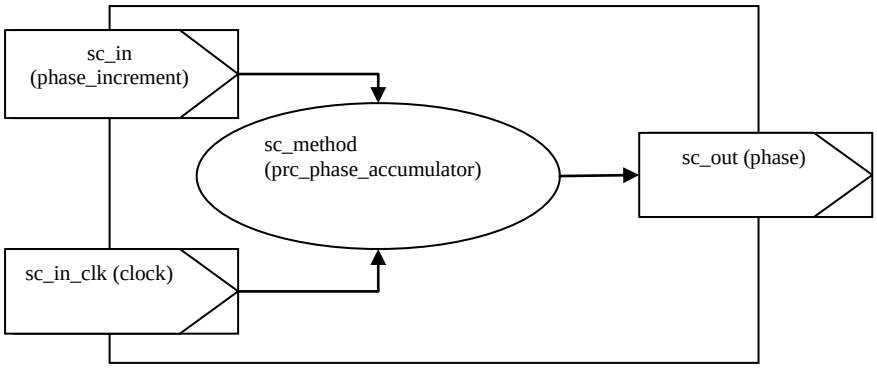


Figure 4.6: sc_module (phase accumulator)

Figure 4.7 shows the simulation result of the phase accumulator module. As shown in the figure the module phase accumulator is capable of adding the phase increment value to the previous content of the phase accumulation register on every clock pulse, the phase increment value used for this test is 21474836, the size of the phase accumulation was 30 bits and the clock frequency was 50 MHz.

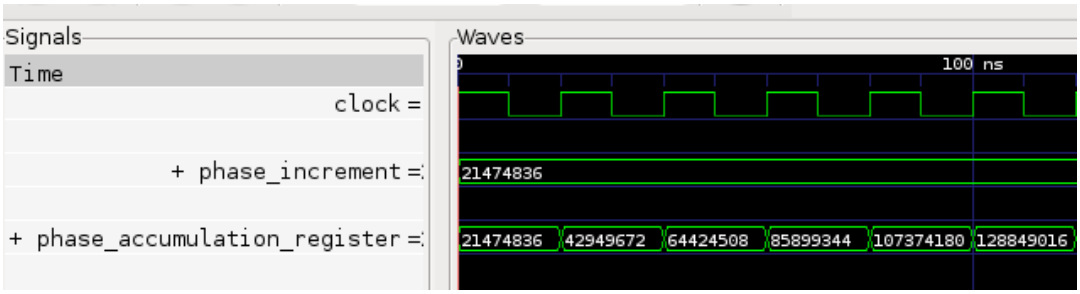


Figure 4.7: Simulation result of the phase accumulator module

The phase accumulator module was also tested to confirm whether the phase accumulation register over flows at the correct time for any particular frequency, as shown in Figure 4.8 the phase accumulation register overflowed every microsecond as expected for a frequency of 1 MHz.

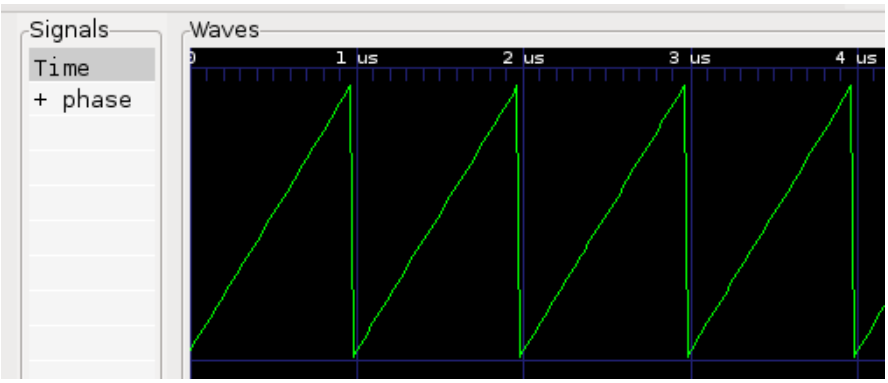


Figure 4.8: Simulation result for the overflow of the phase accumulation register

4.5.3 phase_to_amplitude module

This module is an implementation of a sine wave phase to amplitude converter; it represents the look up table (LUT) of a DDFS. The module uses the phase values to access the members of an array holding the values representing a sine wave. Only the 8 most significant bits of the 30 bit phase accumulation register are used to generate sine wave amplitude values. The output amplitude information is 16 bits in size. As shown in Figure 4.9 it has one input port identified with sc_in and one output port identified with sc_out. The module has one process which is of kind SC_METHOD.

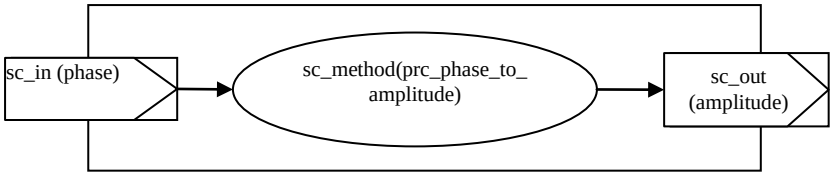


Figure 4.9: sc_module (phase to amplitude)

The output sequence of the array in the ph_to_amplitude module is given by:

$$amplitude[i] = ((\left(\sin\left(i * 2 * \frac{\pi}{256}\right) + 1\right) * (32768)) \qquad 4.8$$

Where:

amplitude[i] is the amplitude value (for the i^{th} phase value)

Sin is the sine function

256 is the number of quantization levels achievable using an 8 bit register

The +1 in the equation is used to ensure that only positive values are generated; a unipolar output was preferred because most DACs are designed to output unipolar voltages.

The 32768 is half the number of quantization levels achievable using a 16 bit register.

Equation 4.8 is used for initializing the look up table only. The amplitude values in the look up table are rounded off to integer values no bigger than 2^{16}-1 = 65535. Thereafter the values are simply looked up from the table. The output sequence is:

$$S[i] = amplitude[i] \qquad 4.9$$

Where: i is the phase value

S[i] is the i^{th} amplitude sample

amplitude[i] is the amplitude value at the i^{th} position in the look up table.

The number of sine wave reconstruction samples generated by the phase to amplitude converter is inversely proportional to the output frequency or the phase increment value.

Figure 4.10 shows the simulation result of the phase to amplitude module. As shown in the figure the module is capable of generating an amplitude value for any phase value on every clock pulse.

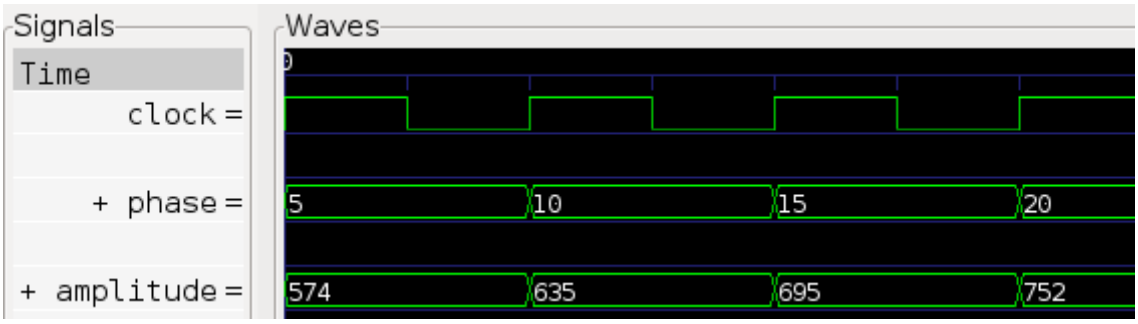


Figure 4.10: Simulation result of the phase to amplitude module

The phase to amplitude module was also tested to confirm whether it can generate signals with the expected period for any particular frequency, as shown in Figure 4.11 the module generates a complete sine wave every microsecond as expected for a frequency of 1 MHz.

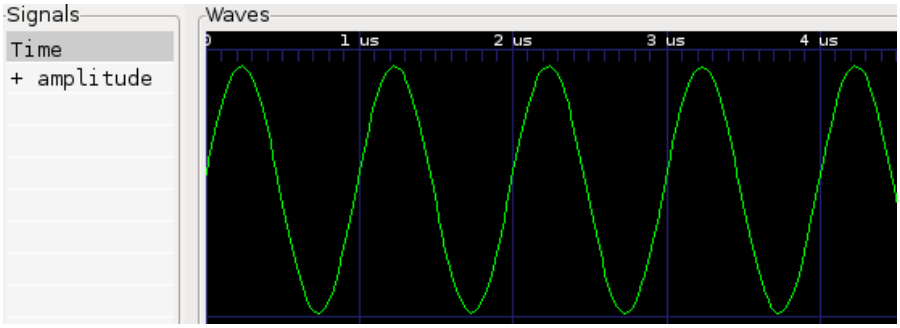


Figure 4.11: Simulation result for frequency period testing

For the signal generator designs that make use of amplitude dithering, the amplitude information stored in the look up table was reduced (scaled) so that the original signal plus the dither would stay within the non-saturating region. Scaling involved normalizing each of the 16 bit entries in the look up table so that the sinusoid amplitude equals 64 16-bit quantization steps less than the full scale value (Flanagan and Zimmerman, 1995). Equation 4.10 was used to generate the samples required to initialize the scaled look up table:

$$amplitude[i] = (\left(\sin\left(i * 2 * \frac{\pi}{256}\right) + 1\right) * (32768)/1.000977532) \qquad 4.10$$

Where:

amplitude[i] is the amplitude value (for the i^{th} phase value).

Sin is the sine function.

256 is the number of quantization steps achievable using an 8 bit register.

The 32768 is half the number of quantization levels achievable using a 16 bit register.

1.000977532 is the figure used to scale each of the amplitude samples, so that each of them is 64 16-bit quantization steps less than the full scale value. The removal of 64 16-bit quantization steps is required to prevent an overflow of the 16-bit register used in holding the sum of the amplitude and a dither signal whose maximum value is $2^6 = 64$. The size of the dither signal (6 bits) is explained in section 4.5.5.

4.5.4 Monitor module

This module helps in recording the various values generated by the signal generator model i.e. frequency, phase, amplitude and the phase accumulator output value. The values are sent out to the console and also saved in a text file. As shown in Figure 4.12 it has four input ports identified with sc_in and one process which is of kind SC_METHOD.

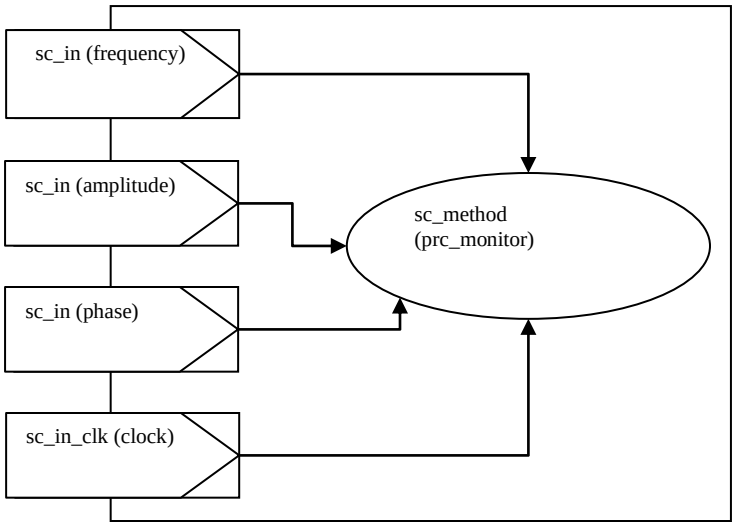


Figure 4.12: sc_module (monitor)

4.5.5 phase dithering module

This module is an implementation of the phase dithering process; it is responsible for the generation and addition of a dither signal to the phase information. As shown in Figure 4.13 it has two input ports identified with sc_in and one output port identified with sc_out. The module has one process which is of kind SC_METHOD, it implements a linear feedback shift register pseudo-random number generator and an adder. A discussion on phase dithering is given in section 3.4.

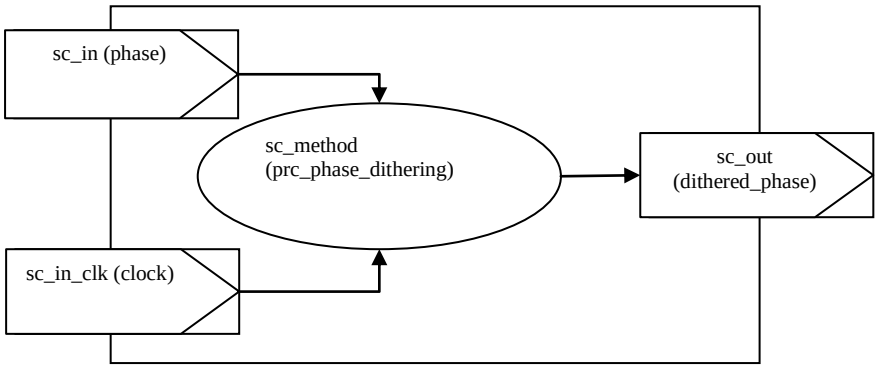


Figure 4.13: sc_module (phase dithering)

Using equation 3.20, the size of the phase dithering signal for this module would be 30-8 = 22 bits. Figure 4.14 shows the simulation result of the phase dithering module. As shown in the figure the module is capable of generating and adding a random number to a phase value on every clock pulse.

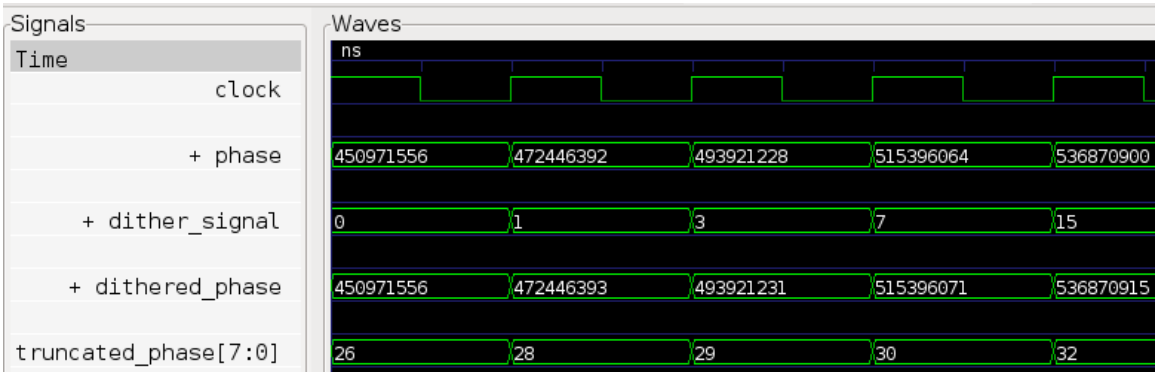


Figure 4.14: Simulation result of the phase dithering module

4.5.6 Amplitude dithering module

This module is an implementation of the amplitude dithering process; it is responsible for the generation and addition of a dither signal to the amplitude information. As shown in Figure 4.15 it has two input ports identified with sc_in and one output port identified with sc_out.

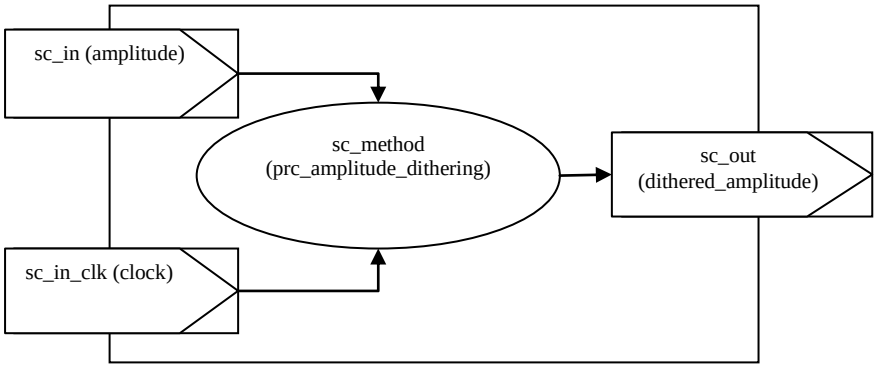


Figure 4.15: sc_module (amplitude dithering)

The module has one process which is of kind SC_METHOD; it implements a linear feedback shift register pseudo-random number generator and an adder. A discussion on dither signal generation is given in section 3.7.

Figure 4.16 shows the simulation result of the amplitude dithering module. As shown in the figure the module is capable of generating and adding a random number to the amplitude information on every clock pulse. The size of the dither signal is 6 bits while the clock frequency used is 50 MHz.

Using equation 3.20, the size of the amplitude dithering signal for this module would be 16-10 = 6 bits. A discussion on amplitude dithering is given in section 3.5.

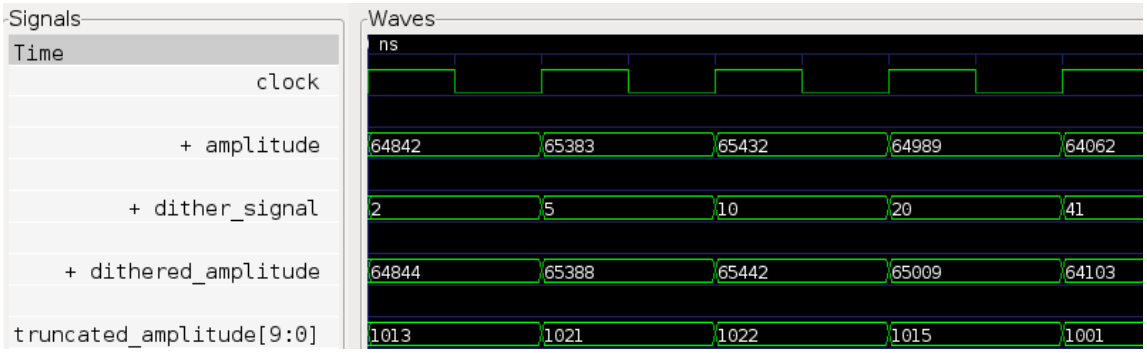


Figure 4.16: Simulation result of the amplitude dithering module

4.5.7 Saw tooth, square and triangle wave generator

This module generates the saw tooth, square and triangle waveforms using the phase information as follows:

1) The square wave is generated using the most significant bit of the phase value; when it is high the 10 bit square waveform amplitude register is set to its maximum value otherwise it is set to zero.

2) The saw tooth waveform is directly generated from the 10 most significant bits of the 30 bit phase accumulator because the phase accumulator output consists of an increasing value which rolls over when the phase accumulation register is full.
3) The triangle wave is generated using 10 most significant bits of the 30 bit phase accumulator; when the phase accumulator value is below half its maximum value the value is channeled directly to the triangle wave amplitude register otherwise it is inverted before being channeled to the register.

As shown in Figure 4.17 the saw tooth, square and triangle waveform generator has one input port identified with sc_in and three output ports identified with sc_out. The module has one process which is of kind SC_METHOD.

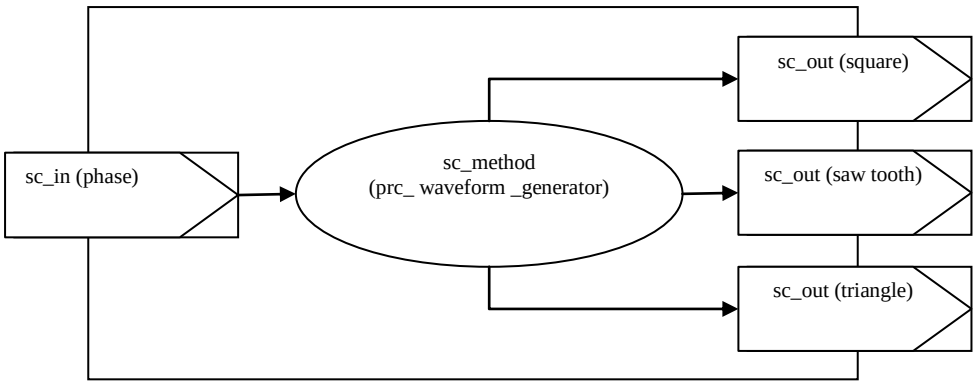


Figure 4.17: sc_module (sawtooth_square_triangle_waveform_generator)

Figure 4.18 shows the simulation result of the saw tooth, square and triangle waveform generator module. As shown in the figure the module is capable of generating saw tooth, square and triangle waveforms. The frequency value used for this test is 1 MHz, the size of the phase information was 30 bits and the waveform amplitude registers are 10 bits is size.

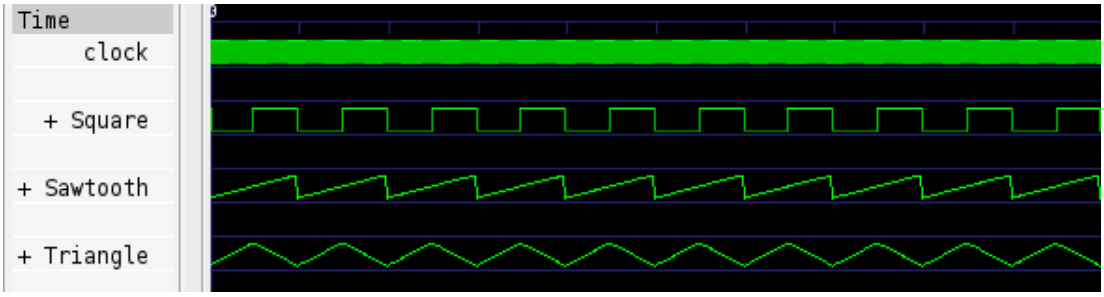


Figure 4.18: Simulation result of the saw tooth, square and triangle waveform generator module

4.5.8 Waveform selection multiplexor

This module receives the sine, saw tooth, square and triangle waveforms data and outputs only one of this waveforms depending on the control signal received from the interface. Table 4-3 shows the waveform selection control signal and the corresponding output waveform.

Table 4-3 Waveform selection control signals

Waveform selection value	**Waveform**
1	Square
2	Sine
3	Saw tooth
4	Triangle

As shown in Figure 4.19 the waveform selection multiplexor has four input ports identified with sc_in and one output port identified with sc_out. The module has one process which is of kind SC_METHOD.

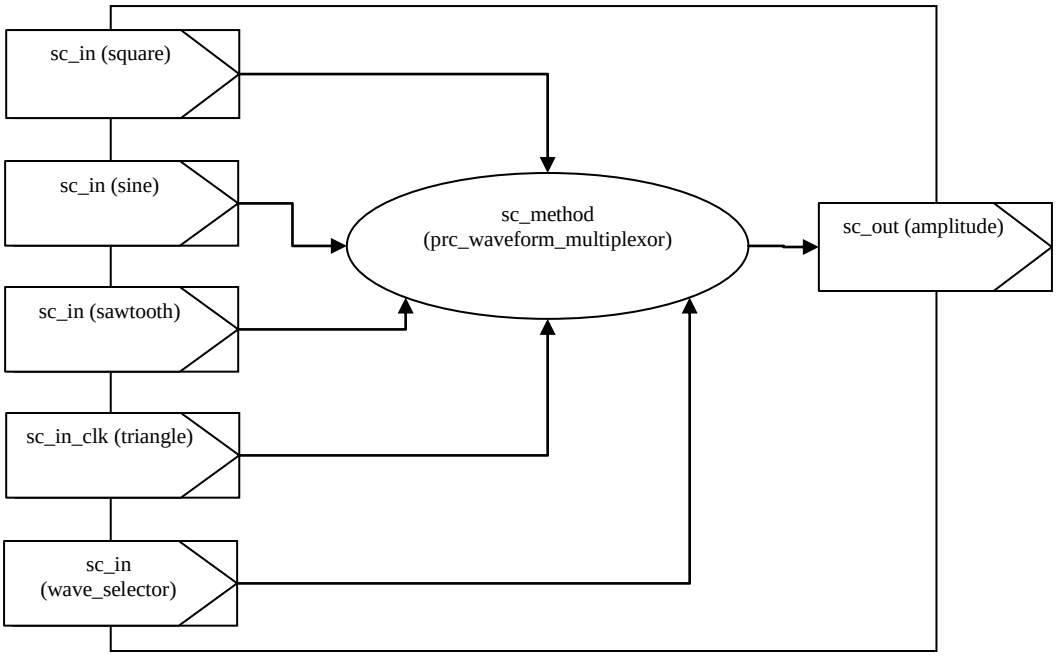


Figure 4.19: sc_module (waveform_selection_mulitplexor)

Figure 4.20 to 4.23 shows the simulation result of the waveform multiplexor module. As shown in the figure the module capable of generating square, sine, saw tooth and triangle waveforms depending on the wave selection control signal as defined in Table 4-3. The frequency value used for this test is 1 MHz and the waveform amplitude registers is 10 bits in size.

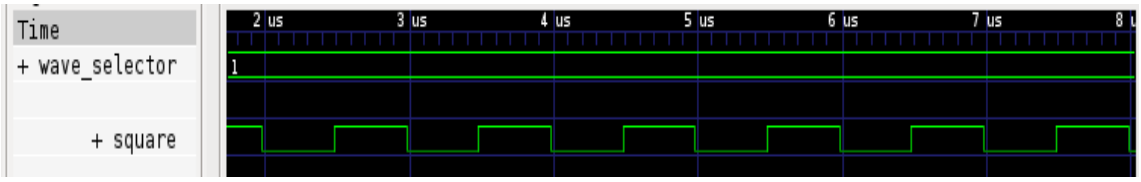


Figure 4.20: square wave simulation result of the waveform_selection_mulitplexor module

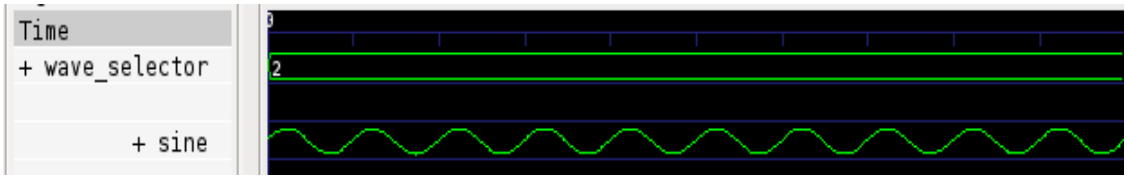


Figure 4.21: sine wave simulation result of the waveform_selection_mulitplexor module

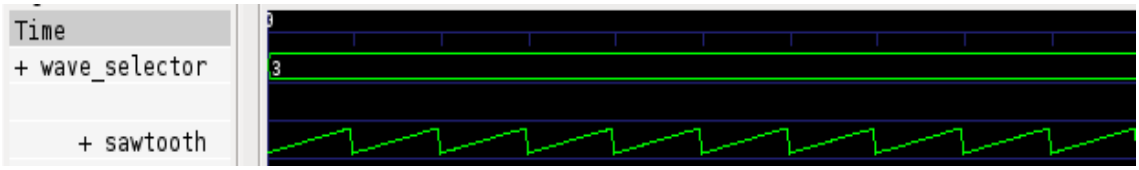


Figure 4.22: Saw tooth wave simulation result of the waveform_selection_mulitplexor module

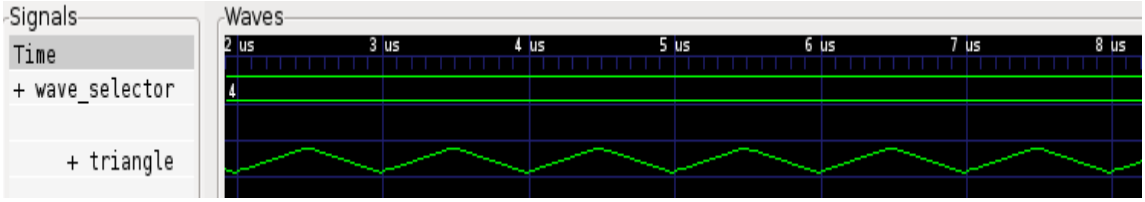


Figure 4.23: Triangle wave simulation result of the waveform_selection_mulitplexor module

4.5.9 Integration of signal generator modules

Four versions of the signal generator were created depending on how the modules discussed in section 4.5.1 to 4.5.8 were integrated:

1. Signal generator design without dither
2. Signal generator design with phase dithering
3. Signal generator design with amplitude dithering
4. Signal generator design with phase and amplitude dithering.

Objective

The main objective of creating four different versions of the signal generator design was to identify the design that would result in the optimal spur reduction. Sections 4.6 to 4.9 describe each of the four modules and the results obtained from their simulation.

The following are the attributes that are common to the four signal generator designs; these and other constraints are also presented and explained in section 4.3:

a) The phase information which is 30 bits in size is truncated to 8 bits to facilitate the use of a small look up table (LUT) in the phase to amplitude converter. This is done for the sine wave only; for the other waveforms, the phase information is truncated to ten bits. The ten bits limit was determined by the resolution of the DAC to be used, see section 4.3.3.6 for an explanation of this figure.
b) The sine amplitude word length is 16 bits before truncation to 10 bits, truncating the amplitude word length was necessary because it would result in a narrower data path compatible with low cost and readily available digital to analog converters.
c) 50 MHz clock was used.
d) The DAC has not been included in the models because the investigation on the signal generator design that will result in the largest spur reduction mainly focuses on the spurs that are caused by the digital part of a DDFS and not the analog part which consists of the DAC.
e) Dithering was only applied in the generation of the sine wave.

4.6 Signal generator design with truncation and without dither

The main attribute of this design is that no dither signal was applied to the phase or amplitude information; as a result spurs caused by truncating the phase and amplitude information are expected in the output spectrum of this model. The model was used as a control in identifying the contribution of the various methods of dithering on the output

spectrum of a DDFS based signal generator. This was done by comparing its SFDR and noise floor performance to that of models that made use of dithering.

Figure 4.24 shows how some of the modules discussed in section 4.5.1-8 were interconnected in order to create the signal generator design without dither; for clarity the figure shows the part related to the sine wave generation only. The model was simulated as described in section 4.2.Wave form generation and output spectrum simulation results for this model will be discussed in section 4.11.

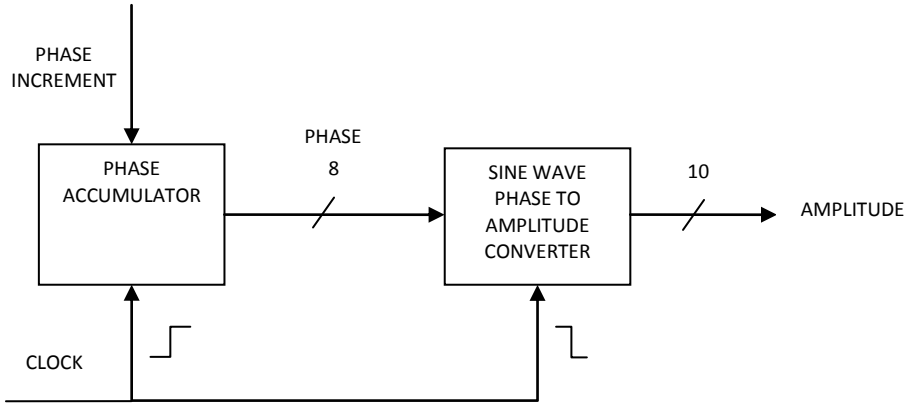


Figure 4.24: signal generator model without dither

4.7 Signal generator design with phase dithering

The main attributes of this design is that a 22 bit dither signal is added to the phase information before truncation; it is expected to reduce the level of spurs caused by phase information truncation. A discussion on phase dithering is provided in section 3.4. The design was used in identifying the contribution of the phase dithering technique on the output spectrum of the DDFS based signal generator. This was done by comparing its performance to that of the design that had no dithering used.

Figure 4.25 shows how the modules discussed in section 4.5.1-8 were interconnected in order to create the signal generator design with phase dithering; for clarity the figure shows the part related to the sine wave generation only. The module was simulated as described in section 4.2; wave form generation and output spectrum simulation results for this module will be presented in section 4.11.

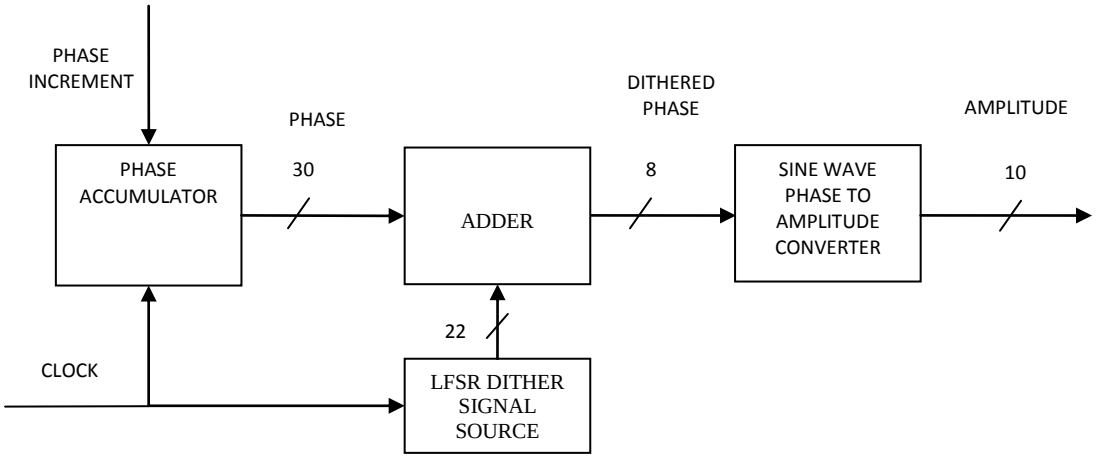


Figure 4.25: Signal generator design with phase dithering

4.8 Signal generator design with Amplitude dithering

The main attributes of this design is that a 6 bit dither signal is added to the amplitude information before truncation to 10 bits, it is expected to reduce the level of spurs caused by amplitude quantization, a discussion on amplitude dithering is provided in section 3.5. The design was used in identifying the contribution of amplitude dithering on the output spectrum of a DDFS based signal generator. This was done by comparing its SFDR and noise floor performance to that of a design that had no dithering used.

Figure 4.26 shows how the modules discussed in section 4.5.1-8 were interconnected in order to create the signal generator design with amplitude dithering; for clarity the figure shows the part related to the sine wave generation only. The design was

simulated as described in section 4.2; wave form generation and output spectrum simulation results for this module will be presented in section 4.11.

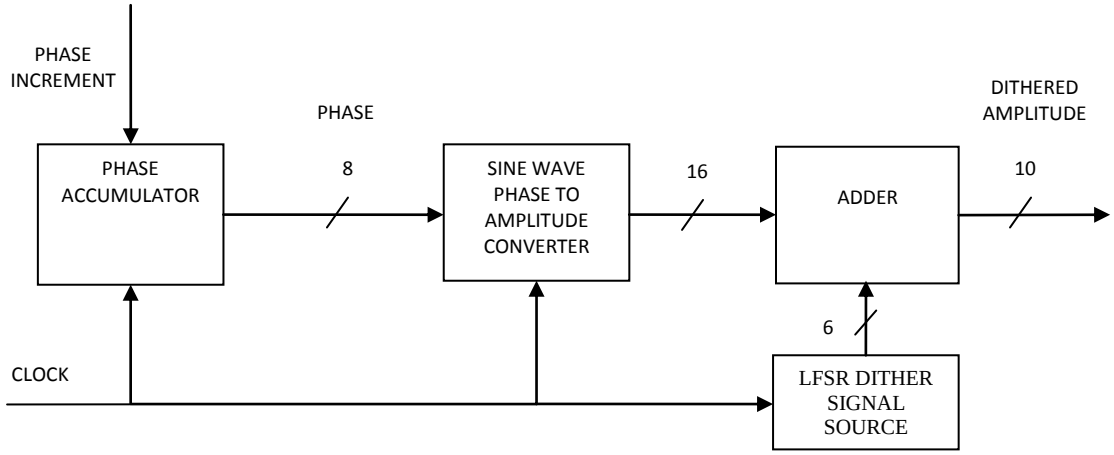


Figure 4.26: Signal generator design with amplitude dithering

4.9 Signal generator design with phase and amplitude dithering

The main attribute of this design are as follows:

a) A 6 bit dither signal is added to the amplitude information before truncation to 10 bits. It is expected to reduce the level of spurs caused by amplitude quantization.

b) A 22 bit dither signal is added to the phase information before truncation.

The design was used to identify the contribution of a combination of phase and amplitude dithering on the output spectrum of the proposed DDFS based signal generator. This was done by comparing its SFDR and noise floor performance to that of the model that did not have dithering.

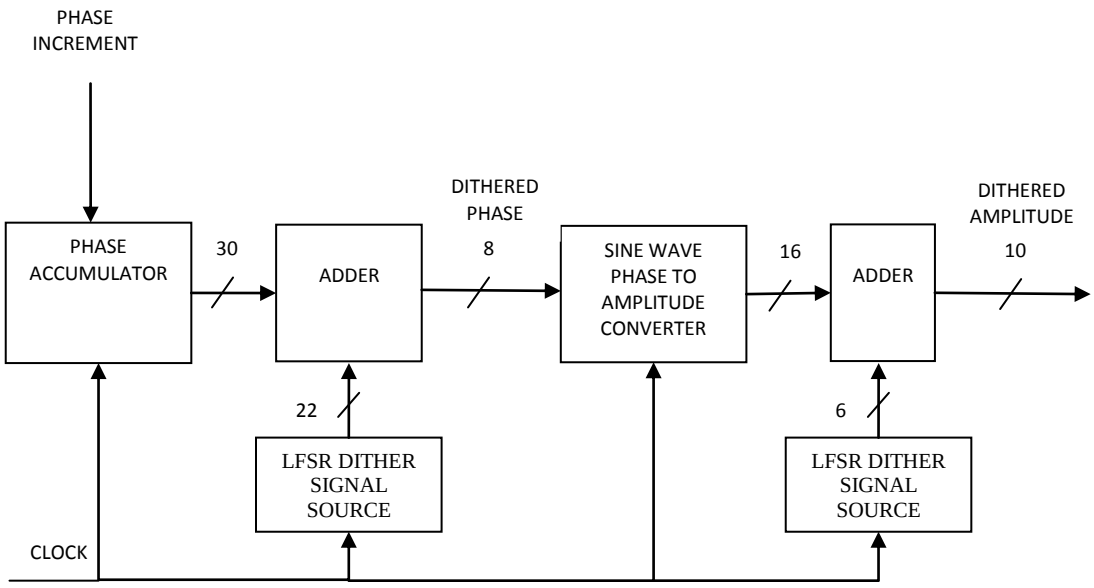


Figure 4.27: Signal generator design with phase and amplitude dithering

Figure 4.27 shows how the modules discussed in section 4.5.1-8 were interconnected in order to create the DDFS module with phase and amplitude dithering; for clarity the figure shows the part related to the sine wave generation only. The module was simulated as described in section 4.2; wave form generation and output spectrum simulation results for this module will be presented in section 4.11.

4.10 Wave form generation simulation results

Figure 4.28 to 4.31 shows the waveforms that were generated after simulating the signal generator model without dither, with phase dithering, with amplitude dithering and with a combination of phase and amplitude dithering. Only one figure has been presented for the four models because no difference was observed in the wave forms generated by the four modules for the test frequency used. From the figures it can be observed that the four signal generator designs were capable of generating sine, square, triangle and saw

tooth waveforms depending on the wave type control signal. The output signal was at an expected test frequency of 1 MHz.

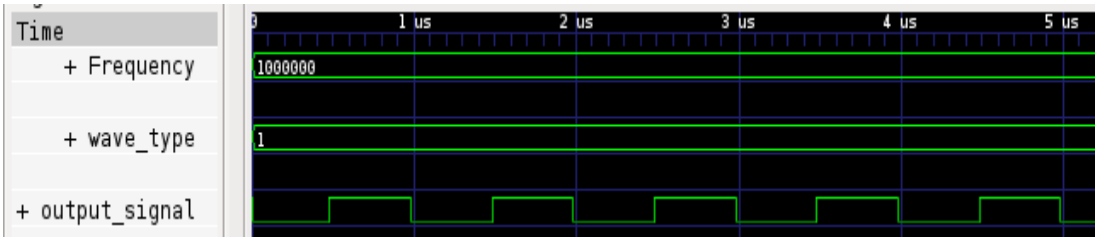


Figure 4.28: Square wave simulation of signal generator design

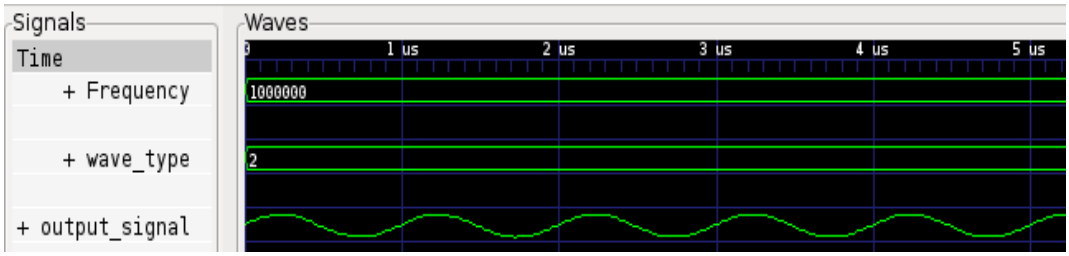


Figure 4.29: Sine wave simulation of signal generator design

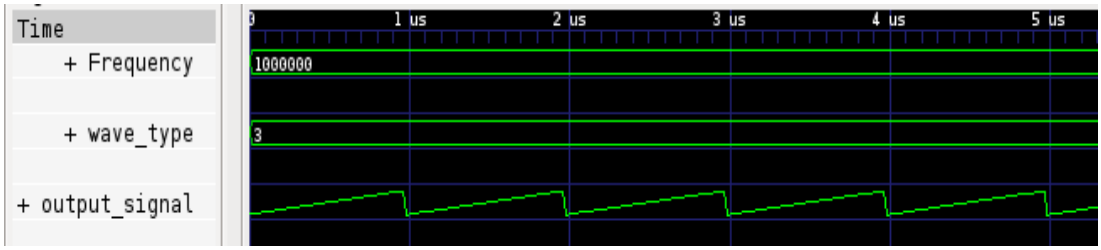


Figure 4.30: Saw tooth wave simulation of signal generator design

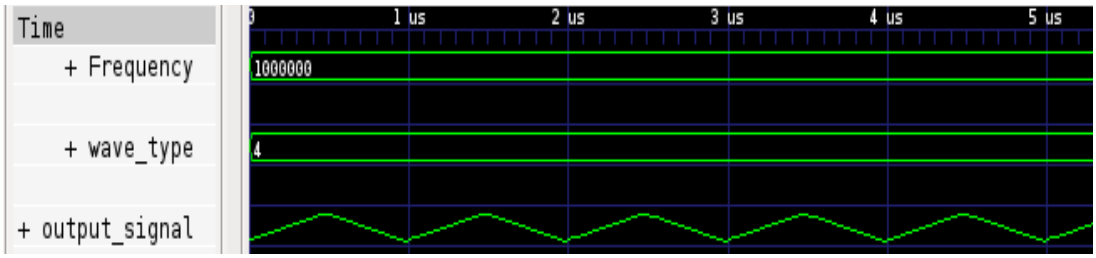


Figure 4.31: Triangle wave simulation of signal generator design

4.11 SystemC simulation output spectrum results

Four SystemC models of the signal generator were created; a model with no dither signal, with phase dithering only, with amplitude dithering only and one that had a combination of both phase and amplitude dithering. Figure 4.32 (a), (b), (c) and (d) show the observed output spectrum results for these models. In the figures, e + n on the x axis, where n is an integer has the same meaning as 10 raised to power +n.

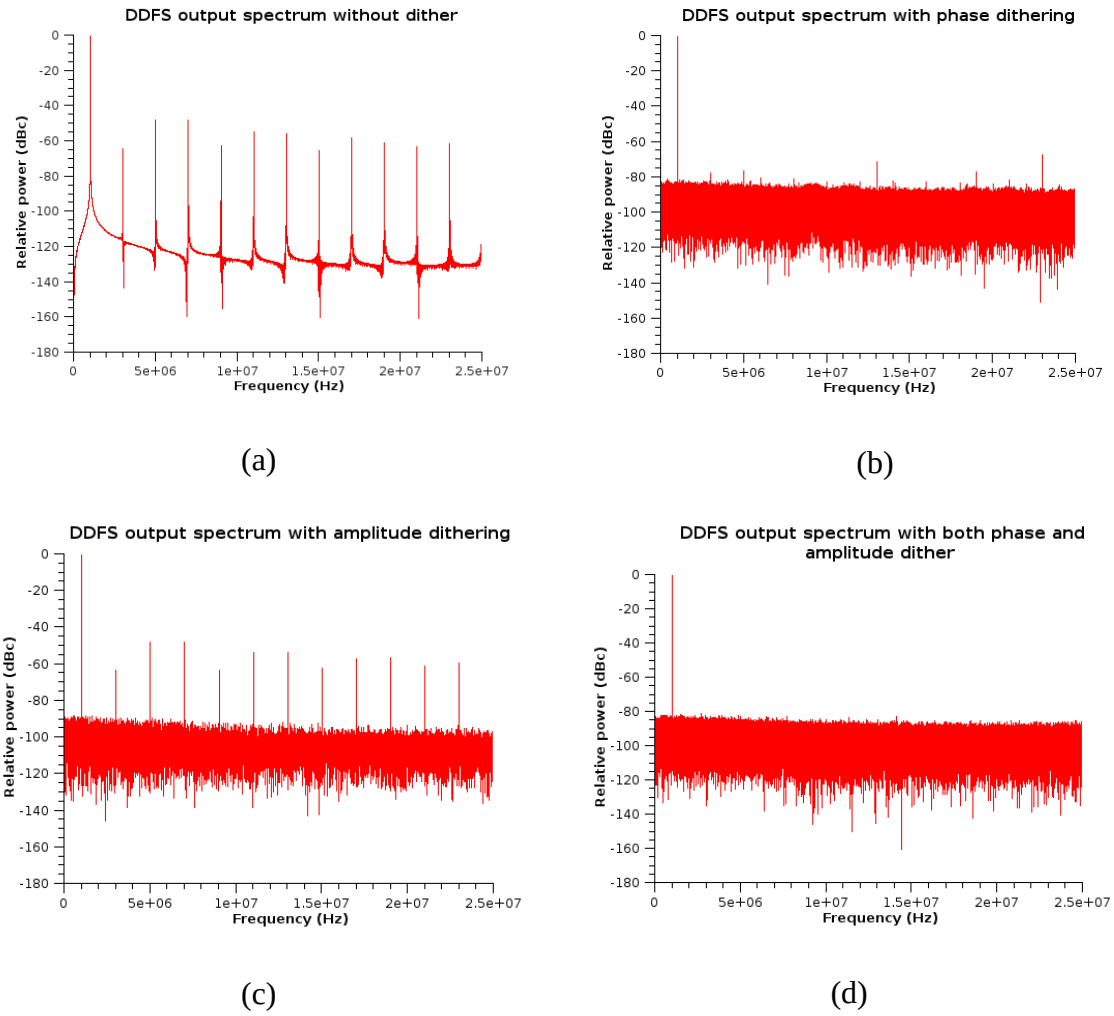


Figure 4.32: DDFS output spectra (a) with no dither signal, (b) with phase dithering (c) with amplitude dithering and (d) with both phase and amplitude dithering.

The spectrums in Figure 4.32 were obtained by taking a Fast Fourier Transform (FFT) of the output sine wave data using QtiPlot. QtiPlot is a program used for two- and three-dimensional graphical presentation of data sets and for data analysis. The simulation parameters used were j=30, k=8, m=10, f_{clk}=50MHz, f_{out}=1MHz (five more results for different frequencies are presented in table 4-4). For the DDFS with no dither signal the observed SFDR was 48 dBc and the noise floor was at -130 dBc, for the DDFS with phase dithering an increase in SFDR to 68 dBc was observed and the noise floor also increased to -88 dBc. For the DDFS with amplitude dithering no increase in SFDR was observed since it was at 48 dBc however the noise floor increased to -99dBc. For the DDFS with both phase and amplitude dithering an increase in SFDR to 88 dBc was observed, the noise floor also increased to -88 dBc.

4.12 Expected SFDR and NF results

This section presents the calculation of expected SFDR and NF results for the proposed DDFS based signal generator design.

4.12.1 Expected SFDR and NF results for DDFS without phase and amplitude truncation

When phase truncation does not exist in a DDFS, the level of spurs in its output spectrum is determined by the errors resulting from the finite precision of the samples stored in the look up table. For such a DDFS design it is assumed that the energy of amplitude quantization spurs is concentrated in one spur, the resulting SFDR can be determined using the equation:

$$SFDR = (6.02M + 1.76) \text{ dBc} \quad (4.11)$$

Where m is the number of bits in the amplitude word without truncation and dBc is a unit representing decibels with respect to the carrier. Using 16 as the number of amplitude bits for the proposed signal generator design, the expected SFDR for the signal generator model without phase and amplitude truncation would be:

$$SFDR = (6.02 \times 16 + 1.76) = 98.08 \text{ dBc}$$

A noise floor is not expected for a DDFS model that does not have phase and amplitude information truncation because the energy of the spurs caused by amplitude quantization will be concentrated in one spur.

4.12.2 Expected SFDR and NF results for DDFS with truncation (without dithering)

The maximum level of spurs for a DDFS whose phase information is truncated can be estimated by (Vankka, 2001):

$$SFDR = 6.02k \text{ dBc} \quad (4.12)$$

Where:

SFDR is the spurious free dynamic range

k is the number of phase bits used

The expected SFDR value for the DDFS model with truncation is:

$$SFDR = 6.02 \times 8 = 48.16 \text{ dBc}$$

The expected noise floor (which in this case is a noise to signal ratio) can be calculated from an equation given in (Vankka, 2001):

$$NF = -\frac{C}{N} \cong -(1.76 + 6.02m + 10\log_{10}(\frac{Pe}{4})) \text{ dBc} \quad (4.13)$$

Where:

NF is the noise floor.

m is the word length of the output amplitude word.

dBc represents decibels with respect to the carrier.

C is the carrier power

N is the noise power

Pe is the numerical period of the phase accumulator output sequence; it can be calculated using equation 3.16.

Using equation 4.13 the expected noise floor is:

$$NF = -(1.76 + 6.02 \times 10 + 10 \times log_{10}\left(\frac{\frac{2^{30}}{4}}{4}\right)) = -140 \text{ dBc}$$

The size of the word length used for the phase address and amplitude register is 8 and 10 bits respectively. These values were selected because they have previously been used in commercially available DDFS designs and hence they were perceived to be acceptable sizes of phase and amplitude information. By extension, the acceptable SFDR and Noise Floor values for a DDFS design with such figures would be 48 dBc and -140 dBc respectively as calculated using equation 4.12 and 4.13.

4.12.3 Expected SFDR and NF results for DDFS with phase dithering

The maximum level of spurs for a DDFS model with phase dithering can be estimated by (Vankka, 2001):

$$SFDR = -(7.84 - 12.04k)\,\text{dBc} \qquad 4.14$$

Where:

k is the number of bits remaining after truncating the phase accumulator word

dBc is a unit representing decibels with respect to the carrier.

SFDR is the spurious free dynamic range.

The expected SFDR for the DDFS model with phase dithering is:

$$SFDR = -(7.84 - 12.04 \times 8) = 88.48 \text{ dBc}$$

The expected noise floor for the DDFS model with phase dithering (which in this case is a noise to signal ratio) can be calculated from an equation given in (Vankka, 2001):

$$-\frac{C}{N} \cong -(6.02k - 9.94 + 10\log_{10}(S)) \text{ dBc} \qquad 4.15$$

k is the number of bits remaining after truncating the phase accumulator word while *S* is the number of points in the output spectrum of the DDFS, S is equal to the numerical period of the phase accumulator (*Pe*); it is calculated using equation 3.16.

Using equation 4.15 the expected noise floor is:

$$NF = -\left(6.02 \times 8 - 9.94 + 10 \times log_{10}\left(\frac{2^{30}}{4}\right)\right) = -123 \text{ dBc}$$

4.12.4 Expected SFDR and NF results for DDFS with amplitude dithering

The SFDR of a DDFS model with amplitude dithering can be estimated by (Vankka, 2001):

$$SFDR = (6.02M + 1.76) \text{ dBc} \qquad 4.16$$

Where m is the number of bits in the amplitude word before truncation and dBc is a unit representing decibels with respect to the carrier. The expected SFDR for the DDFS model with amplitude dithering and phase truncation is:

$$SFDR = 6.02 \times 16 + 1.76 = 98.08 \text{ dBc}$$

The expected noise floor (which in this case is a noise to signal ratio) can be calculated from an equation given in (Vankka, 2001):

$$-\frac{C}{N} \cong -(1.76 + 6.02m + 10\log_{10}(\frac{Pe}{4})) \text{ dBc} \qquad 4.17$$

Where:

m is the word length of the Output amplitude word.

dBc represents decibels with respect to the carrier.

Pe is the numerical period of the phase accumulator output sequence, it can be calculated using equation 3.16.

Using equation 4.17 the expected noise floor with amplitude truncation is:

$$NF = -(1.76 + 6.02 \times 10 + 10 \times log_{10}\left(\frac{\frac{2^{30}}{4}}{4}\right)) = -140 \text{ dBc}$$

4.12.5 Expected SFDR and NF results for DDFS with phase and amplitude dithering

The equations used to calculate the expected value of spurious free dynamic range and noise floor for the DDFS model with phase and amplitude dithering are similar to those used for the model with phase dithering only. The assumption made here is that since the figures of SFDR and noise floor for the model with phase dithering are higher than those of the model with amplitude dithering, they would be the ones observed in the output spectrum of the DDFS with both phase and amplitude dithering. Hence the expected SFDR and NF values for the signal generator model with phase and amplitude dithering would be 88 and -123 respectively.

4.13 SystemC simulation SFDR values

The SystemC simulation SFDR and NF values were obtained using the following steps;

1. Storing time and corresponding amplitude sine wave data generated after simulating a SystemC model.
2. Using QtiPlot (data analysis tool) to perform a Fast Fourier Transform (FFT) on the sine wave form data so that spectral information could be obtained.
3. Plotting a graph of power spectral density versus frequency using the FFT data generated in step 2.
4. The SFDR reading corresponded to the second largest peak in the graph described in step 3.
5. The NF reading was identified after ignoring any harmonic, spurious and DC components in the frequency spectrum.

4.14 Discussion of results

Table 4-2 and 4-3 contain a summary of SFDR and NF results respectively that were obtained from simulating the SystemC models of the signal generator and expected theoretical values derived from the appropriate equations as presented in section 4.12. From the figures presented in the table 4-2 and 4-3 it can be observed that:

1. The SystemC model with phase dithering results in an SFDR that is higher than the one observed in the model without dither and lower than the theoretical value. The increase in SFDR is expected of DDFS models that use phase dithering. However, the failure to meet the expected theoretical SFDR value can be attributed to the presence

of amplitude quantization spurs in the output signal. The observed noise floor when compared to the case without dithering and the theoretical value is observed to increase; this is consistent with theory because dithering leads to an increase in the noise floor.

2. The SystemC model with amplitude dithering leads to an SFDR that is similar to the one observed in the model without dither and lower than the expected theoretical value. This could have been caused by the presence of phase truncation spurs in the output signal. The observed noise floor when compared to the case without dithering and its corresponding theoretical value is observed to increase, this is consistent with theory because dithering leads to an increase in the noise floor.

3. The System model with both phase and amplitude dithering results in a SFDR that is higher than the one observed in the model without dither and similar to the expected theoretical value. This is possible because phase dithering reduced the spurs caused by phase truncation while amplitude dithering minimized the spurs caused by amplitude quantization. The noise floor of the model with both phase and amplitude dither is higher than that of the model without dither, this is expected because dithering leads to an increase in the noise floor.

4. The observed values of noise floor are generally higher than their corresponding theoretical values. The cause of this difference is the number of points used in the FFT analysis; 500,000 points were used instead of the required 268,435,456 (using equation 4.23). The use of more than 500,000 points was observed to cause computational challenges for the ordinary desktop computer used, specifications of the desktop

computer used for the FFT analysis is provided in appendix D. The used numbers of points were enough to avoid masking the expected improvement in SFDR.

Table 4-2: SFDR results

	SYSTEMC SIMULATION SFDR VALUE	**THEORETICAL SFDR VALUE**	**DEVIATION OF SYSTEMC SFDR FROM THEORETICAL VALUE**
WITHOUT PHASE AND AMPLITUDE INFORMATION TRUNCATION	Not done	98	N/A
WITHOUT DITHERING			
1 MHz	48	48	0
5 MHz	45	48	3
10 MHz	47	48	1
15 MHz	43	48	5
20 MHz	43	48	5
24 MHz	48	48	0
WITH PHASE DITHERING			
1 MHz	68	88	20
5 MHz	62	88	26
10 MHz	61	88	27
15 MHz	62	88	26
20 MHz	61	88	27
24 MHz	68	88	20
WITH AMPLITUDE DITHERING			
1 MHz	48	98	50
5 MHz	45	98	53
10 MHz	45	98	53
15 MHz	45	98	53
20 MHz	45	98	53
24 MHz	48	98	50
WITH PHASE AND AMPLITUDE DITHERING			

1 MHz	88	88	0
5 MHz	88	88	0
10 MHz	88	88	0
15 MHz	88	88	0
20 MHz	88	88	0
24 MHz	88	88	0

Table 4-3: NF results

	SYSTEMC SIMULATION NF VALUE	**THEORETICAL NF VALUE**	**DEVIATION OF SYSTEMC NF FROM THEORETICAL VALUE**
WITHOUT PHASE AND AMPLITUDE INFORMATION TRUNCATION	Not done	NF is not expected	
WITHOUT DITHERING			
1 MHz	-130	-140	-10
5 MHz	-118	-140	-22
10 MHz	-110	-140	-30
15 MHz	-104	-140	-36
20 MHz	-97	-140	-43
24 MHz	-108	-140	-32
WITH PHASE DITHERING			
1 MHz	-88	-123	-35
5 MHz	-88	-123	-35
10 MHz	-88	-123	-35
15 MHz	-88	-123	-35
20 MHz	-88	-123	-35
24 MHz	-88	-123	-35
WITH AMPLITUDE DITHERING			
1 MHz	-99	-140	-41
5 MHz	-99	-140	-41
10 MHz	-99	-140	-41
15 MHz	-99	-140	-41
20 MHz	-99	-140	-41

24 MHz	-99	-140	-41
WITH PHASE AND AMPLITUDE DITHERING			
1 MHz	-88	-123	-35
5 MHz	-88	-123	-35
10 MHz	-88	-123	-35
15 MHz	-88	-123	-35
20 MHz	-88	-123	-35
24 MHz	-88	-123	-35

4.15 Conclusion of results

From the examination of previous DDFS designs, it was observed that a phase address of 8 bits is considered to be not too large or too small, hence by extension an acceptable figure of SFDR calculated using the 8 bit phase address would be 48 dBc. One of the purposes of carrying out modeling and simulation in this study was to identify the method of dithering that would lead to the largest improvement in the SFDR. From table 4-2 it was observed that the SystemC model that had phase and amplitude dithering resulted in the largest SFDR (largest reduction in spurs), the resulting SFDR was also the one closest to the expected SFDR when phase truncation is not used. This led to the conclusion that the phase and amplitude method of dithering would be the best for spectral purity improvement in the DDFS based signal generator.

CHAPTER 5

FPGA IMPLEMENTATION OF DDFS SIGNAL GENERATOR

The DDFS technique of waveform generation can be implemented using several technologies. This includes the use of a microcontroller (MCU) (Popa and Sorana, 2007), an application specific integrated circuit (ASIC) (Analog devices, 2003), a digital signal processor (DSP) (Sia et al., 2007) and a field programmable gate array (FPGA) (Sharma and Upadhyaya, 2010). FPGA Implementation was found necessary for this work because better performance could be achieved by avoiding inter-chip connections (Hsieh et al., 2003) and also because the additional hardware required by the dithering technique could easily be made inside the FPGA at no extra cost. This chapter describes how the signal generator was implemented in an FPGA.

5.1 Overview of FPGA devices

A field-programmable gate array (FPGA) is a semiconductor device that can be configured by the designer after manufacturing, hence the name field-programmable. FPGAs are programmed using a logic circuit diagram or a source code in a hardware description language (HDL) to specify how the FPGA will work.

FPGAs contain programmable logic components called logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

The FPGA used for this work (M1AFS 1500) belongs to the Actel fusion family of mixed-signal FPGAs. Figure 5.1 shows the architecture of the fusion device as provided in (Actel Corporation, 2007).The following is a summary of some of the features that made this FPGA (Actel Corporation, 2010) suitable for the implementation of the signal generator design:

- 1,500,000 system gates
- 350 MHz system performance
- Internal 100 MHz RC Oscillator (accurate to 1%)
- Low Power Consumption
- Soft ARM® Cortex™-M1 Fusion Devices (M1)

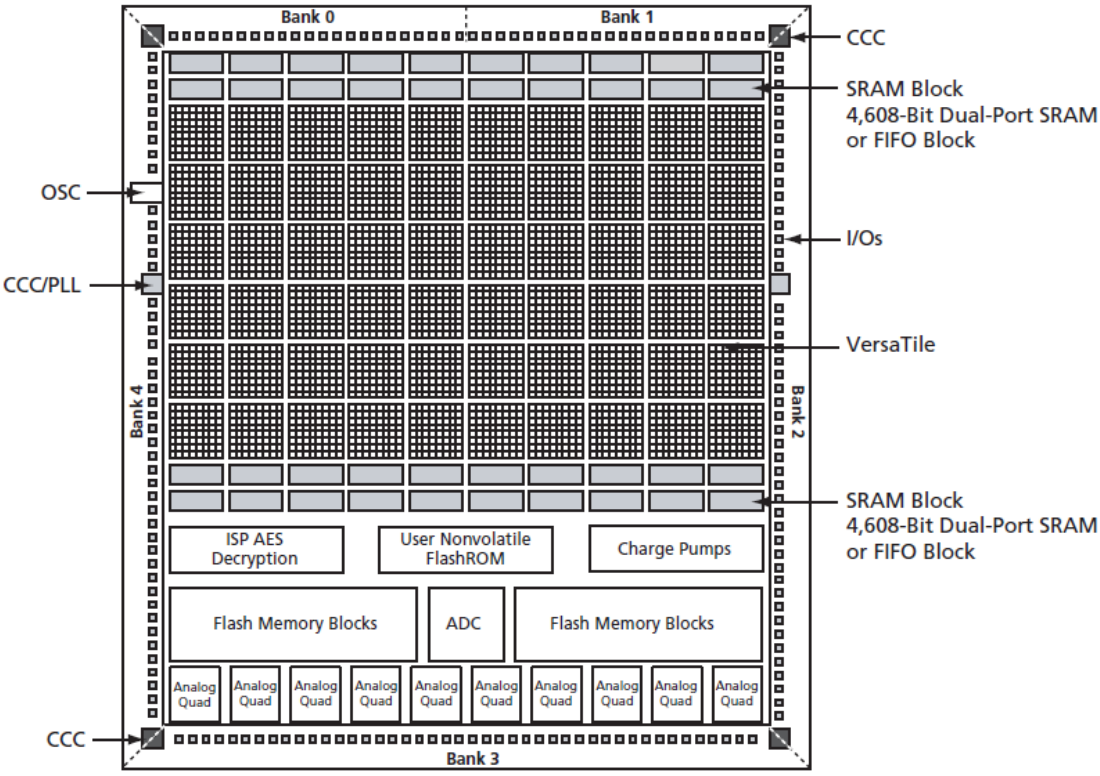


Figure 5.1:Fusion Device Architecture Overview

5.2 FPGA implementation methodology

The signal generator design preferred for implementation at the FPGA level was the one with phase and amplitude dithering; the main reason being that this design was observed to result in the largest improvement in SFDR at the modeling and simulation stage in chapter four. The comparison was made between a signal generator design without dither, with phase dithering and with amplitude dithering.

Actel Libero IDE version 9.0 design tools were used to synthesize, place-and-route the signal generator design. The FPGA used was M1AFS 1500 in the FGG484 package, it was on the Fusion Embedded Development Kit (Appendix C). The major parts of the DDFS signal generator design were cortex^{TM}-M1 processor, digital to analog converter (DAC) outside the FPGA, a DDFS module which contained; a phase accumulator, phase to amplitude converter read only memory (ROM), waveform multiplexor, phase dithering module and an amplitude dithering module. The external DAC used for this project was DAC5652A. It is a dual, 10-bit 275 MSPS digital to analog converter. A description of the signal generator prototypes' modules is presented in section 5.2.2.1-7

5.2.1 Libero IDE Design Flow

To implement the design on FPGA, the Libero IDE Design flow was used. It consists of six steps (Actel Corporation, 2010):

Step One-Design Creation

This step involves planning the design and using Design Entry tools (such as

SmartDesign) to enter it as HDL (VHDL or Verilog), structural schematic, or mixed-mode (schematic and RTL).

Step Two-Design Verification-Functional Simulation

After defining the design, verifying that it functions the way it was intended is done. Its testbench is created using WaveFormer Pro, while functional simulation of the schematic or HDL design is done using ModelSim VHDL or Verilog simulator.

Step Three – Synthesis/EDIF Generation

Synplify Pro AE is used to generate an Electronic Design Interchange Format (EDIF) netlist of the design. It is possible to re-verify the design "post-synthesis" using the VHDL or Verilog ModelSim simulator used in step two. While all RTL code must be synthesized, pure schematic designs are automatically "netlisted" out via the Libero IDE tools to create a structural VHDL or structural Verilog netlist.

Step Four – Design Implementation

After functionally verifying that the design works, the next step is to implement the design using the Actel Designer software. The Designer software automatically places and routes the design and returns timing information. The tools that come with Designer can further be used to optimize the design. SmartTime is used to perform static timing analysis on the design, ChipEditor or ChipPlanner to customize the I/O macro placement, MultiView Navigator for I/O customization, SmartPower for power analysis, and NetlistViewer to view the netlist.

Step Five – Timing Simulation

After completing the design implementation, verification that the design meets timing specifications is done. The testbench is created using WaveFormer Pro while ModelSim VHDL or Verilog simulator is used to perform timing simulation.

Step Six – Device Programming

Once the design is complete, and the results from timing simulation are good, creation of a programming file follows. Depending upon the device family in use Fuse, Bitstream, or Standard Test and Programming Language (STAPL) programming file can be generated.

5.2.2 Description of the DDFS signal generator design implemented in the FPGA

Figure 5.2 shows a block diagram of the DDFS signal generator implemented in FPGA. As shown in Figure 5.2 the systems' communication interface receives waveform configuration parameters from the user, it then generates a phase increment value and waveform selection data. The phase increment is used by the phase accumulator to generate a continuously increasing phase value that can be used by the phase to amplitude converter to generate sine wave samples or by the saw tooth, square and triangle wave generator to generate the mentioned waveforms. The waveform selection data is used by the waveform selection multiplexor as a control signal for determining the type of

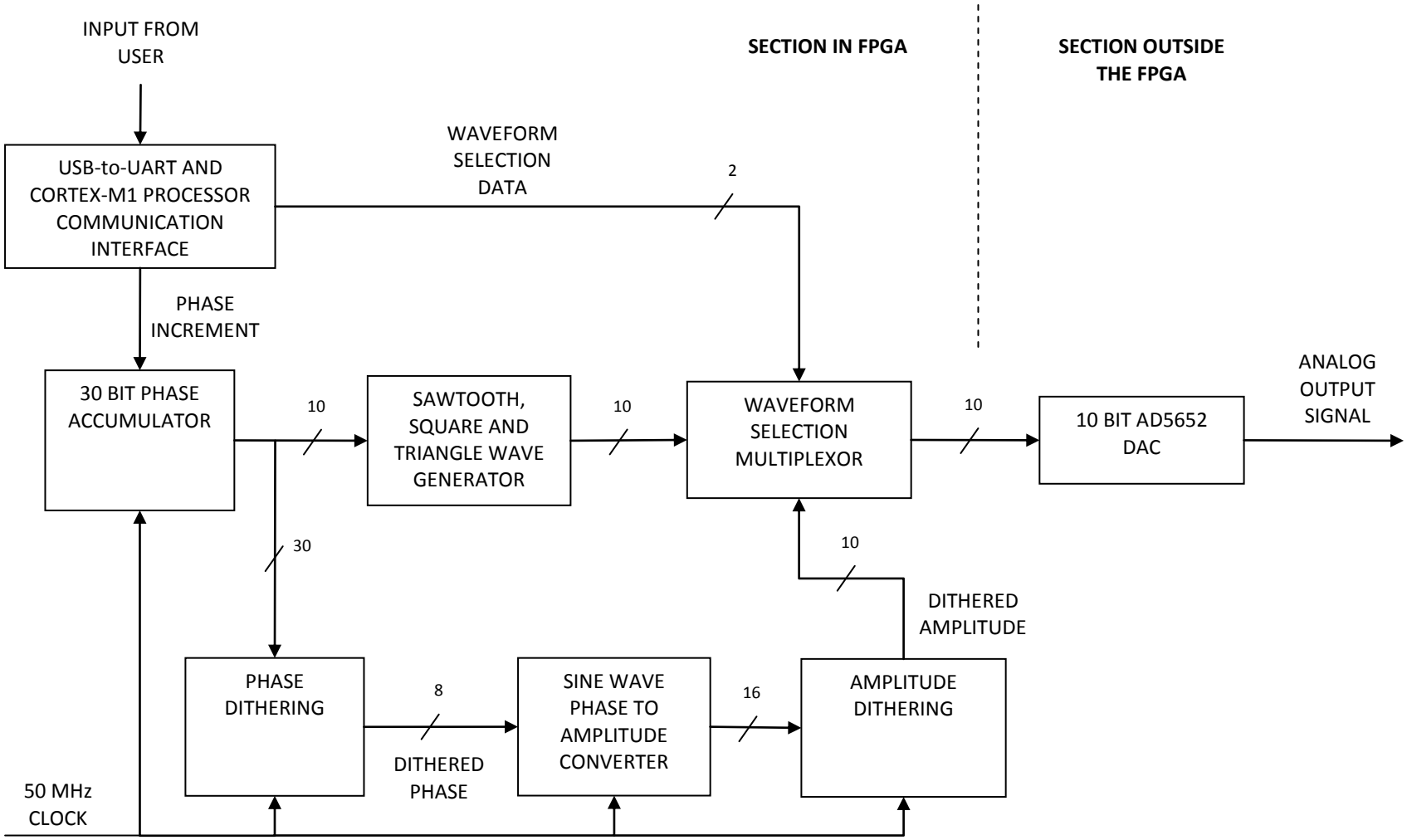


Figure 5.2: DDFS signal generator block diagram

waveform that should constitute its current output. The phase dithering and amplitude dithering modules add a dither signal to each of the phase and amplitude samples respectively. The DAC converts the digital samples of the generated waveforms to an analog format. The following section discusses the signal generator prototype's modules.

5.2.2.1 Communication Interface

The interface is made up of an UART in the Cortex-M1 processor system which connects to an off-chip USB-to-UART chip. This allows communication with the target system via a COM port on a desktop computer using the HyperTerminal. It facilitates the selection of frequency and wave form. It is also possible to use a combination of a liquid crystal display (LCD) and three buttons on the Fusion Embedded Development Kit to control the signal generator as an alternative to the desktop computer; this is achieved by loading software corresponding to the preferred method of communication in the processor's memory.

5.2.2.2 Cortex^{TM}-M1 processor System description

The DDFS signal generator design contains a Cortex-M1 processor system; its main tasks are to receive the desired frequency and type of waveform from the user. The frequency information is used to calculate the phase increment value while the waveform information is used to determine the type of waveform to be generated. The 32 bit GPIO block of the processor system is used to send out this data. The phase increment value is sent out using the first 30 bits of GPIO output port and connected to the phase accumulator, the remaining two bits of GPIO output port are used to send out the waveform type information to the waveform multiplexor.

The Cortex-M1 processor system hardware was created using smart design, as described in (Actel Corporation, 2009). The software was made using Actel SoftConsole IDE v3.1 as described in (Actel Corporation, 2009). The main operation flow of the software in the cortex™-M1 processor System is illustrated in Figure 5.4.

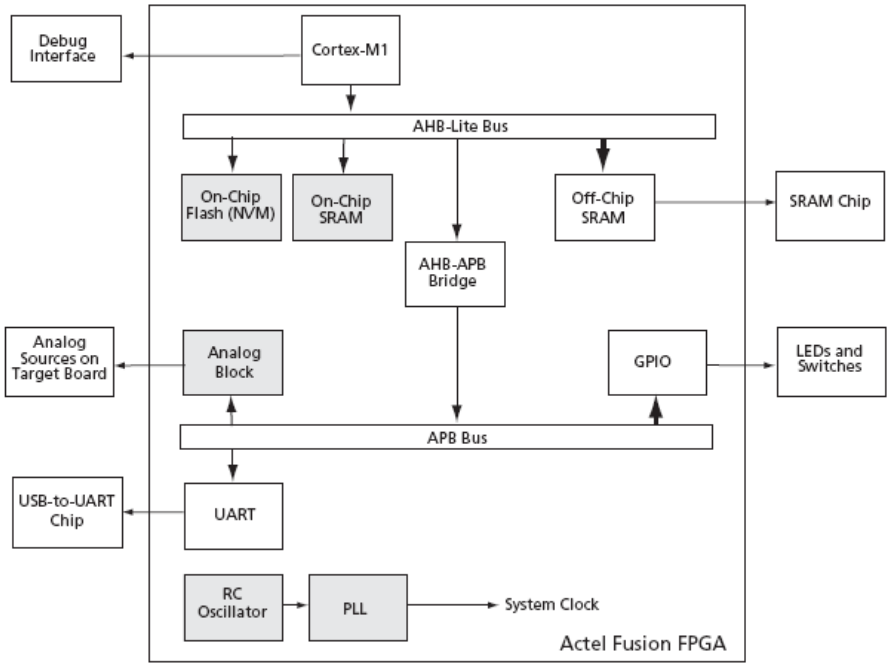


Figure 5.3: Block diagram of the cortex™-M1 processor System (Actel Corporation, 2009)

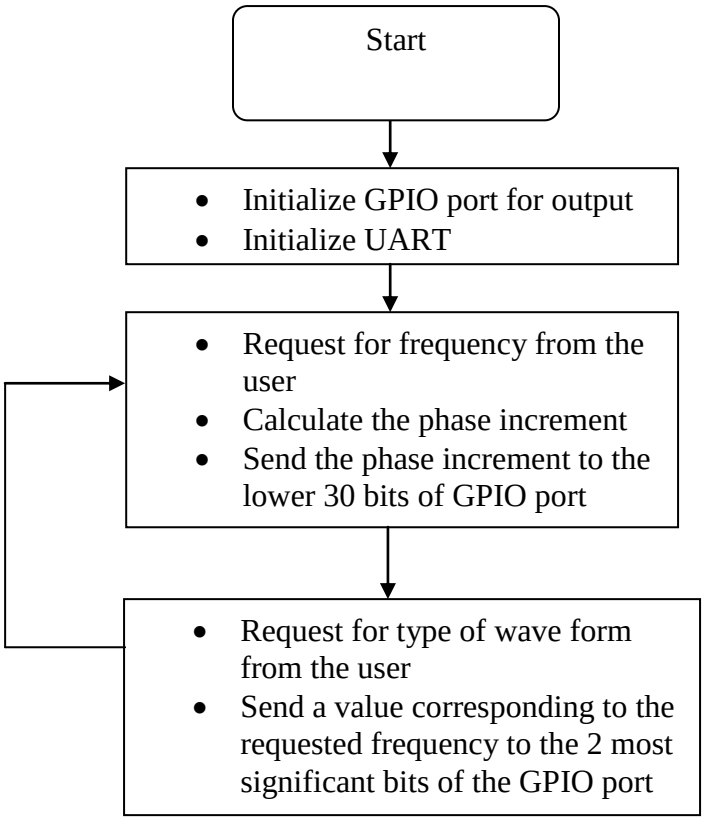


Figure 5.4: Main software routine flow

5.2.2.3 Digital to Analog Converter (DAC)

The DAC is responsible for converting waveform samples generated inside the FPGA from digital format to analog. It was connected externally to the FPGA as shown in Figure 5.2. Ten bits containing the waveform data are connected to the inputs of the DAC, subsequently the DAC converts this data to an analog signal of peak to peak voltage one volt. The DAC used in this project is DAC5652A; it is a dual, 10-bit 275 MSPS digital to analog converter. Reference (Texas Instruments, 2009) gives a detailed description of the DAC. The circuit diagram and printed circuit board (PCB) layout for this DAC was made using Proteus schematic capture software release 7.1. The circuit diagram is presented in appendix B.

5.2.2.4 Phase accumulator

This module receives a 30 bit phase increment word from the cortex™-M1 processor System and uses it to produce a linearly increasing digital value. The accumulator is an intellectual property (IP) core found in the libero IDE catalog. It was instantiated in a SmartDesign canvas named DDFS for interconnection with other modules. 8 Most Significant Bits (MSBs) of the accumulator are used by the phase to amplitude converter, 10MSBs are used to generate a saw tooth and a triangle wave while the MSB is used to create a square wave.

Figure 5.5 shows the simulation result of the phase accumulator module. As shown in the figure the phase accumulator module is capable of adding the phase increment value to the previous content of the phase accumulation register on every clock pulse, the phase increment value used for this test is 21474836, the size of the phase accumulation was 30 bits and the clock frequency was 50 MHz.

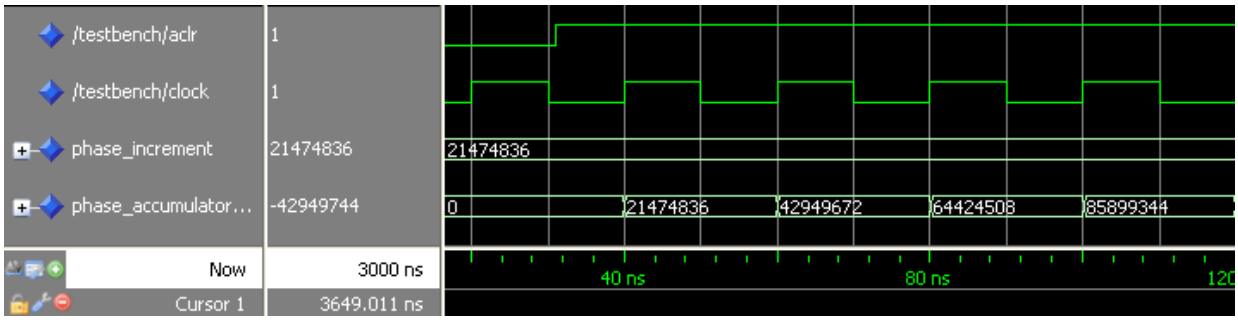


Figure 5.5: Simulation result of the phase accumulator module

The phase accumulator was also tested to confirm whether the phase accumulation register over flows at the correct time for any particular frequency. As shown in Figure 5.6 the phase accumulation register overflowed every 1000 nanoseconds as expected for a frequency of 1 MHz.

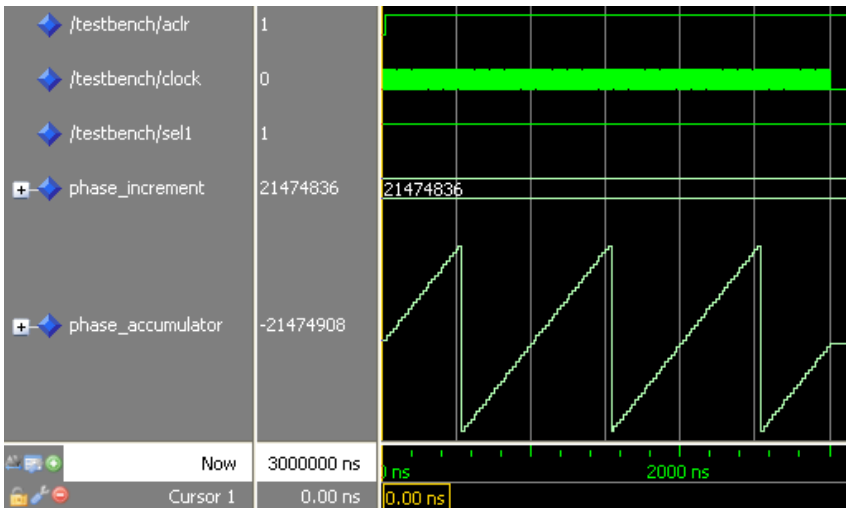


Figure 5.6: Simulation result for the overflow of the phase accumulation register

5.2.2.5 Phase to amplitude converter (ROM)

The phase to amplitude converter is a sine wave look up table (LUT) which converts the phase accumulator value to an amplitude value. The look up table maps the full scale of the phase value output by the phase accumulator to one cycle of a sine wave. As the phase value increases from 0 to full scale, one sine wave is created. This sine wave lookup table is generally implemented using a read only memory (ROM) which stores the sine trigonometric function. In this project the ROM was created using VHDL and then instantiated in a SmartDesign canvas named DDFS for interconnection with other modules. The ROM uses an 8 bit address value obtained from the phase accumulator and outputs a 16 bit value representing the sine wave; only 10 bits of this data are used to generate the analog signal. Figure 5.7 shows a part of the VHDL code for the LUT, the full code is available in appendix A. The VHDL LUT is similar to the phase to amplitude converter described earlier in section 4.5.3.

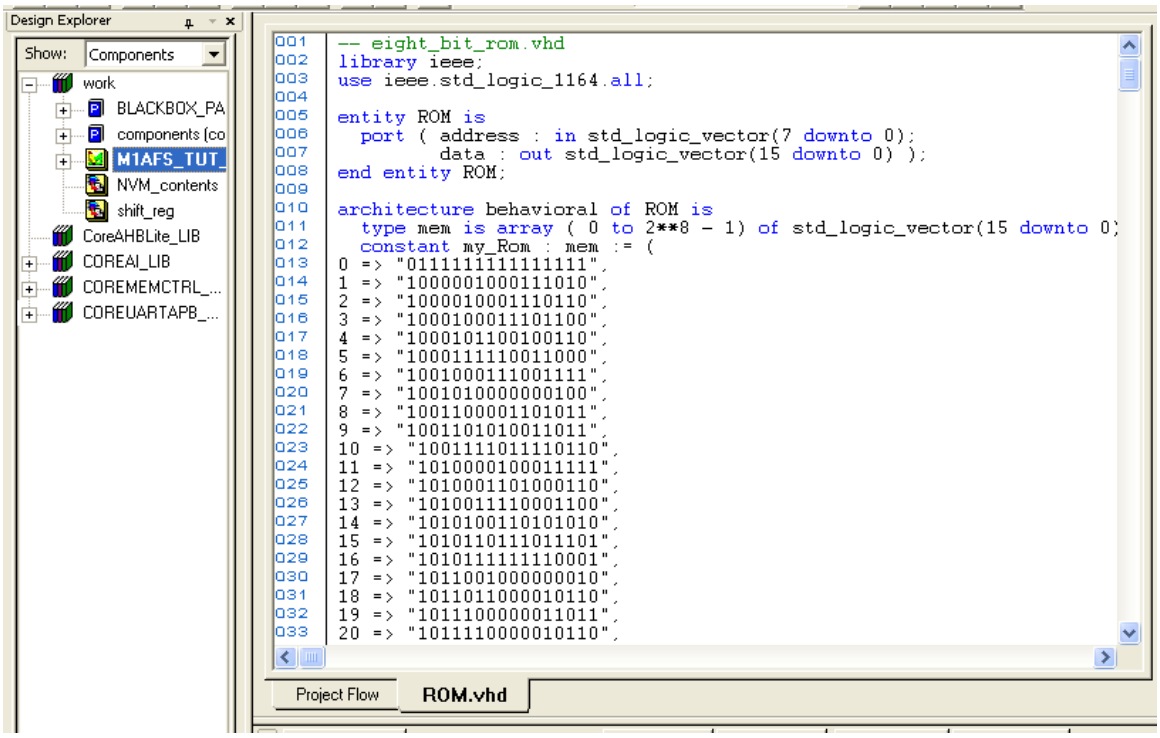


Figure 5.7: Sample VHDL code for the LUT

Figure 5.8 shows the simulation result of the phase to amplitude converter module. As shown in the figure the module phase to amplitude converter is capable of generating an amplitude value for any phase value on every clock pulse.

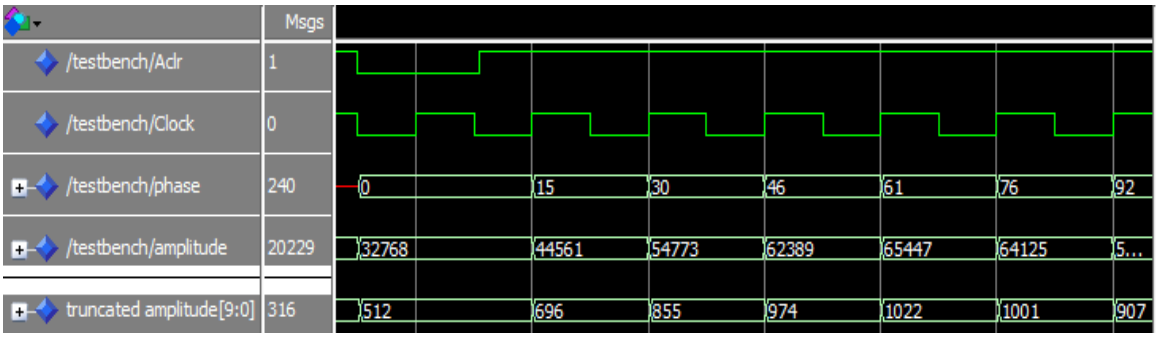


Figure 5.8: Simulation result of the phase_to_amplitude module

The phase to amplitude converter module was also tested to confirm whether it can generate signals with the expected period for any particular frequency, as shown in Figure 5.9

the module generates a complete sine wave every 1000 nanoseconds as expected for a frequency of 1 MHz.

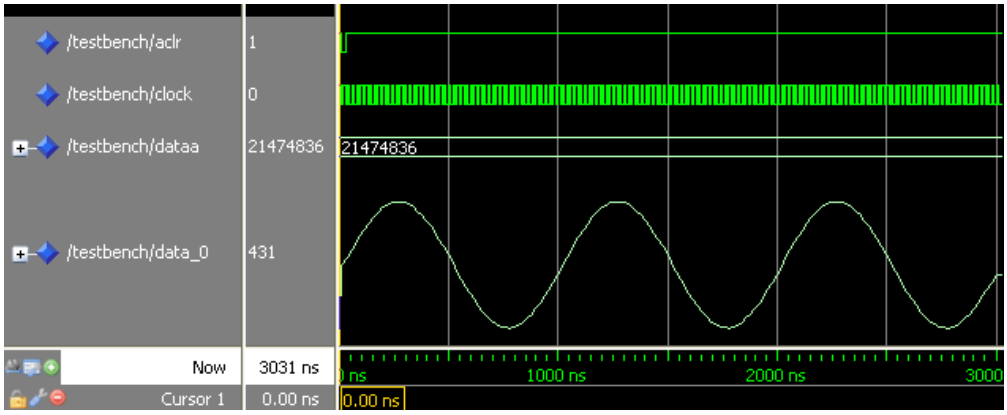


Figure 5.9: Simulation result for frequency period testing

5.2.2.6 Saw tooth, Square and Triangle wave generator

This module receives 10 MSBs from the phase accumulator and uses them to generate saw tooth, square and triangle wave form as follows:

1) The square wave is generated by connecting the most significant bit of the 10 bit phase value to all the bits of a ten bit register; when the MSB of the phase value is high the 10 bit square waveform amplitude register is set to its maximum value otherwise it is set to zero.
2) The saw tooth waveform is directly generated from the 10 most significant bits of the 30 bit phase accumulator because the phase accumulator output consists of an increasing value which rolls over when the phase accumulation register is full.
3) The triangle wave generation process involves inverting all values below one half of the 10 bit full scale value, then subtracting one quarter of the full scale value. The one quarter scale shift is performed to keep the signal swing centered on the one half scale value

(Landry, 1999). The triangle wave logic was implemented using a multiplexor and inverter IP cores which were instantiated in the square, saw tooth and triangle wave generator SmartDesign canvas.

Figure 5.10 shows how the square, saw tooth and triangle wave generator was implemented. It was instantiated in a SmartDesign canvas named DDFS for interconnection with other modules.

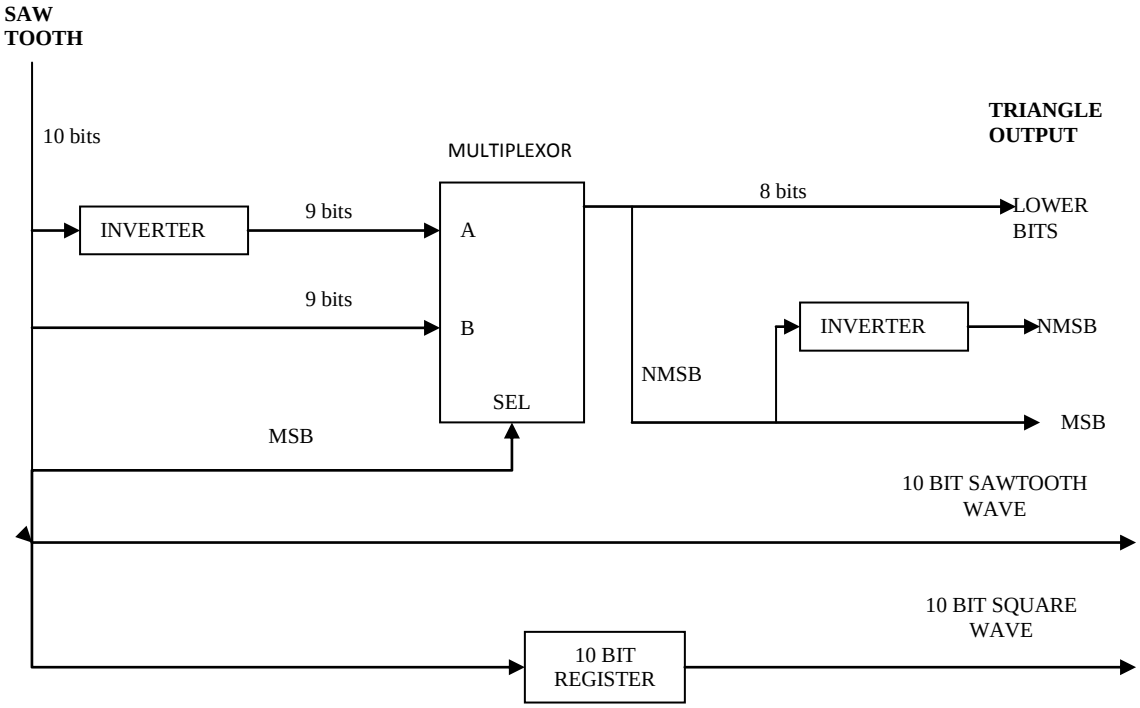


Figure 5.10: Block diagram of the saw tooth, square and Triangle wave generator

Figure 5.11 shows the simulation result of the saw tooth, square and triangle wave generator module. As shown in the figure the module is capable of generating the saw tooth, square and triangle waveforms from the phase information. The size of the phase information used is 10 bits, the size of the triangle wave amplitude is 10 bits and the clock frequency is 50 MHz.

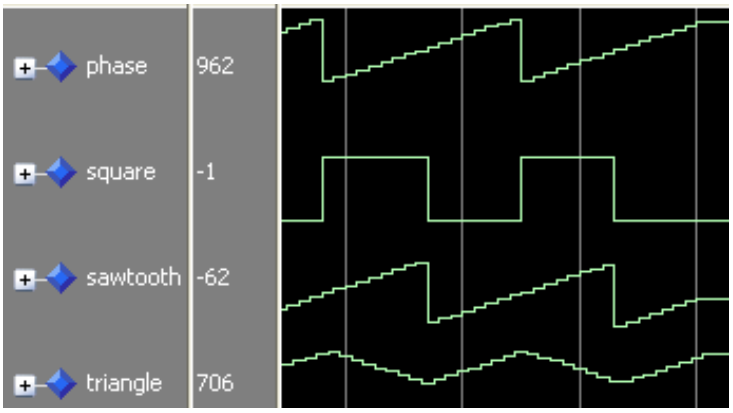


Figure 5.11: Simulation result of the saw tooth, square and triangle wave generator module

5.2.2.7 Waveform multiplexor

The waveform multiplexor receives two bits (2 MSBs) from the cortex™-M1 processor System and uses their value to determine which type of waveform is going to be channeled to the DAC. Digits 0,1,2,3 are used to choose between a square wave, sine wave, saw tooth wave and a triangle wave respectively. The other set of inputs to the multiplexor are sine, square, saw tooth and triangle waveform data from their respective generators. The waveform multiplexor was implemented using an IP core found in the libero IDE catalog, it was instantiated in a SmartDesign canvas named DDFS for interconnection with other modules. Figure 5.12 shows the block diagram of the waveform multiplexor.

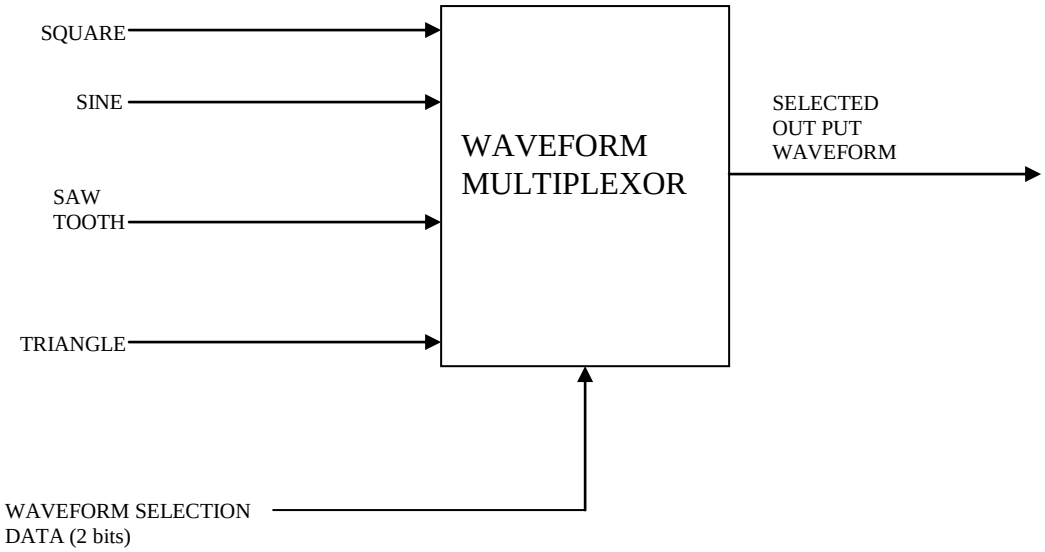


Figure 5.12: Waveform multiplexor

Figures 5.13 to 5.16 show the simulation results of the Waveform multiplexor module. As shown in the figures the module Waveform multiplexor is capable of allowing only the selected waveform at its output. The bits sel0 and sel1 are used to choose between the waveforms as shown in table 5-1 below.

Table 5- 1: Waveform selection codes

Sel0	**Sel1**	**Waveform**
0	0	Square
0	1	Saw tooth
1	0	Sine
1	1	Triangle

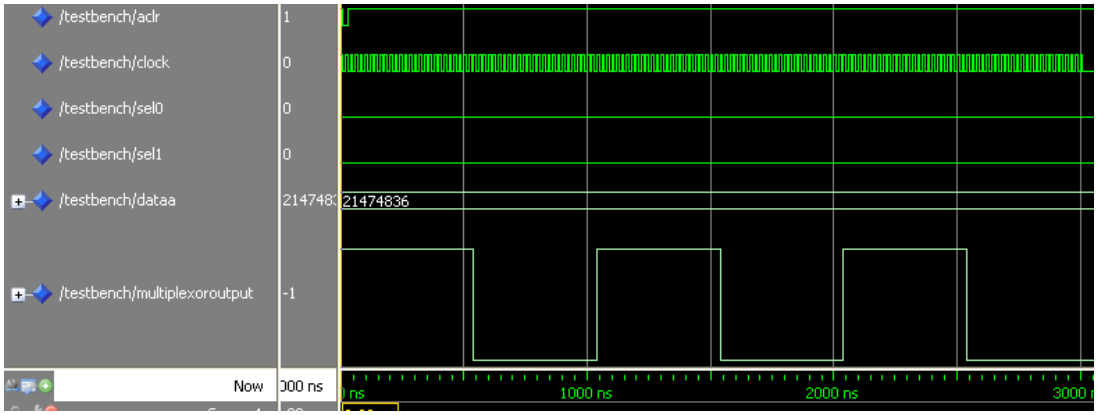


Figure 5.13: Square wave simulation for the waveform multiplexor module

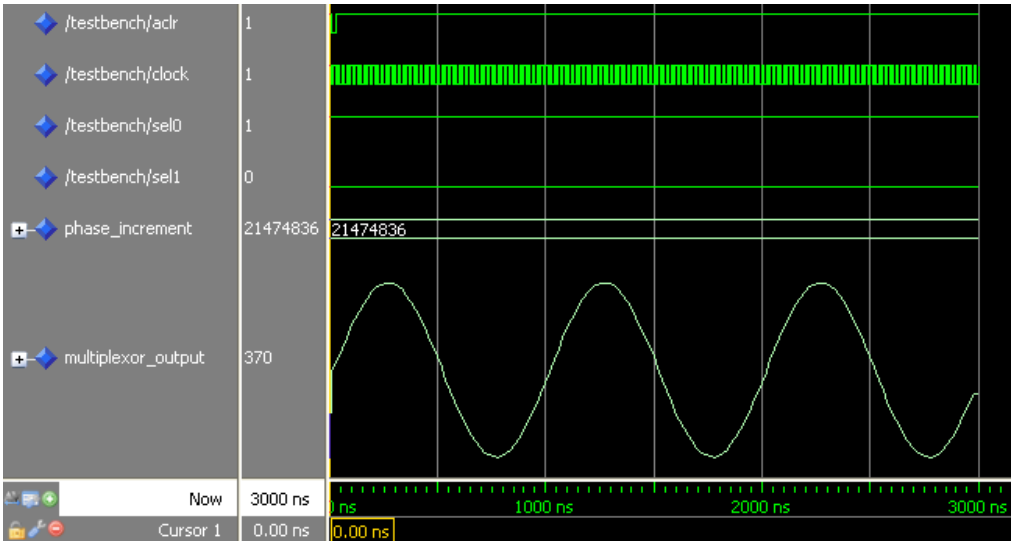


Figure 5.14: Sine wave simulation for the waveform multiplexor module

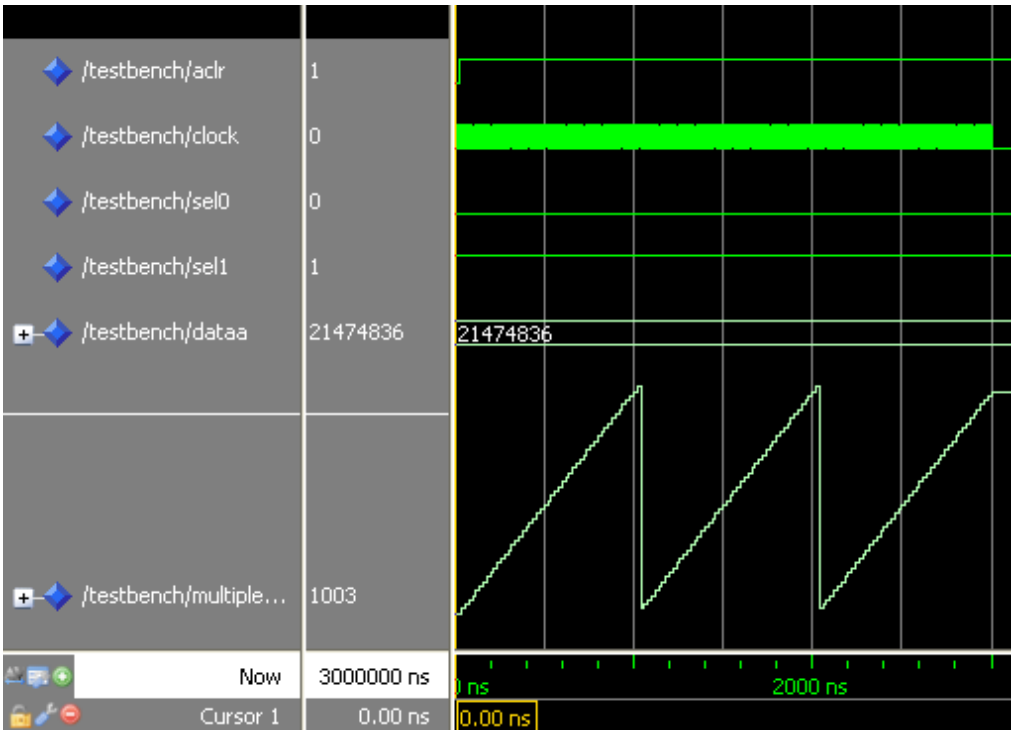


Figure 5.15: Saw tooth wave simulation for the waveform multiplexor module

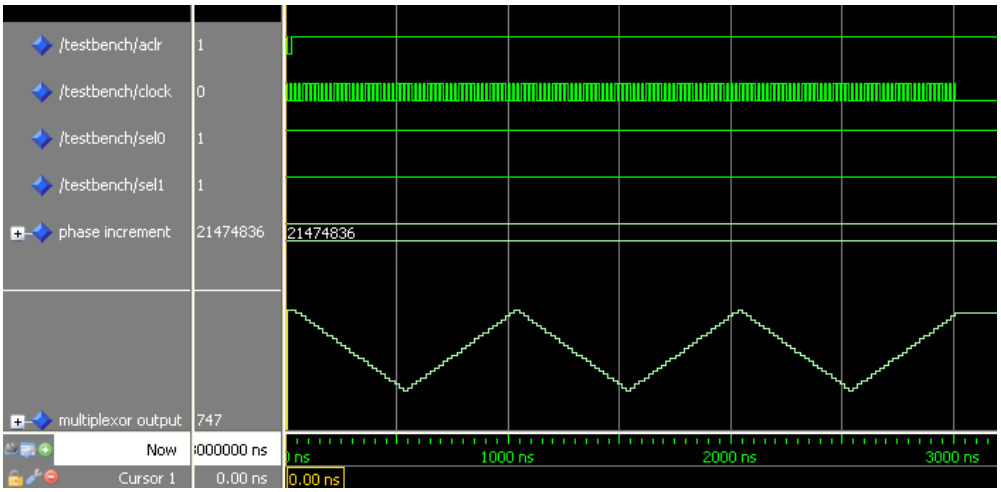


Figure 5.16: Triangle wave simulation for the waveform multiplexor module

5.2.2.8 Phase dithering module

The phase dithering module shown in Figure 5.17 is responsible for generating a dither signal and adding this dither signal to the phase information. It consists of an adder and a linear

feedback shift register pseudo random number generator that is composed of a Serial-In/Parallel-Out shift register and two exclusive or gates. A discussion on phase dithering and dither signal generation is presented in section 3.4 and 3.7 respectively. The adder, shift register and exclusive or gates are IP cores that were instantiated in the phase dithering module SmartDesign canvas and interconnected. The phase dithering module was also instantiated in a SmartDesign canvas named DDFS and connected to the phase accumulator.

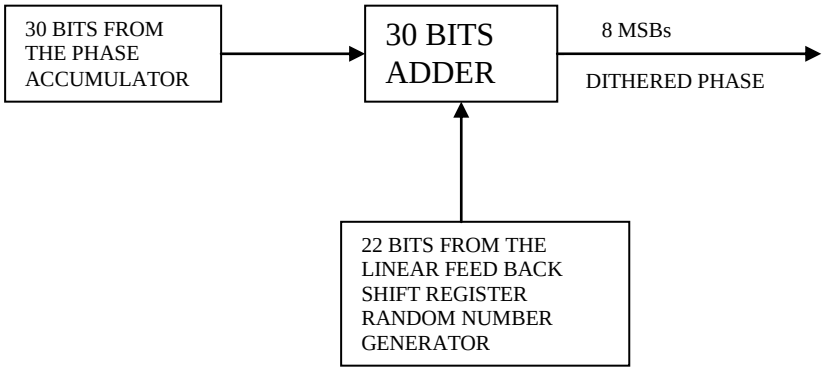


Figure 5.17: Phase dithering module

Figure 5.18 shows the simulation result of the phase dithering module. As shown in the figure the phase dithering module is capable of generating and adding a dither signal to the phase information on every clock pulse, the size of the dither signal used is 22 bits, the size of the phase information was 30 bits and the clock frequency was 50 MHz.

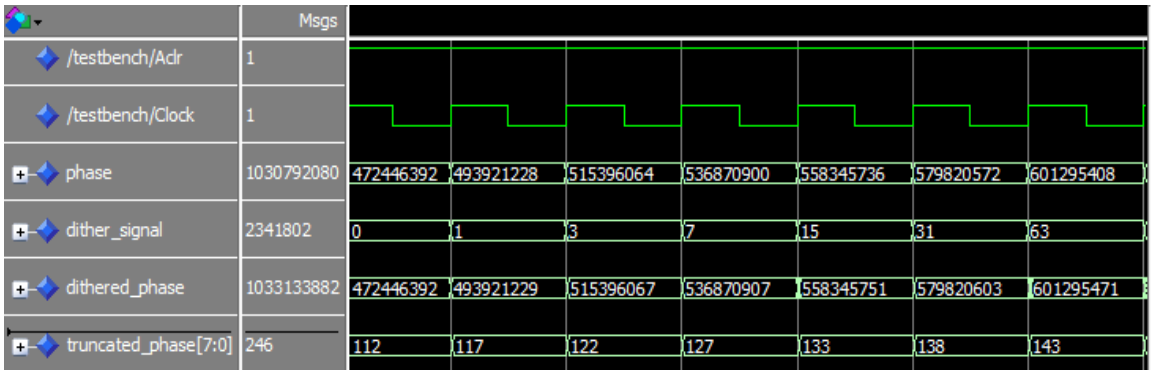


Figure 5.18: Simulation result of the phase dithering module

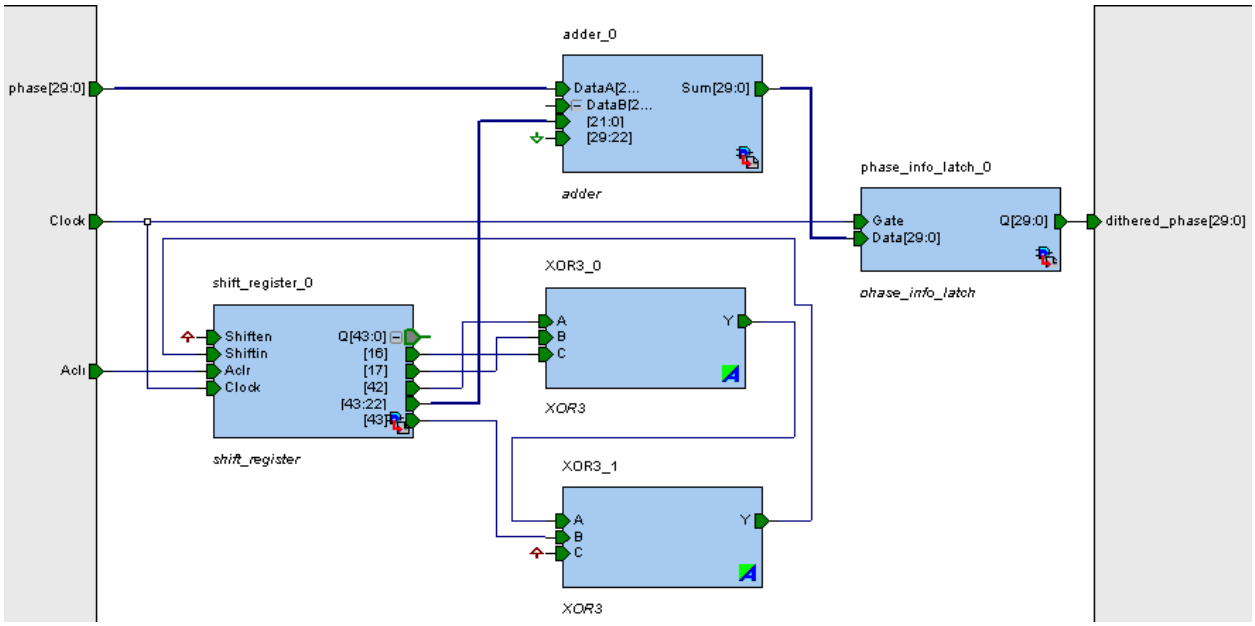


Figure 5.19: FPGA implementation of the phase dithering module

5.2.2.9 Amplitude dithering module

The amplitude dithering module shown in Figure 5.20 is responsible for generating a dither signal and adding this dither signal to the amplitude information obtained from ROM. It consists of an adder and a linear feedback shift register pseudo random number generator that is composed of a Serial-In/Parallel-Out shift register and two exclusive OR gates. A discussion on amplitude dithering and dither signal generation is presented in section 3.5 and 3.7 respectively. The adder, shift register and exclusive OR gates are IP cores that were instantiated in the amplitude dithering module smart design canvas and interconnected; the amplitude dithering module was also instantiated in a SmartDesign canvas named DDFS and connected to the phase to amplitude converter and waveform multiplexor.

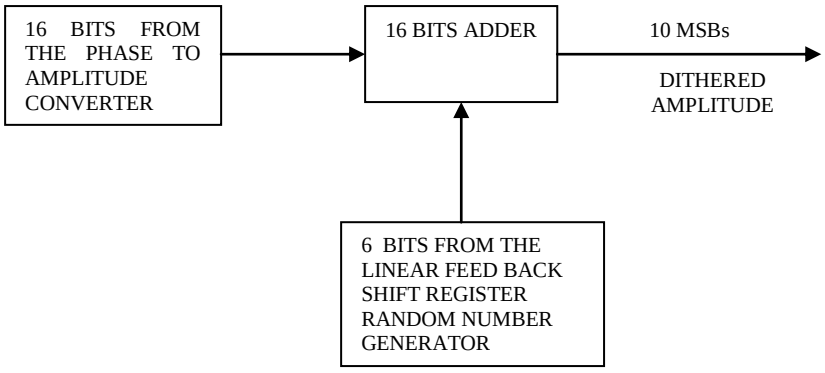


Figure 5.20: Amplitude dithering module

Figure 5.21 shows the simulation result of the amplitude dithering module. As shown in the figure the amplitude dithering module is capable of adding a dither signal to the amplitude information on every clock pulse, the size of the dither signal used is 6 bits, the size of the amplitude information was 16 bits and the clock frequency was 50 MHz.

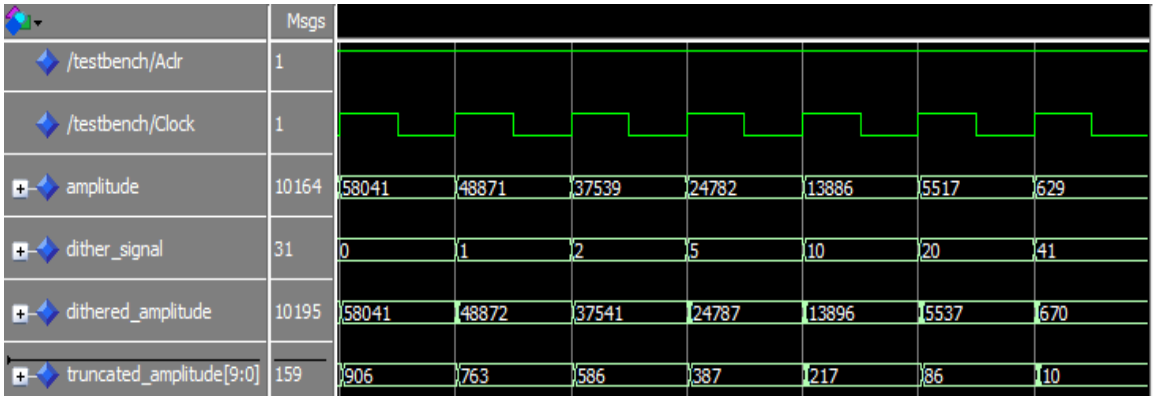


Figure 5.21: Simulation result of the amplitude dithering module

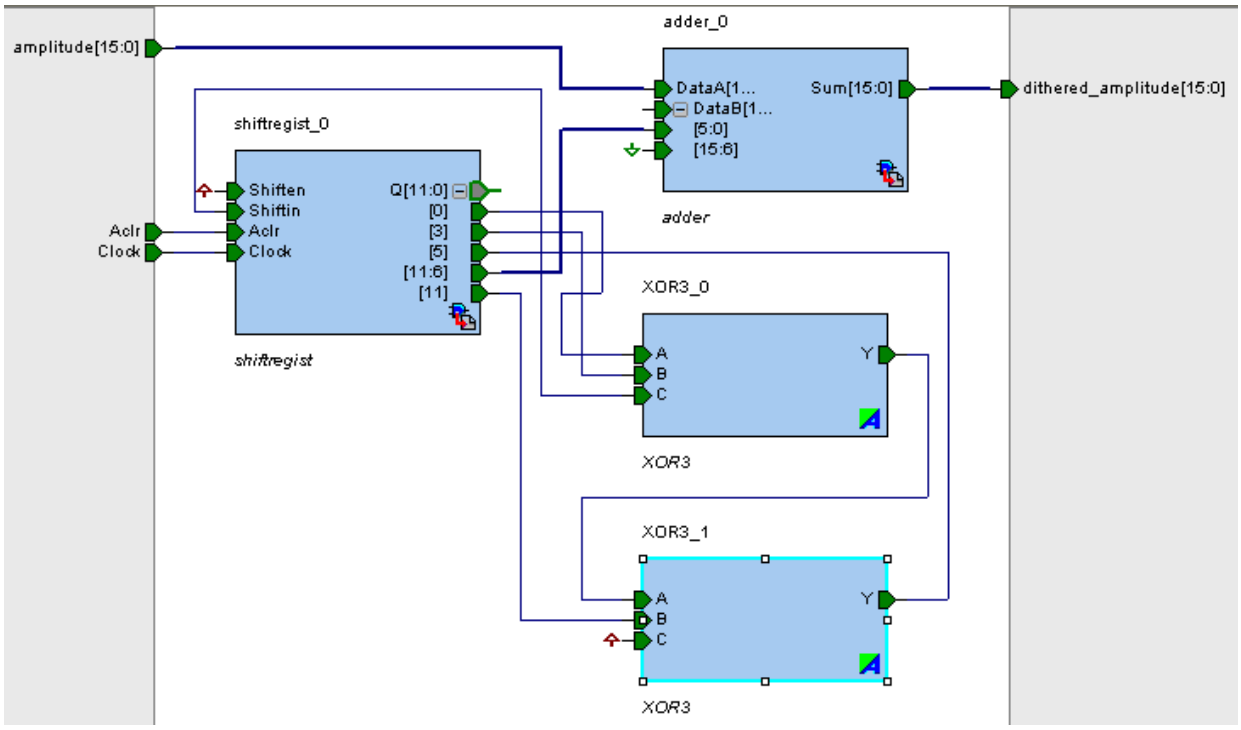


Figure 5.22: FPGA implementation of the amplitude dithering module

Figure 5.23 shows how the various modules (presented in section 5.2.2.1-9) of the signal generator prototype were interconnected in the SmartDesign canvas named DDFS.

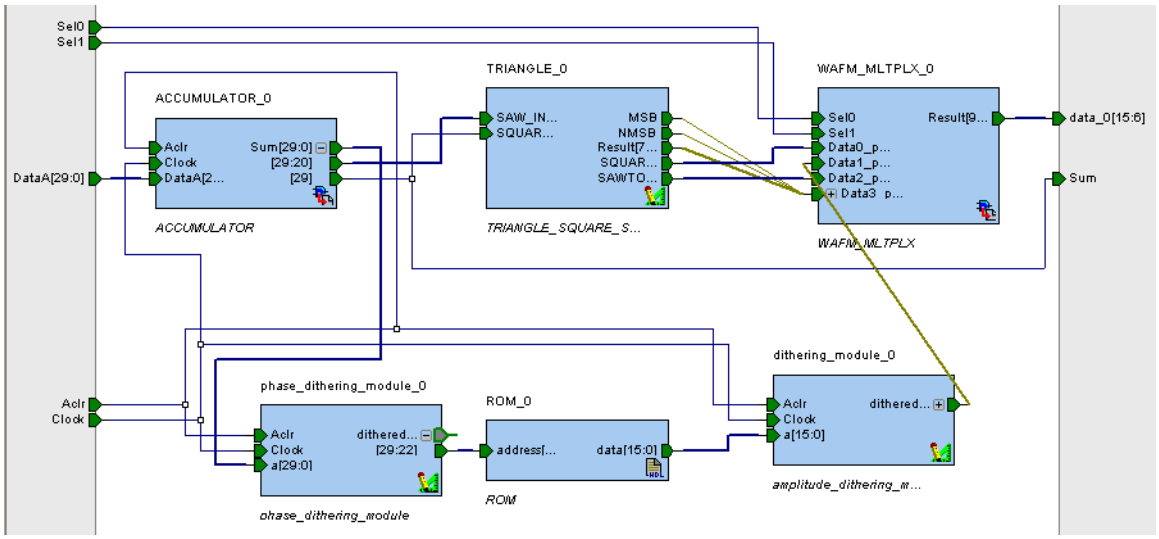


Figure 5.23: Signal generator modules

5.3 Testing Method

Testing of the signal generator prototype was done in order to verify whether it could generate wave forms of correct shape and expected frequency. In addition, figures of SFDR and noise floor for this prototype were to be obtained at this stage. The test setup included the Fusion FPGA Embedded Development Kit Evaluation Board, DAC5652module, USB 2.0 high-speed cable, oscilloscope and desktop computer.

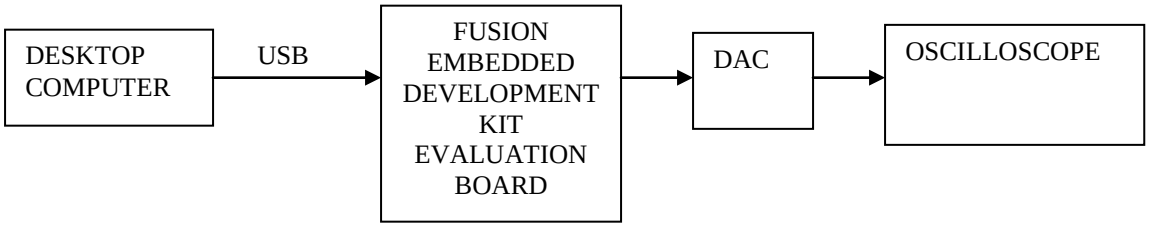


Figure 5.24: Hardware Test Setup

In order to test the signal generator, frequency and waveform data are sent to the FPGA. The interface used to request and send the data is made up of an UART in the Cortex-M1 processor system which connects to an off-chip USB-to-UART chip. This allows communication with the Fusion Embedded Development Kit Evaluation Board via a COM port on a desktop computer using the HyperTerminal. The routine for the software used to request and send wave form data is discussed in section 5.2.2.2.

Two sets of results were collected from testing the signal generator prototype with phase and amplitude dithering. The first set was for waveform analysis while the second was for spectral analysis. Waveforms generated after testing the signal generator prototype with phase and amplitude dithering are shown in Figures 5.25 to 5.29. From the figures it can be seen that the model with phase and amplitude dithering was able to generate sine, square, triangle and saw tooth wave forms. The waveforms were observed to be of expected frequency (1 MHz) and expected shape. In addition the prototype was capable of generating signals at the targeted

resolution of 0.047 Hz as can be observed in Figure 5.29. Output spectrum results are presented in section 5.4.

Figures 5.30 to 5.33 show 10 MHz sample waveforms while Figures 5.34 to 5.37 show additional samples at a frequency of 25 MHz. When the sample waveforms are compared it can be observed that the waveforms generated get distorted as the frequency increases from 1 MHz to 25 MHz. One possible explanation of the sine, saw tooth and triangle wave distortion is that the samples used in generating these waveforms reduce as the output frequency of the DDFS increases. The deterioration in the quality of the square wave can be attributed to stray capacitances in the circuit which cause it to behave like a low pass filter, thereby eliminating the high frequency components of the square wave.

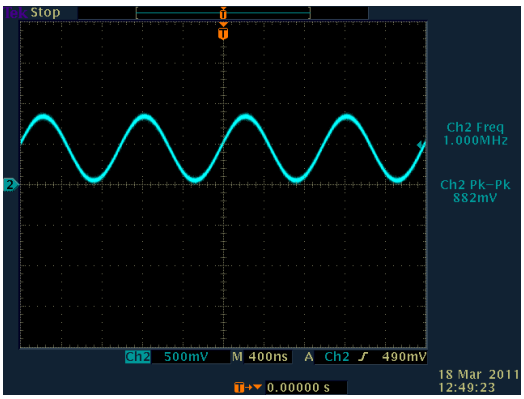


Figure 5.25: 1 MHz Sine wave output

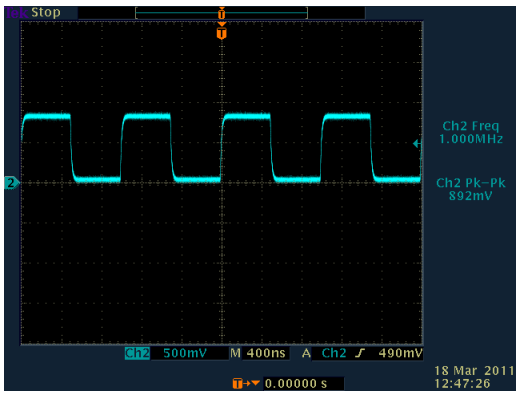


Figure 5.26: 1 MHz Square wave output

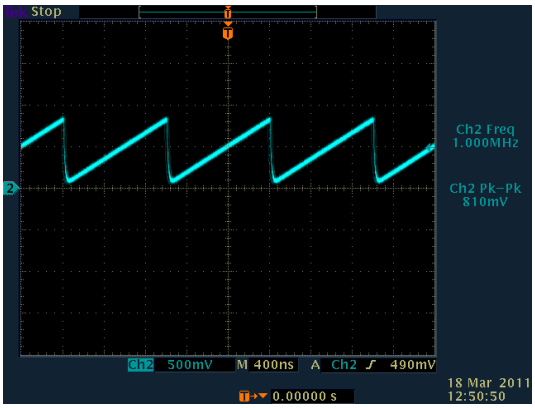


Figure 5.27: 1 MHz Saw tooth wave output

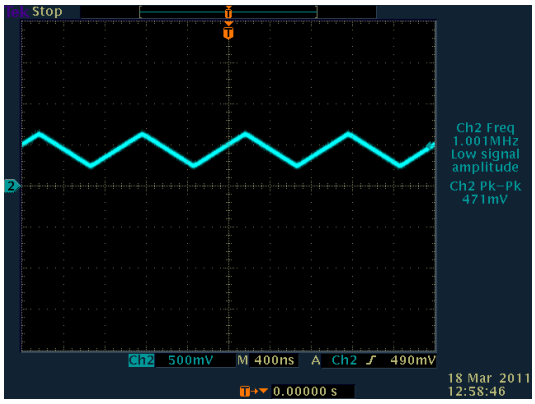


Figure 5.28: 1 MHz Triangle wave output

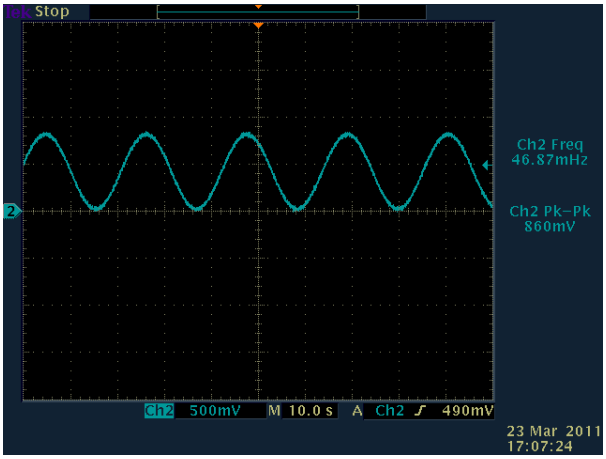


Figure 5.29: 0.047 Hz Sine wave output

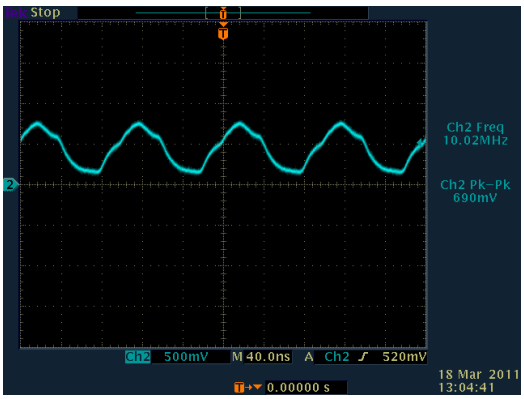


Figure 5.30: 10 MHz Sine wave output

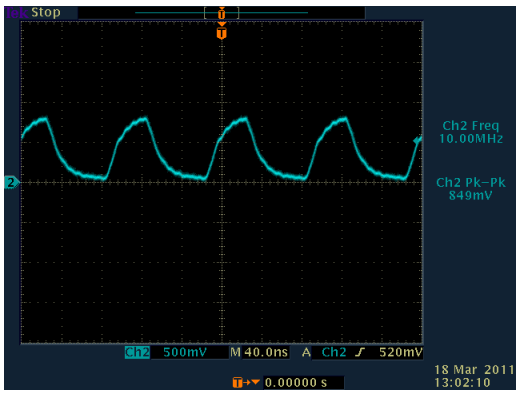


Figure 5.31: 10 MHz Square wave output

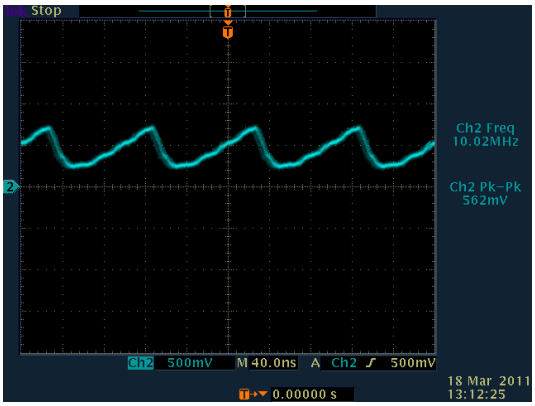


Figure 5.32: 10 MHz Saw tooth wave output

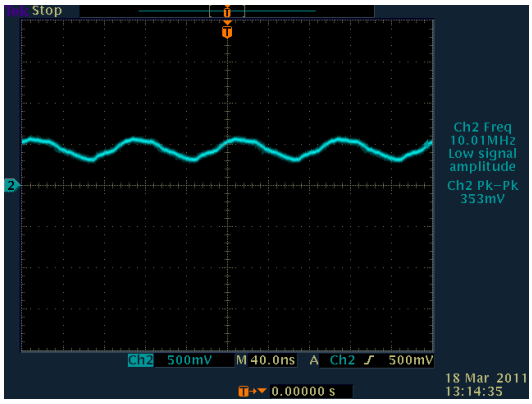


Figure 5.33: 10 MHz Triangle wave output

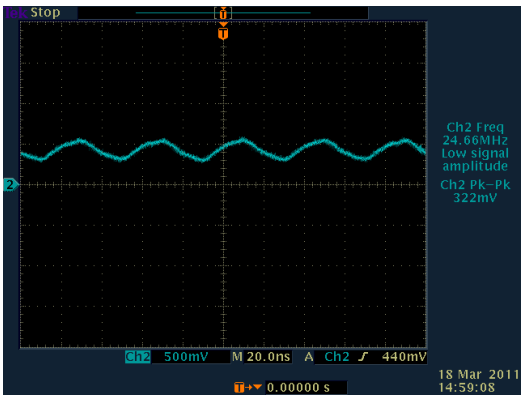


Figure 5.34: 25 MHz Sine wave output

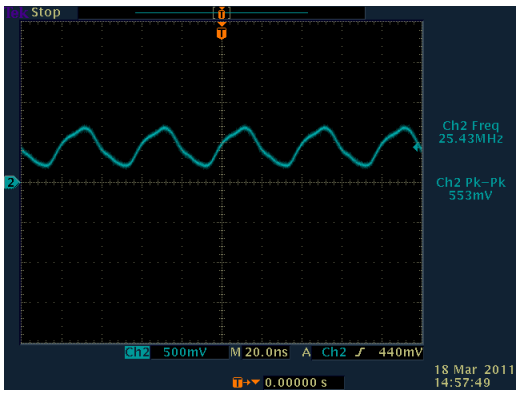


Figure 5.35: 25 MHz Square wave output

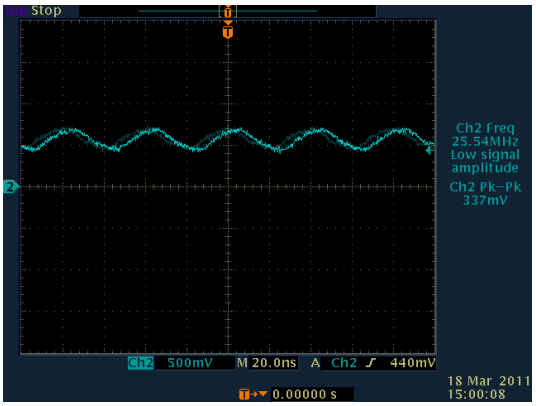


Figure 5.36: 25 MHz Saw tooth wave output

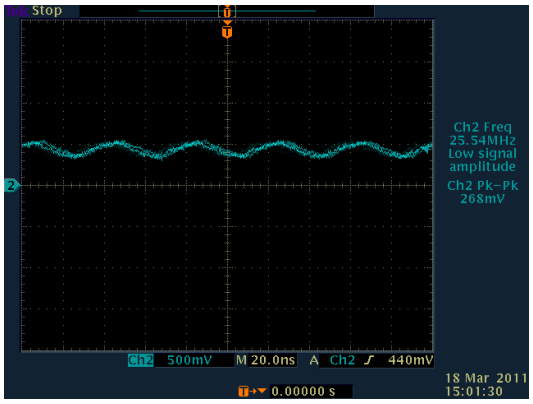
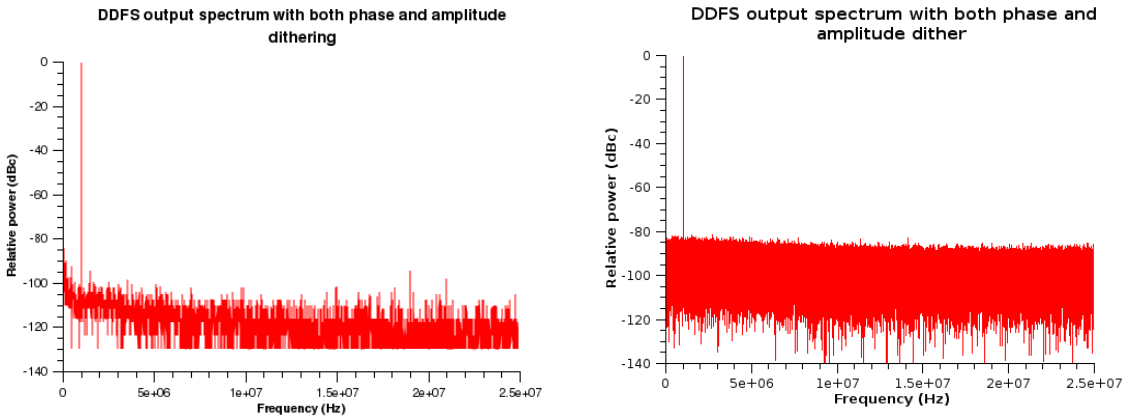


Figure 5.37: 25 MHz Triangle wave output

5.4 FPGA implementation output spectrum results

Figure 5.38 (a) shows the observed output spectrum result for the signal generator prototype. The spectrum was obtained using the math function of the TDS3014B oscilloscope. The parameters used to generate the sine waves whose spectral properties are shown in Figure 5.38 (a) were j = 30, k = 8, m = 10, f_{out} =1MHz. The observed SFDR was 85 dBc and the noise floor was at -116dBc. More SFDR and NF results for different frequencies are presented in Table 5-2.

For comparison purposes, the output spectrum result of the signal generator prototype (Figure 5.38 (a)) and that of the corresponding SystemC signal generator design (Figure 5.38 (b)) are presented together. From the two figures it can be observed that the performance of the signal generator prototype closely matches that of the SystemC simulation model.



(a) (b)

Figure 5.38: DDFS output spectra (a) FPGA prototype testing result, (b) SystemC simulation result

5.5 Discussion of FPGA implementation testing results

Tables 5-2 and 5-3 contain a summary of SFDR and NF results respectively that were obtained from simulating the SystemC models, the expected theoretical values derived from the appropriate equations as presented in section 4.12, and those obtained from testing the signal generator FPGA implementation.

Table 5-2: SFDR results

OUTPUT FREQUENCY	THEORETICAL SFDR VALUE (dBc)	SYSTEMC SIMULATION SFDR VALUE (dBc)	FPGA PROTOTYPE TESTING SFDR VALUE (dBc)	DEVIATION OF FPGA PROTOTYPE SFDR FROM THEORETICAL VALUE (dBc)
1 kHz	88	88	102	-14
1 MHz	88	88	102	-14
5 MHz	88	88	85	3
10 MHz	88	88	79	9
15 MHz	88	88	79	9
20 MHz	88	88	62	26
24 MHz	88	88	60	28
25 MHz	88	88	59	29

Mean deviation of FPGA prototype SFDR value from the theoretical value;

$$= \frac{-14-14+3+9+9+26+28+29}{8} = 9.5 \text{ dBc}$$

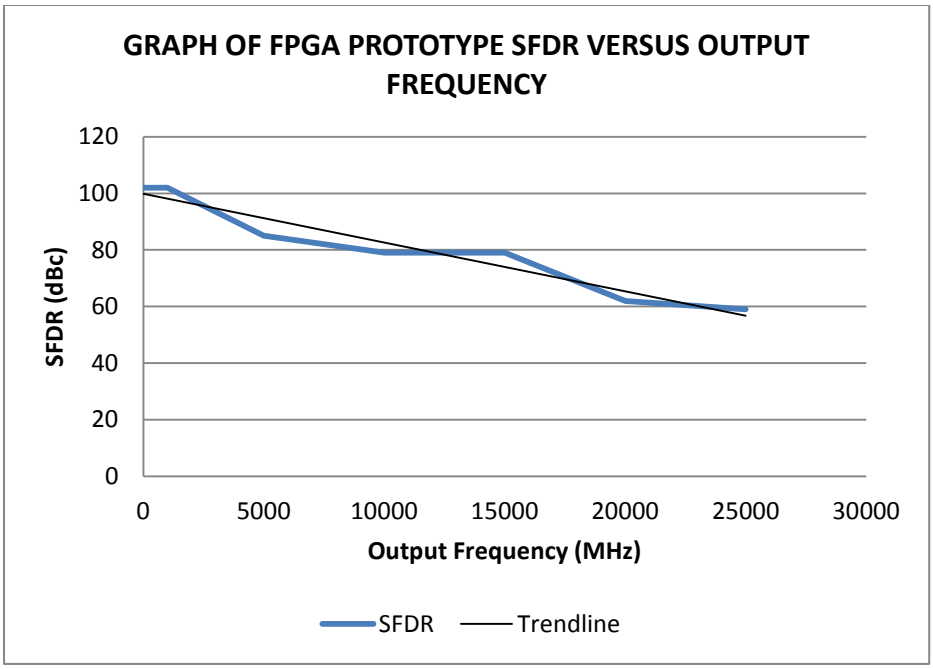


Figure 5.39: Graph of FPGA prototype SFDR versus output frequency

Table 5-3: NF results

OUTPUT FREQUENCY	THEORETICAL NF VALUE (dBc)	SYSTEMC SIMULATION NF VALUE (dBc)	FPGA PROTOTYPE TESTING NF VALUE (dBc)	DEVIATION OF FPGA PROTOTYPE NF FROM THEORETICAL VALUE (dBc)
1 kHz	-123	-88	-116	-7
1 MHz	-123	-88	-116	-7
5 MHz	-123	-88	-115	-8
10 MHz	-123	-88	-111	-12
15 MHz	-123	-88	-106	-17
20 MHz	-123	-88	-101	-22
25 MHz	-123	-88	-101	-22

Mean deviation of FPGA prototype NF value from the theoretical value;

$$= \frac{-7-7-8-12-17-22-22}{7} = -13.6 \text{ dBc}$$

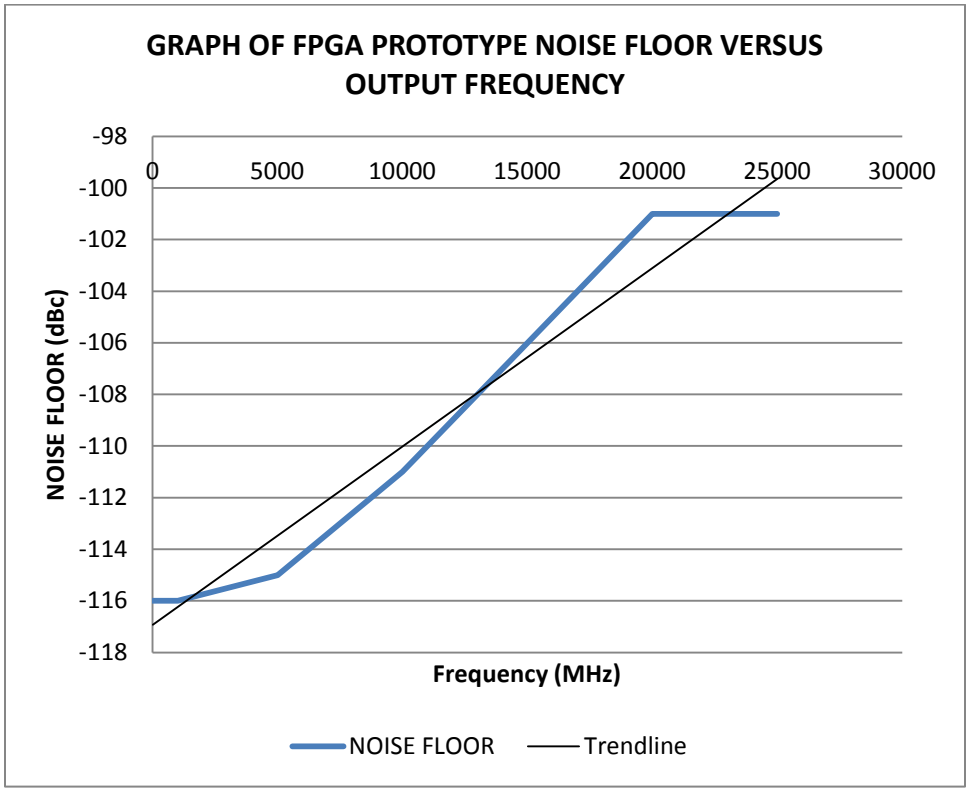


Figure 5.40: Graph of FPGA prototype NF versus output frequency

From the figures presented in the table 5-2 and 5-3 it can be observed that:

1. The FPGA prototype with both phase and amplitude dithering had an SFDR that differed slightly from the one observed in the corresponding SystemC model and the estimated theoretical value by an average of 9.5 dBc. The observed high SFDR is anticipated because phase dithering was expected to reduce the spurs caused by phase truncation while amplitude dithering was used to minimize the spurs caused by amplitude quantization. However the SFDR reduces as the output frequency increases as can be observed in Figure 5.39. A possible explanation of why the SFDR gets lower as the output frequency increases is that the number of samples generated by the DDFS for reconstructing the sine wave reduces as

the output frequency increases; hence the waveforms get distorted gradually as the output frequency increases. The distortion is observed as spurs in the frequency domain. Equation 5.1 shows how the output frequency is related to the number of samples. From the equation it can be observed that for a constant clock frequency, the number of samples decrease as the output frequency increases.

$$n = \frac{F_{clk}}{F_{out}} \qquad 5.1$$

Where:

n is the number of samples in waveform

F_{clk} is the clock or sampling frequency

F_{out} is the output frequency

2. The observed values of noise floor are on average -13.6 dBc higher than their corresponding theoretical values; for example it can be seen in table 5-3 that at frequency of 5 MHz the NF is -115 dBc while the theoretical NF value is -123 dBc. The cause of this difference is the number of points used in the FFT analysis; 10,000 points were used instead of the required 268,435,456 (using equation 4.23). The use of more than 10,000 points was not possible because the oscilloscope used (TDS3014B) for the FFT analysis could support the FFT analysis of 10,000 points only. However the used numbers of points were enough to avoid masking the expected improvement in SFDR.
3. From Figure 5.40 the noise floor is observed to increase as the output frequency increases. One possible cause could be the deterioration of the quality of the waveform as the output frequency increases, as explained in the first observation.

5.6 Conclusion of results

From Table 5-2 it can be observed that the FPGA prototype that had phase and amplitude dithering resulted in an SFDR that is close to the one observed at the modeling and simulation level and the expected theoretical value. However, the SFDR decreased as output frequency increased mainly due to the fact that the number of samples in the output waveform decreased as the output frequency increased.

CHAPTER 6

CONCLUSION AND RECOMMENDATION

6.1 Conclusion

The aim of this research was to design and implement a spectral purity optimized DDFS based signal generator in an FPGA, but the research also includes an investigation on the dithering technique of DDFS spur reduction and the use of the relatively new modeling and simulation language called SystemC. Although there is a lot of material on DDFS, dithering and SystemC that already exists in the literature, the author believes that this research is the first to do the following;

1. Demonstrate how to use SystemC in the modeling and simulation of DDFS designs that use phase dithering, amplitude dithering and a combination of phase and amplitude dithering. The models can be useful to someone who wants to understand the dithering technique of spur reduction or to carry out more research on dithering. In addition, use of these SystemC models would enable the user to benefit from the advantages of SystemC over other languages such as faster simulation time and the ability to model both at the behavior and architecture level of the design.
2. Compare using the same DDFS design the performance of three methods of dithering namely; phase dithering, amplitude dithering and a combination of phase and amplitude dithering. The results of this comparison showed that the phase and amplitude method of dithering resulted in the most reduction in spurs. This information could be useful to a designer who wants to apply the dithering technique of spur reduction but is not sure which method of dithering to choose.

The results of the investigation in dithering proved that the phase and amplitude method

of dithering provided the largest reduction in spurs when compared to phase dithering only or amplitude dithering only. One possible explanation for this observation is that truncating the phase information and finite word length representation of amplitude information in a DDFS results in two sets of spurs; those caused by phase information truncation and the ones caused by finite precision of the amplitude samples, hence spur reduction efforts in such a DDFS design should target the two sets of spurs.

The implemented FPGA prototype of the signal generator had phase and amplitude dithering due to the fact that this design was observed to offer the highest spectral purity at the modeling and simulation level. The prototype was capable of generating sine, square, saw tooth and triangle wave forms, the frequency resolution was 0.047 Hz, maximum output frequency was 25 MHz, sine wave SFDR was improved from 48 dBc to 85 dBc and the noise floor was -116 dBc.

6.2 Recommendation

The expense of phase dithering is an increased noise floor while the penalty of amplitude dithering is a reduced dynamic range and a loss of the amplitude information caused by the need to scale the amplitude information so that the original signal plus the dither will stay within the non-saturating region. Future studies can focus on trying to overcome these challenges faced by dithering.

While SystemC was used for modeling and simulation, the crafting of the signal generator on FPGA used vendor specific tools which were not related to SystemC. There is therefore a need to integrate SystemC all the way down to realization of the signal generator on FPGA.

REFERENCES

Vankka, J., 2001, Direct Digital Synthesizers: Theory, Design and Applications, Kluwer Academic Publishers, London.

Popa, M., and Sorana, A., 2007, Programmable signal generator based on advanced tricore microcontrollers, IEEE international conference on signal processing and communication (ICSPC 2007) , 24-27 November 2007 , Dubai United Arab Emirates.

Analog devices, 2003, AD9833, application note, www.analog.com, last accessed on 27 august 2009.

Sia, L. H., Jamuar S. S., Sidek, R. M., and Marhaban, M., H., 2007, Digital signal-Processor-Based waveform generator, Measurement science and technology 18, 35- 40.

Sharma, R. K., and Upadhyaya G., 2010, Memory Reduced and Fast DDS Using FPGA, International Journal of Computer Theory and Engineering, vol. 2, no. 4, pp 1793-8201.

Dubey, R., 2009, Introduction to Embedded System Design Using Field Programmable Gate Arrays, Springer, London.

Chimakurthy, L. S. J., Ghosh, M., Dai, F. F., and Jaeger, R. C., 2006, A Novel DDS Using Nonlinear ROM Addressing With Improved Compression Ratio and Quantization Noise, IEEE transactions on ultrasonics, ferroelectrics, and frequency control, vol. 53, no. 2,pp 274-283.

Kamboj, B., and Mehra, R., 2012, Efficient FPGA Implementation of Direct Digital Frequency Synthesizer for Software Radios, International Journal of Computer Applications, Volume 37, No.10, pp 0975 – 8887.

Xiaoqin, W., and Yin, S., 2007, Design and Implementation of DDS Based on VHDL, The Eighth International Conference on Electronic Measurement and Instruments, ICEMI'2007, pp2-33-2-37, Aug. 16 2007-July 18.

"Open SystemC initiative" [online]. Available: http://www.sytemc.org, last accessed on 20 July 2012.

Agostinelli, M., Priewasser, R., Huemer, M., Marsili S., and Straeussnigg, D., 2010, SystemC-AMS modeling and simulation of digitally Controlled DC-DC converters, Applied Power Electronics Conference and Exposition (APEC), 2010 Twenty-Fifth Annual IEEE, pp 170-175, 21-25 Feb. 2010.

Vachoux, A., and Grimm, C., 2003, Analog and Mixed Signal Modelling with SystemC-AMS, IEEE International Symposium on Circuits and Systems (ISCAS).

Slepička, D., 2000, Noise floor in ADC testing, Measurement (2000) Volume: 2, Issue: 4, Pages: 4-8.

Cordeses, L., 2004, Direct digital synthesis: A tool for periodic waveform generation (part 1), IEEE signal processing magazine, pp 50-54.

Cordeses, L., 2004, Direct digital synthesis: A tool for periodic waveform generation (part 2), IEEE signal processing magazine, pp 110-117.

Flanagan, M. J. , and Zimmerman, G. A., 1995, Spur-Reduced Digital Sinusoid Synthesis, IEEE transactions on communications, vol. 43, No. 7, pp2254-2262.

Xinguang, T., et al, 2009, DDFS spurious signals due to amplitude quantization in absence of phase-accumulator truncation, Journal of Systems Engineering and Electronics, Vol. 20, No. 3, pp 485-492.

Xilinx, 2007, XAPP210 (v1.3), Linear Feedback Shift Registers in Virtex Devices, www.xilinx.com, last accessed on 27 august 2009.

Bhasker, J., 2002, A SystemC Primer, Star Galaxy Publishing, Allen town.

Gajski, D. D., Abdi, S., Gerstlauer, A., and Schirner, G., 2009, Embedded System Design, Springer, London.

Silicon laboratories, 2003, AN123-DS11, using the DAC as a function generator application note. www.silabs.com, last accessed on 27 august 2009.

Texas Instruments, 2009, SLAS535A, DAC5652A datasheet, www.ti.com, last accessed on 3 June 2011.

Hsieh, J., Tsai, G., and Lin, M., 2003, Using FPGA to implement a N-channel Arbitrary Waveform Generator with Various Add-on Functions, proceedings of 2003 international conference ,15-17 December 2003, pp 296-298.

Actel Corporation, 2007, 51700092-011-0, Actel Fusion Handbook, www.actel.com, last accessed on 10 July 2011.

Actel Corporation, 2010, 5-02-9124-25, Libero IDE v9.0 User's Guide, www.actel.com, last accessed on 10 July 2011.

Actel Corporation, 2009, 50200154-2, ARM® Cortex™-M1 Embedded Processor Hardware Development Tutorial for Fusion Mixed-Signal FPGAs, www.actel.com, last accessed on 10 July 2011.

Actel Corporation, 2009, 50200162-1, ARM® Cortex™-M1 Embedded Processor Software Development Tutorial for Fusion Mixed-Signal FPGAs, www.actel.com, last accessed on 10 July 2011.

Landry, M. W., 1999, Digital frequency synthesizer, United States patent number 5, 931,891.

APPENDICES

Appendix A: SystemC header file for the phase accumulator module

```
/*
 * phase_accumulator.h
 *
 *  Created on: Nov 4, 2012
 *      Author: maina
 */

#ifndef PHASE_ACCUMULATOR_H_
#define PHASE_ACCUMULATOR_H_

#include "systemc.h"

SC_MODULE (phase_accumulator) {

	sc_in<sc_uint<30> > phase_increment;

	sc_in_clk clock;

	sc_out<sc_uint<30> > phase;

	sc_uint<30> phase_var, phase_increment_var;

	void prc_phase_accumulator();

	SC_CTOR (phase_accumulator) {

		SC_METHOD (prc_phase_accumulator);

		sensitive << clock.pos();

		dont_initialize();
	}
};


#endif /* PHASE_ACCUMULATOR_H_ */
```

Appendix B: SystemC implementation file for the phase accumulator module

```
/*
 * phase_accumulator.cpp
 *
 *  Created on: Nov 4, 2012
 *      Author: maina
 */
#include "phase_accumulator.h"

#include <iostream>

void phase_accumulator:: prc_phase_accumulator() {


        //sc_uint<30> phase_var,phase_increment_var;

        phase_increment_var = phase_increment.read();

        phase_var=(phase_increment_var + phase_var);

        phase.write(phase_var);


}
```

Appendix C: DAC5652A digital to analog converter

The following are the features and circuit diagram of the DAC used in this project (Texas Instruments, 2009):

- 275 MSPS Update Rate
- Single Supply: 3.0 V to 3.6 V
- HighSpurious-FreeDynamicRange (SFDR):
  80 dBc at 5 MHz
- Signal to noise ratio: 63 dB
- High Third-Order Two-Tone Intermodulation)
  (IMD3): 78 dBc at 15.1 MHz and 16.1 MHz
- Independent or Single Resistor Gain Control
- Dual or Interleaved Data
- On-Chip 1.2-V Reference
- Low Power: 290 mW
- Power-Down Mode: 9 mW
- Package: 48-Pin Thin-Quad Flat Pack (TQFP)

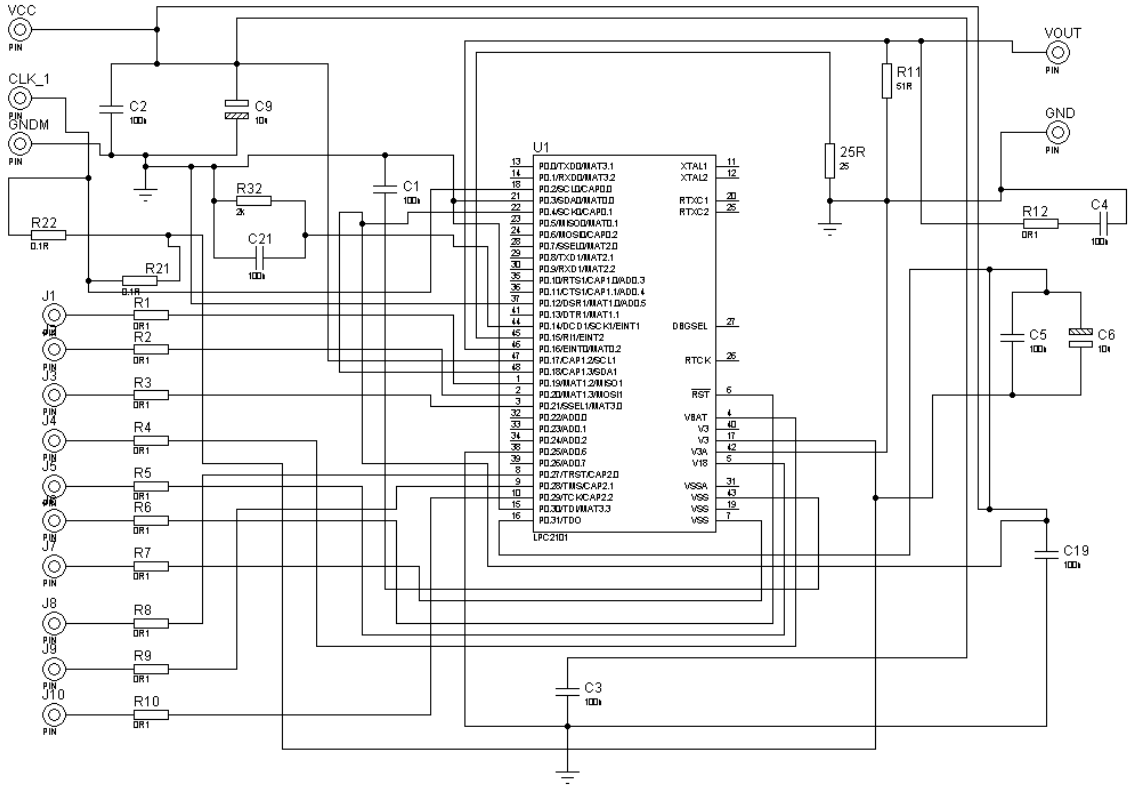


Figure B-1: circuit diagram of the DAC

Appendix D: fusion embedded development kit

Fusion Embedded Development Kit Contents

- Fusion Embedded Development Kit board (Figure C-1) with M1-Enabled Fusion FPGA(M1AFS1500)
- Low-Cost Programming Stick (LCPS) for programming the M1AFS1500 FPGA
  External 5 V Power Supply
- USB 2.0 high-speed cables
- Packet of jumpers
- Quick Start Guide
- Actel Libero® Integrated Design Environment (IDE) software DVD

Figure C-1: Fusion Embedded Development Kit Evaluation Board with LCPS Attached

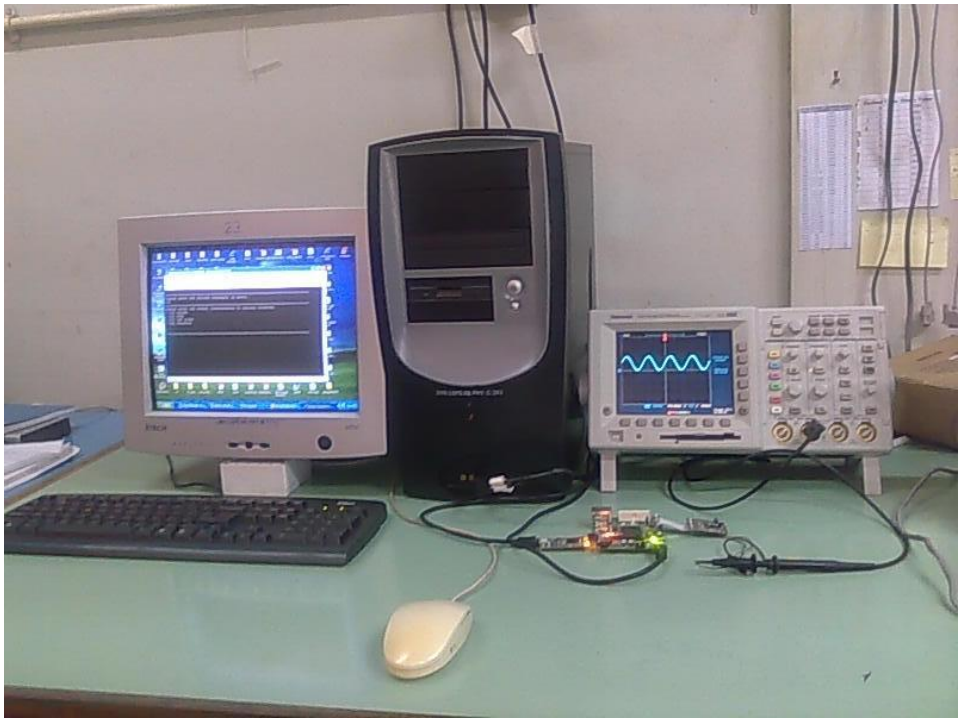


Figure C-2: A picture of the Hardware Test setup

Appendix E: Specifications of the Desktop computer Used in the FFT analysis

Computer model	HP Compaq
Operating system	Debian Linux
CPU	Pentium Dual-Core E5200
Speed	2.50 GHz
RAM	1.96 GB