M. Tech VLSI & Embedded systems

Sri Harsha Chintamaneni

2022

IMPLEMENTATION OF DIGITAL MODULATIONS ASK, PSK AND FSK IN VERILOG USING VEDIC MULTIPLIERS

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

VLSI DESIGN & EMBEDDED SYSTEMS

Submitted by

SRI HARSHA CHINTAMANENI

2K20/VLS/20

Under the supervision of

Dr. N. S. RAGHAVA

ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

May, 2022

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Sri Harsha Chintamaneni, Roll No. 2K20/VLS/20 student of M.Tech VLSI Design & Embedded Systems, hereby declare that the project Dissertation titled "**Implementation of Digital Modulations ASK, PSK and FSK in Verilog using Vedic Multipliers**" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Ch. Sriharsha

Place: Delhi **SRI HARSHA CHINTAMANENI**

Date:

ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "**Implementation of Digital Modulations ASK, PSK and FSK in Verilog using Vedic Multipliers**" which is submitted by Sri Harsha Chintamaneni, Roll No 2K20/VLS/20 Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date:

Dr. N. S. RAGHAVA

SUPERVISOR

PROFESSOR

ACKNOWLEDGEMENT

A fruitful development of a project work cannot be arranged solely by the efforts of the person to whom the project is allocated, it also necessitates the assistance and supervision of those who assisted in the conclusion of the project. I'd like to thank everyone who assisted me with this study and encouraged me during my studies.

It gives me a great sense of pleasure to present the Project Dissertation of the MTech. I owe special debt of gratitude to Dr. N. S. Raghava, Department of Electronics and Communication Engineering, Delhi Technological University (Formerly Delhi College of Engineering) for his full support and assistance during the development of the work. His sincerity, thoroughness and perseverance have been a constant source of inspiration. It is only his cognizant efforts that our endeavours have seen light of the day.

SRI HARSHA CHINTAMANENI (2K20-VLS-20)

Date:

ABSTRACT

India, the sacred land of deep cultural heritage, has long demonstrated the efficacy of mathematics by introducing a faster and more efficient method of obtaining mathematical results through the innovative ideas of ancient vedic mathematics. It enables us to answer nearly all mathematical problems in much less time. Every fraction of a second counts in today's competitive environment to stay ahead. Modulation technique used in various communications over the radio carrier, is crucial to any of the wireless communication systems. The significant portion of wireless transmissions are digital today, and due to the limited bandwidth available, the encoding type is more essential than ever. The basic purpose of modulation today is to incorporate as much information as possible into the smallest bandwidth. The spectral efficiency goal examines how rapidly data may be transferred within the particular available bandwidth. To achieve and enhance spectral bandwidth efficiency, a variety of strategies have evolved. This dissertation intends to serve as a bridge between modern day digital modulations and the ancient Indian vedic mathematics. An extensive work has been done to implement a 32-bit vedic multiplier using the most efficient sutra (algorithm) called Urdhva Triyakbhyam. A Verilog code for generating sinusoidal signals had been written using a technique called Direct Digital Synthesis and by using these the present-day digital modulations namely Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying (PSK) are built. This work is carried out in Xilinx vivado simulation software. An idea to implement modulation schemes using vedic multiplier and sinusoidal waveforms in Xilinx software with Verilog coding has been implemented. The implemented BASK proved impressive in the amount of total on chip power consumption. The power results obtained from Xilinx power report are furnished herewith.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

ACRONYM	ABBREVIATION
AM	Amplitude Modulation
ASK	Amplitude Shift Keying
BASK	Binary Amplitude Shift Keying
BCD	Binary Coded Decimal
BFSK	Binary Frequency Shift Keying
BPSK	Binary Phase Shift Keying
BRAM	Block RAM
CMOS	Complementary Metal Oxide Semiconductor
DAC	Digital to Analog Converter
DDS	Direct Digital Synthesis
DPSK	Differential Phase Shift Keying
ESL	Electronic System Level
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
IC	Integrated Circuit
IDE	Integrated Design Environment
IP	Intellectual Property
ISE	Integrated Synthesis Environment
LFM	Linear Freq Modulation
LSB	Least Significant Bit
LUT	Look Up Table
MATLAB	MATrix LABoratory
MSB	Most Significant Bit
RCA	Ripple Carry Adder
ROM	Read Only Memory
VHDL	Verilog Hardware Description Language

CHAPTER 1

INTRODUCTION

1.1 Overview

The main idea behind this project is to show how to create digital modulations using the Verilog hardware description language. It has been difficult to incorporate sinusoidal signals as analog waves in Xilinx Vivado software. Despite this limitation the developed designs are successfully simulated, and power reports of all three primary modulation techniques are evaluated. This work contains the analysis of the results as well as a comprehensive examination of vedic multipliers, which have been used as the framework for building the digital modulation techniques.

1.2 Literature Review

Due to the rising demand for faster and more efficient electronic devices, Vedic multiplier circuit design with high performance and low power consumption has become a prominent research area for researchers throughout the world in recent years. This section serves as the foundation for this dissertation work. For acquiring a thorough understanding of publications on related topics and tools that have been used. These are papers published in prominent conferences or journals by authors with comparable interests. This section contains a literature review of these publications that deal with vedic multipliers and digital modulations. The review of literature is established as follows:

P. N. Murthy et al: [1] The BASK, BFSK, BPSK, and QAM modulators were discussed, with the sinusoidal output being observed through the Xilinx Vivado system generator and the Direct Digital Synthesis (DDS) approach. Using a multiplexer and a demultiplexer, the modulations were created. The four waveforms stated earlier are generated by the input 1:4 demux, which are then passed to the 4:1 mux to construct the output data stream. Xilinx simulations have been used to observe the output waveforms.

N. Pallavi et al: [2] proposed the design part as well as the implementation of Linear Frequency Modulation (LFM utilising DDS The MATLAB tool and also the Xilinx system are both used in the design. FPGA board is used to implement the generated

code. LFM having frequencies of 60MHz and 200MHz are created and studied using Xilinx simulations.

B. R. Jammu et al: [3] This paper demonstrates FPGA-based application of digital modulation techniques BASK, BFSK, BPSK and DPSK that can be used in digital communication education courses. The user has control over both the input carrier signal as well as the modulating waveform. The modulators here were designed in System Generator of Xilinx and then integrated into the FPGA Spartan board. The gains as indicated by this study include the use of lesser digital blocks, the power to control input frequencies, and compatibility with Xilinx FPGA boards. Xilinx ISE 14.7 was employed to gather experimental results such as power, temperature, and utilisation reports.

A. Sharma et al: [4] This paper provides a general overview of architecture using Verilog HDL-based simulations for foundational and broadly used modulation schemes such as BASK, BFSK, BPSK, and QPSK. Although the concept of sinusoidal signal generation is not new, this paper strived to optimise it by employing sampling and quantization concepts in the time and amplitude domains, respectively. Model Sim and Xilinx ISE are used to run the simulations

Du Weitao et al: [5] An efficient concept for DDS IP core generator has been demonstrated in this paper. A wireless connectivity ROM compacted DDS circuit is created using this Core generator. For DDS architecture, this employs linear slope ratio interpolation. This generator's effectiveness is dependent on ROM compression, which effectively reduces waveform ROM size. An FPGA board is used to verify the results.

Yogita Bansal et al: [6] Multipliers are used in a plethora of circuit designs, spanning high-speed arithmetic, digital signal processing, and Multiplier and Accumulator units, to highlight a few. The need for more efficient and quick multipliers is becoming more evident. Although there are numerous ways of improving the performance of a multiplier, including using the booth algorithm or the Wallace tree method, multipliers from vedic mathematics have demonstrated to be the quickest and require the least amount of electricity. This study uses a vedic sutra called urdhva triyakbhyam to demonstrate the use of vedic multipliers in performing the basic multiplication operation. The methods, benefits, and drawbacks of vedic multipliers are also examined in this work.

Yeshwant Deodhe et al: [7] They presented regarding the actual planning and execution of an eight-bit CMOS vedic multiplier. This paper explored a novel method for leveraging vedic sutras (algorithms) to dramatically reduce computation steps when compared to the traditional multiplication procedures. The drawings are created in Tanner and evaluated in T-Spice using 180nanometers CMOS technology. A concept for a high linearity CMOS low power multiplier is presented. According to the findings, the proposed model saves power consumption by up to 75% while occupying 750 um2 of space.

A. K. Mehta et al: [8] The Vedic BCD multiplier and the modified binary to BCD converter were analysed and examined. Decimal arithmetic is useful when more precise data processing is demanded. This work uses multiplexers and BCD adders as the foundation for building a BCD 8421vedic multiplier as well as a modified binary to BCD converter. The VHDL implementation findings demonstrate that the vedic BCD multiplier produces 2x faster results and takes up 4x less space than the conventional BCD multiplier.

F. Quadri et al: [9] Discusses the digital modulation techniques used in FPGAs. As a summary, a survey of several digital modulation techniques using various methodologies and tools is delivered. This research focuses on the implementation of three popular modulation techniques ASK, FSK, and PSK. The verilog language was coded using Xilinx software, and the simulated results were authenticated using the model sim software. The modulators are also created in the MATLAB/SIMULINK environment to analyse the many parameters among the various established designs, that influence the choosing of a certain modulation approach.

1.3 Tools Adopted – Xilinx Vivado

Vivado is an integrated design environment (IDE) that launched in the year 2012 with frameworks relating to system and IC level based on a shared expandable dataset and a routine debugging process. It also has electronic system-level (ESL) design implements for verification and synthesis of some of the C-predicated algorithm based Ips Standard packaging of RTL IP have been reused at system integration level for all building blocks in the system and their respective verification methods [10]. The Vivado Design Suite aids designers in enhancing their productivity by optimizing their procedures. It is built on the latest Xilinx devices and features an improved user interface, which can help

minimize design complexity. The Vivado Design Suite simplifies the implementation of complex design projects by providing tools that can analyse and optimize multiple design metrics. The new Vivado Design Suite has a comprehensive set of tools that replace the previously existing ISE Design Suite of tools. It features a shared, scalable data model that enables whole of the design process to run in the memory space without the need of translating any intermediate file formats.

1.4 Structure of Work

The flow in which this thesis is organised has been described in this section as follows:

Chapter 1: This chapter describes the broad perspective of the thesis work, including the software tools used for verilog coding, as well as a brief introduction to the work that follows. It also covers the literary works of several authors who have comparable interests in this field of work. Much of their work has been briefly described in order to obtain a general understanding of the overall gist of their work. The references have been added to the references section of this thesis work.

Chapter 2: Describes the basic foundation for the thesis work, which is based on vedic mathematics. A 32-bit multiplier based on vedic mathematics is constructed. All multipliers are built on the base of a 2-bit fundamental vedic multiplier cell. To arrive at the 32-bit multiplier, each multiplier uses the previously built multiplier. The implemented circuits are displayed as a block diagram with their respective Xilinx vivado simulations to validate the multiplication output results, as well as the verilog code explanation.

Chapter 3: Sinusoidal signals, such as sine and cosine, were generated using DDS. This chapter discusses the generation of these sinusoidal signals, as well as the code used to generate them. The Xilinx vivado simulation results were also included.

Chapter 4: This chapter will provide an understanding of the three main digital modulated signals. Every one of these modulation patterns has been thoroughly examined, as well as the usage of vedic mathematics to produce the final modulated output. The RTL implementation schematics and simulations from Xilinx are given.

Chapter 5: This thesis closes with a discussion of the Xilinx vivado results.

CHAPTER 2

VEDIC MULTIPLIERS

2.1 Chapter Outline

In Sanskrit the word 'Veda' translates to 'knowledge'. Vedic math is quite comparable to human mathematical calculations. It reduces the number of steps needed to calculate multiplication while also improving accuracy. Vedic multipliers utilise less power and memory than traditional multipliers. A multiplier is an important component in practically every processor and adds significantly to the system's total area and power usage. [7][6]. Sri Bharati Krishna Tirthaji (1884-1960), between the years of 2000 and 2010, a Sanskrit, math, history, and philosophy scholar uncovered Vedic math concepts from ancient Indian literature. It is founded on 16 Vedic sutras having 14 sub sutras (algorithms), which are applicable to various streams of math for easier, highly optimized and quick to use calculations. Three sutras and two sub-sutras are provided for multiplication out of the total of 16, they are Urdhva-tiryakbhyam (vertical & cross-wise), Nikhilam Navatashcaramam Dashatah (everything from 9 but last digit from 10), Anurupyena (Proportionality), Ekanyuena Purvena (By one lower than the one before it) and Antyayordasake'pi (numbers where the last digit adds up to 10). Because at least one operand must be in the near power of 10, Nikhilam is not considered as a universal approach for decimal numbers. Anurupyena, which provides the solution to this issue, is not a good alternative because it involves multiplying or dividing the divisor by an appropriate proportionality constant to bring it closer to a decimal base. The proportionality constant must not be a power of 10, as dividing it by one multiplier makes the procedure unappealing in comparison to previous approaches. Antyayordasakepi and EkanyunenaPurvena are specific to astronomy, but Urdhva-Tiryagbhyam is a universally accepted method for all multiplications [8].

2.2 2-bit vedic multiplier

The 2-bit vedic multiplier is the fundamental multiplier cell that serves as the foundation for all of the other vedic multipliers mentioned in this dissertation. This utilises the Urdhva Triyakbhyam Sutra (Algorithm) of Vedic mathematics, which translates to vertical and crosswise which can be seen in Fig. 2.1.

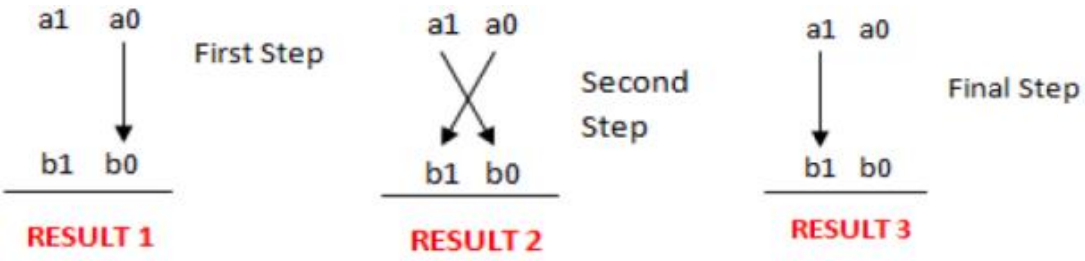


Fig. 2.1 Urdhva Triyakbhyam Algorithm

Multiplication has a multiplier and a multiplicand as usual, but the process is done by multiplying the first two bits of multiplicand and multiplier, and next cross multiplying of side numbers of multiplicand with numbers in multiplier, finally the last digits are multiplied parallelly to complete the multiplication process

2.2.1 Design of 2-bit vedic multiplier

The block level representation of the theory discussed above is utilised to implement a 2-bit multiplier in Xilinx vivado using the Hardware Description Language Verilog. The same is depicted in Figure. 2.2, where two half adders have been used.

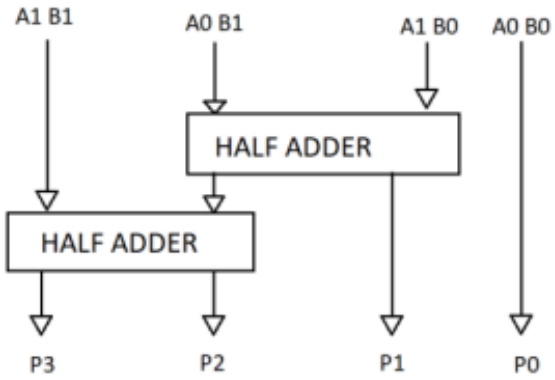


Fig. 2.2 Block Diagram of 2-bit vedic multiplier

The inputs for this multiplier are a 2-bit multiplicands namely A[1:0] and B[1:0]. When A and B when they are multiplied in binary system produces a 4-bit result shown in Fig. 2.2 as P[3:0] where P[3] is the MSB and P[0] is the LSB. The inputs to the first half adder is the A1B0 and A0B1 as shown in Fig. 2.1 as Result 2. The result of this addition gives a carry bit and sum bits represented by P[1]. The carry is taken forward and given as input to the second half adder along with A1B1 shown as final step in Fig. 2.2 The

sum ad carry generated by the second half adder gives the P[3:2] of the result. The result P[0] can be directly obtained by a simple AND gate. Refer the Fig. 2.3

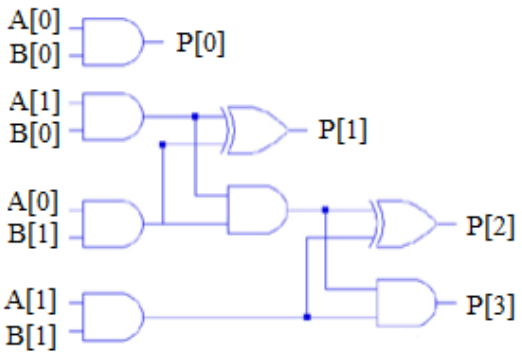


Fig. 2.3 Gate level structure of 2-bit vedic multiplier

2.2.2 RTL implementation of 2-bit vedic multiplier

The code employs four basic 2-bit AND gates and 2 half adder modules. The half adders are in-turn composed of two EXOR gates and two AND gates. as shown in Fig. 2.3. A0, A1, B0, B1 are declared as inputs. Output is declared as a 4-bit value.

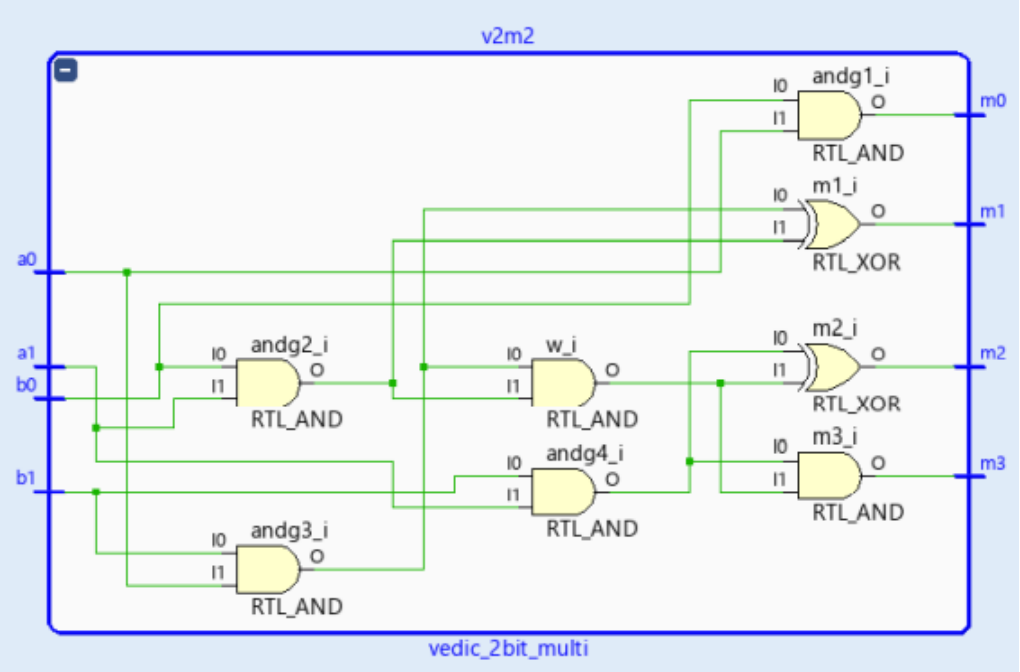


Fig. 2.4 RTL implementation of 2-bit vedic multiplier

The Xilinx synthesis and RTL simulation gives the gate level netlist of the verilog code synthesized. Where the module is defined as vedic_2bit_multi and the inputs and outputs are defined outside the module.

2.2.3 Xilinx simulation results of 2-bit vedic multiplier

The simulations are carried out in Xilinx vivado. The figure below shows the output of a 2-bit vedic multiplier. The multiplication result of a[1:0] and b[1:0] is stored in m[3:0]. A sample multiplication of 0*2, 3*1, 2*3, 1*1 can be seen in Fig. 2.5 The maximum value for a 2-bit binary is 11 which is 3 in decimal. When two such are multiplied, it gives out a result of 1001 in binary which is 9 in decimal. Since for the output 9 requires 4 binary bits, the output m is taken to be a 4-bit binary output.

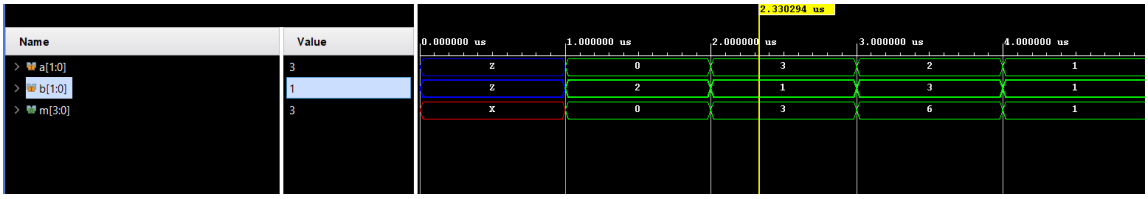


Fig. 2.5 Xilinx simulation output of 2-bit vedic multiplier

2.3 4-bit vedic multiplier

A 4-bit multiplier in vedic mathematics utilises the basic 2-bit vedic multiplier as depicted in Fig. 2.6. The figure below shows the vertical and cross multiplication algorithm of vedic mathematics. A 4-bit multiplier takes two 4-bit binary numbers A[3:0] and B[3:0] to produce a 8-bit result.

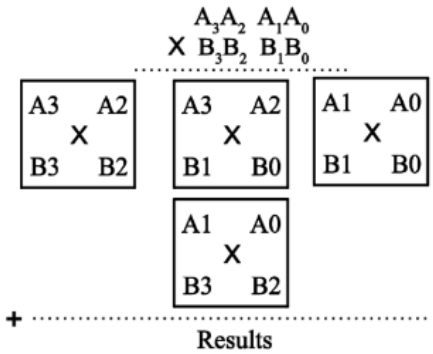


Fig. 2.6 4-bit vedic multiplier depicting Urdhva Triyakbhyam algorithm

The vertical multiplications are depicted as A[3] A[2] B[3] B[2] which takes the first two bits from MSB and A[1] A[0] B[1] B[0] which takes the last two bits from the LSB. The cross multiplication for 4-bit is obtained by MSB and LSB multiplications of the binary inputs.

2.3.1 Design of 4-bit vedic multiplier

The block level design of 4-bit vedic multiplier is as shown in Fig. 2.7 below. To multiply two 4-bit binary numbers they can be divided into two parts A1A0 as one block and A3A2 as another. Similarly, B can also be divided into two blocks. Each subblock will be having its own 2-bit multiplication. This gives an idea that 4-bit multiplication can be implemented with the help of 2-bit multiplier blocks created previously. The block diagram is shown in Fig. 2.7 utilising 4 modules of 2-bit multipliers and 3 modules of 4-bit RCAs and 1 half adder.

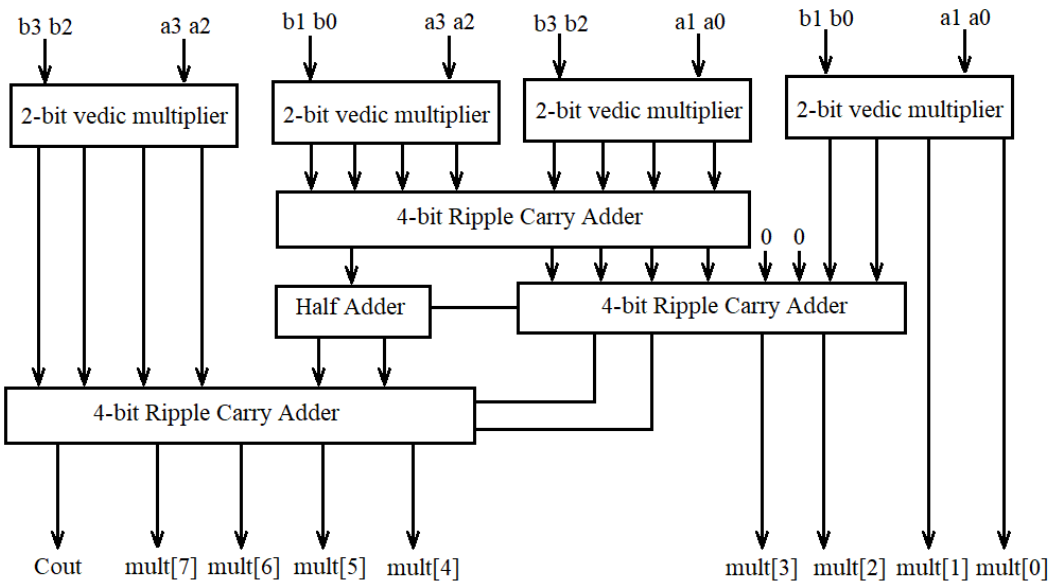


Fig. 2.7 Block diagram of 4-bit vedic multiplier

This work uses the basic ripple carry adder. For an optimised version other faster and efficient adder such as Carry Look Ahead adder can also be used. The design of 4-bit ripple carry adder is as shown below in Fig. 2.8

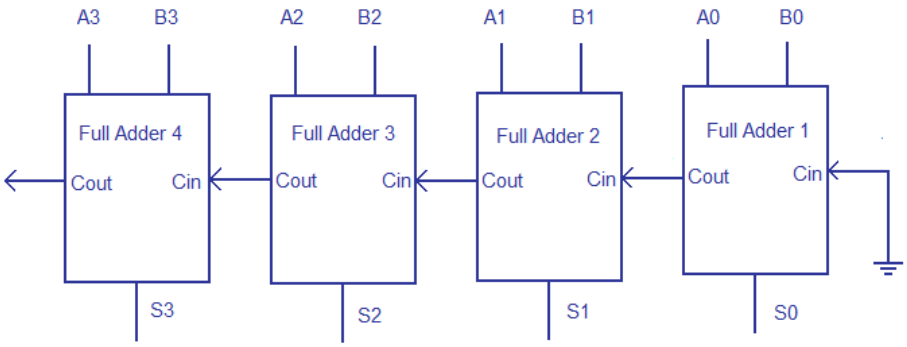


Fig. 2.8 4-bit Ripple Carry Adder

An N-bit RCA can be better viewed as a cascaded structure of N- Full adders where the carry of one Full adder is propagated to another. This gives the notion that the carry is rippled from one adder to another hence, the name ripple carry adder.

The structure of N-bit RCA consists of basic element called Full Adder. The structure of a Full Adder using logical gates is as shown in Fig. 2.9 below

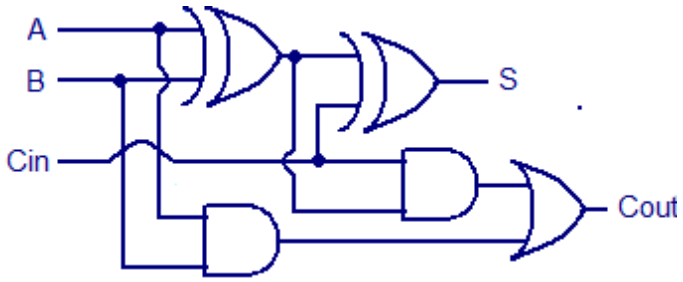


Fig. 2.9 Structure of Full Adder

The sum part of the adder is obtained from output S and Carry part from output C_{out}. Using this structure a RCA of any number of bits can be easily constructed by simply cascading them. The operation of this is as follows: Since, initially there is no carry generated the C_{in} for Full Adder 1 is taken as logic 0 by simply grounding it. The outputs S_0 and C_{out} of Full Adder 1 is obtained after certain delay because of the internal logic gate delays of the Full Adder. The carry from this stage is carried forward to the next adder Full Adder 2. This operation is performed untill the final sum S_3 and C_{out} are obtained.

It is to be noted that only after the final S_3 and C_{out}, the entire result is taken to be of valid. All the immediate results are invalid. Thus a RCA produces valid output after the final adder gives its output. This accounts for a major delay in RCAs.

Advantages of RCA:

Designing is not complex – Designing an RCA requires only full adders.

Area Efficient – This gives a compact implementation by reducing the area. The chip density can be greatly increased.

Disadvantages of RCA:

Delay in obtaining the result – The major drawback in RCA is the time required to produce final result. It is the sum of delays of all the full adders connected in cascade.

2.3.2 RTL implementation of 4-bit vedic multiplier

The code uses modules of half adder, RCA and 2-bit vedic multiplier. The verilog code for this multiplier discussed briefly below.

```
module four_bit_vedic_mult(input [3:0]a, [3:0]b, output [7:0]mult);
```

The module is taken as Vedic4bit_mult with inputs a and b of 4 bits size and output is taken as mult of 8 bits.

```
vedic_2bit_multi v2m0(a[0], a[1], b[2], b[3], m1[3], m1[2], m1[1], m1[0]);
```

A code line depicting the insertion of 2-bit vedic multiplier module

```
RCA4bit rca0(m2[3:0], m1[3:0], 1'b0, rcac1s[3:0], rcac1c);
```

Code for inserting the RCA module inside 4-bit vedic multiplier module where the output of second and third 2-bit multipliers in Fig. 2.7 are taken as input. Another input for RCA is taken as bit 0. This produces the output RCA sum and carry. Sum is a 4-bit output for a 4-bit adder and carry is 1 bit.

```
half_add_dataflow had0(rcac1c, rcac2c, had0s, had0c);
```

The half adder takes the input the carry generated from two RCA adders and produces the output sum and carry.

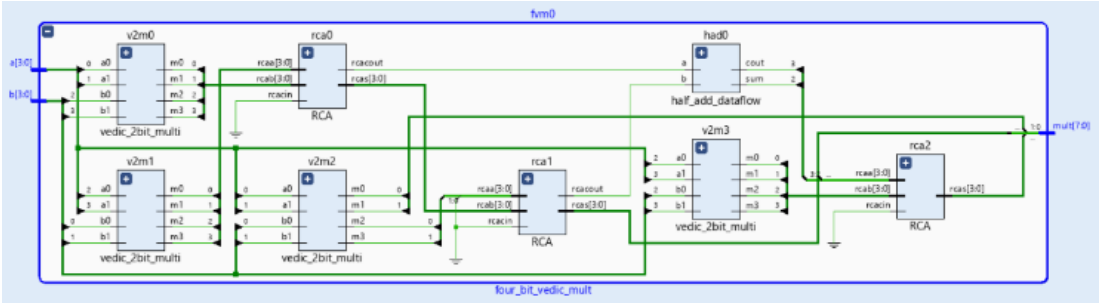


Fig. 2.10 RTL implementation of 4-bit vedic multiplier

The Xilinx synthesis and RTL simulation gives the gate level netlist of the verilog code synthesized. Where the module is defined as four_bit_ved_mult and the inputs and outputs are defined outside the module.

2.3.3 Xilinx simulation results of 4-bit vedic multiplier

The simulations are carried out in Xilinx vivado. The figure below shows the output of a 4-bit vedic multiplier. The multiplication result of a[3:0] and b[3:0] is stored in mult[7:0]. A sample multiplication of 3*12, 10*7, 15*15, 13*0, 9*9, 11*4 can be verified in Fig. 2.11 The maximum value for a 4-bit binary is 1111 which is 15 in decimal. When two such numbers are multiplied, it gives out a result of 1110 0001 in binary which is 225 in decimal. Since for the output 225 requires 8 binary bits, the output mult is taken to be a 8-bit binary output

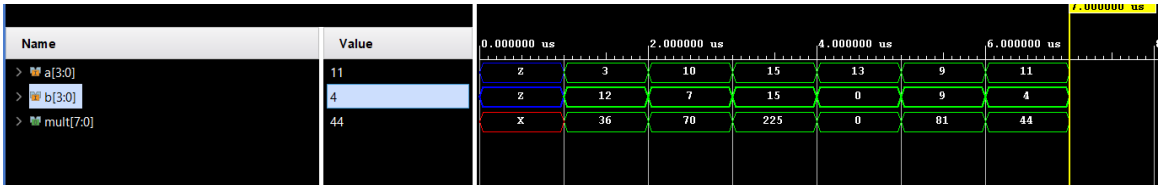


Fig. 2.11 Xilinx simulation output of 4-bit vedic multiplier

2.4 8-bit vedic multiplier

This multiplier has similar design and implementation compared to the 4-bit multiplier. It uses four modules of 4-bit vedic multiplier along with three modules of 8-bit RCA. The 8-bit RCA is developed from the 4-bit RCA adder. The verilog code has been written and the output simulations are verified in Xilinx vivado

2.4.1 Design of 8-bit vedic multiplier

4-bit vedic multiplier acts as the basic building module of 8-bit Vedic multiplier depicted in Fig. 2.12 can be constructed by using four similar four-bit Vedic multipliers and three eight-bit RCAs (consisting 2 inputs of 8 bits each) are required. The 4-bit Vedic multiplier takes the inputs, and the eight-bit output taken from the multiplier. Now, the 1st RCA adder input taken to be the output value of the second and third 4-bit vedic multipliers generating an output totalling to eight bits along with one carry.

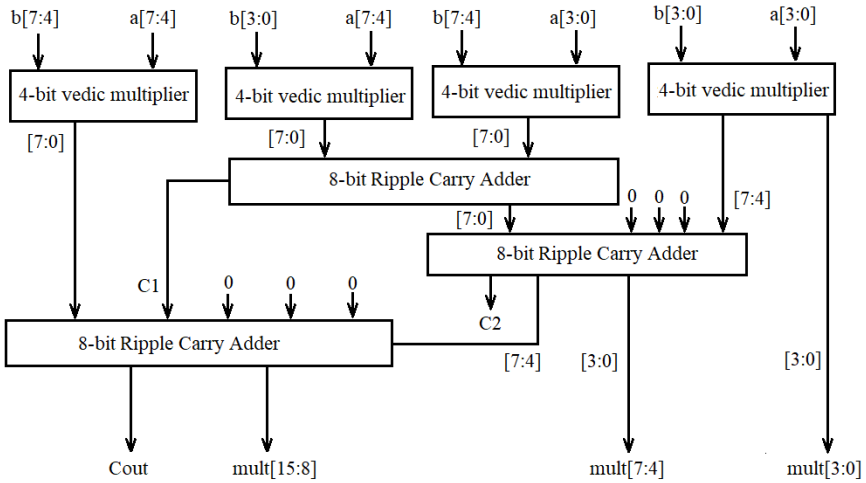


Fig. 2.12 Block diagram of 8-bit vedic multiplier

The Second RCA will add the eight-bit output of the first RCA and four bits taken from the output of first 4-bit multiplier, other 4 bits are taken to be 0. As a result, the output is given to be eight bits along with one carry in this case the carry generated is discarded. The third RCA will add the output taken from the fourth 4-bit multiplier and four output bits of second RCA, remaining four carry bits of the initial RCA are 0s. The initial vedic multiplier output along with mult[7:4], mult[15:8] and carry bit generated from the last RCA together make the output of the eight bit multiplier shown in Fig. 2.12

2.4.2 Implementation of 8-bit vedic multiplier

The code uses modules of RCA and 4-bit vedic multiplier. The verilog code for this multiplier discussed briefly below.

```
module eightbit_ved_mult( input [7:0]a,
                          input [7:0]b,
```

output [15:0]mult);

The module is taken as eightbit_ved_mult with inputs a and b of 8 bits size and output is taken as mult of 16 bits.

four_bit_ved_multi fvm0(a[3:0], b[3:0], o1[7:0]);

A code line depicting the insertion of 4-bit vedic multiplier module

eight_rca erca1(o2[7:0], o3[7:0], 1'b0, rcas1[7:0], cout1);

Code for inserting the RCA module inside 8-bit vedic multiplier module where the output of second and third 4-bit multipliers in Fig. 2.12 are taken as input. Another input for RCA is taken as bit 0. This produces the output RCA sum and carry. Sum is a 8-bit output for a 8-bit adder and carry is 1 bit.

The Xilinx synthesis and RTL simulation gives the gate level netlist of the verilog code synthesized. Where the module is defined as eightbit_ved_mult and the inputs a[7:0], b[7:0] and outputs mult[15:0] are defined outside the module. All the modules inside 8-bit vedic multiplier are clearly depicted in Fig. 2.13

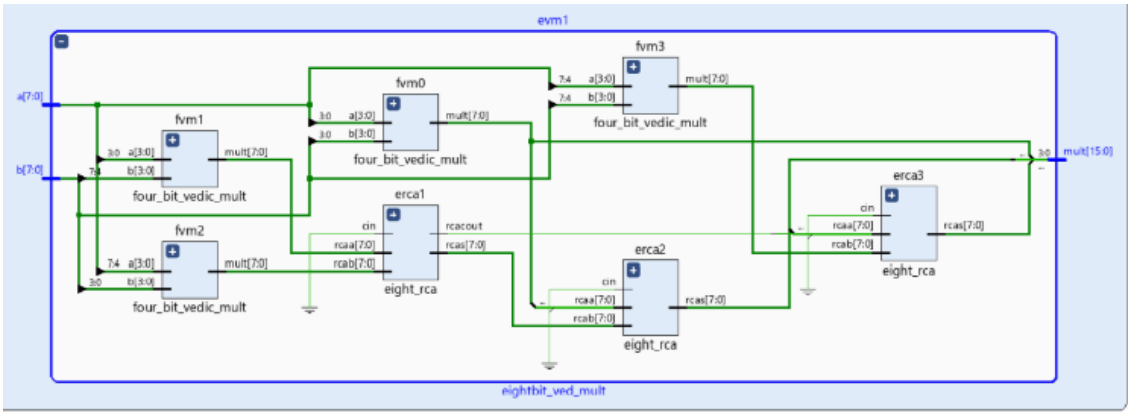


Fig. 2.13 RTL implementation of 8-bit vedic multiplier

2.4.3 Xilinx simulation results of 8-bit vedic multiplier

The simulations are carried out in Xilinx vivado. The figure below shows the output of a 8-bit vedic multiplier. The multiplication result of a[7:0] and b[7:0] is stored in mult[15:0]. A sample multiplication of 64*2, 255*255, 12*25, 23*0, 19*200, 111*111 can be verified in Fig. 2.14

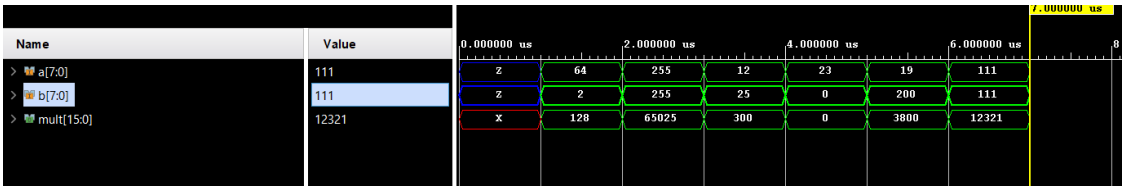


Fig. 2.14 Xilinx simulation output of 8-bit vedic multiplier

The maximum value for a 8-bit binary is 1111 1111 which is 255 in decimal. When two such numbers are multiplied, it gives out a result of 1111 1110 0000 0001 in binary which is 65,025 in decimal. Since for the output 65,025 requires 16 binary bits, the output mult is taken to be a 16-bit binary output

2.5 16-bit Vedic Multiplier

This multiplier has similar design and implementation compared to the 8-bit multiplier. It uses four modules of 8-bit vedic multiplier along with three modules of 16-bit RCA. The 16-bit RCA is developed in similar manner as the 8-bit RCA adder. The verilog code has been written and the output simulations are verified in Xilinx vivado

2.5.1 Design of 16-bit vedic multiplier

The 16-bit multiplier block has a similar structure as of 8×8 blocks as in Fig. 2.15. 8-bit multiplier block acts as the starting block for constructing 16-bit Vedic Multiplier. The same 16-bit module shown in Fig. 2.15 can be constructed by utilising four 8-bit Vedic multipliers and three 16-bit RC Adders (having 2 inputs of 16 bits) are required. The 8-bit Vedic multiplier takes the inputs, and the output of the multiplier is given as 16 bits. Now, the input of the 1st RCA adder is the output of the 2nd and 3rd 8-bit multipliers which gives output of 16 bits and one carry.

The RCA-2 will add the output of RCA-1 and 8 bits from the generated output of initial Vedic multiplier, all the other bits are taken as 0. As a result, 16-bit output and the corresponding carry are obtained and this carry can be discarded. The RCA-3 will add the output of the last 8-bit multiplier along with the output of RCA-2, the other bits have been as depicted in the Fig. 2.15. The combined results from all the RCAs along with the carry generated from RCA-3 will be taken to be the output of eight-bit vedic multiplier.

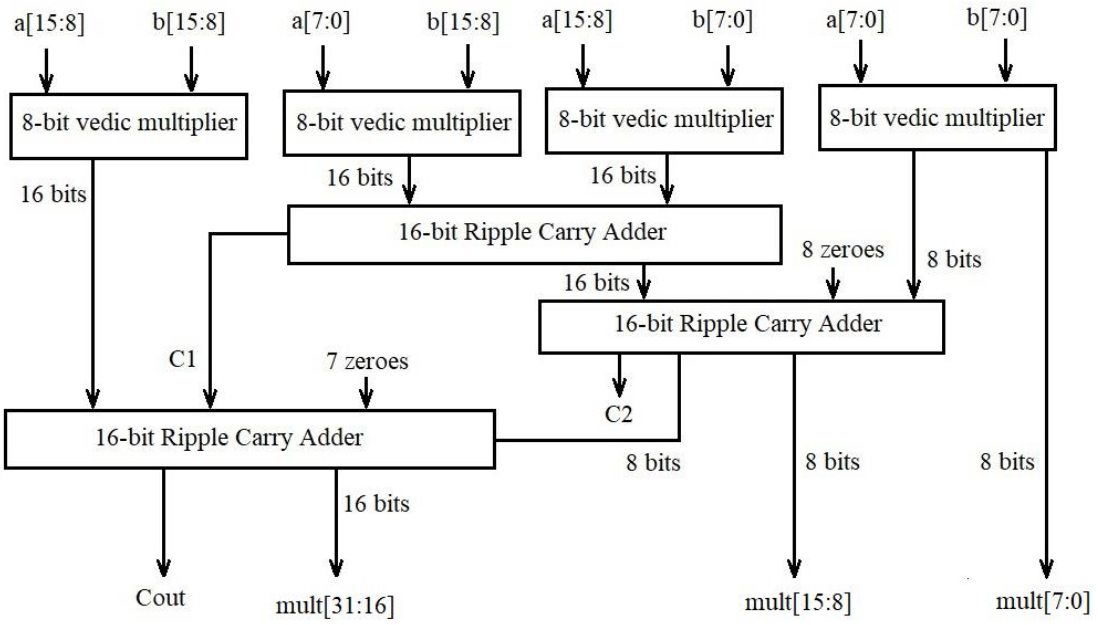


Fig. 2.15 Block diagram of 16-bit vedic multiplier

2.5.2 Implementation of 16-bit vedic multiplier

The code uses modules of 16-bit RCA and 8-bit vedic multiplier. The verilog code for this multiplier discussed briefly below.

```
module sixteenbit_vedic_mult(
                    input [15:0]a,
                    input [15:0]b,
                    output [31:0]mult
                    );
```

The module is taken as sixteenbit_vedic_mult with inputs a and b of 16 bits size and output is taken as mult of 32 bits.

```
eightbit_ved_mult evm1(a[7:0], b[7:0], o1[15:0]);
```

A code line depicting the insertion of 8-bit vedic multiplier module

```
sixteen_rca srca1(o2[15:0],o3[15:0],1'b0,rcas1[15:0],rcacout1);
```

Code for inserting the RCA module inside 16-bit vedic multiplier module where the output of second and third 8-bit multipliers in Fig. 2.15 are taken as input. Another input

for RCA is taken as bit 0. This produces the output RCA sum and carry. Sum is 16-bit output for a 16-bit adder and carry is 1 bit.

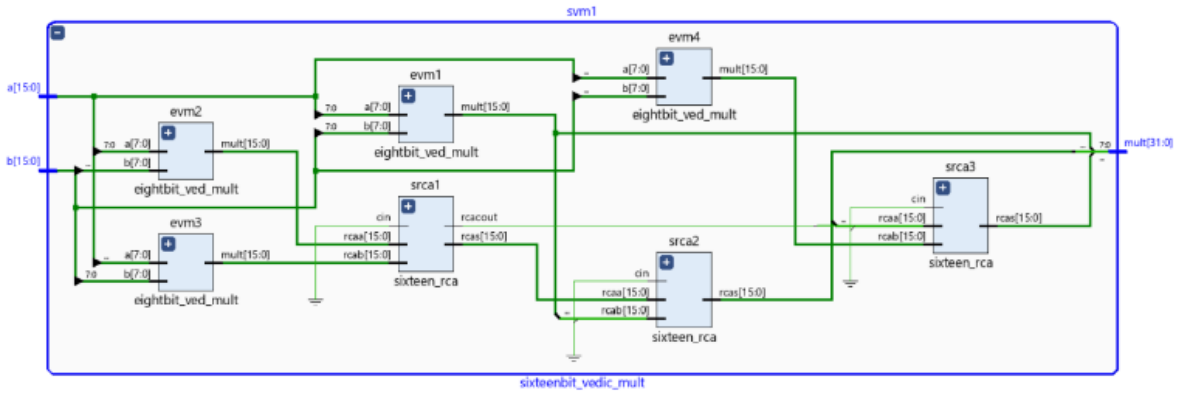


Fig. 2.16 RTL implementation of 16-bit vedic multiplier

The Xilinx synthesis and RTL simulation gives the gate level netlist of the verilog code synthesized. Where the module is defined as sixteenbit_vedic_mult and the inputs a[15:0], b[15:0] and outputs mult[31:0] are defined outside the module. All the modules inside 16-bit vedic multiplier are clearly depicted in Fig. 2.16

2.5.3 Xilinx simulation results of 16-bit vedic multiplier

The simulations are carried out in Xilinx vivado. The figure below shows the output of a 16-bit vedic multiplier. The multiplication result of a[15:0] and b[15:0] is stored in mult[31:0]. A sample multiplication of 65535*65535, 4660*4369, 135*0, 38200*28422, 7341*9, 39321*39321 can be verified in Fig. 2.17

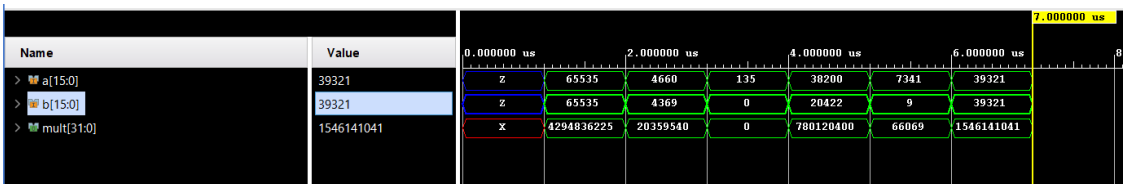


Fig. 2.17 Xilinx simulation output of 16-bit vedic multiplier

The maximum value for a 16-bit binary is 1111 1111 1111 1111 which is 65535 in decimal. When two such numbers are multiplied, it gives out a result of 1111 1111 1111 1110 0000 0000 0000 0001 in binary which is 4294836225 in decimal. Since for the

output 4294836225 requires 32 binary bits, the output mult is taken to be a 32-bit binary output

2.6 32-bit vedic multiplier

This multiplier has similar design and implementation compared to the 16-bit multiplier. It uses four modules of 16-bit vedic multiplier along with three modules of 32-bit RCA. The 32-bit RCA is developed in similar manner as the 16-bit RCA adder. The verilog code has been written and the output simulations are verified in Xilinx vivado

2.6.1 Design of 32-bit vedic multiplier

The design of 32×32 block follows the same procedure as discussed for the previous vedic multipliers in a simplified diagram as in Fig. 2.18. 16-bit Multiplier Module is taken to act as the basic constructing module of 32-bit Vedic multiplier shown in Fig. 2.18 can be constructed by using four blocks of 16-bit Vedic multipliers and three similar blocks of RCA which are essentially having 2 inputs of 32 bits each are required. The 16-bit Vedic multiplier takes the inputs, and the output of the multiplier is 32 bits. Now, the RCA-1 input is taken as the output of the middle multipliers which each of these gives output of 32 bits and along with one generated carry.

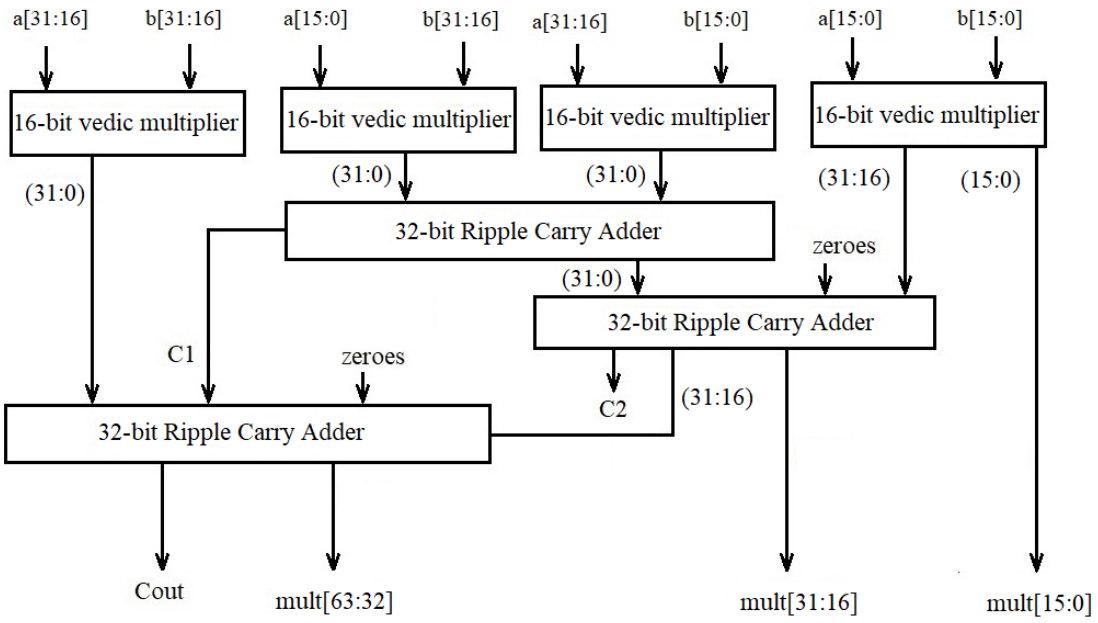


Fig. 2.18 Block diagram of 32-bit vedic multiplier

The RCA-2 adds the RCA-1and the initial block of 16-bit multiplier along with sixteen zeroes. This results an output of 32 bits and also one carry which is being discarded. The RCA-3 will add the output of the last module of 16-bit multiplier and RCA-2, other 16 are carry bits of 1st RC Adder and fifteen 0s. Now, the output of 32-bit multiplier is mult[15:0] output of 1st 16-bit multiplier, mult[31:16] is output of 2nd RC Adder and mult[63:32] is output of 3rd RCA adder. These outputs along with the carry generated from third RCA will add up to the 64-bit output for 32-bit vedic multiplier

2.6.2 RTL Implementation of 32-bit vedic multiplier

The code uses modules of 32-bit RCA and 16-bit vedic multiplier. The verilog code for this multiplier discussed briefly below.

```
module thirtytwobit_vedic_mult(
                                    input [31:0]a,
                                    input [31:0]b,
                                    output [63:0]mult
                                   );
```

The module is taken as thirtytwobit_vedic_mult with inputs a and b of 32 bits size and output is taken as mult of 64 bits.

```
sixteenbit_vedic_mult svm1(a[15:0],b[15:0],o1[31:0]);
```

A code line depicting the insertion of 16-bit vedic multiplier module

```
thirtytworca ttrca1(o2,o3,1'b0,rcares1,cout1);
```

Code for inserting the RCA module inside 32-bit vedic multiplier module where the output of second and third 16-bit multipliers in Fig. 2.18 are taken as input. Another input for RCA is taken as bit 0. This produces the output RCA sum and carry. Sum is 32-bit output for a 32-bit adder and carry is 1 bit.

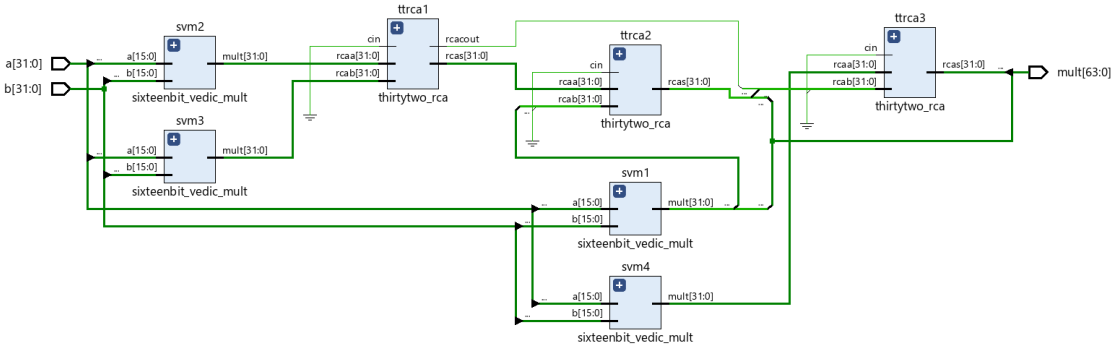


Fig. 2.19 RTL implementation of 32-bit vedic multiplier

2.6.3 Xilinx simulation results of 32-bit vedic multiplier

The simulations are carried out in Xilinx vivado. The figure below shows the output of a 32-bit vedic multiplier. The multiplication result of a[31:0] and b[31:0] is stored in mult[63:0]. A sample multiplication of FFFFFFFF*FFFFFFFF, 0abcdefa*efabcdef, 0000feda*feda0000, f456d231*76549721, 45982841*0 can be verified in Fig. 2.20

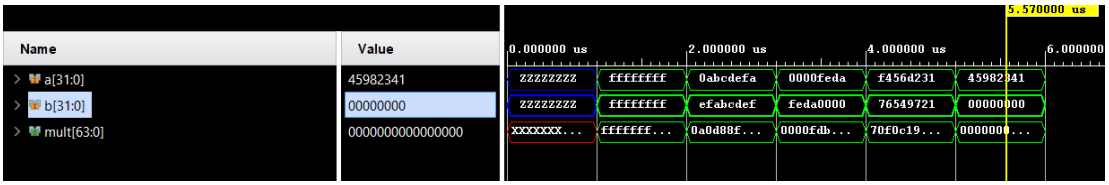


Fig. 2.20 Xilinx simulation output of 32-bit vedic multiplier

The maximum value for a 32-bit binary is FFFF FFFF in hexadecimal system. When two such numbers are multiplied, it gives out a result of FFFF FFFE 0000 0001 in hexadecimal system. Since for the output FFFF FFFE 0000 0001 requires 64 binary bits, the output mult is taken to be a 64-bit binary output

CHAPTER 3

SINUSOID GENERATION

3.1 Chapter Outline

This chapter describes how to generate the sinusoidal signals sine and cosine using Xilinx vivado's Direct Digital Synthesis (DDS) IP core generator. Direct Digital Synthesis (DDS) is a waveform generator that uses a frequency synthesizer. This approach generates sine, square, triangle, and sawtooth waves, all of which can be tweaked to a great degree of amplitude and frequency precision. It works by digitally recording the points of a waveform. The waveform is then reconstructed by recalling these digital data. The hardware description language code used to generate them will be briefly described as well. The fact that sine and cosine signals are 90º out of phase with each other makes it easier to generate them.

Xilinx vivado simulations were used to generate the appropriate results. The output signals have been validated. These generated sinusoidal signals serve as the foundation for the digital modulations addressed in Chapter 4.

3.2 Direct Digital Synthesis (DDS)

Direct digital synthesis (DDS) converts a analog continuous time waveform digitally. DDS devices are small and power-efficient. They can provide rapid switching between output frequencies, excellent frequency resolution, and operating across a wide frequency range. Sinusoidal waveforms are constructed by the core and can be employed in a wide range of applications.

Many sectors have made it a need to be able to reliably produce and regulate waveforms of varied frequencies and characteristics. Important design factors are convenience, compactness, and low cost. In both communications and industrial applications, the DDS technique is quickly gaining favor for solving frequency (or waveform) generating requirements.

A phase accumulator, a mechanism of phase and amplitude conversion of a typical sinusoidal, and DAC are the three primary components of a DDS device's internal circuitry. At a certain frequency, a DDS generates a sine wave. The clock frequency

and the binary value together are entering a special register for frequency tuning. The phase accumulator receives its main input from the frequency register, which is a binary number. If a sine LUT is employed, phase accumulator calculates the phase address, which then sends the DAC the digital amplitude relative to the sine of phase angle. The DAC then translates that number to an analogue voltage or current measurement. For every cycle of the clock, a constant phase increment value which is defined by the binary value adds to the accumulator to form a sinusoidal waveform having single frequency. If the increment value is made substantial, the accumulator will rapidly step resulting in a sinusoidal waveform having higher. When the angle increment is minimal, the number of steps needed by the accumulator increases significantly which results in a sinusoidal of very low frequency. A Phase Generator and a SIN/COS Lookup Table (phase to sinusoid conversion) make up a DDS. A lookup table strategy serves as a standard approach for digital generation of complex sinusoidal by storing their sample value [2].

Theory of Operation

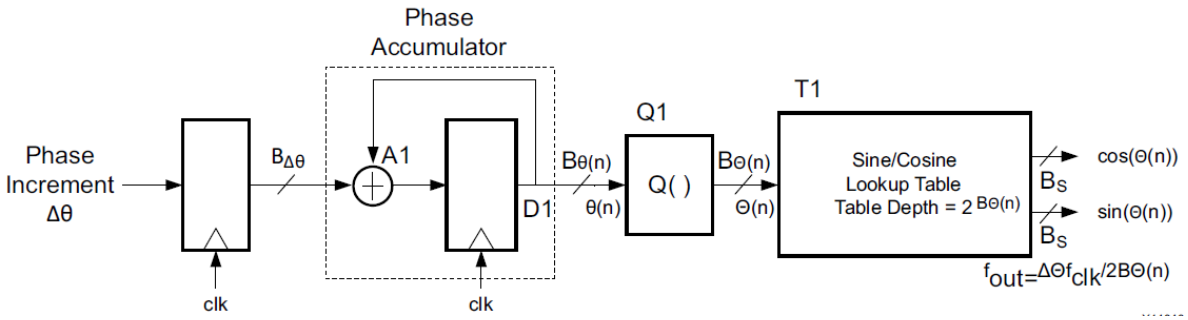


Fig. 3.1 Phase Generator of DDS

The angular phase range of sinusoidal transmissions is 0 – 2π. The digital implementation follows the same pattern. In the DDS implementation, the carry function of the counter allows the phase accumulator to operate as a phase wheel.

Envision the sinusoidal waveform oscillation as a vector circling around a phase circle to grasp this aspect. Each indicated point on the phase wheel relates to a sine wave cycle's corresponding point. Analyze the sine function of the angle generating a relatively similar sine wave as the vector moves around the circle. One complete cycle of the output sinusoidal waveform is produced by rotating the vector around the phase

axis at a consistent pace. The phase accumulator generates uniformly distributed angular values as the vector rotates linearly around the phase axis. The points on the loop of the output sinusoidal waveform correspond to the values of the phase accumulator.

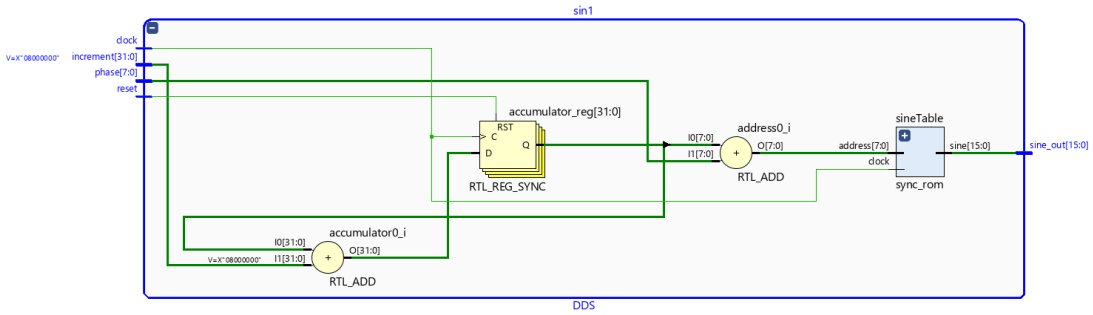


Fig. 3.2 DDS block diagram generated

The number of observations for every cycle diminishes as the output frequency rises. The maximum basic output frequency of a sinusoidal DDS is $\frac{f_c}{2}$, because sampling theory mandates that at least 2 different samples for each cycle are necessary to recreate the output waveform. However, for real world scenarios, the output frequency is restricted to a fraction to improve overall quality of the output waveform.

The DDS Compiler core's standard mode employs phase truncation. The integrator generates a phase slope, to be mapped to a sinusoid through the LUT. phase angle having two precision levels are denoted separately as $\theta(n)$ and $\Theta(n)$ These angles are supplied to a lookup table's address port, which performs the phase-space to time mapping. If an analogue output is required, the DDS sends these samples to a DAC and a low-pass filter, which produces an analogue waveform with the desired frequency structure [1]. The basis waveform's quarter wave symmetry are used to create a waveform. For the case in study, the two MSB bits of $\Theta(n)$ are used for phase mapping. Because the memory requirements are lowered, this design is more resource efficient, allowing for either reduced FPGA block RAMs or lesser distributed memory.

There are two types of applications that now use DDS method of waveform formation: DDS's combination of spectrum performance and frequency-tuning resolution. Communications design engineers frequently choose this when they need fast-changing frequency sources with a wide phase variation and low spurious. With reference to PLL the DDS method actually increases controllability in frequency domain, as a LO, and sometimes for direct Radio communications also

A DDS is also used as a customizable signal generator in various biotechnological and industry related applications. Since DDS is digitally configurable, it is possible to change the frequency and phase of a waveform without needing to alter the external necessary components when using standard analogue waveforms generators.

DDS allows for actual time frequency alterations to find resonance frequencies or mitigate for temperature fluctuation. To use DDS in variable frequency sources to monitor impedance inside an impedance sensor, to develop pulse modulated waveforms, or perhaps to research loss in Local area networks or telephone lines are instances of such applications.

3.3 DDS Code

This subsection made a legitimate attempt at explaining the program that is being used to construct the sinusoidal frequencies that would use the Direct Digital Synthesis technique, which had already been taken into account in detail in subsection 3.2.

The module which has been used contains the following elements taken as input and output to the module.

```
module sincos (clock, reset, increment, phase, sine_out, cos_out);
input clock, reset;
input [31:0] increment ;
input [7:0] phase;
output wire signed [15:0] sine_out,cos_out;

reg [31:0] accumulator;
```

Clock – Acts as the main input for supplying the entire module with continuously running clock pulses

Reset – Used to reset the counter value

Increment – acts as an increment for generating the sinusoidal signal. This takes up 32 bits

Phase – represents the phase of the sinusoid. This is taken as an 8-bit input

Sine_out – Outputs the 16-bit signed magnitude sinusoid signal sine

Cos_out – Outputs the 16-bit signed magnitude signal cosine

Accumulator – Is a 32-bit register that serves the purpose of storing the intermediate outputs obtained during the code run

```
always@(posedge clock)
begin
if (reset)
accumulator <= 0 ; // increment phase accumulator
else accumulator <= accumulator + increment;
end
```

Represents the simple module for incrementing the phase value of the sinusoid by utilising the 32-bit accumulator register. A simple if and else statements are presented. The if statement takes the argument reset and if the reset=1'b1 then it will go inside the if loop and executed the following statements within it.

To link the value stored in the accumulator register sine cos lookup table is needed. This requires creating another verilog module having the inputs clock, address, cos and sine. Sine and cos are assigned an initial value of hexadecimal number of 16 bits which is 2bytes worth of data

```
assign sine_out = 16'haa ;
assign cos_out = 16'haa ;
```

The sine and cosine wave ROM table produces twos complement 16-bit approximation of a sine and cosine waveform if it is given an input phase address.

The ROM module begin with an always block which is triggered at the positive edge of every clock cycle.

For every phase address the values of sine and cosine are stored in a table. The values of quarter waves are stored. The remaining points for the sine and cosine waves are generated by simply utilising the fact that both sine and cosine are perfectly synchronized waveforms with repetitive values for various cycles.

Once the sine waveform has been stored in the look up table the values of the same for co-sinusoidal can be obtained by simply phase shifting the values by 90º

As an example, consider

8'h00: sine = 16'h0000

The above line indicates that at phase address 00 representing the phase angle 0º the value of sine is 0. As sin(0)=0. These zero value is stored digitally as a 16-bit binary input 16'h0000.

Since the value of co-sinusoidal at 90º is 0 which is mathematically represented as

$$\mathrm{Cos}\left(\pi/_2\right) = 0$$

Post applying the 90º the value of phase address in hexadecimal number system is given by

8'h40: cos = 16'h0000

The output sine wave in signed hexadecimal has a total of 16-bits. 16 binary bits gives a total of 256 values. These values represent the phase of the sinusoid. Usually one full cycle of sine wave will have 0 to 2π. Where the sine wave starts at 0º reaches the peak value at 90º falls down to 0 at π goes into the negative half cycle at $3\pi/_2$ and again reaches the zero value at 2π thus completing one full cycle. The entire 360º range can be classified into 4 regions. These 4 regions are represented by 256 phase values.

Region 1	0º – 90º	8'h00 – 8'h40
Region 2	90º – 180º	8'h40 – 8'h80
Region 3	180º – 270º	8'h80 – 8'hC0

Region 4	270º – 360º	8'hC0 – 8'hFF

The maximum hexadecimal number for 8bits is FF. The Sinusoidal values are taken accordingly. For 360º duration. Each region above depicts the quarter wave of the sinusoid. All 4 regions together make up one full cycle of the sinusoidal wave.

Similarly the same can be observed for a cosine sinusoidal wave by simply shifting the Sinusoidal wave by 90º by phase. The respective regions for the cosine sinusoid is given below

Region 1	90º – 180º	8'h00 – 8'h40
Region 2	180º – 270º	8'h40 – 8'h80
Region 3	270º – 360º	8'h80 – 8'hC0
Region 4	360º – 90º	8'hC0 – 8'hFF

Sine and cosine will have the exact same values given the phase difference is taken to be 90º between each and every value. The same has been shown above.

Sine waveform one quarter cycle (Region 2, 90º – 180º) :

40h →3fffh	41h→3ffah	42→3febh
43h →3fd2h	44h →3fb0h	45h →3f83h
46h →3f4dh	47h →3f0dh	48h →3ec4h
49h →3e70h	4ah →3e14h	4bh →3dadh
4ch →3d3dh	4dh →3cc4h	4eh →3c41h
4fh →3bb5h	50h →3b1fh	

Cosine waveform one quarter cycle (Region 1, 90º – 180º) :

00h→3fffh	01h→3ffah	02h→3febh

03h→3fd2h	04h→3fb0h	05h→3f83h
06h→3f4dh	07h→3f0dh	08h→3ec4h
09h→3e70h	0ah→3e14h	0bh→3dadh
0ch→3d3dh	0dh→3cc4h	0eh→3c41h
0fh→3bb5h	10h→3b1fh	

Given the 90º phase shift in the angle the values for the sine and cosine remains equally similar.

3.4 DDS Testbench Code

Every verilog code requires a testbench code to perform simulations. Testbenches are composed of non-synthesizable verilog code that creates design inputs and verifies that the results are appropriate. We can employ a modelling tool that lets us to examine waveforms instantaneously. This mechanism is provided by Xilinx's Vivado a widely accessible software package.

The very first aspect of developing a testbench is to construct a verilog module that serves as the test's top level.

we would like to design a module without any input output values. The reason to do it is because we would like the testbench component to be entirely self-contained.

The code below demonstrates how to construct an initial module that would function as our test platform.

We should then implement the architecture that we are investigating after building a testbench component. This permits us to attach signals to the structure and invigorate the code.

The testbench code begins by declaring the reg and wire data types as given below

```
reg clk_50, clk_25, reset

reg [31:0] index;
```

```
wire signed [15:0]  sin_tb_out;
```

Two types of clocks are required to generate the sinusoidal waveform. clk_50 and clk_25 generates sinusoids of two different frequency components. sin_tb_out which is a 16-bit signed wire data type generates the sinusoid output waveform.

Initialization of clocks and index need to use initial begin and always begin statements of verilog language.

The generation of clocks can happen in two different methods. We could therefore write a program to do this in two contexts together within initial block and always block. The vhdl delay function can then be used to arrange the transitions.

Whenever it refers to the system clock, we have used the forever term to keep things operating throughout our experiments.

Users execute an inverted clock every 1 ns that use this technique, producing in an operating clock speed of 1GHz.

This clock speed was being used solely to provide a speedy computation time. In reality, 1GHz frequency speeds in FPGAs are really not achievable, hence the benchmarking tool frequency must reflect the equipment clock rate.

The verilog program below illustrates how well clock and resetting pulses in our simulation environment are handled.

```
// Clock-1

initial begin

clk = 1'b0;

forever #1 clk = ~clk;

end


// Clock-2

initial begin

reset = 1'b1;
```

```
#10
reset = 1'b0;
end
```

For incrementing the index register the following verilog program using always begin statements have been shown

```
always @ (posedge clk_50) begin
        index  <= index + 32'd1;
    end
```

Every increment of the clk_50 pulse at the positive edge of the clock the index register is incremented by a value of 1 in a 32-bit decimal number

3.5 DDS Simulation Results

The code structure discussed in section 3.2 follows the same for generating both the sine and co sine sinusoidal signals. The simulations from the Xilinx vivado software have been presented in Fig. 3.2 From the figure it can be noted that whenever the value of the sine waveform is 0 at the same instant of time we can observe that the value of co sine waveform is 1. This verifies the basic sine and co sine waveforms. The frequency of these waveforms can be altered by making changes in the program code of verilog to obtain different frequencies of both sine and co sine.

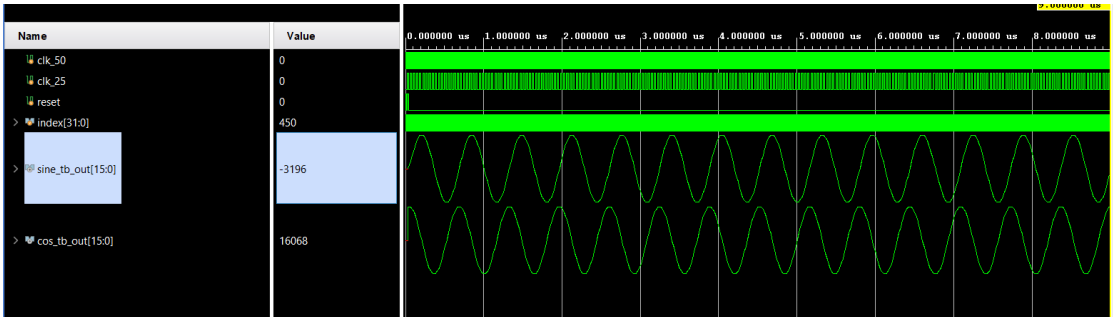


Fig. 3.3 Generation of sine and cosine waveforms

CHAPTER 4
DIGITAL MODULATIONS

4.1 Modulation Overview

The transmission of digitally altered carrier analog waveforms between two end points is termed as digital modulation in a communications system. These signals are also known as radio waves or radio signals as they travel through the atmosphere of earth. These find applications is many of the wireless systems in communication. Traditional analogue modulation systems, such as amplitude modulation (AM), frequency modulation (FM), and phase modulation (PM), are frequently replaced by digital schemes which are in more modern way. which provide several significant advantages over traditional analogue systems, including simplicity in processing methods, multiplexing, and these also studied to have better noise margin.

The word digital communications is a bit of a misnomer, as it can signify different things to different individuals. Digital communications in this context refers to systems in which carrier waves having higher frequency are modulated by their counterpart waves at lower frequencies digital radio signals and systems in which digital pulses are transmitted (digital transmission). Because digital transmission technologies transfer data in digital form, a real path between the transmitter and receiver is required, such as a wire made of some metals, coaxial cable, or even in the case of light transmission, the optical fibre cable. The carrier facility in digital radio systems could be a physical cable or open space. The nature of the modulating signal distinguishes digital radio systems from traditional analogue modulation communications systems. Analog carriers are used in both analogue and digital modulation systems to transmit information. The information signal in analogue modulation systems is analogue, but the information signal in digital modulation systems is digital, which could be computer generated data or digitally encoded analogue signals.

Binary data contains strings of 1’s and 0’s. If this data is to be transmitted over copper wires, they can be directly transmitted as appropriate voltage levels. Data in the form of digital signals is difficult to transmit long distances or to radiate into free space. To overcome this problem sinusoidal carrier is added to binary data and resultant signal is transmitted using antenna. This process usually termed as modulation. The modulation

process involves keying the voltage, time or angle of the sinusoidal carrier in some manner that variates according to input digital stream.

There are three digital bandpass transmission schemes

1. Amplitude Shift keying (ASK)
2. Phase Shift Keying (PSK)
3. Frequency Shift Keying (FSK)

A digitally modulated signal termed amplitude shift keying (ASK) is obtained when the transmitted signal is digitized and the carrier's amplitude is altered proportionally to the data signal. Frequency shift keying (FSK) is created in the cases where carrier frequency is altered proportionally to the data signal, while phase shift keying (PSK) is generated when the carrier angle is varied proportionally to the data signal.

Band pass signals can be of two types

1. Binary
2. M-ary

A binary bandpass signals have two symbols which are represented by binary logic values of 1 and 0. whereas M-ary has M different symbols each having log2M bits.

This dissertation focuses mainly on the Binary bandpass signal transmissions ASK, PSK and FSK.

4.2 Amplitude Shift Keying (ASK)

In the context of digital communications, ASK is a modulation procedure that assigns two or more than two discrete voltage levels to a sinusoid. Digital modulation systems include this form of modulation. The term "keying" is important here since it refers to the transmission of a digital signal via a channel.

Two input signals are needed for ASK, a sequence of binary data and a carrier signal. The most crucial note to memorise is that the carrier needs to have wider range than the input data signal

The voltage of the input data v of the input binary signal fluctuates in accordance with the carrier frequency in ASK method. The carrier frequency and its time intervals are multiplied with the input data sequence in ASK. The similar process is repeated for all time intervals in between the time interval of input data signal. If the input digital signal is at logic level high for a given time, will be seen at the output terminals with an increase in the amplitude level. The main aim of amplitude keying approach is to modify or improve the amplitude properties of the data signal in accordance to the carrier. The incoming binary data and sinusoidal signal carrier are simply applied to two inputs of a product modulator (balanced modulator) to form an ASK signal [9]. Here, in this dissertation a vedic multiplier which acts as a product modulator is utilized to obtain ASK output signal. The block diagram of the ASK modulator can be as shown in Fig. 4.1

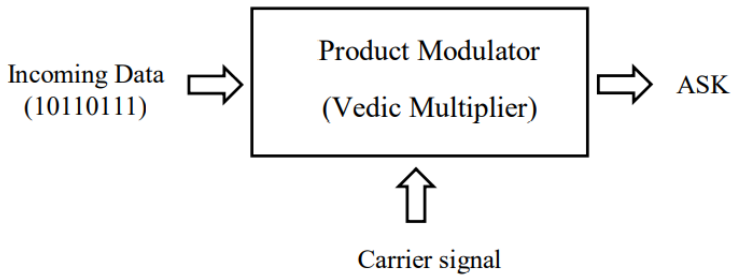


Fig. 4.1 Block diagram of ASK generation

Let m(t) be the data signal which can be a string of 1's and 0's. c(t) be the carrier sinusoid signal which is represented as

$$c(t) = Accos2\pi fct$$

where Ac is the carrier signal amplitude, fc is the carrier signal frequency.

The output when these two are multiplied is the output ASK signal

$$\begin{aligned} ASK &= m(t) * Accos2\pi fct \\ &= 1 * Accos2\pi fct \; ; \quad \text{When } m(t) = \text{logic } 1 \\ &= 0 * Accos2\pi fct = 0 \; ; \text{ When } m(t) = \text{logic } 0 \end{aligned} \tag{1}$$

In its basic sense, a radio frequency burst is only transmitted when a binary 1 appears and is silenced when a binary 0 appears

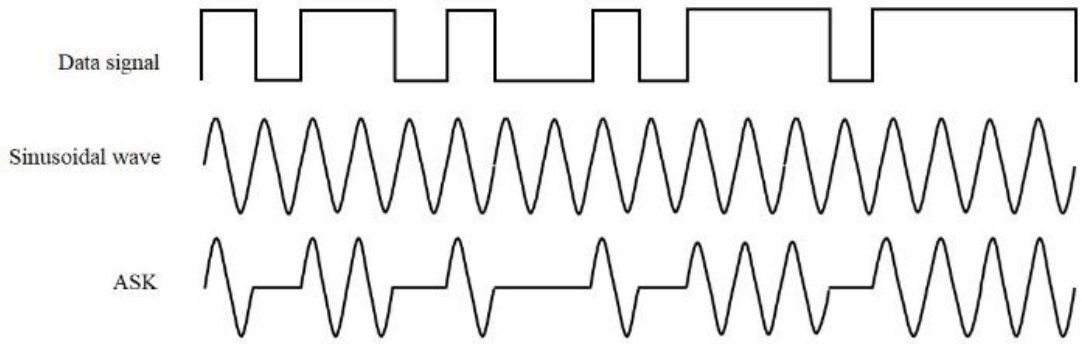


Fig. 4.2 Sample waveforms of ASK generation

In communications, modulation is very significant. Applications for amplitude shift keying are listed below. They are:

Low frequency radio frequency applications

Devices for smart home devices

Devices for industrial control systems

Area network for wireless

Monitoring methods for tyre pressure

As a result, ASK (amplitude shift keying) is a digital signal processing technique for enhancing the amplitude properties of a binary signal input. However, its shortcomings limit it. And the other modulation scheme, FSK, can solve these disadvantages.

In ASK information is transmitted by change in amplitude as can be seen from the Fig. 4.2 where the input data for example is taken as 101101001011101111 which is represented by return to zero signal levels where logic 1 is represented by maximum

voltage level and logic 0 is represented by zero volts. The sinusoidal signal is generated by using Direct Digital Synthesis (DDS) using Xilinx Vivado software A detailed study on DDS is presented in [5]. when sinusoidal is multiplied with logic 1 the output will be the same sinusoidal signal. Similarly, when it is multiplied with logic 0 i.e., zero volts it gives an output of 0 volts ASK. This is clearly represented in the Fig. 22 Therefore, it is sometimes implied that ASK, PSK, and FSK are "binary" methods (BASK, BPSK, and BFSK), which means that the signal characteristics only change between two values. ASK in here can also be represented as BASK. At the changeover points, there are abrupt discontinuities. As a result, the signal's bandwidth becomes unnecessarily wide [10]. Before transmission, band limiting is usually applied to ensure that the breaks are rounded by exposing to limitations in the frequency bands. The digital message or the modulated signal itself can be subjected to band limitation. Often the data rate is set to be a multiple of the carrier frequency.

4.2.1 ASK Code

The verilog coding for ASK modulation using Direct Digital Synthesis that has been used in the project has been presented briefly as listed

The module is declared with the inputs din which is a 16-bit digital data input and clk input. The output is taken to be a ASK signed 16-bit output

```
module ASK(
        input [15:0]din,
        output signed [15:0] ASK,
        input clk
    );
```

The clock inputs for generating the sinusoid that needs to be used in the generation of ASK signal remains the same as discussed in DDS module. This ASK module uses two clocks for synchronising purpose

```
reg clk_50, clk_25, reset;
reg [31:0] index;
wire signed [15:0] sine;
```

By utilising the Sixteen-bit vedic multiplier generated previously the basic concept of ASK generation as depicted in the Fig. 4.2 can be administered as seen below

```
sixteenbit_vedic_mult svm1(.a(din),.b(sine),.mult(ASK));
```

The 16 bit input from the 16-bit vedic multiplier A[15:0] can be assigned to be the input digital data signal of 16 binary bits. Similarly another input to the sixteen bit vedic multiplier B[15:0] is used to input the sine wave obtained from the output of DDS module.

The two inputs, binary data input signal and sinusoidal signal are multiplied together using the 16-bit vedic multiplier to finally obtain the output ASK modulated signal

4.2.2 ASK RTL implementation

In the RTL implementation of the verilog code for ASK modulation the output functionality of the generated ASK signal is verified. The RTL synthesis tool generates the synthesized schematic

The netlist is shown graphically in the schematic. It's created to:

- View the netlist as a graphical representation.
- Go through the gates, hierarchies, and connections.
- Trace and expand logic cones
- Examine the design.
- Gain a better understanding of what's going on inside the design.

The Fig. 4.3 taken from the Xilinx vivado presents the elaborated schematic, we understand in what way the device has construed your code. In Synthesized and Implemented design, we see the gates generated by the synthesis tool.

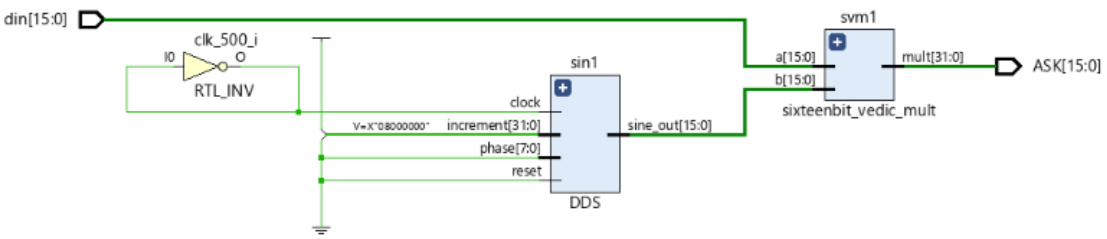


Fig. 4.3 Xilinx RTL implemented schematic of ASK

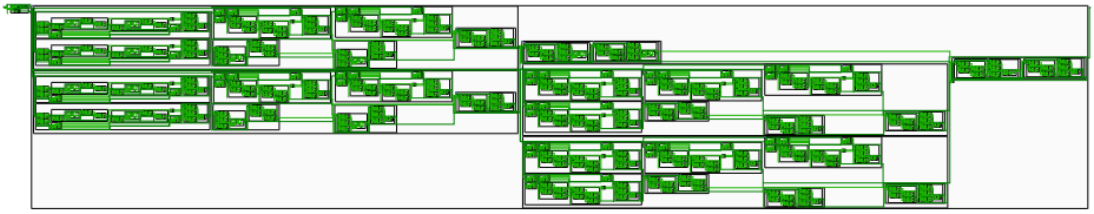


Fig. 4.4 Xilinx RTL Elaborated Schematic of ASK

The overall RTL blocks can be seen as shown in Fig. 4.3 sixteenbit_vedic_mult contains 8-bit, 4-bit and 2-bit vedic multipliers inside it. 8-bit contains 4-bit and 2-bit inside it. 4-bit contains 2-bit inside it. The block showing input and output for a DDS is also shown with clock signal, phase value, increment value and reset signal. This block produces a sinusoidal as an output wave.

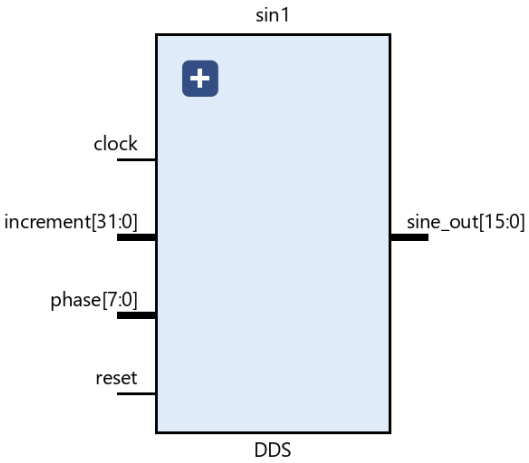


Fig. 4.5 DDS module Xilinx implemented block

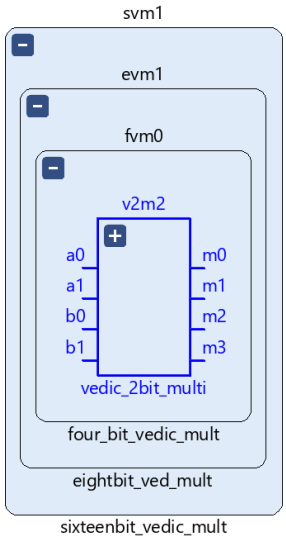


Fig. 4.6 Sixteen-bit vedic multiplier Xilinx implemented block

As can be seen from the Fig. 4.3 the generation of ASK using vedic multiplier can be clearly understood. The vedic multiplier takes the input data signal as one of it's input and the other input is provided from the output of DDS module. The vedic multiplier generates the 16 big signed output ASK waveform

4.2.3 ASK Synthesized Design

The synthesized implementation from Xilinx vivado of the ASK modulator has been presented in the Fig. 4.7 and the elaborated design obtained by expanding the design to the leaf cells has been presented in Fig. 4.8 It can be seen that entire connections between each and every module, input and output connections in the figure.

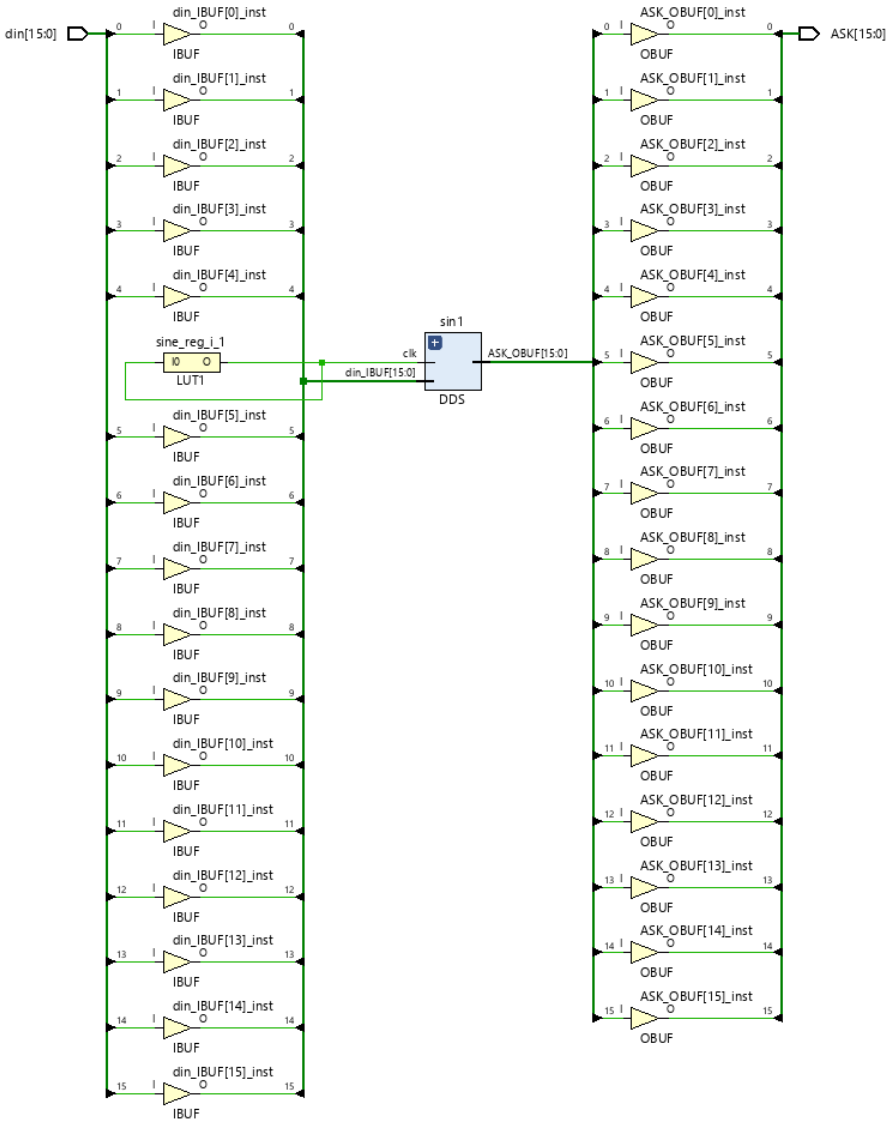


Fig. 4.7 Xilinx synthesized design of ASK

Fig. 4.8 Xilinx elaborated synthesized design of ASK

4.2.4 ASK Simulation Waveforms

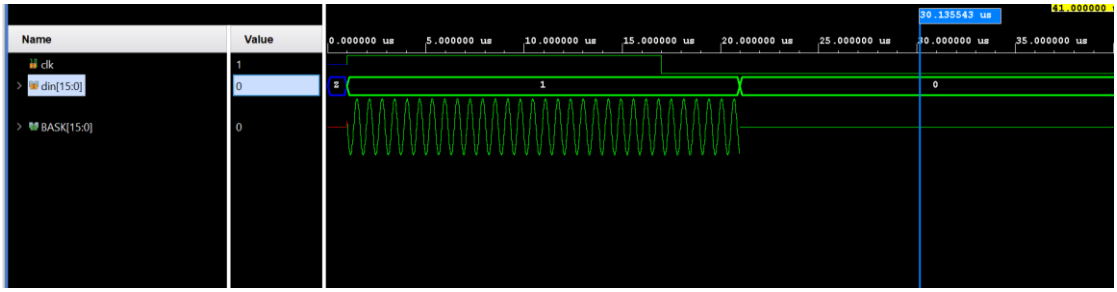


Fig. 4.9 Xilinx output of ASK simulations

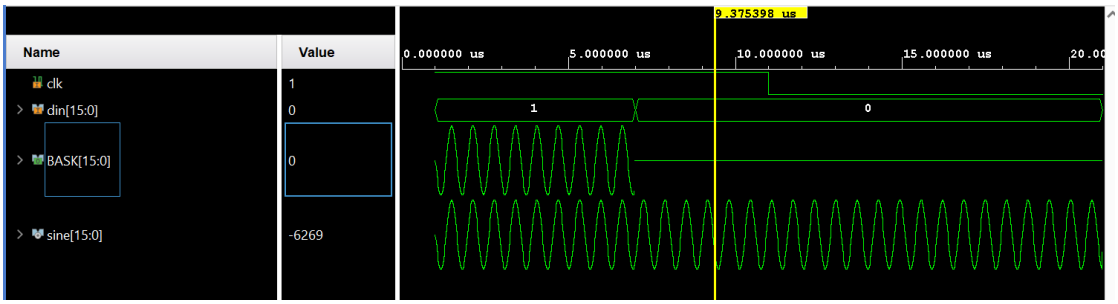


Fig. 4.10 ASK with sinusoidal waveform

The sinusoidal signal is generated by using Direct Digital Synthesis (DDS) using Xilinx Vivado software A detailed study on DDS is presented in [4]. when sinusoidal is multiplied with logic 1 the output will be the same sinusoidal signal. Similarly, when it is multiplied with logic 0 i.e., zero volts it gives an output of 0 volts ASK. This is clearly represented in the Fig. 4.10 Therefore, it is sometimes implied that ASK, PSK, and FSK are "binary" methods (BASK, BPSK, and BFSK), which means that the signal characteristics only change between two values. ASK in here can also be represented as BASK. At the changeover points, there are abrupt discontinuities. As a result, the signal's bandwidth becomes unnecessarily wide [7]. Before transmission, band limiting is usually applied, in which circumstance these breaks are rounded. The data signal can be exposed to limitations at frequency band level. Often the data rate is set to be a multiple of the carrier frequency.

4.2.5 ASK Power Report

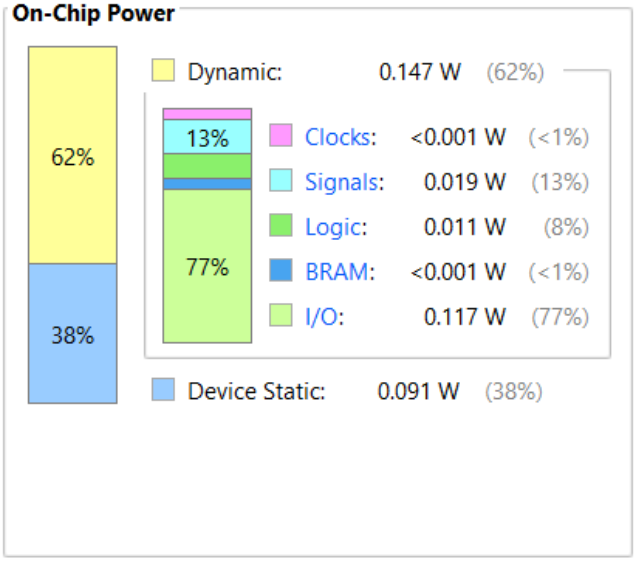
The Xilinx tool calculates power for the implemented design and generates a power report [3][4] as shown in Fig. 4.11

The power components in the power report are divided into static and dynamic. Static power is when the device is idle it still consumes some power This can be because of leakage currents flowing in the transistors. Dynamic power is when the device is powered on and various components like clocks, signals, logic, BRAM and Input output signals make up the dynamic component of power. The static and dynamic power sum up to Total on-chip power. A particular circuit can have an estimated power which is the design power budget. The power budget margin is the amount of power saved from the estimated budget. Junction temperature is the highest operating temperature of the

semiconductor. Thermal margin is the difference between the current and maximum temperature.

As can be seen the total power consumption is 0.239 W which is 1.761 W less than the defined power budget 2 W. The percentage of dynamic power is reported as 62% and static power at 38%.

The detailed power consumption of clocks taking up <0.001 W power which amounts to <1% of the total power consumed, signals taking 0.019W amounting to 13% of the total power consumption, logic taking up 0.011 W amounting to 8% of the total power consumption, BRAM consuming <0.001 W amounting to <1% total power consumption similar to the clocks, and it is interesting to note that the I/O amounts to 0.117 W which is 77% taking the majority of the dynamic component of the power



Total On-Chip Power:	**0.239 W**
Design Power Budget:	**2 W**
Power Budget Margin:	**1.761 W**
Junction Temperature:	**25.6°C**
Thermal Margin:	59.4°C (22.6 W)

Fig. 4.11 Power report of ASK

4.3 Phase Shift Keying (PSK)

In a radio telecommunications system, the term PSK, or phase shift keying, is commonly used. This process is generally used in transmission of data signal. In

comparison to additional modulation forms, it permits data to be transported through telecommunications transmission more efficiently. Presently this method of communication is preferred widely, with many communication transmission formats. each having their own compensations and difficulties.

This method communicates data by modifying the angle of the carrier, repeatedly recognized as a reference signal. This scheme of modulation contains increments of angle which in turn can be allotted binary logic values encoding the identical quantity of bits. The exact phase denotes the symbol formed by each bit pattern.

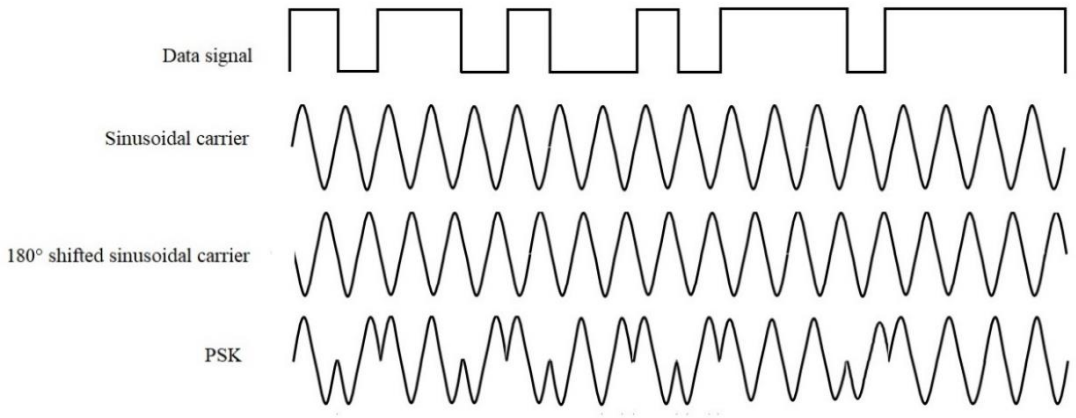


Fig. 4.12 Sample waveforms of PSK generation

In PSK the information is transmitted in the form of phase shift. The Fig. 4.12 shows the sample waveform. Where sinusoidal carrier is taken to be sinusoidal and the waveform also incorporates sinusoidal carrier with 180° shift which is generated by DDS and shifted using Verilog coding. PRK (phase reversal keying) or 2PSK are other names for PSK. This type of phase-shift keying makes use of two 180-degree split phases. This is why it's sometimes also referred to as 2-PSK.

The general equation for a PSK wave is given by

$$S_{PSK}(t) = \sqrt{\frac{2E_b}{T_b}}\cos\left(2\pi f_c t + \pi(1-n)\right) n = 0,1 \qquad (2)$$

Where Eb is the average energy per bit. Tb is the bit duration. f_c is the carrier frequency. The value of n yields two phases. In the equation 2, The value of n is dependent on the logic level of data signal. when n=1 the value of output is the input sinusoidal signal

The product modulator takes the input sinusoidal carrier1 and when n =0 the output is a sinusoidal with a ' – ' in front of it i.e. 180° shifted sinusoid.

This dissertation work utilises the sinusoidal obtained from the output of DDS as the input and by using Verilog coding we impart 90° shift to the co-sinusoidal thereby producing a total shift of 180°.

Thus, vedic multiplier multiplies the data signal and sinusoidal either with 0° shift or 180° shift based on the logic state of the input data signal to produce an output PSK modulated waveform.

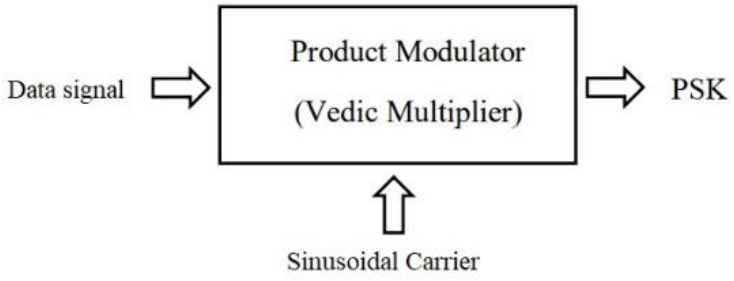


Fig. 4.13 Block diagram of PSK generation

The output waveform will be similar to sinusoidal carrier for a logic 1 data signal and the output is 180° out of phase with sinusoidal carrier in case of a logic 0 data signal. The output waveform for a sample binary data of 101101001011101111 is as depicted in the Fig. 4.12

The following are some of the benefits of phase-shift keying.

In contrary to FSK, this sort of PSK allows information to be transmitted with a radio communications transmission signal more efficiently.

When compared to ASK modulation, it is less susceptible to defects and consumes a similar bandwidth.

The following are some of phase-shift keying's drawbacks.

When compared to ASK modulation, the efficiency of bandwidth of PSK is lower.

It's a reference signal that's not coherent.

The binary information can be deciphered by determining the phase levels of the signal. Recovery and detection algorithms are extremely challenging.

Because of reference signal for demodulation is not stable, the fault might accumulate with time, resulting in inaccurate demodulations.

PSK could be used for the following purposes.

Biostatistics, Bluetooth, Near Field Communication, RFID or even in many of the additional wireless communications

Optical Fibre

LO

WDM

4.3.1 PSK Code

The verilog coding for PSK modulation using Direct Digital Synthesis that has been used in the project has been presented briefly as listed

The module is declared with the inputs din which is a 16-bit digital data input and clk input. The output is taken to be a PSK signed 16-bit output

```
module PSK(
        input [15:0]din,
        output signed [15:0] PSK,
        input clk
    );
```

The clock inputs for generating the sinusoid that needs to be used in the generation of PSK signal remains the same as discussed in DDS module. This PSK module uses two clocks for synchronising purpose namely clk_50 and clk_25. It also uses a sixteen-bit wire named as din_bar which stores the negated value of the input data signal. Along

with the sine wave output from DDS this PSK modulation also requires testbench_out, p_out and shifted_180 declared as a wire data type of 16 bits signed inputs

```
wire [15:0] din_bar;

reg clk_50, clk_25, reset;
reg [31:0] index;
wire signed [15:0]  testbench_out,p_out,shifted_180,sine;
```

To create a negated value of data input signal which will function in synchronization with the input data logic value the following verilog code has been written

```
assign din_bar=din-1'b1;
```

This utilises the assign keyword from the verilog code to assign the value of din-1'b1 to the din_bar. This ensures that whenever input data signal is 0. The output waveform will be imparting a negated value to apply the phase shift. When the din logic value is 1, the assign statement gives the output value of 1-1=0. This will ensure that the output PSK waveform will not be applied with 180^{o} phase shift.

By utilising the Sixteen-bit vedic multiplier generated previously the basic concept of PSK generation as depicted in the Fig. 4.12 can be administered as seen below

```
sixteenbit_vedic_mult svm1(.a(din),.b(sine),.mult(p_out));

sixteenbit_vedic_mult svm2(.a(din_bar),.b(sine),.mult(shifted_180));
```

The 16 bit input from the 16-bit vedic multiplier A[15:0] can be assigned to be the input digital data signal of 16 binary bits. Similarly another input to the sixteen bit vedic multiplier B[15:0] is used to input the sine wave obtained from the output of DDS module. This will give out the output waveform with 0^{o} phase shift taken to be as p_out.

Another vedic multiplier of 16 bits is instantiated to generate another sinusoid with 180^{o} out of phase to the p_out generated waveform. One input of the multiplier is taken to be the din_bar obtained form the assign statements. Another input is taken to be the sine wave output from DDS module. When these two inputs are multiplied this produces an output waveform termed as shifted_180.

To obtain the final PSK waveform another assign statement has been used as follows

```
assign PSK[15:0]=p_out[15:0]+shifted_180[15:0];
```

Both the outputs from each of the vedic multipliers employed are taken and a simple addition operation is performed to finally obtain PSK modulated waveform.

4.3.2 PSK RTL Implementation

In the RTL implementation of the verilog code for PSK modulation the output functionality of the generated ASK signal is verified. The RTL synthesis tool generates the synthesized schematic

RTL level schematic when elaborated through the Xilinx vivado tool generates the figure as shown in Fig. 4.14. This is helpful in the cases where the user can comprehend the interpretation of their code to essentially see the gate level synthesis

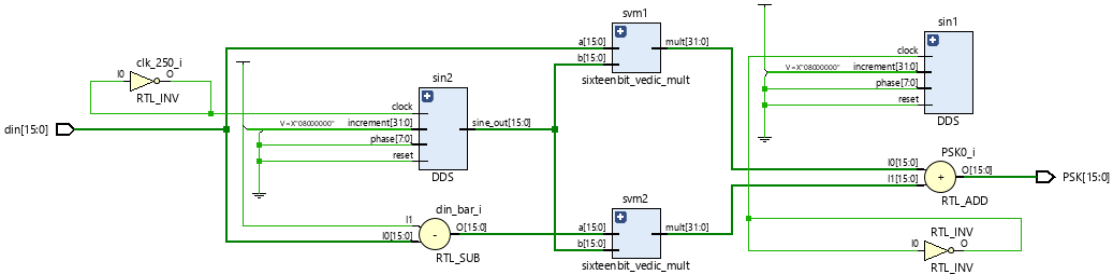


Fig. 4.14 Xilinx RTL implemented schematic of PSK

Fig. 4.15 Xilinx RTL Elaborated Schematic of PSK

4.3.3 PSK Synthesized Design

The synthesized implementation from Xilinx vivado of the PSK modulator has been presented in the Fig. 4.16 and the elaborated design obtained by expanding the design

to the leaf cells has been presented in Fig. 4.17 It can be seen that entire connections between each and every module, input and output connections in the figure.

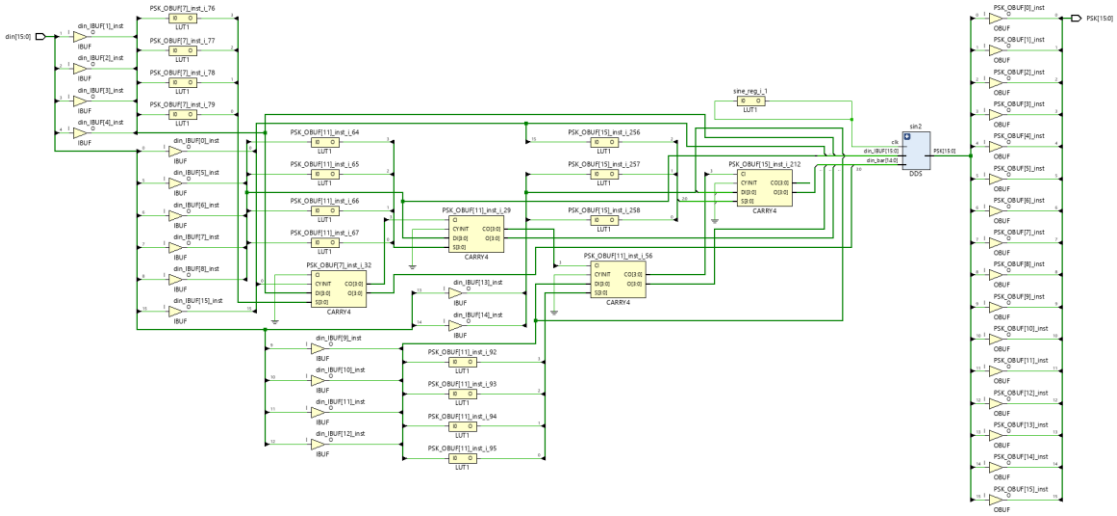


Fig. 4.16 Xilinx synthesized design of PSK

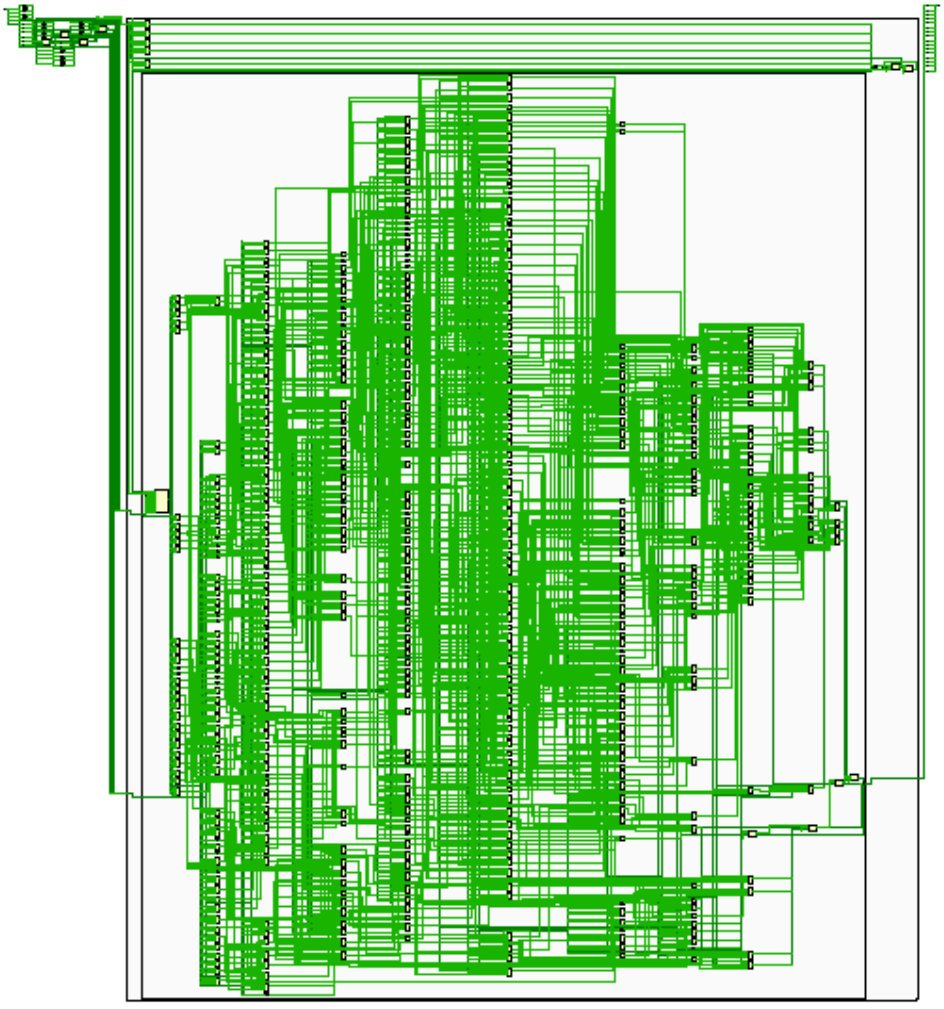


Fig. 4.17 Xilinx elaborated synthesized design of PSK

4.3.4 PSK Simulation Waveforms

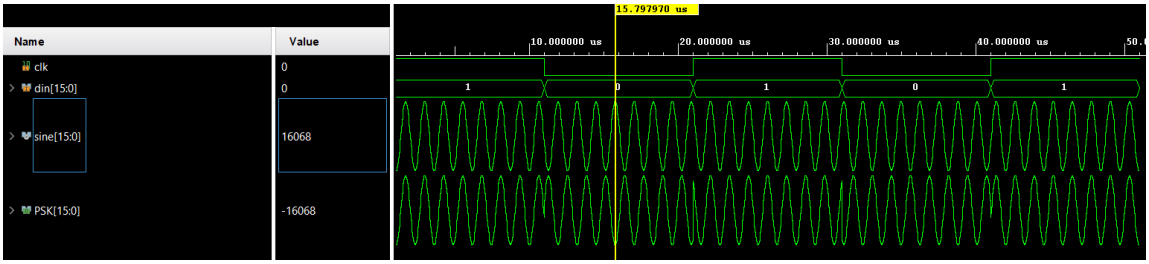


Fig. 4.18 Xilinx output of PSK simulations

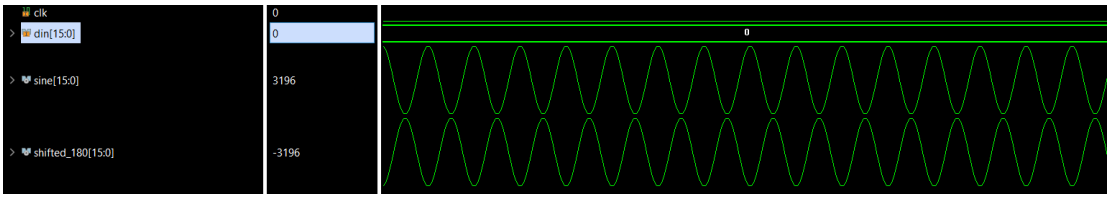


Fig. 4.19 Xilinx output of 180° phase shifted sinusoidal waveforms

The simulation results obtained from the Xilinx vivado are presented in the figures 4.18 and 4.19. The Fig. 4.19 shows two sinusoidal waveforms sine and shifted_180 which are 180^{o} out of phase to each other.

From the Fig. 4.18 the basic functionality of the PSK modulation depicted in the previous section in the Fig. 4.12 can be verified. Whenever the input data signal is 1'b1 the output PSK waveform will be the sine waveform. When the input data signal is 1'b0 the output PSK waveform will be shifted_180 waveform. So that When the input data signal changes from 1-0 or from 0-1the phase of the PSK output waveform changes.

4.3.5 PSK Power Report

The Xilinx tool calculates power for the implemented design and generates a power report [3][4] as shown in Fig. 4.20

The power components in the power report are divided into static and dynamic. Static power is when the device is idle it still consumes some power This can be because of leakage currents flowing in the transistors. Dynamic power is when the device is powered on and various components like clocks, signals, logic, BRAM and Input output

signals make up the dynamic component of power. The static and dynamic power sum up to Total on-chip power. A particular circuit can have an estimated power which is the design power budget. The power budget margin is the amount of power saved from the estimated budget. Junction temperature is the highest operating temperature of the semiconductor. Thermal margin is the difference between the current and maximum temperature.

As can be seen the total power consumption is 0.299 W which is 1.701 W less than the defined power budget 2 W. The percentage of dynamic power is reported as 69% and static power at 31%.

The detailed power consumption of clocks taking up <0.001 W power which amounts to <1% of the total power consumed, signals taking 0.032W amounting to 16% of the total power consumption, logic taking up 0.040 W amounting to 19% of the total power consumption, BRAM consuming <0.001 W amounting to <1% total power consumption similar to the clocks, and it is interesting to note that the I/O amounts to 0.135 W which is 63% taking the majority of the dynamic component of the power

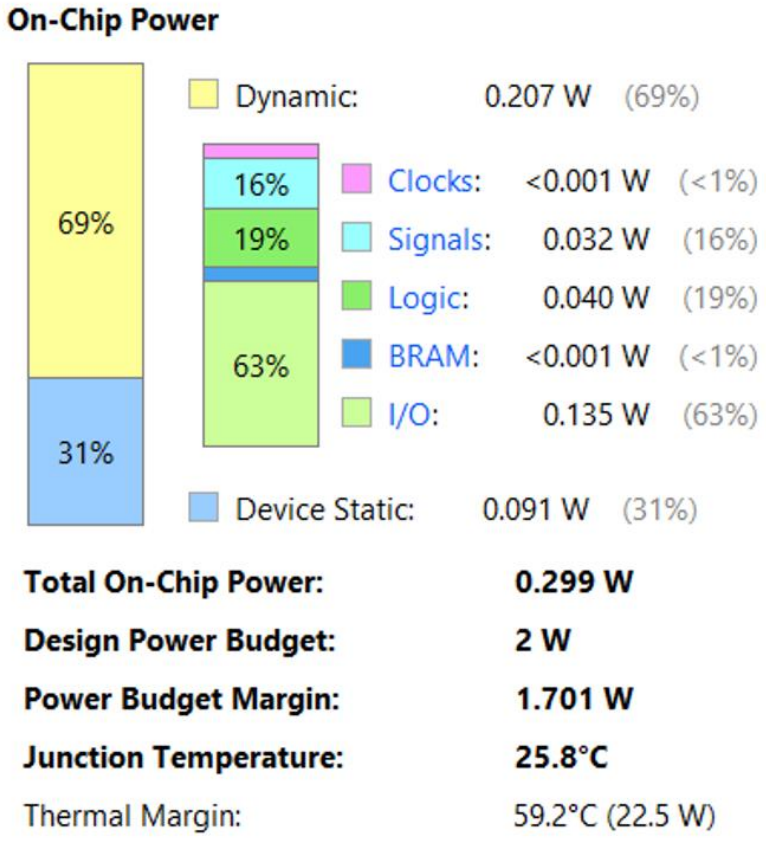


Fig. 4.20 Power report of PSK

4.4 Frequency Shift Keying (FSK)

Frequency shift keying, commonly recognized as FSK, one among the digital modulation technique that are discussed as part of this work. The qualities which all waveforms possess are its amplitude, frequency, and phase.

Modulation process to progresses any of these properties. Because the modulation practice has numerous compensations. The size of the antenna can be made smaller or to avoid the multiplexing of the signal, reduce the component of the noise are some among the numerous benefits FSK offers.

The carrier signal is used to change or improve the frequency features of an input binary sequence. Major downsides of ASK is volatility in its amplitude levels. As a result, it is only employed in a few cases. It also results in surplus power being unused because of low power efficiency. Frequency Shift Keying is recommended to alleviate these disadvantages. Binary Frequency Shift Keying is another name for FSK (BFSK).

This frequency shift keying method demonstrates how the carrier signal changed the frequency properties of a binary sequence. Binary communication can take place by a carrier signal with frequency shifts in FSK. The block diagram of Frequency modulator used in this dissertation is shown in Fig. 4.21 below

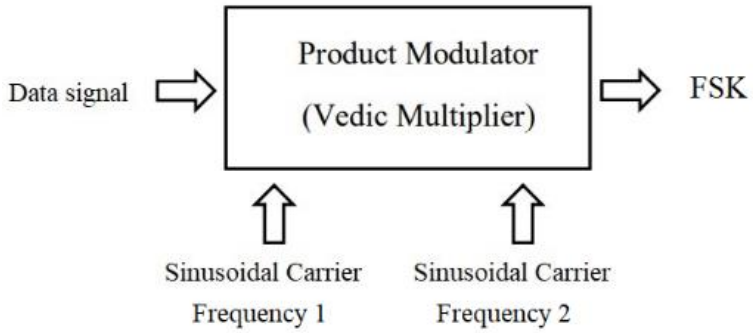


Fig. 4.21 Block diagram of FSK generation

FSK modulated signals are created uses carrier components that differs in their frequency levels. FSK modulated transmissions are expressed in the form of distinct frequencies. "Sinusoidal Carrier Frequency 1" and "Sinusoidal Carrier Frequency 2" are the corresponding frequencies. Logic 1 was defined by Sinusoidal Carrier Frequency 1, and logic 0 was defined by Sinusoidal Carrier Frequency 2. Only one distinction

exists between the two carrier signals: one of the carrier sources has a higher frequency than the other.

Frequency shift keying takes in two sinusoidal signals of different frequencies and produces a frequency modulated output depending upon the logic level of input data signal. The output FSK wave will be at high frequency when the input data signal is at logic level 1 and the output FSK wave will be at lower frequency when the input data signal is at logic 0. The same is represented in diagrams. Refer to Fig. 4.22. Here the high frequency signal is taken as sinusoidal wave of frequency f_1 and low frequency signal is taken as sinusoidal wave of frequency f_2. Here f_1 & f_2 taken as general representation and they can be of any frequencies as long as $f_1 > f_2$.

$$S_{FSK}(t) = \sqrt{\frac{2E_b}{T_b}} cos(2\pi f_1 t), \qquad 0 \le t \le T_b,$$

$$= \sqrt{\frac{2E_b}{T_b}} cos(2\pi f_2 t), \qquad 0 \le t \le T_b, \qquad (3)$$

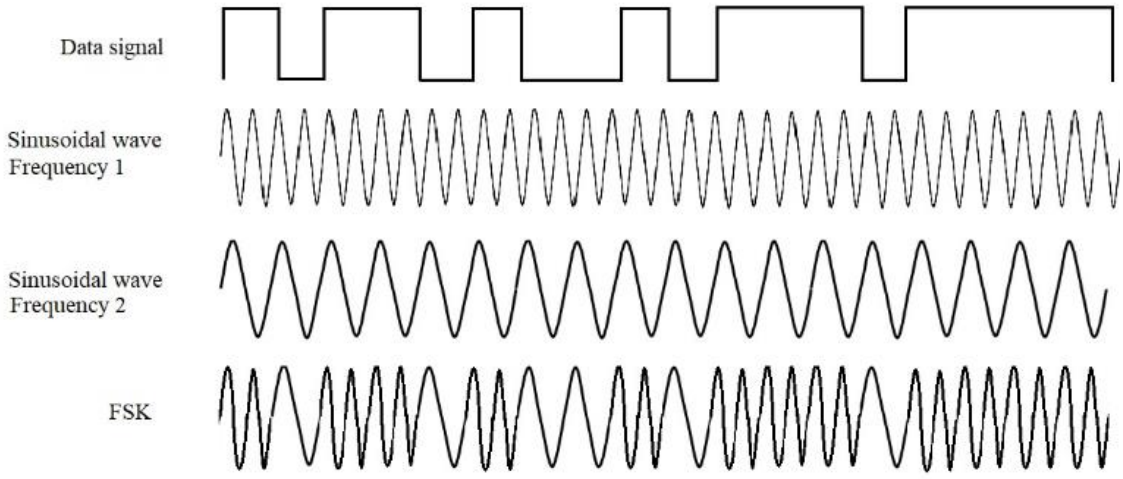


Fig. 4.22 Sample FSK waveforms

The pros and downsides of frequency shift keying are given below.

Advantages

The circuit is built in a simple manner.

No variations in amplitude

Allows for a high data transfer rate.

Error probability is low.

Has significant signal to noise ratio

Noise immunity is stronger than the ASK

With FSK, error-free transmission is achievable.

High-frequency radio communications benefit from it

Disadvantages

It necessitates more bandwidth than either the ASK or the PSK transmissions
This FSK can only be used in low-speed modems due to the massive bandwidth demands.
In comparison to phase shift keying, the BER in AEGN channel is lower.

4.4.1 FSK Code

The verilog coding for FSK modulation using Direct Digital Synthesis that has been used in the project has been presented briefly as listed

The module is declared with the inputs din which is a 16-bit digital data input and clk input. The output is taken to be a FSK signed 16-bit output

```
module FSK(
        input [15:0]din,
        output signed [15:0] FSK,
        input clk
    );
```

The clock inputs for generating the sinusoid that needs to be used in the generation of FSK signal remains the same as discussed in DDS module. This FSK module uses two clocks for synchronising purpose namely clk_50 and clk_25. It also uses a sixteen-bit wire named as din_bar which stores the negated value of the input data signal. Along with the sine wave output from DDS this FSK modulation also requires testbench_out,

testbench_out_1, sine_freq_1, sine_freq_2 declared as a wire data type of 16 bits signed inputs

```
wire [15:0] din_bar;

reg clk_50, clk_25, reset;

reg [31:0] index;

wire signed [15:0] sine_out,sine_out_1,sine_freq_1,sine_freq_2;
```

To create a negated value of data input signal which will function in synchronization with the input data logic value the following verilog code has been written

```
assign din_bar=din-1'b1;
```

This utilises the assign keyword from the verilog code to assign the value of din-1’b1 to the din_bar. This ensures that whenever input data signal is 0. The output waveform will be imparting a negated value to apply the phase shift. When the din logic value is 1, the assign statement gives the output value of 1-1=0.

By utilising the Sixteen-bit vedic multiplier generated previously the basic concept of FSK generation as depicted in the Fig. 4.23 can be administered as seen below

```
sixteenbit_vedic_mult svm4(.a(din),.b(sine_out),.mult(sine_freq_1));

sixteenbit_vedic_mult svm5(.a(din_bar),.b(sine_out_1),.mult(sine_freq_2));
```

The 16 bit input from the 16-bit vedic multiplier A[15:0] can be assigned to be the input digital data signal of 16 binary bits. Similarly another input to the sixteen bit vedic multiplier B[15:0] is used to input the sine wave obtained from the output of DDS module. This will give out the output waveform having frequency 1 taken to be as sine_freq_1.

Another vedic multiplier of 16 bits is instantiated to generate another sinusoid with frequency 2. One input of the multiplier is taken to be the din_bar obtained form the assign statements. Another input is taken to be the sine wave of lesser frequency output

from DDS module. When these two inputs are multiplied this produces an output waveform with frequency 2 termed as sine_freq_2.

To obtain the final FSK waveform another assign statement has been used as follows

assign FSK[15:0]=sine_freq_1[15:0]+sine_freq_2[15:0];

Both the outputs from each of the vedic multipliers employed are taken and a simple addition operation is performed to finally obtain FSK modulated waveform.

4.4.2 FSK RTL Implementation

In the RTL implementation of the verilog code for FSK modulation the output functionality of the generated ASK signal is verified. The RTL synthesis tool generates the synthesized schematic

vivado tool generates the elaborated schematic shown in Fig. 4.14. This is helpful in the cases where the user can comprehend the interpretation of their code to essentially see the gate level execution.

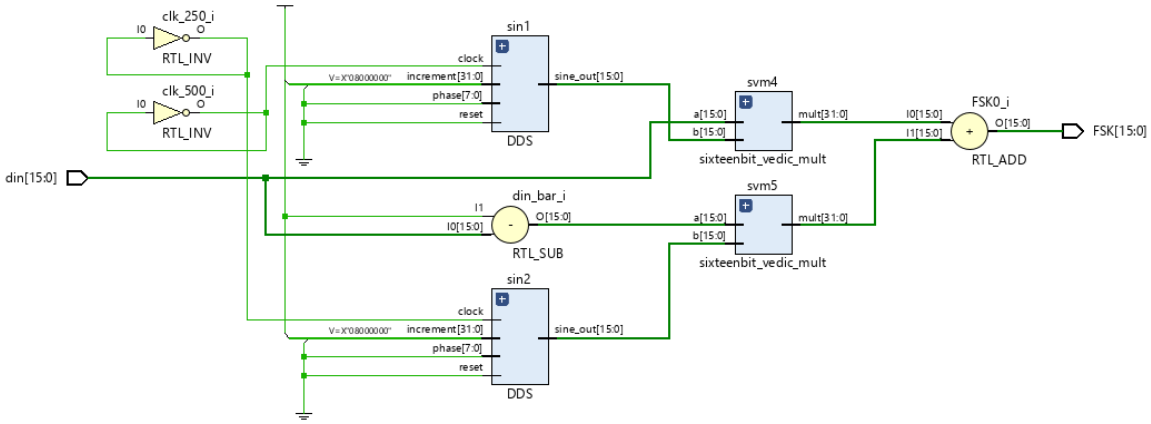


Fig. 4.23 Xilinx RTL implemented design of FSK

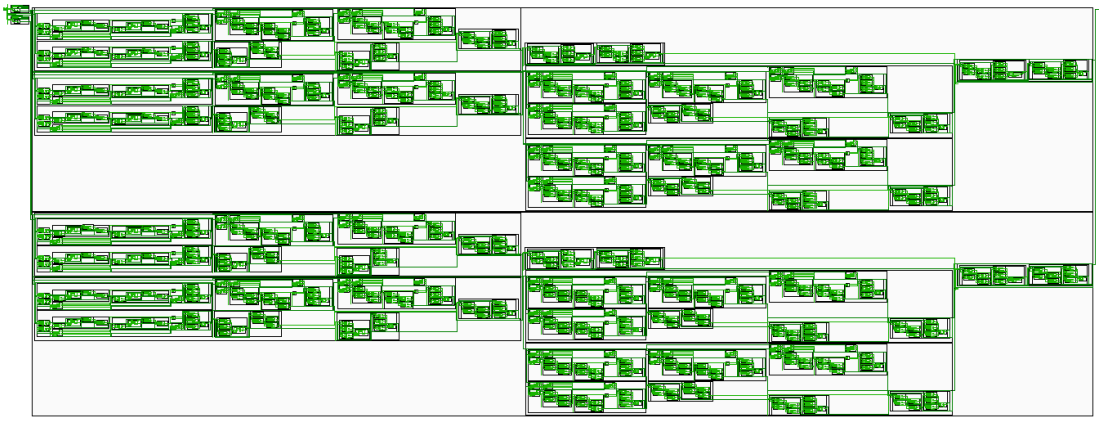


Fig. 4.24 Xilinx RTL Elaborated Schematic of FSK

4.4.3 FSK Synthesized Design

The synthesized implementation from Xilinx vivado of the PSK modulator has been presented in the Fig. 4.25 and the elaborated design obtained by expanding the design to the leaf cells has been presented in Fig. 4.26 It can be seen that entire connections between each and every module, input and output connections in the figure.

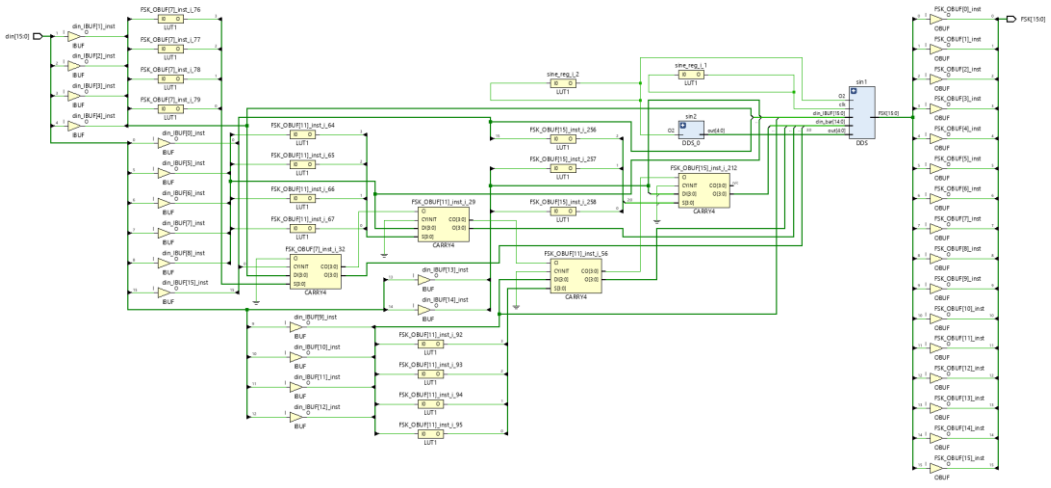


Fig. 4.25 Xilinx synthesized design of FSK

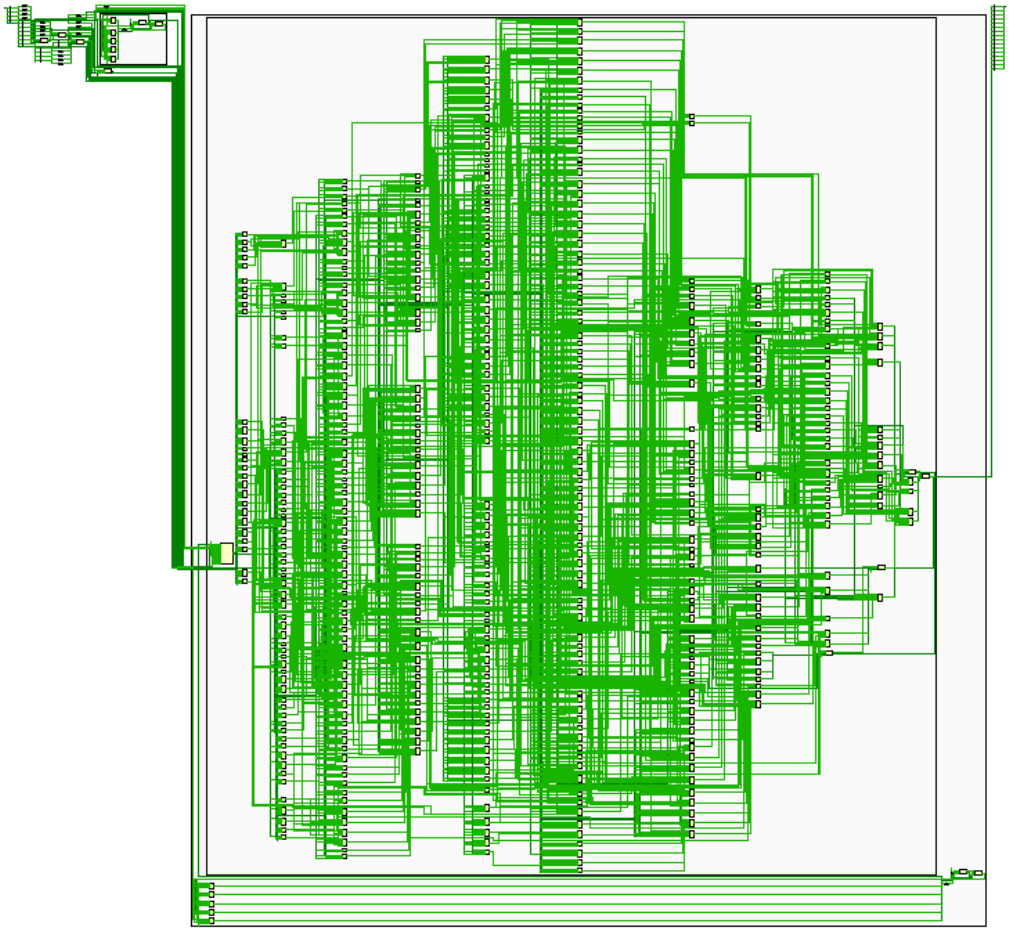


Fig. 4.26 Xilinx elaborated synthesized design of FSK

4.4.4 FSK Simulation Waveforms

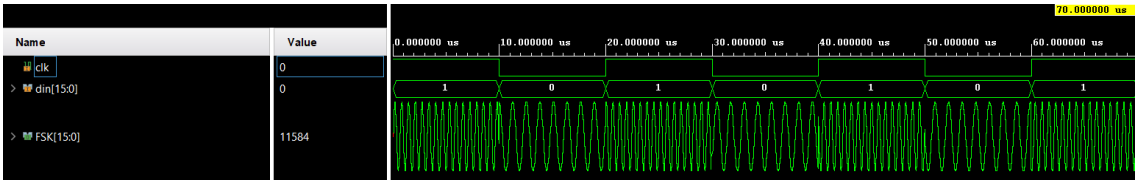


Fig. 4.27 Xilinx output of FSK simulations

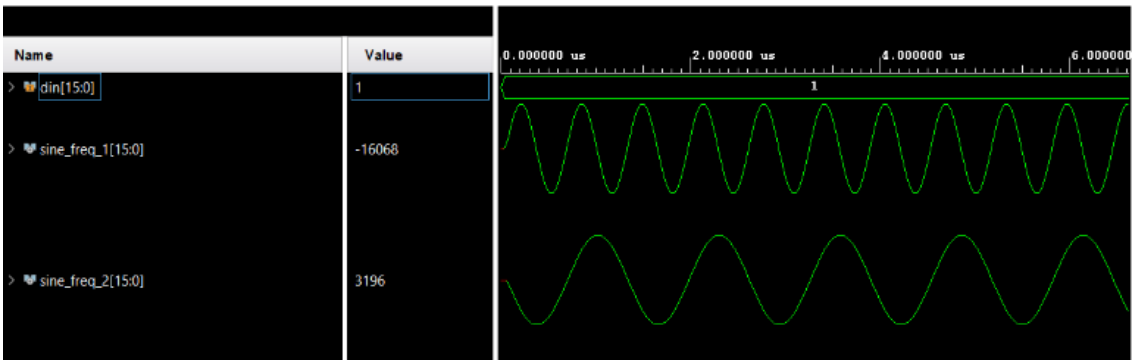


Fig. 4.28 Xilinx output of sinusoidal waveforms with different frequencies

The simulation results obtained from the Xilinx vivado are presented in the figures 4.27 and 4.28. The Fig. 4.28 shows two sinusoidal waveforms sine_freq_1 and sine_freq_2 which are having different frequencies and also the frequency of sine_freq_1 is ensured to be greater than the frequency of sine_freq_2.

From the Fig. 4.27 the basic functionality of the FSK modulation depicted in the previous section in the Fig. 4.22 can be verified. Whenever the input data signal is 1'b1 the output FSK waveform will be having the sine waveform of frequency sine_freq_1. When the input data signal is 1'b0 the output FSK waveform will be having the sine waveform of frequency sine_freq_2.

4.4.5 FSK Power Report

The Xilinx tool calculates power for the implemented design and generates a power report [3][4] as shown in Fig. 4.29

The power components in the power report are divided into static and dynamic. Static power is when the device is idle it still consumes some power This can be because of leakage currents flowing in the transistors. Dynamic power is when the device is powered on and various components like clocks, signals, logic, BRAM and Input output signals make up the dynamic component of power. The static and dynamic power sum up to Total on-chip power. A particular circuit can have an estimated power which is the design power budget. The power budget margin is the amount of power saved from the estimated budget. Junction temperature is the highest operating temperature of the

semiconductor. Thermal margin is the difference between the current and maximum temperature.

As can be seen the total power consumption is 0.293 W which is 1.707 W less than the defined power budget 2 W. The percentage of dynamic power is reported as 69% and static power at 31%.

The detailed power consumption is as follows, clocks taking up <0.001 W power which amounts to <1% of the total power consumed, signals taking 0.028W amounting to 14% of the total power consumption, logic taking up 0.039 W amounting to 19% of the total power consumption, BRAM consuming <0.001 W amounting to <1% total power consumption similar to the clocks, and it is interesting to note that the I/O amounts to 0.135 W which is 65% taking the majority of the dynamic component of the power

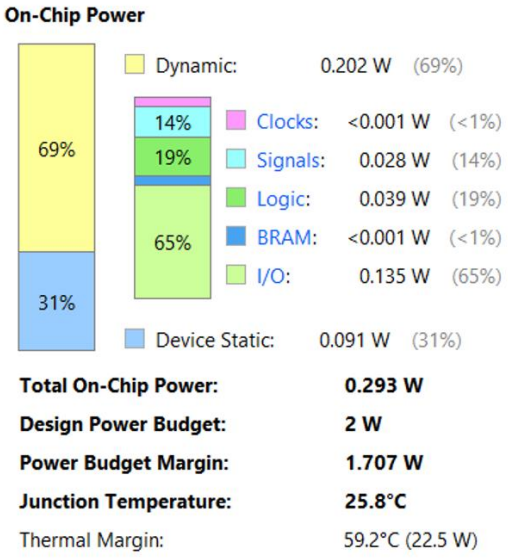


Fig. 4.29 Power report of FSK

CHAPTER 5

CONCLUSIONS AND RESULTS

A 32-bit multiplier using ancient Indian mathematics called Vedic mathematics has been implemented using Verilog coding in Xilinx Vivado Software tool. This was achieved by first creating a 2x2 multiplier using Urdhva Triyagbhyam Principle of vedic mathematics. The subsequent blocks of multipliers are created based on this. A 4x4, 8x8, 16x16 and finally 32x32 multipliers are created. The output of each was verified by running the Xilinx simulation.

A Sine and Cosine waveform using the concept of Direct Digital Synthesis has been studied and implemented to obtain the analog waveforms of these sinusoids.

A digital modulation technique has been implemented by using the vedic multiplier and sinusoid carrier signal. The design is synthesized to obtain gate level netlist and implemented to obtain the power consumption of the entire project. The on – chip power with static and dynamic power consumptions are clearly stated.

TABLE I

POWER COMPARISON OF ASK, PSK AND FSK

	ASK	PSK	FSK
Static power	0.091 W	0.091 W	0.091 W
Dynamic power	0.147 W	0.207 W	0.202 W
Clocks	<0.001 W	<0.001 W	<0.001 W
Signals	0.014 W	0.032 W	0.028 W
Logic	0.014 W	0.040 W	0.039 W
BRAM	< 0.001 W	< 0.001 W	< 0.001 W
I/O	0.118 W	0.135 W	0.135 W
Total on chip power	0.238 W	0.299 W	0.293 W

Design power budget	2 W	2 W	2 W
Power budget margin	1.762 W	1.701 W	1.707 W
Junction temperature	25.6 ºC	25.8 ºC	25.8 ºC
Thermal margin	59.4 ºC	59.2 ºC	59.2 ºC

Future Scope

This project has wide level key concepts for implementing different projects based on the same idea that has been presented throughout this work. The future scope for this project has been detailed below:

- This work uses Ripple Carry Adder for generating the Multiplier design. For further scope RCA can be replaced with other efficient and low power consuming adders such as Carry Look Ahead Adder to further reduce the power consumption and make this design more efficient for low power applications.
- Various other Vedic algorithms such as Nikhilam can be utilised for better results in the mathematic and arithmetic operations.
- Area optimization of the DDS algorithm can be carried out as can be seen from the synthesised and RTL implemented designs, most of the chip area is being consumed by LUT in the DDS module.

REFERENCES

[1] P. N. MURTHY, "Design and Simulation of Reconfigurable Digital Modulator using Vivado System Generator," 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), 2019, pp. 1366-1370, doi: 10.1109/ICECA.2019.8821965.

[2] N. Pallavi, P. Anjaneyulu, P. B. Reddy, V. Mahendra and R. Karthik, "Design and implementation of linear frequency modulated waveform using DDS and FPGA," 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), 2017, pp. 237-241, doi: 10.1109/ICECA.2017.8212806.

[3] B. R. Jammu, H. K. Botcha, A. V. Sowjanya and N. Bodasingi, "FPGA implementation of BASK-BFSK-BPSK-DPSK digital modulators using system generator," 2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT), 2017, pp. 1-5, doi: 10.1109/ICCPCT.2017.8074203.

[4] A. Sharma, S. Majumdar, A. Naugarhiya, B. Acharya, S. Majumder and S. Verma, "VERILOG based simulation of ASK, FSK, PSK, QPSK digital modulation techniques," 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2017, pp. 403-408, doi: 10.1109/I-SMAC.2017.8058380.

[5] Du Weitao and Yang Zhanxin, "Design of area efficient DDS IP core generator," 2015 12th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), 2015, pp. 443-446, doi: 10.1109/ICEMI.2015.7494230.

[6] Yogita Bansal , Charu Madhu , Pardeep Kaur, “HIGH SPEED VEDIC MULTIPLIER DESIGNS”, Proceedings of 2014 RAECS UIET Punjab University Chandigarh.

[7] Yeshwant Deodhe, Sandeep Kakde, Rushikesh Deshmukh,”Design and Implementation of 8-bit Vedic Multiplier using CMOS Logic”, The First International Conference on Machine Intelligence and Research Advancement-ICMIRA-2013 Katra, India, 978-0-7695-5013-8/13 ,2013 IEEE

[8] A. K. Mehta, M. Gupta, V. Jain and S. Kumar, "High performance vedic BCD multiplier and modified binary to BCD converter," 2013 Annual IEEE India Conference (INDICON), 2013, pp. 1-6, doi: 10.1109/INDCON.2013.6725995.

[9] F. Quadri and A. D. Tete, "FPGA implementation of digital modulation techniques," 2013 International Conference on Communication and Signal Processing, 2013, pp. 913-917, doi: 10.1109/iccsp.2013.6577189.

[10] C. Erdoğan, I. Myderrizi and S. Minaei, "FPGA Implementation of BASK-BFSK-BPSK Digital Modulators [Testing Ourselves]," in IEEE Antennas and Propagation Magazine, vol. 54, no. 2, pp. 262-269, April 2012, doi: 10.1109/MAP.2012.6230771