DIGITAL IMPLEMENTATION OF TRIGONOMETRIC FUNCTIONS VIA FPGA DEVICES

Shakir Salman Ahmad

DECEMBER, 2017

DIGITAL IMPLEMENTATION OF TRIGONOMETRIC FUNCTIONS VIA FPGA DEVICES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF
ÇANKAYA UNIVERSITY

BY
SHAKIR SALMAN AHMAD

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER
IN
THE DEPARTMENT OF
ELECTRONIC AND COMMUNICATION ENGINEERING

DECEMBER, 2017

Title of the Thesis : **DIGITAL IMPLEMENTATION OF TRIGONOMETRIC FUNCTIONS VIA FPGA DEVICES**

Submitted by **SHAKIR SALMAN AHMAD**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

[illegible]

Prof. Dr. Can ÇOĞUN
Director

I certify that this thesis satisfies all the requirements by way of thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Sıtkı Kemal İDER

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, by way of thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Orhan GAZİ

Supervisor

Examination Date: 19.12.2017

Examining Committee Members

Assoc. Prof. Dr. Orhan GAZİ	(Çankaya Univ.)	[illegible]
Assist. Prof. Dr. Javad RAHABI	(Turkish Aeronautical Association Univ.)	[illegible]
Assist. Prof. Dr. Göker ŞENER	(Çankaya Univ.)	[illegible]

STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, by way of required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name	: Shakir Salman Ahmad
Signature	: [illegible]
Date	: 19.12.2017

ABSTRACT

DIGITAL IMPLEMENTATION OF TRIGONOMETRIC FUNCTIONS VIA FPGA DEVICES

AHMAD, Shakir Salman

M.Sc., Department of Electronics and Communication Engineering

Supervisor: Assoc. Prof. Dr. Orhan GAZİ

December. 2017, 52 Pages

As the technology improves, the importance of dynamic hardware design gains more and more attention in the scientific world. FPGA devices are used for digital circuit design purposes, and recently a significant improvement has been observed in FPGA technology. Many communication systems employed in wireless and satellite systems employ FPGAs.

Trigonometric functions, such as sine or cosine, are vital components of communication systems. With the invention of coordinate rotation digital computer, i.e., CORDIC in short, algorithm in 1956, it became possible to generate the trigonometric functions in digital devices, such as calculators, microcontrollers, and electronic chips etc.

In this thesis work, we first implement the CORDIC algorithm in MATLAB environment, and analyze the iteration number of the algorithm considering the accuracy of the calculated sine or cosine value. Next, we implement the CORDIC algorithm in FPGA environment using the VHDL programming language. For this purpose, we used the FGPA board involving SPARTAN-3 FPGA chip produced by the XILINX company. Later, we followed an alternative approach for the generation of sine signal. For this purpose, we generated 1Hz sine signal in MATLAB and considering the ratio among sine samples, we generated an integer sequence keeping approximately the same ratio among samples, and using this integer sequence we generated sine signal in FPGA platform and using the D/A converter, we observed the generated sine signal on OSCILLOSCOPE screen. With this alternative approach, it became possible to generate sine signal with any frequency and much less hardware complexity. To generate different sine signals with different frequencies, we divide the clock of the FPGA device by a desired amount and use it while sending the sine samples to the output port.

Keywords: Cordic algorithm, FPGA, VHDL, and Cosine function

ÖZET

FPGA CİFAZLARI İLE TRİGONOMETRİK FONKSİYONLARIN DİJİTAL GERÇEKLEŞTİRİMİ

AHMAD , Shakir Salman

YL, Elektronik ve Haberleşme Mühendisliği Bölümü

Danışman: Doçent Dr. Orhan GAZİ

Aralık 2017, 52 Sayfa

Teknolojinin gelişimi ile birlikte dinamik donanım yapılarının tasarlanması gün geçtikçe daha da önem kazanmaktadır. FPGA cihazları dinamik donanımlar tasarlanmak için kullanılan ve gittikçe dünyada popülaritesi artan elektronik ünitelerdir. FPGA cihazları günümüzde birçok kablosuz ve uydu iletişim sistemlerinde kullanılmaktadır.

Sinüs ve kosinüs gibi trigonometrik fonksiyonlar iletişim sistemlerinin vazgeçilmez unsurlarıdırlar. CORDIC algoritmasının 1956 yılında icadı ile trigonometrik fonksiyonların sayısal elektronik cihazlarındaki gerçekleştirimlerinin önü açılmıştır. Bugün günümüzde kullanılan birçok elektronik cihaz, mesela hesap makinesi, mikrokontroller ve elektronik yongalar CORDIC algoritmasını kullanmaktadır.

Bu tez çalışmasında ilk olarak CORDIC algoritmasının MATLAB ortamında gerçekleştirimini yapıyoruz ve algoritmada kullanılan yineleme sayısı ile elde edilen sinüsün veya kosinüsün gerçek değerinden ne kadar sapma gösterdiğini inceliyoruz. Daha sonra ise algoritmayı FPGA ortamında VHDL programlama dili kullanarak gerçekleştiriyoruz. Bunun için XILINX firmasının üretmiş olduğu SPARTAN-3 yonga içeren FPGA platformu kullanıyoruz. Tezin üçüncü çalışmasında sinüs veya kosinüs sinyallerini üretmek için daha basit bir yol takip ediyoruz. Bunun için frekansı 1Hz olan sinüs veya kosinüs sinyallerinin bir periyodunu MATLAB ortamında üretiyoruz. Daha sonra üretilen sayılar arasındaki oranı göz önüne alarak bir tam sayı dizini oluşturuyoruz. Oluşturulan tam sayı dizinini FPGA ortamında taşıyarak VHDL dili ile FPGA cihazının saat darbe sayısı istediğimiz gibi ayarlayarak herhangi bir frekansa sahip olan sinüs veya kosinüs sinyalini üretiyoruz. Kullandığımız bu yöntem CORDIC algoritmasına göre daha basit ve donanım karmaşıklığı çok daha az olmaktadır.

Anahtar kelimeler: Cordic Algoritması, FPGA, VHDL, Kosinüs fonksiyon.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Assoc. Prof. Dr. Orhan GAZİ for his supervision, special guidance, suggestions, and encouragement through the development of this thesis.

It is a pleasure to express my special thanks to my family for their valuable support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

Introduction to ISE Suite

1.1. Definition and Background

''Xilinx ISE (**I**ntegrated **S**ynthesis **E**nvironment)" is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different motivations, and configure the target device with the programmer.

Xilinx ISE is a software development platform for FPGA supplied from Xilinx, and it was tightly-coupled to the architecture of given chips, and cannot be used with FPGA products from other retailers. The Xilinx ISE is fundamentally utilized for circuit design and synthesis, while ISIM or the ModelSim logic simulator is utilized for framework level analysis. Different parts dispatched with the Xilinx ISE include the a Software Development Kit (SDK) , Embedded Development Kit (EDK), and ChipScope Pro.

Since 2012, Xilinx ISE has been stopped for VDS, that aids the same roles by way of ISE with extra highlights for framework on a chip progression. Xilinx released the last version of ISE in October 2013 (version 14.7), and states that "ISE" has moved into the supporting period of its item life cycle, and there are not any more arranged ISE out.

1.2. Overview of ISE

All parts of design flow are controlled by ISE. Through the interface of Project Navigator (PG)ö we can get the majority of the tools implementation design and entry design. By this way, we can get files and documents related with our project.

1.2.1. Project Navigator Interface

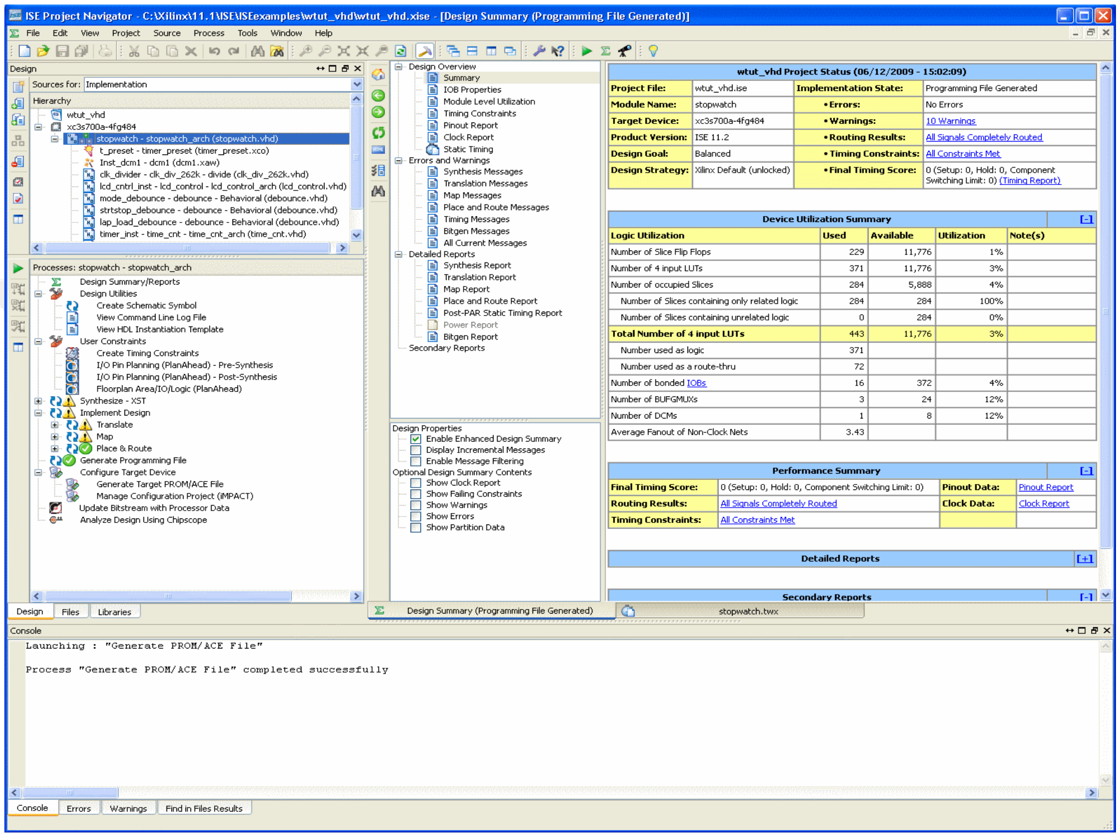
Practically the Interface of Project Navigator is classified into four types of panel sub-windows as explained in Figure 1. On the top-left of screen we see file, design and libraries panel are displayed and they lead to source file of the project. At the lower part of the Project Navigator we have the console, warnings and errors which show status messages' warnings and errors. We see the multi-document- interface (MDI) window indicate to workspace. We can use it to show schematics, text files, design reports, and simulation of waveforms. Every window might be resized, undocked from PN, moved to another area inside the fundamental PG window, tiled, layered, or shut. Boards might be opened or shut by utilizing the View then Panels then menu choices.

1.3. Design Panel

1.3.1. Sources View

We can see the name of project by the sources view in order to display the name of project documents of user the required device and source of design files related with design view choices. The Design View ("Sources for") drop-down rundown at the highest point of the Sources tab enables you to see just those source documents related with the design view we choose, for example, Synthesis/Implementation or Simulation.

Every file has an icon referring to the type file (HDL, text, core, schematic). In order to know the source type and their related icons we can use help in ISE™ . For this purpose we can chose help and then select help contents. And search about “source file type” after click on index tab. The sign ‘+’on the icon of the file refers to it has more than one level of hierarchy and we can open these levels. We can expand and show the contains of the file by click on the sign '+'and the file will be opened by indicating on the file name then double click.



ISE Project Navigator - C:\Xilinx\11.1\ISE\ISEexamples\wtut_vhd\wtut_vhd.xise - [Design Summary (Programming File Generated)]

Sources for: Implementation

Hierarchy
- wtut_vhd
- xc3s700a-4fg484
  - stopwatch - stopwatch_arch (stopwatch.vhd)
    - t_preset - timer_preset (timer_preset.xco)
    - Inst_dcm1 - dcm1 (dcm1.xaw)
    - clk_divider - clk_div_262k - divide (clk_div_262k.vhd)
    - lcd_cntrl_inst - lcd_control - lcd_control_arch (lcd_control.vhd)
    - mode_debounce - debounce - Behavioral (debounce.vhd)
    - strtstop_debounce - debounce - Behavioral (debounce.vhd)
    - lap_load_debounce - debounce - Behavioral (debounce.vhd)
    - timer_inst - time_cnt - time_cnt_arch (time_cnt.vhd)

Processes: stopwatch - stopwatch_arch
- Design Summary/Reports
- Design Utilities
  - Create Schematic Symbol
  - View Command Line Log File
  - View HDL Instantiation Template
- User Constraints
  - Create Timing Constraints
  - I/O Pin Planning (PlanAhead) - Pre-Synthesis
  - I/O Pin Planning (PlanAhead) - Post-Synthesis
  - Floorplan Area/IO/Logic (PlanAhead)
- Synthesize - XST
- Implement Design
  - Translate
  - Map
  - Place & Route
- Generate Programming File
- Configure Target Device
  - Generate Target PROM/ACE File
  - Manage Configuration Project (iMPACT)
- Update Bitstream with Processor Data
- Analyze Design Using Chipscope

- Design Overview
  - Summary
  - IOB Properties
  - Module Level Utilization
  - Timing Constraints
  - Pinout Report
  - Clock Report
  - Static Timing
- Errors and Warnings
  - Synthesis Messages
  - Translation Messages
  - Map Messages
  - Place and Route Messages
  - Timing Messages
  - Bitgen Messages
  - All Current Messages
- Detailed Reports
  - Synthesis Report
  - Translation Report
  - Map Report
  - Place and Route Report
  - Post-PAR Static Timing Report
  - Power Report
  - Bitgen Report
- Secondary Reports

Design Properties
- [✓] Enable Enhanced Design Summary
- [ ] Display Incremental Messages
- [ ] Enable Message Filtering

Optional Design Summary Contents
- [ ] Show Clock Report
- [ ] Show Failing Constraints
- [ ] Show Warnings
- [ ] Show Errors
- [ ] Show Partition Data

wtut_vhd Project Status (06/12/2009 - 15:02:09)

Project File:	wtut_vhd.ise	**Implementation State:**	Programming File Generated
Module Name:	stopwatch	**• Errors:**	No Errors
Target Device:	xc3s700a-4fg484	**• Warnings:**	10 Warnings
Product Version:	ISE 11.2	**• Routing Results:**	All Signals Completely Routed
Design Goal:	Balanced	**• Timing Constraints:**	All Constraints Met
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	0 (Setup: 0, Hold: 0, Component Switching Limit: 0) (Timing Report)

Device Utilization Summary [-]

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	229	11,776	1%	
Number of 4 input LUTs	371	11,776	3%	
Number of occupied Slices	284	5,888	4%	
Number of Slices containing only related logic	284	284	100%	
Number of Slices containing unrelated logic	0	284	0%	
Total Number of 4 input LUTs	443	11,776	3%	
Number used as logic	371			
Number used as a route-thru	72			
Number of bonded IOBs	16	372	4%	
Number of BUFGMUXs	3	24	12%	
Number of DCMs	1	8	12%	
Average Fanout of Non-Clock Nets	3.43			

Performance Summary [-]

Final Timing Score:	0 (Setup: 0, Hold: 0, Component Switching Limit: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports [+]

Secondary Reports [-]

Design Summary (Programming File Generated) | stopwatch.twx

Console

```
Launching : "Generate PROM/ACE File"

Process "Generate PROM/ACE File" completed successfully
```

Figure 1: Project Navigator

1.3.2. Processes View

The Processes view is important context and we can change its state depending on what we choose of the source type in sources part and top-level of source in our project. By this way, we choose the necessary function for setting, in order to run our design and analyzing. By using processes, we get several functions such as:

1-Report or design summary.

We can access the report of design and results data and message from summary design.

2- Utilities of design.

We can access from utilities of design create schematic symbol, view command log file and view HDL instantiation template.

3- Users constraints.

We get several branches such as: create timing constraints, I/O pin planning (planned Ahead)-pre synthesis, I/O pin planning (planned Ahead)-post synthesis and floor plan areal /IO/logic (planned Ahead).

4- Syntheses

Synthesis contains many parts such as: view of RTL (register transfer level) schematic, view technology schematic, check syntax and great post- synthesis simulation model.

5- Implement design.

Implement of design part contains several levels

A- Translate.

Translate contains generate post-translate simulation model.

B- Map

Map contains several levels: Generate post-map static timing, analyze post-map static timing, manually place and route (FPGA Editor) and generate post-map simulation model.

C- Place and Route.

Place and Route panel has several types such as: Generate post-place and route static timing, analyze post-place and route static timing, analyze timing/floor plan design (plan Ahead), view/edit routed design (FPGA Editor), Analyze power distribution (x power Analyzer), Generate text power report, generate post-place and route simulation model, Generate IBIS model, view IBIS model, Back-annotate pin locations, view locked pin constraints and generating programming file.

D-Configure target device.

Configure target device contains: generate target PROM/ACE file, manage configuration project (IMACT) and analyze design using chip scope.

1.3.3. Files Panel

We get a list of flat sortable source files in our project from the files panel. We can have sorted these file in view by any columns, we can modify and view properties to every file by selecting the option source properties after indicating the file and right clicking on it.

1.3.4. Libraries Panel

HDL libraries and related source files of HDL can be managed by the tab of libraries we can get, show the libraries and edit their libraries related with sources.

1.3.5. Console Panel

From project navigator and from run process, the Console panel gives us all output in standard state. We see error in symbol (x) in red color and yellow color for warning in symbol (i) with information about warning by way of a message.

1.3.6. Errors Panel

Only messages of error have been displayed here. Other massages are neglected out consol.

1.3.7. Warnings Panel

If we have error of synthesis or message in console panel of warning or errors to location of error in HDL file. We select the messages of errors or by right click and chose the option go to source then HDL file will open and the indicator transfer to line where the errors lies. Only messages of warning displayed in this panel. Other console messages are neglected out. We can go to http://www.xilinx.com/support website to find answer to our problems and errors

1.3.8. Workspace

The Workspace refers to the place of design editors with viewers with analysis tools when it is opened. ISE Text Editor, Schematic Editor, Timing Constraint Editor, Design Summary & Report Viewer, RTL and Technology Viewers, and Timing Analyzer are included. Other tools such by way of Plan Ahead for I/O planning and floor planning, ISE Simulator (ISim), 3rd party Text Editors, XPower Analyzer, and iMPACT open in separate windows outside the main PN environment when invoked.

1.3.9. Report Viewer & Design Summary

We get a summary of main design data from design summary also get all the reports and message from implementation and synthesis tools and gives important information about our project and we can overview information summary of device utilization performance data collected from place and route report summary and constraint information from all reports individual links.

1.4. ISE Console window

The ISE console window provides narrative from the design processes by way of they operate. Most of the scripts that the ISE runs and messages that are emitted by the various processes can be gotten by way of a running display in the Console window—seeing it while a design is being treated can be quite informative. Infrequently, some message will flash by and trigger an insight into a design's characteristics or defects. When the processing of a design has ended or finished, the "Errors" or "Warnings" tab will filter all of the process dialog down to the important details.

1.5. ISE Reports/Edit window

This shows a viewer for the numerous reports produced in the Processes window. It also becomes a very useful syntax-coloring editor for Verilog and VHDL sources in the Sources window.

CHAPTER TWO

Fundamental of VHDL and FPGA

2.1. Definition and Background on VHDL

VHDL means VHSIC Hardware Description Language. VHSIC is itself acronym to "Very High Speed Integrated Circuits" furnished through United States Department of Defense in 1980s. It is a language to a hardware description. It is a language to a system or electronic circuit behavioral description. It is the language for circuit simulation in addition to circuit synthesis [2].

Its early version is VHDL 87, next developed to a so-called VHDL 93. It is the first hardware description language and innovative standardized via Institute of Electrical and Electronics Engineers with IEEE 1076 standard. Furthermore, IEEE 1164, has been added to present a multi-valued logic system.

A major encouragement inspiration for utilizing VHDL (or its competitor, Verilog) is that VHDLs are a standard, technology/vendor standalone language, and is to portable and reusable. VHDL have two topmost instantaneous applications that are: 1) to area of ASICs (Application Specific Integrated Circuits) 2) To area of Programmable Logic Devices that comprising FPGAs—Field Programmable Gate Arrays and CPLDs—Complex Programmable Logic Devices. VHDL codes are written for two purposes: 1) it used to manufacture a chip of ASIC or to carry out circuit inside a programmable device such as Xilinx, Atmel and Altera. Presently, numerous multifaceted marketable chips have been designed utilizing this approach, e.g., microcontrollers.

2.1.1. Design Flow in VHDL

As explained previously, VHDL is mainly used for synthesis in FPGA or in PLD or in ASIC. Design flow of VHDL is given in Figure 2 [2]. VHDL codes are saved in a file with extension .vhd and they have similar name as that of ENTITY's name. Compilation is primary stage of a synthesis process [2].

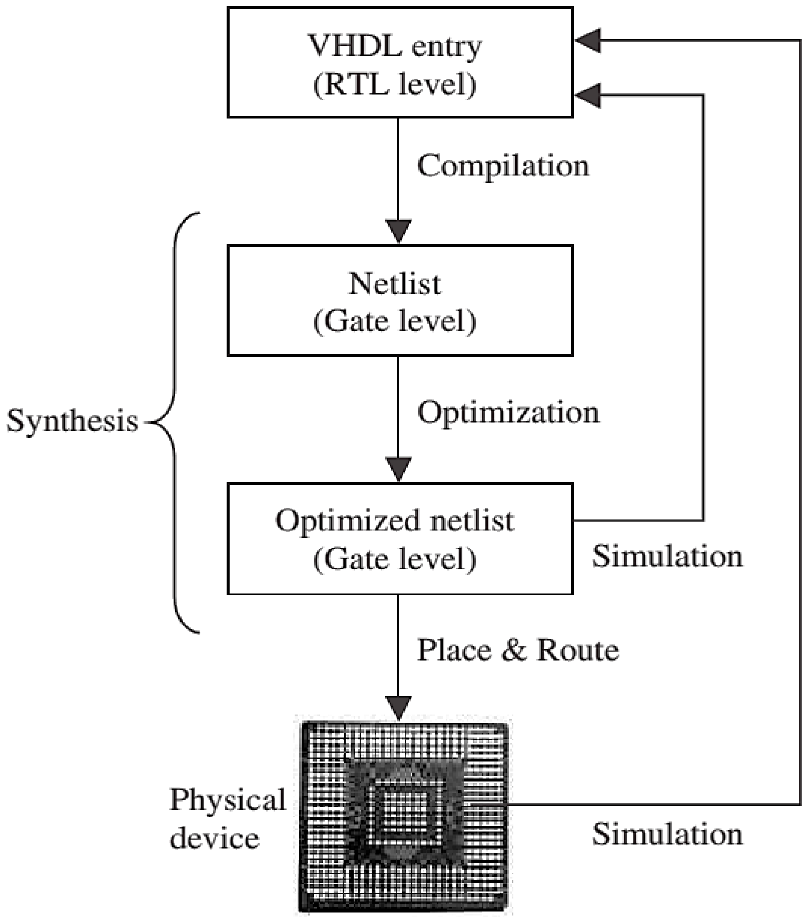


Figure 2 Design Flow Summary [2].

It is a high-level VHDL language that defines every circuit at Register Transfer Level (RTL) and feed it in a net list at gate level. Then, optimization is second step that is done on gate-level. At this level, simulation of design is implemented. Lastly, it possible to generate a physical layout to a FPGA and PLD chips or generate covers to ASIC through place and-route (fitter) software

2.1.2. Code Structure in VHDL

VHDL code is comprised from three components [2]:

1. LIBRARY declarations: Covers a list of libraries that is utilized in design. Such as: ieee, std, work, etc.
2. ENTITY: Indicate   I/O pins of a circuit.
3. ARCHITECTURE: Comprises VHDL code that defines the way circuit should function.

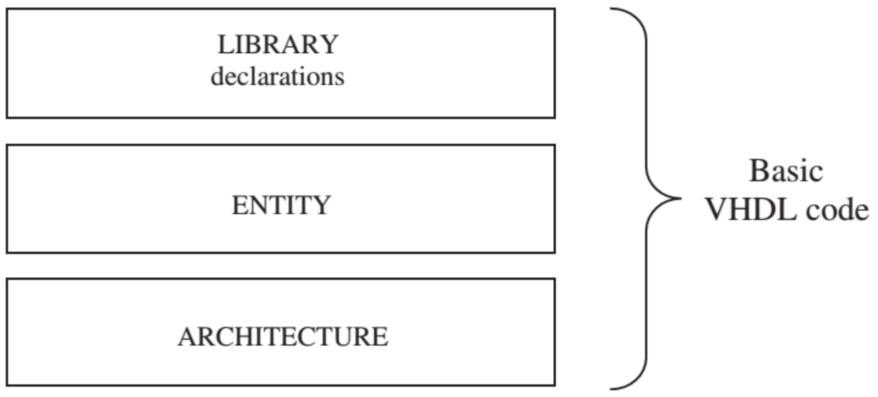


Figure 3. VHDL Code Components [2]

To announce a LIBRARY and to make it visible to design; two lines of code required, one is the name of   library, and the other indicating the name of the package to be used.

```
LIBRARY library_name;
USE library_name.package_name.package_parts;
```

An ENTITY is a list containing all input and output pins named as PORTS. Its syntax is given below.

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

ARCHITECTURE is an explanation that according to it a circuit should function. Its syntax is following:

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

2.2. Definition and Background on FPGA

VLSI circuits have been categorized as shown in Fig. 2.3. FPGA member of a class of devices called field-programmable logic (FPL) [3]. FPLs describe programmable devices covering repetitive fields of elements and small logic blocks. It is seen that an FPGA is an ASIC technology of FPGAs application-specific ICs. Moreover, a design of a classic ASIC needs extra semiconductor dispensation stages essential to an FPL. Extra stages deliver higher-order ASICs in addition of energy consumption benefit but also with high Non-Recurring Engineering (NRE) expenses.

The architecture of FPGA device is given in Fig. 5. Fundamental logic blocks include 4 to 5-bit input tables, 1 or 2-bit output. Routing channel selections varies from short too long. Programmable I/O block with flip-flops is connected to physical boundary of device.

Traditionally, FPGA is not energy efficient, slower and usually given not much functional compared to their fixed ASIC counterparts. Nowadays, FPGAs by Altera

Stratix 5 have come to opposing like ASIC or Xilinx Virtex-7 and ASSP keys by giving considerably less power usage, higher speed, less resources cost, insignificant implementation real-estate, and more potentials to reconfiguration 'on- -fly'. Previously, a project is comprised 6 to 10 ASICs, now same project can be implemented utilizing only one FPGA [3].

2.2.1. FPGA Application

Any computable problem can be solved through FPGA. This is slightly established via fact of FPGA that is utilized to implement a soft microprocessor.

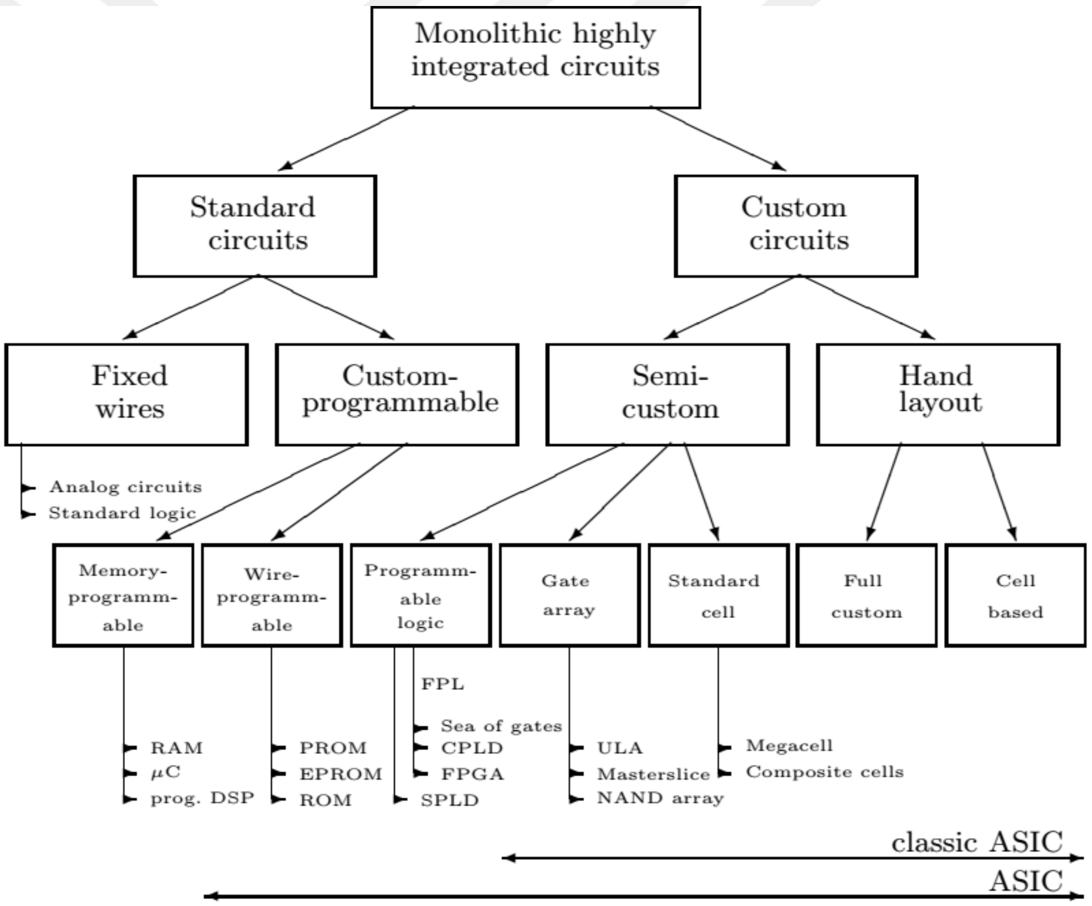


Figure 4 Classification of VLSI circuits [3]

FPGAs initially started as a competition to CPLDs to carry out glue logic to PCBs considering their size, abilities, and speed improved. The created multipliers inside FPGA architectures in late 1990s,

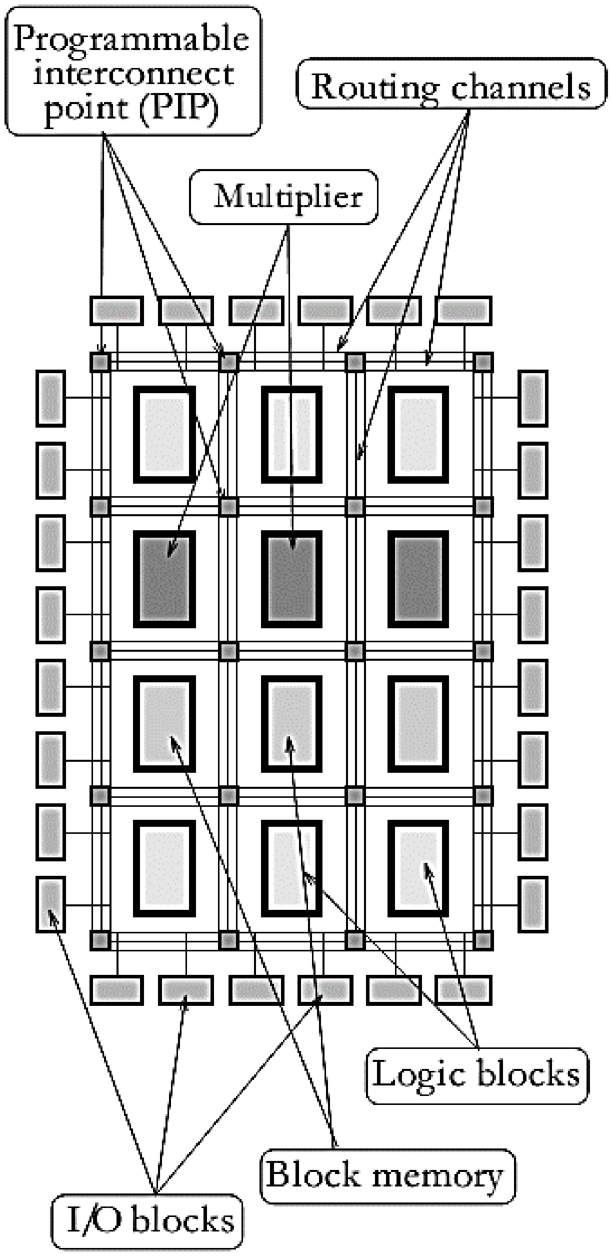


Figure 5 A field-programmable gate array architecture [3]

One more trend in utilizing of FPGAs is hardware acceleration is that it is possible to utilize FPGA to speed up given parts of algorithm and dividend part of calculation between FPGA and a generic processor.

Usually, FPGAs reserved to exact vertical applications where size of manufacture is not large. To these small size applications, premium that companies pay in hardware costs per unit to a programmable chips not less affordable compared to development resources consumed on making an ASIC to a small-size application. Nowadays, fresh cost and performance dynamics broadened variety of feasible applications.

2.2.2. Advantages of FPGA

FPGAs have turned out to be extremely prominent in current years due to its abundance of benefits. FPGA enable parallel processing operations do be done reducing the computation overhead of complex algorithms. Programming time of FPGA can be considered small relative to its ASIC counterparts. Re-programmability property of the FPGAs provide us with great flexibility of circuit design.

NRE cost is zero: NRE denotes to one-time cost of investigating, developing, designing and testing a new product. Since FPGAs are re-programmable and can be utilized without any loss of quality every time, NRE cost is not present. This significantly decreases initial cost of manufacturing ICs since program is applied and verified on FPGAs free of cost. Due to look up table which used by FPGA technology, the execution time will be much less compared to ASIC technology. FPGAs are produced in great numbers, considering their benefits, they can be accepted as cheap in price and they are too friendly to the designer. In addition, power consumption is too less.

CHAPTER THREE

Function Calculation using CORDIC

3.1. Trigonometric functions

Calculation of sine and cosine can be achieved via CORDIC. We start our discussion with those two simple functions to verify simplification of CORDIC. Differences between CORDIC classes not complicated in derivative and concept point of view, hence considering one or two classes can lead to other classes' derivation and calculation. Thereto, considering those classes will serve to provide details to one class of function and extension to another class.

Sine and cosine calculations performed in rotation scenario and named by way of rotation mode. Rotating a (technically) is unit-length vector over any predetermined angle. Let us assume to calculate $cos\,\theta$ and $sin\,\theta$, to give angle θ. If we consider a vector $(X_0, Y_0) \triangleq (1\,,0)$, rotate it on circle $X^2 + Y^2 = 1$, over a sequence of positive angles, $\theta_0\,, \theta_1, \dots, \theta_n$, such that $\theta_0 + \theta_1 + \cdots, \theta_n = \theta$, and finish with vector $(X_{n+1}\,, Y_{n+1})$, then $Y_{n+1} = sin\,\theta$ and $X_{n+1} = cos$. Then, let us take look at rotation technique in CORDIC in rotation mode.

Figure 3.1 gives steps of rotation, by an angle θ_i, of unit-length vector with coordinates (X_i, Y_i) at an angle φ from x axis. Coordinates, (X^*_{i+1}, Y^*_{i+1}), of new vector given as:

$$X^*_{i+1} = cos(\theta_i + \varphi)$$

$$X^*_{i+1} = \cos\varphi \cos(\theta_i) - \sin\varphi \sin(\theta_i)$$
$$X^*_{i+1} = X_i \cos(\theta_i) - Y_i \sin(\theta_i)$$
$$X^*_{i+1} = (X_i - Y_i \tan(\theta_i)) \cos(\theta_i)$$

(1)

$$Y^*_{i+1} = \sin(\theta_i + \varphi)$$
$$Y^*_{i+1} = \sin(\varphi) \cos(\theta_i) - \cos(\varphi) \sin(\theta_i)$$
$$Y^*_{i+1} = X_i \cos(\theta_i) - Y_i \sin(\theta_i)$$
$$Y^*_{i+1} = (X_i - Y_i \tan(\theta_i)) \cos(\theta_i)$$

(2)

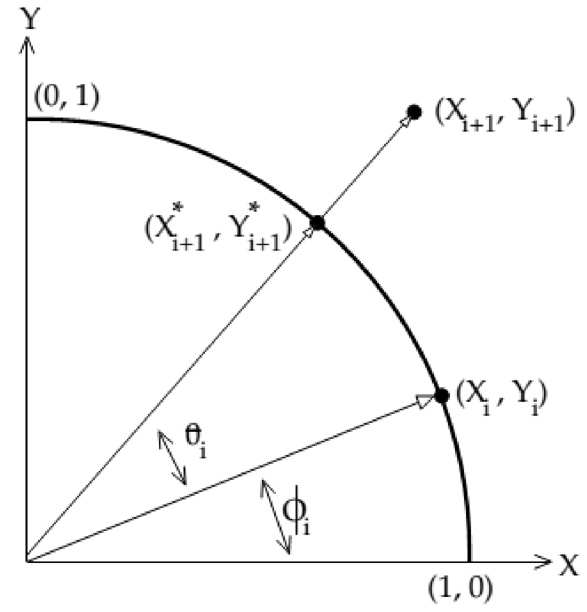


Figure 3.1 Rotation mode CORDIC algorithm.

In CORDIC, rotation of (X_i, Y_i) to (X^*_{i+1}, Y^*_{i+1}), is shown in Figure 3.1 and this rotation can mathematically be expressed as follows

$$X_{i+1} = X_i - Y_i\, tan\,(\theta_i)$$

$$Y_{i+1} = X_i - Y_i\, tan\,(\theta_i))$$

(3)

As we shall see below, utilization of a pseudo-rotation, in place of an exact rotation can reduce complexity and hardware implementation of the algorithm that is most important feature of CORDIC.

A comparison of Equations 1–2 and Equation 3 demonstrations that vector(X_{i+1}, Y_{i+1}), is lengthier than vector (X^*_{i+1}, Y^*_{i+1}), by a factor of$(1/\,cos\,\theta_i)$. Consequently, to obtain $sin\,\theta$ and $cos\,\theta$ at end of final rotations of algorithm, it is important to multiply (scale) each X_i and Y_i by this factor. Then, if magnitudes of rotation angles are constant and total number of rotations also constant, nevertheless of given angle, θ and thus, same situation with recurrences above if θ_i suitably selected then multiplying of scaling factors is also constant. Thereto, rather than scaling by $(1/\,cos\,\theta_i)$ at each given step, i, to $n+1$ iterations, it is adequate to simply multiply (scale) last out come by factor

$$k = \prod_{i=0}^{n} \frac{1}{\cos\theta_i}\,,$$

$$k = \prod_{i=0}^{n} \sqrt{1+\tan(\theta_i)}\,,$$

(4)

Actually, the last process, i.e., scaling by K can be avoided by setting X_0 to the initial value$1/K$. This initial scaling in X subsequently introduces a similar scaling in Y .

Rotation by angle θ_i can be converted into shifting by carefully selecting the rotations angles as will be explained now. Let us select $\theta_i = \tan^{-1} 2^{-i}$, this yields

$$X_{i+1} = X_i - Y_i \; 2^{-i}$$
$$Y_{i+1} = X_i - Y_i \, 2^{-i}$$

(5)

where 2^{-i} signifies a simple shift to right by i bits, and the constant term is

$$k = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}} \, .$$

(6)

Last issue in this presentation is convergence. We might dive into points of interest of that beneath, however we make some preparatory comments here with a specific end goal to finish definition of algorithm.

In order to choose whether to add or subtract at the current iteration in the algorithm, total rotation angle so far is subtracted from θ. If difference is negative, then an addition takes place; and if it is positive, then a subtraction takes place. Alternatively, we can look at the difference between the accumulated angle and target angle. If last rotation was to forward (backward), then next one can be backward (toward). In algorithm, the difference is calculated incrementally by way of $\theta_0 \pm \theta_1, \theta_0 \pm \theta_1 \pm \theta_2, \theta_0 \pm \theta_1 \pm \theta_2 \pm \theta_3,$ etc. and preceding actions thereto correspond to making adjustments according to “how much farther we have to go”. Angle residual is computed in a third variable Z_i.

To summarize, recurrences to computation of sine and cosine are done as

$$X_0 = \frac{1}{k} \, , Y_0 = 0 \, , Z_0 = \theta \; and \; \theta_i = \tan^{-1} 2^{-i}$$

$$X_{i+1} = X_i - s_i \, 2^{-i} \, Y_i$$

$$Y_{i+1} = Y_i + s_i \, 2^{-i} \, X_i$$

$$Z_{i+1} = Z_i - s_i \, \theta_i$$

$$s_i = \begin{cases} 1 \ \ if\, Z_i \ \geq 0 \\ -1 \ \ if\, Z_i \ \leq 0 \end{cases}$$

3.2. Numerical Example for CORDIC

In this section, a numerical example is given about rotation mode of CORDIC algorithm. We are going to evaluate sin and cos of the angle 0.735 radians.

First we need to find and set initial values of algorithms as:

$$Y_1 = 0\,, \qquad Z_1 = 30^{\mathrm{o}} \ \rightarrow 0.523 \text{ rd}, \qquad \mathrm{s}_1 = +1$$

$$k \ = \prod_{i=1}^{10} \sqrt{1 + 2^{-2i}} \ = 1.1644,\ X_1 = \frac{1}{k} = 0.8588$$

Then, iterations are performed as follows:

For $i = 0$,

$\theta_0 = 0.7853\, rd$

$X_2 = X_1 - s_1\, 2^{-1}\, Y_1 \ \rightarrow \ X_2 = 0.8588 - \ 1 \times \ 2^{-1} \times 0 \ = 0.8588$

$Y_2 = Y_1 - s_1\, 2^{-1}\, X_1 \ \rightarrow \ Y_1 = 0 + s_1\, 2^{-1}\, X_1 = \ 0.4294$

$Z_1 = Z_0 - s_1\, \theta_1 \rightarrow Z_1 \ = \ 0.523 - \ s_1\, 0.7853 = -0.2623$

For $i = 1$,

$\theta_1 = 0.4636\, rd\,, \quad Z_1 \ < 0, s_2 = \ -1$

$X_2 = X_1 - s_1\, 2^{-1}\, Y_1 \ \rightarrow \ X_2 = 0.8588 + \ 1 \times \ 2^{-1} \times 0 \ = 0.8588$

$Y_2 = Y_1 - s_1\, 2^{-1}\, X_1 \ \rightarrow \ Y_1 = 0 - s_1\, 2^{-1}\, X_1 = -\, 0.4294$

$Z_2 = Z_1 - s_1\,\theta_1 \rightarrow Z_1 \;=\; -0.262 + \; s_1\,0.4636 = -0.2016$

For $i = 2,$

$\theta_2 = 0.2450\,rd, \qquad Z_2 \;< 0, s_2 = \;-1$

$X_3 = X_2 - s_2\,2^{-2}\,Y_2 \;\rightarrow\; X_2 = 0.8588 + \;1 \times\; 2^{-2} \times 0.4294 \;\; = 0.75145$

$Y_3 = Y_2 + s_2\,2^{-2}\,X_2 \;\rightarrow\; Y_3 = 0.4294 - s_2\,2^{-2}\,X_2 = \;0.2147$

$Z_3 = Z_2 - s_2\,\theta_2 \rightarrow Z_3 \;=\; -0.2016 + \; s_2\,0.2450 = -0.0434$

and so on $\cdots$, and lastly

For $i = 10,$

$\theta_{10} = 0.0020\,rd, \qquad Z_2 \;< 0, s_{10} = \;-1$

$X_{11} = X_{10} - s_{10}\,2^{-10}\,Y_{10} \;\rightarrow\; X_{10} = 0.8659 + \;1 \times\; 2^{-10} \times 0.5002 \;\; = 0.8569$

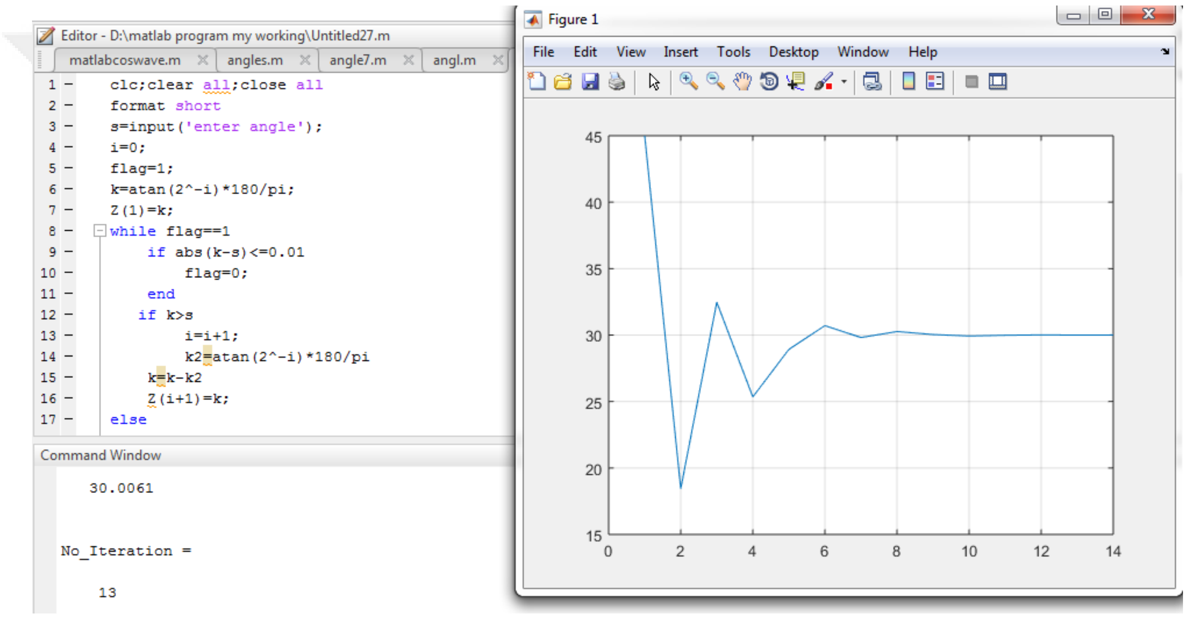
$Y_{11} = Y_{10} + s_{10}\,2^{-10}\,X_{10} \;\rightarrow\; Y_{10} = 0.5002 - 1 \times\; 2^{-10} \times\; 0.8659 = \;0.5002$

$Z_{11} = Z_{10} - s_{10}\,\theta_{10} \rightarrow Z_{10} = \;-0.0002 \;+\; 1 \times 0.002 \;\; = -0.002$

We wrote a program in MATLAB to calculate the cosine and sine values of the angles, and find the number of iterations needed considering the accuracy of the calculated value, the program and its outputs are depicted in Fig. 3.2 and Fig. 3.3.

```
clc;clear all;close all
format short
s=input('enter angle');
i=0;
flag=1;
k=atan(2^-i)*180/pi;
Z(1)=k;
while flag==1
if abs(k-s)<=0.01
flag=0;
end
if k>s
i=i+1;
```

```
k2=atan(2^-i)*180/pi
k=k-k2
Z(i+1)=k;
else
i=i+1;
k2=atan(2^-i)*180/pi
k=k+k2
Z(i+1)=k;
end
end
plot(Z)
grid on; hold on;
_Iteration=i
```



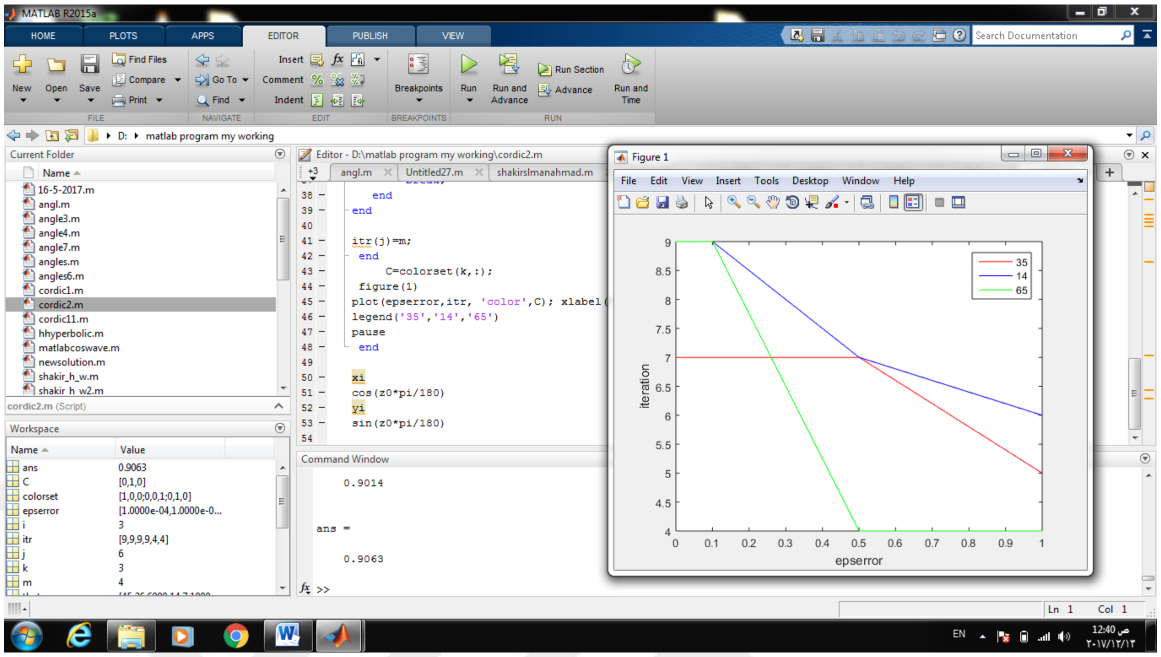
```
    clc;clear all;close all
    format short
    s=input('enter angle');
    i=0;
    flag=1;
    k=atan(2^-i)*180/pi;
    Z(1)=k;
   while flag==1
        if abs(k-s)<=0.01
            flag=0;
        end
       if k>s
            i=i+1;
            k2=atan(2^-i)*180/pi
        k=k-k2
        Z(i+1)=k;
    else
```

```
   30.0061


No_Iteration =

    13
```

Figure 3.2 Number of iterations w.r.t angles.

```
clc;clear all;close all

 %epserror=[3 2 1 0.1 0.05 0.01 0.001 0.0001]

 epserror=[0.0001 0.001 0.01 0.1 0.5  1 ]
 z0_in=[35 14 65]
 colorset={[1 0 0];[0 0 1];[0 1 0]}
 colorset =cell2mat(colorset)
 for k=1:3
 for j=1:length(epserror)
     m=0;
     xi=0.607252935;
yi=0;
z0=z0_in(k);
```

```
zi=z0;

 theta=[45 26.6 14 7.1 3.6 1.8 0.9 0.4 0.2 0.1]
for i=0:length(theta)

    if (zi>=0)
        xi1=xi-(2^-i)*yi;
        yi1=yi+(2^-i)*xi;
        zi1=zi-theta(i+1);
    else
      xi1=xi+(2^-i)*yi;
      yi1=yi-(2^-i)*xi;
      zi1=zi+theta(i+1);
    end
      xi=xi1;
      yi=yi1;
      zi=zi1;

        m=m+1;

   if (abs(zi1)<=epserror(j))

        break;
   end
end

itr(j)=m;
 end
     C=colorset(k,:);
 figure(1)
plot(epserror,itr, 'color',C); xlabel('epserror'),
ylabel('iteration'),hold on;
legend('35','14','65')
pause
 end

xi
cos(z0*pi/180)
yi
sin(z0*pi/180)
```



```
      end
  end

  itr(j)=m;
   end
       C=colorset(k,:);
   figure(1)
  plot(epserror,itr, 'color',C); xlabel(
  legend('35','14','65')
  pause
   end

  xi
  cos(z0*pi/180)
  yi
  sin(z0*pi/180)
```

Name	Value
ans	0.9063
C	[0,1,0]
colorset	[1,0,0;0,0,1;0,1,0]
epserror	[1.0000e-04,1.0000e-0...
i	3
itr	[9,9,9,9,4,4]
j	6
k	3
m	4

```
    0.9014


ans =

    0.9063

>>
```

Figure 3.3 Iteration number w.r.t accuracy parameter.

CHAPTER FOUR

Implementing CORDIC in FPGA using VHDL

4.1. Overview of CORDIC Architectures

Architecture of CORDIC algorithm is presented in this section. Generally, architectures are categorized into two parts: first unfolded and second folded as depicted in Figure 9. We can have folded-architectures by implementing difference equations in hardware for CORDIC algorithm and time multiplexing of whole iterations inside a single efficient part. Folded-architectures give mechanism to an interchange zone into signal processing architectures to timing. Folded architectures has been divided to bit-serial and word-serial architectures subject to whether functional unit carry out logic into one bit or one word for every iteration of traditional CORDIC algorithm [4].

Traditionally, CORDIC algorithm can be implemented using serial bit architecture including, and entire iterations can be performed using the same hardware. In such implementations, computing device gets slow-down and, is not appropriate for high-speed application. An iterative CORDIC architecture is word serial architecture proposed in [5, 6] which is fast compared to classical implementations. Suitable simple angles θ_i are retrieved from lookup-table.

Carry or borrow generation, addition or subtraction and variable shifting operations have degrading speed factors throughout iterations of word serial architecture, however, these operations in traditional CORDIC implementation causes the hardware to function very slowly. Thereto, through unfolding iteration process, drawbacks have been overcome by performing same iteration at each processing elements as shown in Figure 4.1. Unfolded pipelined architecture possess large throughput because of hardwired shifts instead of time and region consuming barrel shifters and removal of ROM that is main benefit of unfolded pipelined architecture over Folded architecture. Pipelined architectures give throughput enhancement by a factor of n to n-bit precision at the price of rising hardware complexity by factor not more than n.

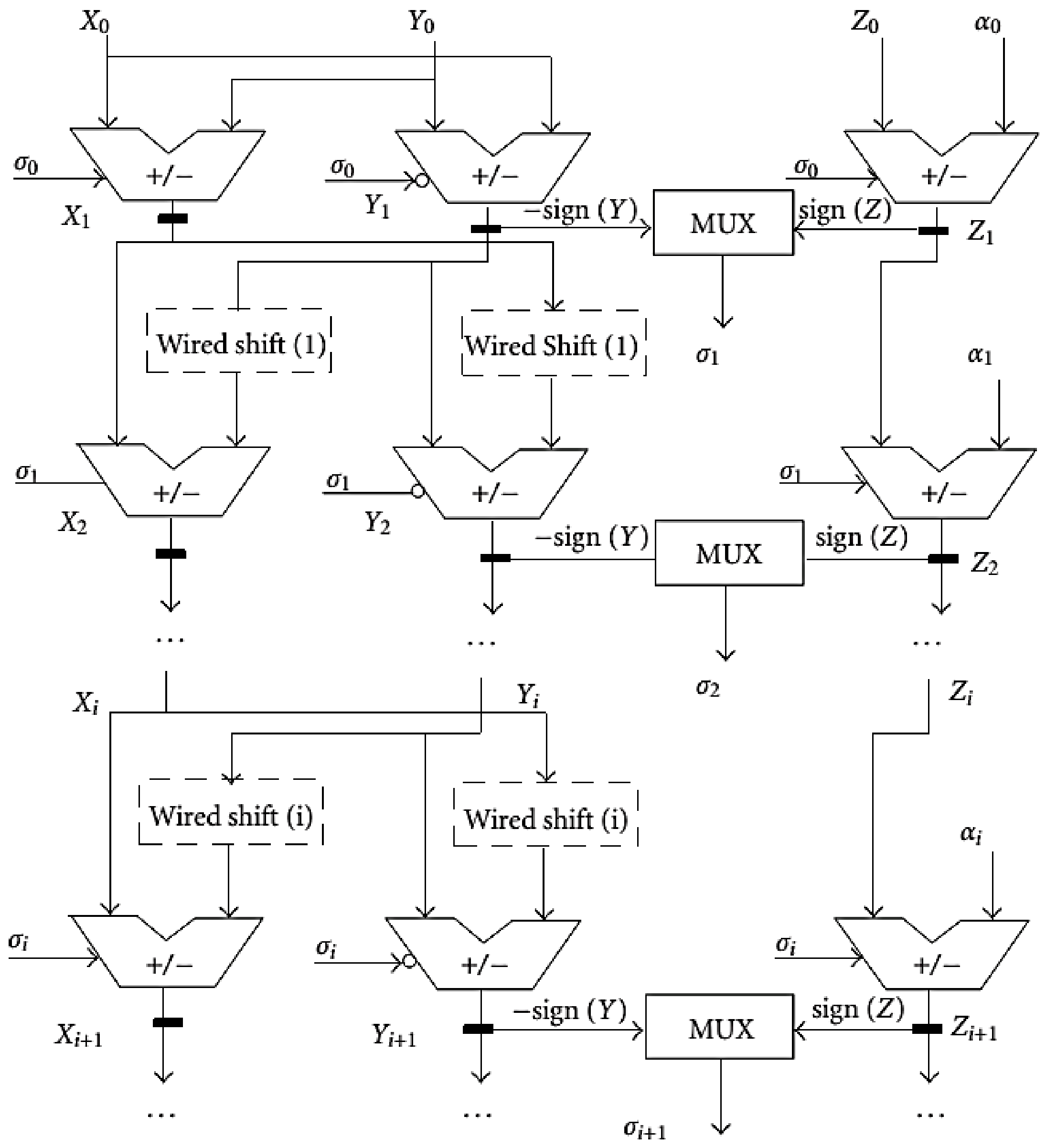


Figure 9 Basic Architecture of CORDIC [4]

4.2. FPGA Implementation to CORDIC Algorithm

Nexys 3™ FPGA Board Reference is used for algorithm implementations. Nexys 3 is a complete, ready-to-utilizing digital circuit development platform involving Xilinx Spartan-6 LX16 FPGA. Spartan-6 is enhanced to high performance logic, and gives not less than 50% more capacity, improved performance, and additional resources over **Nexys 3™** Spartan-3 500E FPGA can be found in [7].

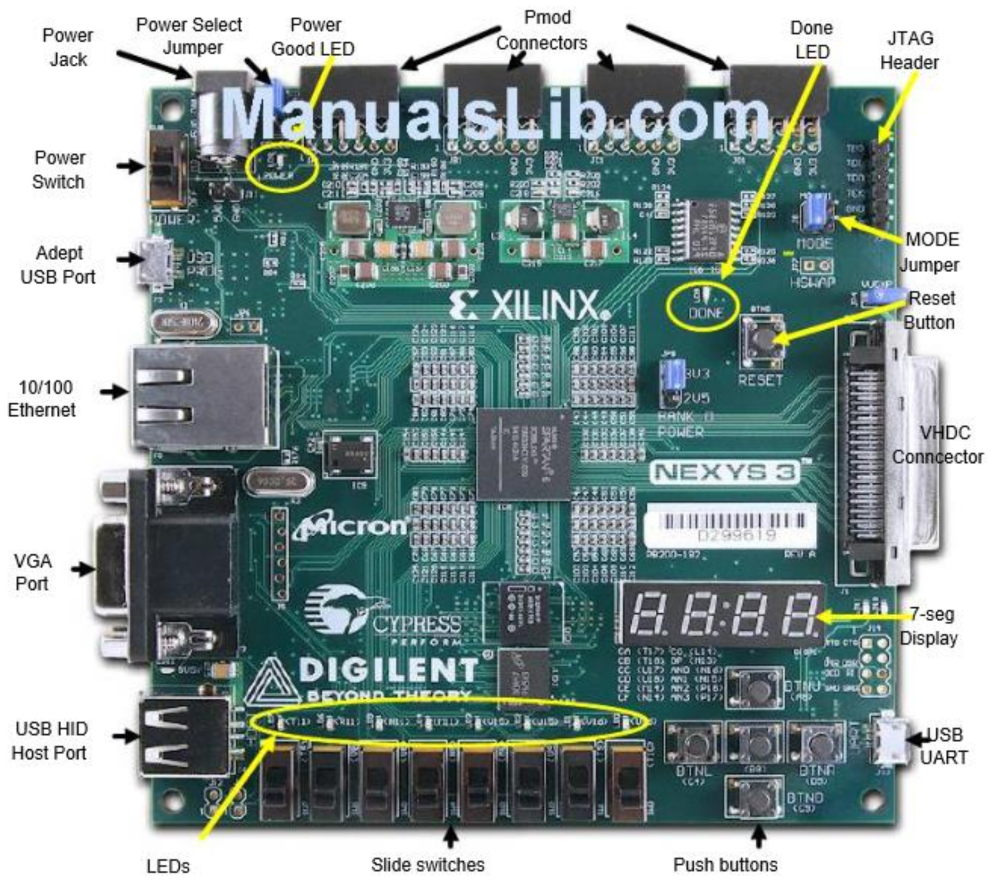


Figure 10 Nexys 3™ FPGA Board Reference

Structures of **Nexys 3™** FPGA Board are:

1. 16 Mbyte Cellular RAM x16
2. Xilnx Sprtan-6 LX16 FPGA a 324-pin BAG package

3. 16 Mbyts SIP PMC non-volatile memory
4. 16 Mbyts parallel PCM non-volatile memory
5. 10/100 Ethernet PHY
6. On-board USB2 port to programming & data xfer
7. USB-UART and USB-HID port (to mouse/keyboard)
8. 8-bit VGA port
9. 100 MHz COMS oscllator
10. 72 I/O routed to expnsion connectors
11. GIPO comprises 8 LEDs, 5 buttons,8 slide switches and 4-digit seven-segment display USB2 programming cable comprised

Besides Spartan-6 FPGA, Nexys 3 gives better group of peripherals comprising 32Mbytes of Micron's latest Phase Change Nonvolatile memory, a 10/100 Ethernet PHY, 16Mbytes of Cellular RAM, a USB-UART port, a USB host port to mice and keyboards, and an enhanced high-speed expansion connector. Large FPGA and broad set of peripherals make Nexys 3 board a perfect host to a wide variety of digital systems, comprising embedded processor designs founded on Xilinx's MicroBlaze. Nexys 3 is well-matched with every Xilinx CAD tools, comprising ChipScope, EDK, and free WebPack. Nexys 3 utilizes Digilent's latest Adept USB2 system that gives FPGA and ROM programming, automatic board tests, virtual I/O, and basic user-data transmission services.

CHAPTER FIVE

Generation of Sinusoidal Signals

In order to generate an output signal having a sinusoidal waveform, we will be use a digital stores at different locations binary-coded amplitudes at equally spaced instants of time during a complete cycle of a sine wave. Those locations are addressed in turn by incomes of a pulse counter and the signals supplied by the memory are passed to a digital/analogue converter from which the required output signal is resultant. The pulse signal supplied to the counter is achieved from a sequence of clock pulses by a programmable divider. To get a frequency change in the output signal (for the purpose of signaling binary data), the frequency of the divider is changed in a sequence of steps so that the overall change in output frequency is less quick than would otherwise be the case.

5.1. Digital to analog Converter

The abbreviation DAC is refers to converts from digital to analog. A device gives an analogue signal from convert's digital data. The digital to analogue converter takes serial numbers of discrete values as input values and gives an analogue signal that is instantaneous with the equivalent numerical value. Several kinds of Digital to Analog converter ICs are obtainable commercially built on this same standard. The R-2R ladder circuit is constructed by a set of resistors of two values. It makes the circuit simpler and economical for several applications.

R-2R biased resistor ladder circuit consumptions only 2 set of resistors- R and 2R. If you need to construct a very fixed DAC, be accurate while selecting the values of resistors that will exactly equal to the R-2R ratio

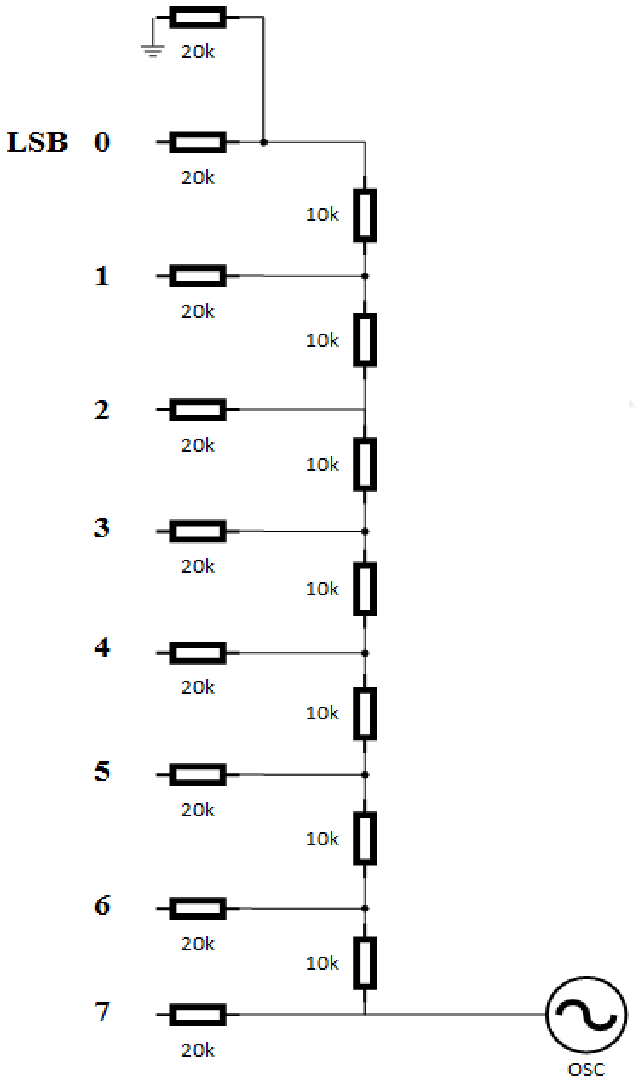


Figure 11 Digital to Analog Converter

5.2. Introduction

In order to test a circuit by software we can use simulator platforms. Conventionally, we can build a digital circuit on board using chips to test logical circuit but this state has some disadvantages. One of the important disadvantages is high cost, because in this method we need heavy expensive lab equipment and hardware. Cost can increase more because many chips may be hooked up and incorrectly destroyed. Some errors often difficult to detect on board. Solution to these problems is design re-usable circuits. In next step, we will explain Xilinx ISE Design Suite 14.5 ISim Simulator.

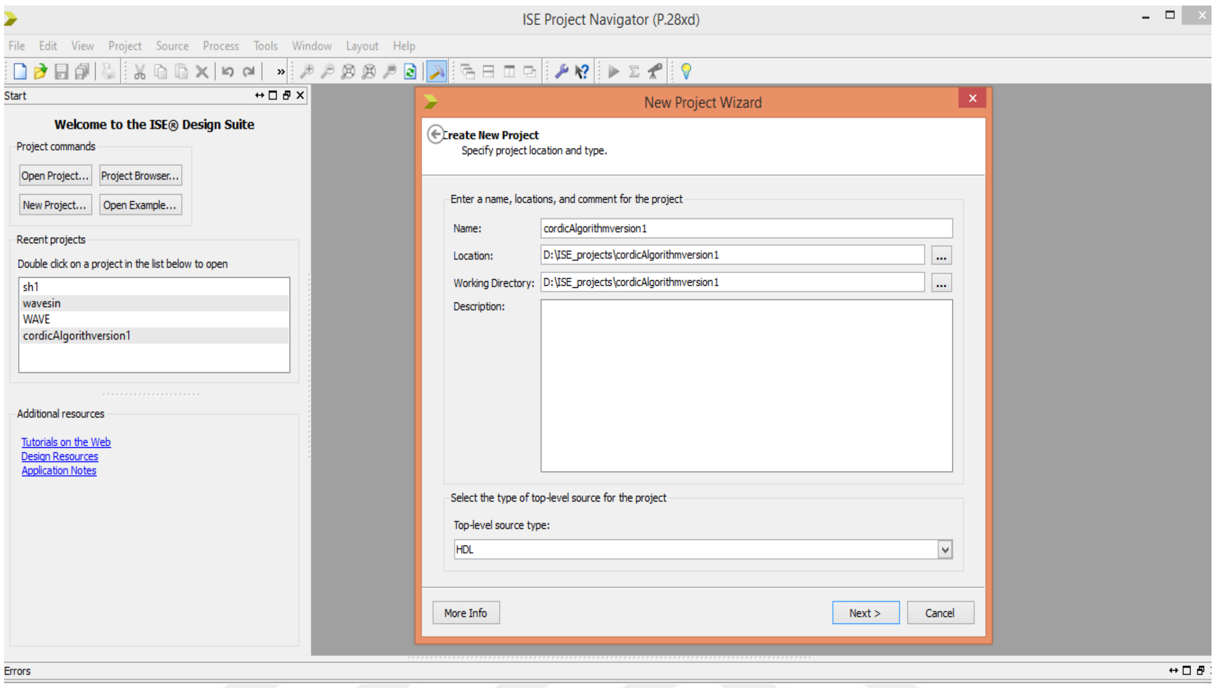
We will learn the following topics:

1. How to use a simulator and learn different capabilities of it?
2. How to create a test bench in (VHDL) in order to be used by simulator?
3. How to verify functionality (CORDIC Algorithm) by using Xilinx ISim Simulator?

5.2.1. Getting Started

In this section, we will describe requirements to conducting behavioral simulations. To create design files in VHDL, we open icon (file) and chose open (new project) option and name it, see Fig. 12.

- To simulate design, we need a bench file to provide stimulus to design.
- Simulation Libraries: We need Xilinx simulation libraries when we use Xilinx primitive or IP core in design.


ISE Project Navigator (P.28xd)
File Edit View Project Source Process Tools Window Layout Help
Start
Welcome to the ISE® Design Suite
Project commands
Open Project... Project Browser...
New Project... Open Example...
Recent projects
Additional resources
Tutorials on the Web
Design Resources
Application Notes
New Project Wizard
More Info
Next >
Cancel
Errors

Double click on a project in the list below to open

sh1
wavesin
WAVE
cordicAlgorithversion1

Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name:	cordicAlgorithmversion1
Location:	D:\ISE_projects\cordicAlgorithmversion1
Working Directory:	D:\ISE_projects\cordicAlgorithmversion1
Description:	

Select the type of top-level source for the project

Top-level source type: HDL

Figure 12 How to create a design file.

5.3. Behavioral Simulation Using ISim

The steps to be followed to simulate your design are:

1. Open your design project (cordicAlgorithversion1).

2. Choose simulation mode.

3. Add a HDL test bench to your existing design.

4. Identify simulations.

5. Determine simulation properties.

6. Simulation performance.

7. Add signals.

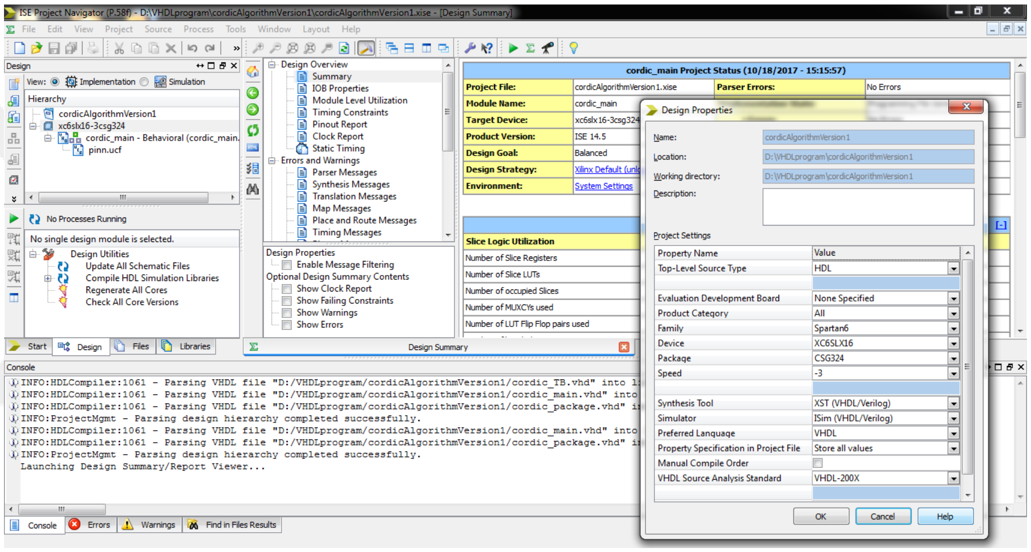
8. Signal analysis.

5.3.1. How to open project

1. Open Project Navigator from all programs Xilinx ISE design suite 14.5.
2. We see project navigator window Figure 13.
3. We have two method to open any project:

   1. Directly open project and chose project from list.

   2. Open existing project on screen and we will see previous project to example (cordicAlgorithversion1) and chose it.



cordic_main Project Status (10/18/2017 - 15:15:57)

Project File:	cordicAlgorithmVersion1.xise	Parser Errors:	No Errors
Module Name:	cordic_main		
Target Device:	xc6slx16-3csg324		
Product Version:	ISE 14.5		
Design Goal:	Balanced		
Design Strategy:	Xilinx Default (unlo		
Environment:	System Settings		

Slice Logic Utilization
Number of Slice Registers
Number of Slice LUTs
Number of occupied Slices
Number of MUXCYs used
Number of LUT Flip Flop pairs used

Design Properties

Name: cordicAlgorithmVersion1
Location: D:\VHDLprogram\cordicAlgorithmVersion1
Working directory: D:\VHDLprogram\cordicAlgorithmVersion1
Description:

Project Settings

Property Name	Value
Top-Level Source Type	HDL
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-3
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-200X

OK Cancel Help

Console

```
INFO:HDLCompiler:1061 - Parsing VHDL file "D:/VHDLprogram/cordicAlgorithmVersion1/cordic_TB.vhd" into li
INFO:HDLCompiler:1061 - Parsing VHDL file "D:/VHDLprogram/cordicAlgorithmVersion1/cordic_main.vhd" into
INFO:HDLCompiler:1061 - Parsing VHDL file "D:/VHDLprogram/cordicAlgorithmVersion1/cordic_package.vhd" in
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
INFO:HDLCompiler:1061 - Parsing VHDL file "D:/VHDLprogram/cordicAlgorithmVersion1/cordic_main.vhd" into
INFO:HDLCompiler:1061 - Parsing VHDL file "D:/VHDLprogram/cordicAlgorithmVersion1/cordic_package.vhd" in
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
Launching Design Summary/Report Viewer...
```

Figure 13. Project navigator window

4. We see two choices, implementation or simulation, See Fig. 14.

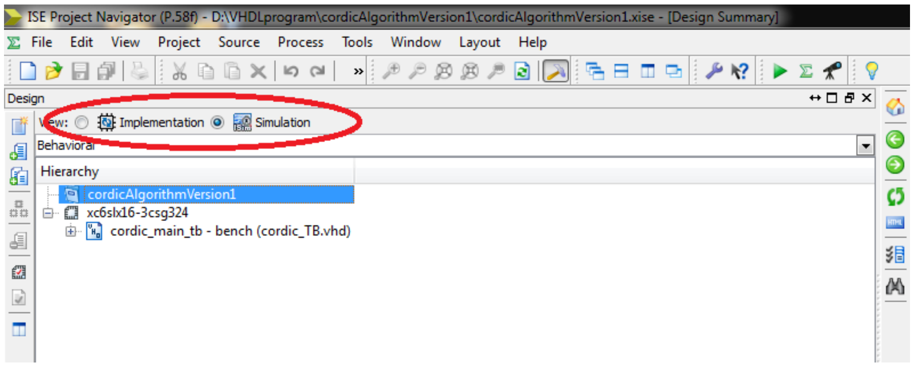

ISE Project Navigator (P.58f) - D:\VHDLprogram\cordicAlgorithmVersion1\cordicAlgorithmVersion1.xise - [Design Summary]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
cordicAlgorithmVersion1
xc6slx16-3csg324
cordic_main_tb - bench (cordic_TB.vhd)

Figure 14 Explanation of the choices for the implementation and simulation options

5. You should use implementation mode when you want to design a circuit using VHDL. On other hand, you should switch to simulation mode when you want to

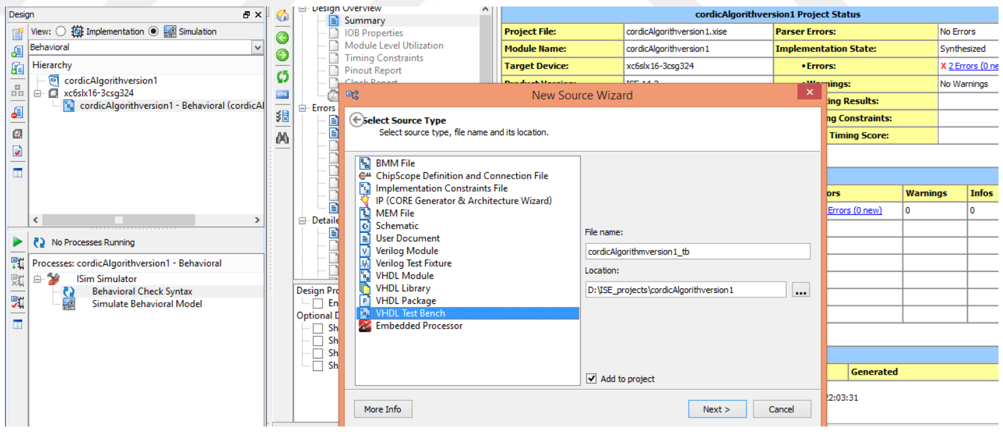
simulate your design. To simulate the design, we should use Simulation mode and chose implementation mode when you want to design a circuit using VHDL.

•When we use simulation mode, we see behavioral check syntax and simulate behavioral model, translate simulation (that is executed after synthesis), mapping (that is implemented after executing design), route simulation (that is achieved after mapping, placement and routing).

5.4. How to add an HDL Test Bench

In order to create a new test bench file to our project and modify by editing it with statements, we should follow these steps:

1. Highlight our project and then click on new source and select VHDL Test Bench and enter “cordicAlgorithversion1_tb”, and click next, see Fig. 15.



File name: cordicAlgorithmversion1_tb

Location: D:\ISE_projects\cordicAlgorithversion1

☑ Add to project

cordicAlgorithversion1 Project Status

Project File:	cordicAlgorithversion1.xise	Parser Errors:	No Errors
Module Name:	cordicAlgorithversion1	Implementation State:	Synthesized
Target Device:	xc6slx16-3csg324	•Errors:	X 2 Errors (0 ne
		•Warnings:	No Warnings

Figure 15.Creating a test bench.

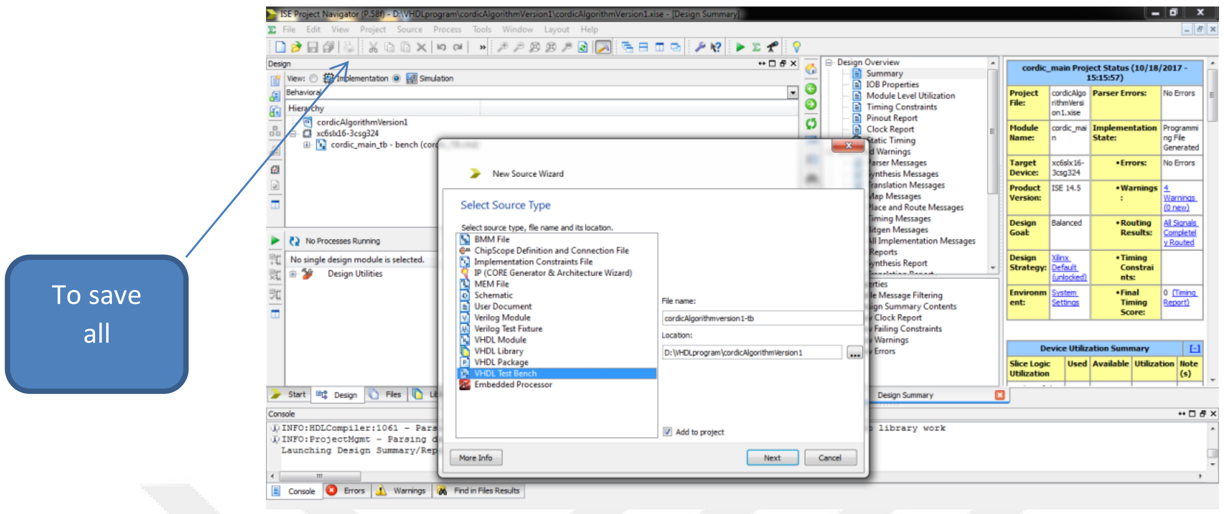


Figure 16 Saving a test bench file

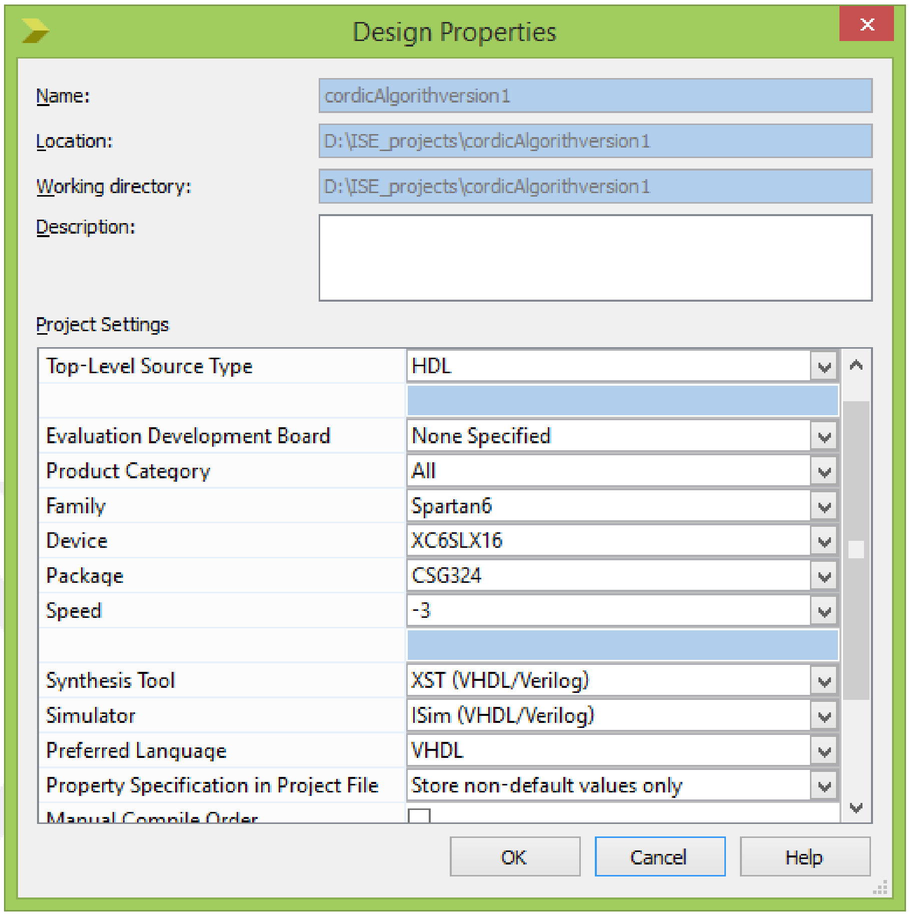
2. We will see new file with our project in Hierarchy pane (cordicAlgorithversion1.vhd) and a skeleton VHDL file in Workspace panel on right hand side of screen.
3. We write a test bench program and save it.
4. To save test bench file see Fig. 16.

5.5. Behavioral Simulation Using ISim

After we get a test bench in our project, we can implement behavioral simulation using ISim and ISE software full integration. ISE software supports ISim to create directory, compile main files, load design, and implement simulation.

How to Select ISim

We can do double-click in device line in hierarchy pane of Project Navigator Design panel, right-click device line and select Design Properties and set Simulator field to ISim(VHDL/Verilog) and press OK., see Fig. 17.


Design Properties

Name: cordicAlgorithversion1

Location: D:\ISE_projects\cordicAlgorithversion1

Working directory: D:\ISE_projects\cordicAlgorithversion1

Description:

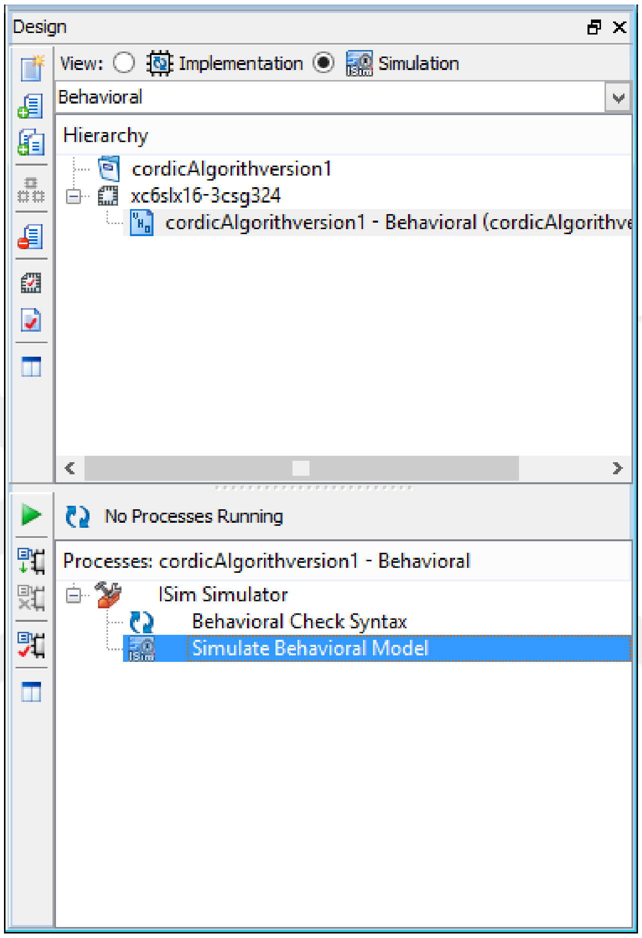
Project Settings

Top-Level Source Type	HDL
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-3
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store non-default values only
Manual Compile Order	☐

Figure 17. How to select ISim

How to locate ISim processes

a. Select Simulation, and select Behavioral from drop-down list and select test bench file (cordicAlgorithversion1- TB), see Fig. 18.


Design

View: ○ Implementation ● Simulation

Behavioral

Hierarchy

cordicAlgorithversion1
xc6slx16-3csg324
cordicAlgorithversion1 - Behavioral (cordicAlgorithve

No Processes Running

Processes: cordicAlgorithversion1 - Behavioral

ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model

Figure 18 How to select Simulate Behavioral Model

b. Expand ISim Simulator in Processes pane to view list of processes.

c. At beginning check syntax, and simulate behavioral model. If all goes fine, we will get message ("Completed successfully).

How to do behavioral simulation:

We can change properties and see behavioral simulation properties by apply these steps:

d. Choice test bench files (cordicAlgorithmversion1_tb).

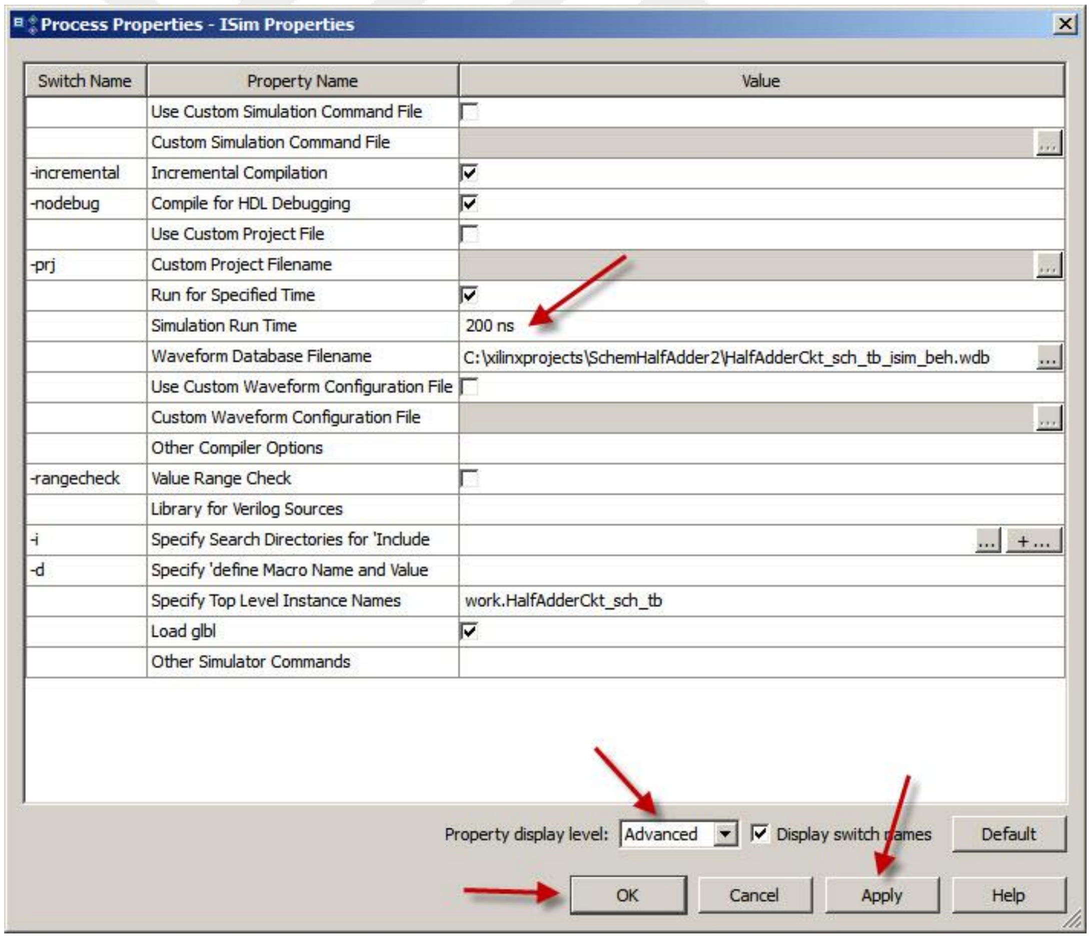
e. Expand ISim Simulator, right-click Simulator Behavioral Model, and choice Process Properties.

f. Set Property display level to Advance in Process Properties dialog box.

g. Convert Simulation Run Time, say 200 ns.

h. Click on apply.

i. Click OK, see Fig. 19.



Process Properties - ISim Properties

Switch Name	Property Name	Value
	Use Custom Simulation Command File	☐
	Custom Simulation Command File	
-incremental	Incremental Compilation	☑
-nodebug	Compile for HDL Debugging	☑
	Use Custom Project File	☐
-prj	Custom Project Filename	
	Run for Specified Time	☑
	Simulation Run Time	200 ns
	Waveform Database Filename	C:\xilinxprojects\SchemHalfAdder2\HalfAdderCkt_sch_tb_isim_beh.wdb
	Use Custom Waveform Configuration File	☐
	Custom Waveform Configuration File	
	Other Compiler Options	
-rangecheck	Value Range Check	☐
	Library for Verilog Sources	
-i	Specify Search Directories for 'Include	
-d	Specify 'define Macro Name and Value	
	Specify Top Level Instance Names	work.HalfAdderCkt_sch_tb
	Load glbl	☑
	Other Simulator Commands	

Property display level: Advanced ☑ Display switch names

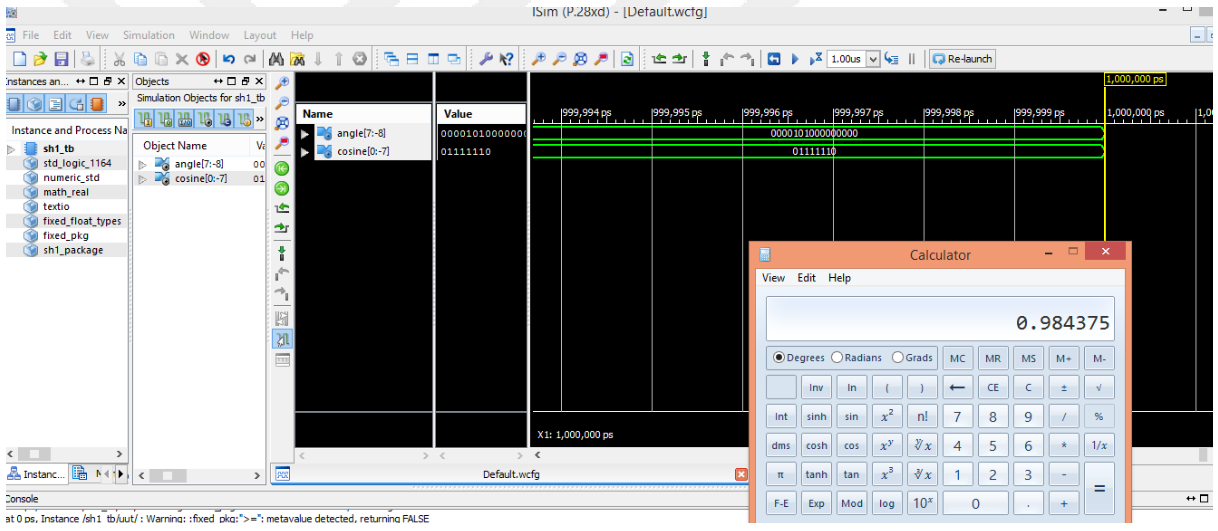
Figure 19. Change properties and behavioral simulation properties

How to perform simulation

To obtain the simulation waveforms, we follow the following steps:

a. Will need to use zoom tools (zoom out and in), and we can click in restart simulation icon .

As an example, if we choose the angle value as 10, and run the cordic algorihm, we get $cos(10) = 0.984375$, see Fig. 20.



Name	Value
angle[7:-8]	0000101000000000
cosine[0:-7]	01111110

0.984375

```
at 0 ps, Instance /sh1_tb/uut/ : Warning: :fixed_pkg:">=": metavalue detected, returning FALSE
```

Figure 20. How to perform simulation

5.6. Information from VHDL Test Bench

We can use test bench program to with timing utility. When we want to write a suitable test bench we need provide simulation data to test the behavior of the circuit.

Every test bench consist three elements:

- Unit under test (UUT)
- Stimulus generate
- Response monitor.

Unit Under Test: A test bench is top-level VHDL program that includes an instantiation of the module under consideration.

We can achieve authentication in simple project test-benches by visual examination of UUT's outputs in simulator's waveform window.

However, a more efficient approach is to include code in test bench so that we can automatically check real UUT output values in contrast to probable values. We consider test bench by self-checking .We will get a message if any error appears during the test.

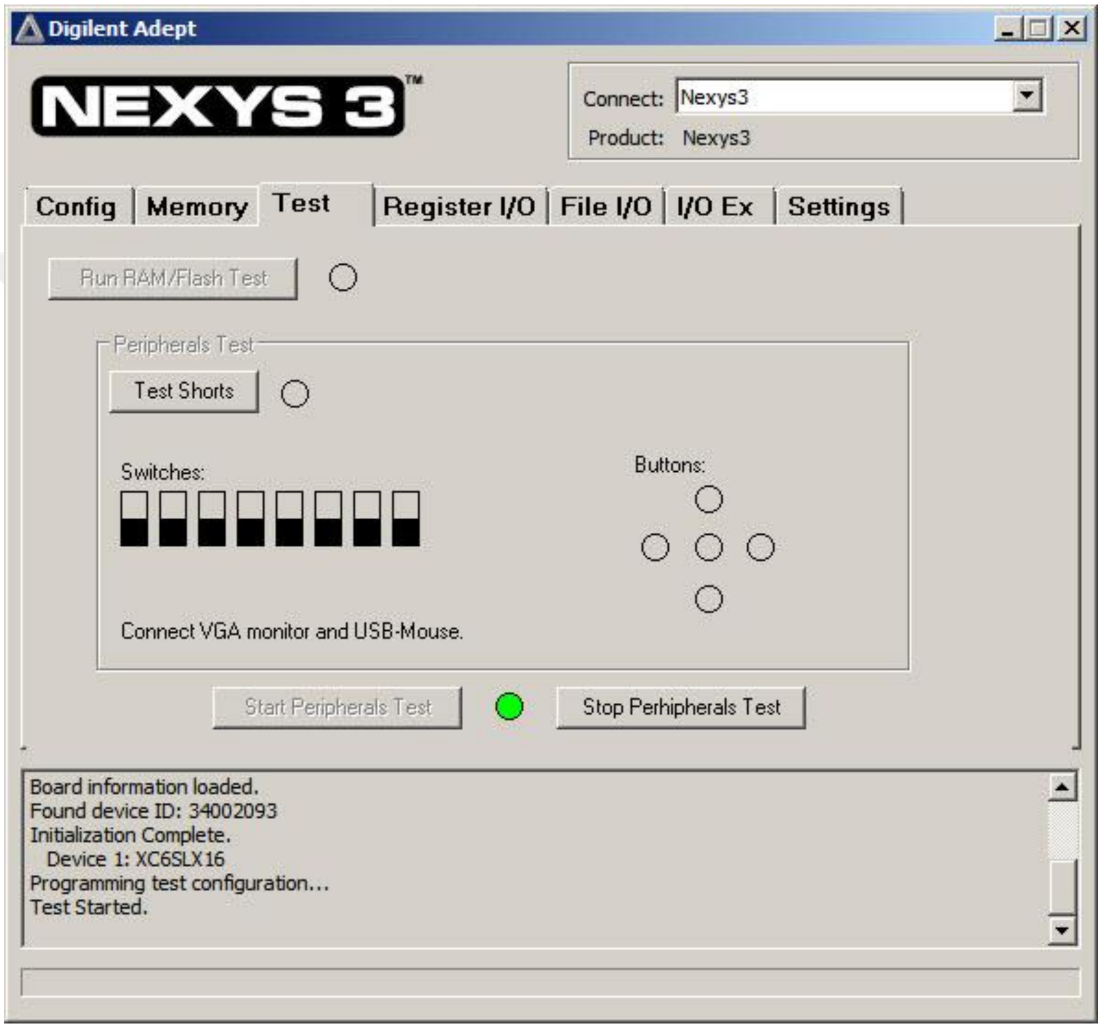
5.7. How to set-up and test NEXYS3 board

Setup of NEXYS 3 boards is not difficult. We will use LEDs by way of outputs. The following steps are pursued for the setting up of FPGA board.

1. We connect NEXYS 3 board to USB cable.
2. We connect USB cable to our computer.
3. Small yellow LED is energized when power is turned on.
4. By using Digilent Adept tool we can test, if digilent NEXYS3 board is active.
5. On Test icon click "Start Peripherals Test".

On LEDs we will have different output values depending on the used angle value.

In order to reset board to industry settings, we press the “reset button’. At top left part of board, we use slide switch to power off, see Fig. 21.



Connect VGA monitor and USB-Mouse.

```
Board information loaded.
Found device ID: 34002093
Initialization Complete.
  Device 1: XC6SLX16
Programming test configuration...
Test Started.
```

Figure 21 Setting up of NEXYS3 board

5.8. Switches and 7-Segments displays

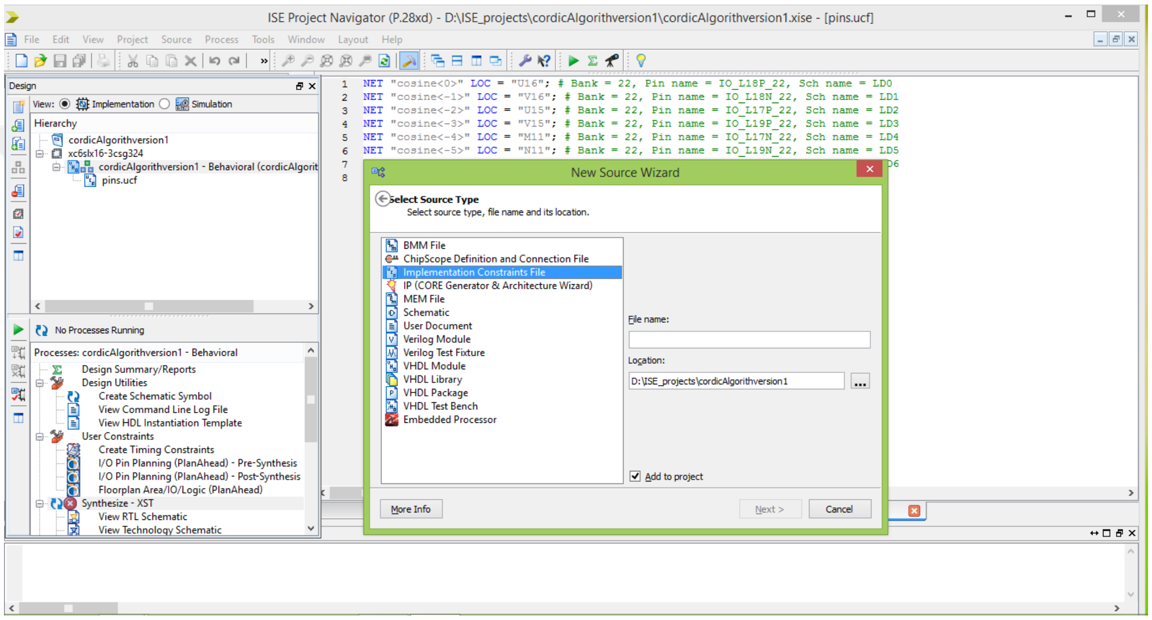
We use NEXYS 3 Board.

We need to know connection and location of DIP switches and LEDs.

5.8.1. LEDs

NEXYS 3 Board affords a sequence of eight LEDs (LD0–LD7) to use. Every LED glows when high”1-logic”is applied on it.

In this thesis, we use the LED locations listed in Table 1 as illustrated in Fig. 22. In Table 1, connections from LEDs to NEXYS 3 Board and UCF constraints are shown.


ISE Project Navigator (P.28xd) - D:\ISE_projects\cordicAlgorithversion1\cordicAlgorithversion1.xise - [pins.ucf]

```
NET "cosine<0>" LOC = "U16"; # Bank = 22, Pin name = IO_L18P_22, Sch name = LD0
NET "cosine<-1>" LOC = "V16"; # Bank = 22, Pin name = IO_L18N_22, Sch name = LD1
NET "cosine<-2>" LOC = "U15"; # Bank = 22, Pin name = IO_L17P_22, Sch name = LD2
NET "cosine<-3>" LOC = "V15"; # Bank = 22, Pin name = IO_L19P_22, Sch name = LD3
NET "cosine<-4>" LOC = "M11"; # Bank = 22, Pin name = IO_L17N_22, Sch name = LD4
NET "cosine<-5>" LOC = "N11"; # Bank = 22, Pin name = IO_L19N_22, Sch name = LD5
```

New Source Wizard

Select Source Type
Select source type, file name and its location.

File name:
Location: D:\ISE_projects\cordicAlgorithversion1
Add to project

File
Edit
View
Project
Source
Process
Tools
Window
Layout
Help
Design
View: Implementation
Simulation
Hierarchy
cordicAlgorithversion1
xc6slx16-3csg324
cordicAlgorithversion1 - Behavioral (cordicAlgorit
pins.ucf
No Processes Running
Processes: cordicAlgorithversion1 - Behavioral
Design Summary/Reports
Design Utilities
Create Schematic Symbol
View Command Line Log File
View HDL Instantiation Template
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAhead) - Pre-Synthesis
I/O Pin Planning (PlanAhead) - Post-Synthesis
Floorplan Area/IO/Logic (PlanAhead)
Synthesize - XST
View RTL Schematic
View Technology Schematic
BMM File
ChipScope Definition and Connection File
Implementation Constraints File
IP (CORE Generator & Architecture Wizard)
MEM File
Schematic
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
Embedded Processor
More Info
Next >
Cancel

Figure 22 Connection from LEDs to NEXYS 3 Board

Table 1: NEXYS 3 (Light Emitting Diodes) LEDs

Description	Location
NET LD0	LOC = U16
NET LD1	LOC = U16
NET LD2	LOC = U15
NET LD3	LOC = U15
NET LD4	LOC = U11
NET LD5	LOC = U11
NET LD6	LOC = U11
NET LD7	LOC = U11

5.8.2. Seven Segment Displays

Digilent NEXYS 3 Board includes 4 multiplexed 7-segment displays to use.

If we need to display on 7-segment displays, we can use the addresses in Table 2 for UCF file.

Table 2: NEXYS 3 (7-Segment display)

Description	Location
NET CA	LOC = T17
NET CB	LOC = T16
NET CC	LOC = U17
NET CD	LOC = M14
NET CE	LOC = N14
NET CF	LOC = L14
NET CG	LOC = M13
NET DP	LOC =N16
NET AN0	LOC = N15
NET AN1	LOC = P19
NET AN2	LOC = P17
NET AN3	LOC = T17

5.8.3. Switches

We see eight slide switches available on NEXYS 3 board. When we put switch at ON state, each DIP switch pull pin of NEXYS 3 Board is connected to the ground, but when DIP switch is at OFF state, pin is drawn high through a 10K resistance.

In Table 3 below we show connections of switches to Digilent NEXYS 3 Board.

Table 3: NEXYS 3 (Slide Switches)

Description	Location
NET SW0	LOC = T10
NET SW1	LOC = T9
NET SW2	LOC = V9
NET SW3	LOC = M8
NET SW4	LOC = N8
NET SW5	LOC = U8
NET SW6	LOC = V8
NET SW7	LOC =T5

5.8.4. Push Buttons

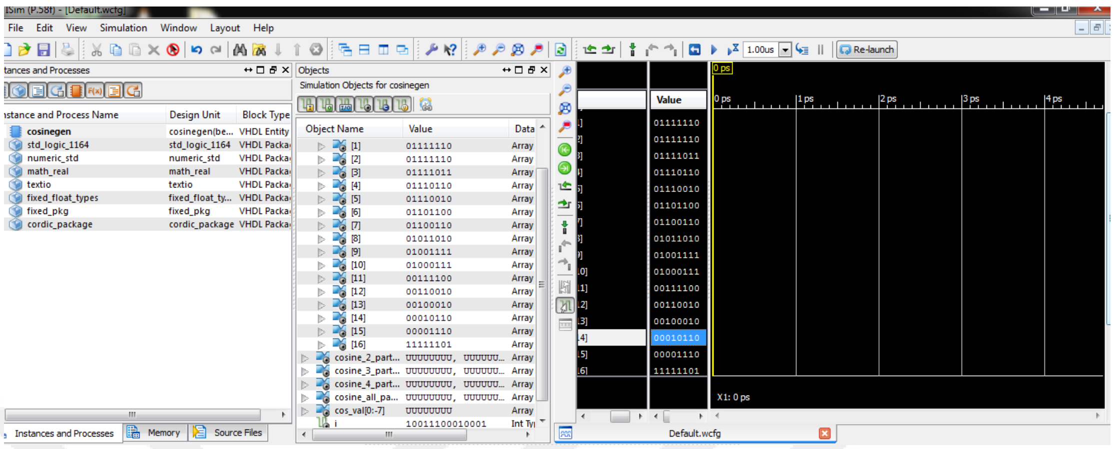
We see five pushbuttons (labeled BTNS through BTNR) on Digilent NEXYS 3 board available to the user. When pushed, each pushbutton is connected to the ground pin of NEXYS 3 Board. Else, pin is drawn high through a 10K resistor.

In Table 4 we see connections from push buttons to Digilent NEXYS 3 Board.

Table 4: NEXYS 3 (Pushbuttons)

Description	Location
NET BTNS	LOC = B8
NET BTNS	LOC = A8
NET BTNS	LOC = C4
NET BTNS	LOC = C9
NET BTNS	LOC = D9

We get the following results, when we use our code as shown below



Instance and Process Name	Design Unit	Block Type
cosinegen	cosinegen(be...	VHDL Entity
std_logic_1164	std_logic_1164	VHDL Packa
numeric_std	numeric_std	VHDL Packa
math_real	math_real	VHDL Packa
textio	textio	VHDL Packa
fixed_float_types	fixed_float_ty...	VHDL Packa
fixed_pkg	fixed_pkg	VHDL Packa
cordic_package	cordic_package	VHDL Packa

Object Name	Value	Data
[1]	01111110	Array
[2]	01111110	Array
[3]	01111011	Array
[4]	01110110	Array
[5]	01110010	Array
[6]	01101100	Array
[7]	01100110	Array
[8]	01011010	Array
[9]	01001111	Array
[10]	01000111	Array
[11]	00111100	Array
[12]	00110010	Array
[13]	00100010	Array
[14]	00010110	Array
[15]	00001110	Array
[16]	11111101	Array
cosine_2_part...	UUUUUUUU, UUUUUU...	Array
cosine_3_part...	UUUUUUUU, UUUUUU...	Array
cosine_4_part...	UUUUUUUU, UUUUUU...	Array
cosine_all_pa...	UUUUUUUU, UUUUUU...	Array
cos_val[0:-7]	UUUUUUUU	Array
i	10011100010001	Int Ty

	Value
1]	01111110
2]	01111110
3]	01111011
4]	01110110
5]	01110010
6]	01101100
7]	01100110
8]	01011010
9]	01001111
10]	01000111
11]	00111100
12]	00110010
13]	00100010
14]	00010110
15]	00001110
16]	11111101

Figure23 Results of cosine values

cos(0.0000)=01111111=1/2+1/4+1/8+1/16+1/64+1/128-----------> = 0.9884375

cos(05.6250)=01111110=1/2+1/4+1/8+1/16+1/32+1/64-------> = 0.984375

cos(11.2500)=01111110=1/2+1/4+1/8+1/16+1/32+1/64------> = 0.984375

cos(16.8750)=01111011=1/2+1/4+1/8+1/16+1/64+1/128-----> = 0.9609375

cos(22.5000)=01110110=1/2+1/4+1/8+1/32+1/64-----------> = 0.921875

cos(28.1250)=01110010=1/2+1/4+1/8+1/64----------------> = 0.890625

cos(33.7500)=01101100=1/2+1/4+1/16+1/32---------------> = 0.84375

cos(39.3750)=01100110=1/2+1/4+1/32+1/64---------------> = 0.796875 cos(45.0000)=01011010=1/2+1/8+1/16+1/64---------------> = 0.703125

cos(50.6250)=01001111=1/2+1/16+1/32+1/64+1/128--------> = 06171875

cos(56.2500)=01000111= 1/2+1/32+1/64+1/128------------> = 0.5546875

cos(61.8750)=00111100=1/4+1/8+1/16+1/32---------------> = 0.46875

cos(67.5000)=00110010=1/4+1/8+1/64----------------------> = 0.390625

cos(73.1250)=00100010=1/4+1/64----------------------------> = 0.265625

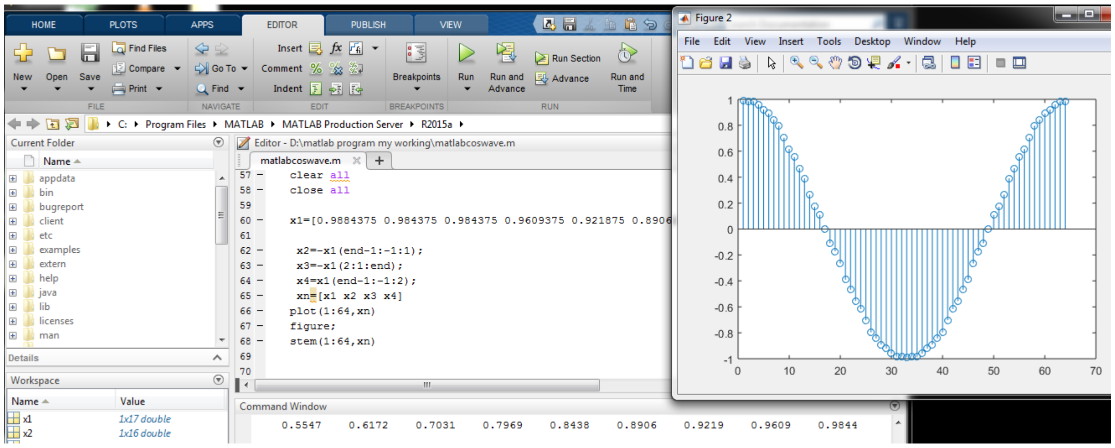
cos(78.7500)=00010110=1/8+1/32+1/64--------------------> = 0.171875

cos(84.3750)=00001110=1/16+1/32+1/64-------------------> = 0.109375

cos(90.0000)=11111101=~zero

The MATLAB code to create a vector representing the values of cosine is written as:

```
clc
Clear all
Close all
 x1=[0.9884375 0.984375 0.984375 0.9609375 0.921875 0.890625 0.84375
0.796875 0.703125 0.6171875 0.5546875 0.46875 0.390625 0.265625
0.171875 0.109375 0];
 x2=-x1(end-1:-1:1);
 x3=-x1(2:1:end);
 x4=x1(end-1:-1:2);
 xn=[x1 x2 x3 x4]
plot(1:64,xn)
figure;
stem(1:64,xn)
```


Editor - D:\matlab program my working\matlabcoswave.m

```
clear all
close all

x1=[0.9884375 0.984375 0.984375 0.9609375 0.921875 0.8906

 x2=-x1(end-1:-1:1);
 x3=-x1(2:1:end);
 x4=x1(end-1:-1:2);
 xn=[x1 x2 x3 x4]
plot(1:64,xn)
figure;
stem(1:64,xn)
```

Name	Value
x1	1x17 double
x2	1x16 double

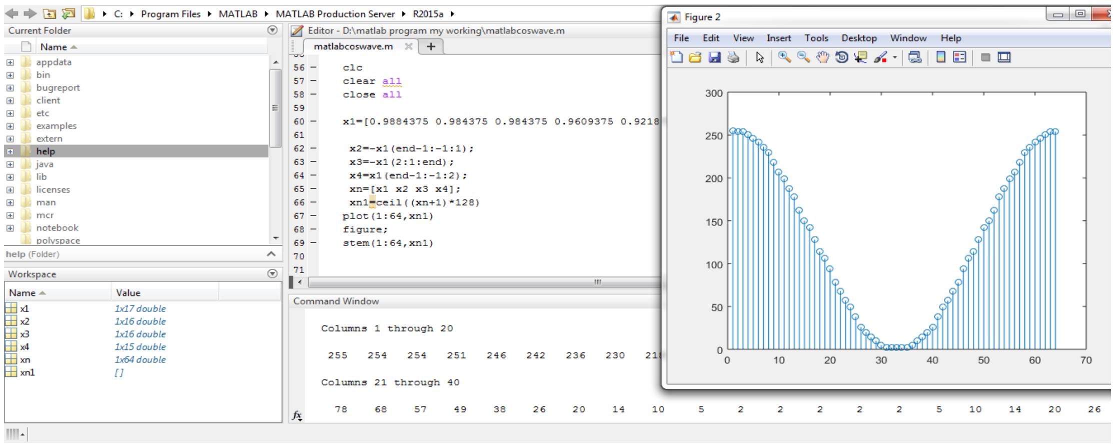
Command Window

```
    0.5547    0.6172    0.7031    0.7969    0.8438    0.8906    0.9219    0.9609    0.9844
```

Figure 2
File Edit View Insert Tools Desktop Window Help
HOME PLOTS APPS EDITOR PUBLISH VIEW

Figure24 Cosine wave

```
clc
clear all
close all

x1=[0.9884375 0.984375 0.984375 0.9609375 0.921875 0.890625 0.84375
0.796875 0.703125 0.6171875 0.5546875 0.46875 0.390625 0.265625
0.171875 0.109375 0];

 x2=-x1(end-1:-1:1);
 x3=-x1(2:1:end);
 x4=x1(end-1:-1:2);
 xn=[x1 x2 x3 x4];
 xn1=ceil((xn+1)*128)
plot(1:64,xn1)
figure;
stem(1:64,xn1)
```


Editor - D:\matlab program my working\matlabcoswave.m

```
clc
clear all
close all

x1=[0.9884375 0.984375 0.984375 0.9609375 0.9218

 x2=-x1(end-1:-1:1);
 x3=-x1(2:1:end);
 x4=x1(end-1:-1:2);
 xn=[x1 x2 x3 x4];
 xn1=ceil((xn+1)*128)
plot(1:64,xn1)
figure;
stem(1:64,xn1)
```

Name	Value
x1	1x17 double
x2	1x16 double
x3	1x16 double
x4	1x15 double
xn	1x64 double
xn1	[]

Command Window

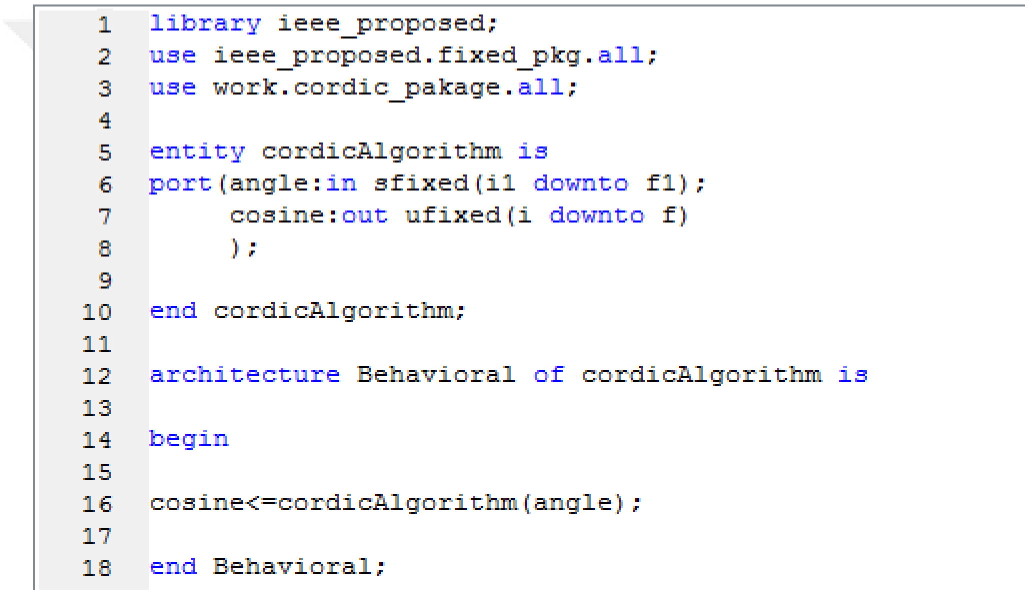
```
  Columns 1 through 20

   255   254   254   251   246   242   236   230   21

  Columns 21 through 40

    78    68    57    49    38    26    20    14    10     5     2     2     2     2     2     5    10    14    20    26
```

Figure 25 Cosine wave with shift

xn1=[255 254 254 251 246 242 236 230 218 207 199 188 178 162 150 142 128 114 106 94 78 68 57 49 38 26 20 14 10 5 2 2 2 2 2 5 10 14 20 26 38 49 57 68 78 94 106 114 128 142 150 162 178 188 199 207 218 230 236 242 246 251 254 254]

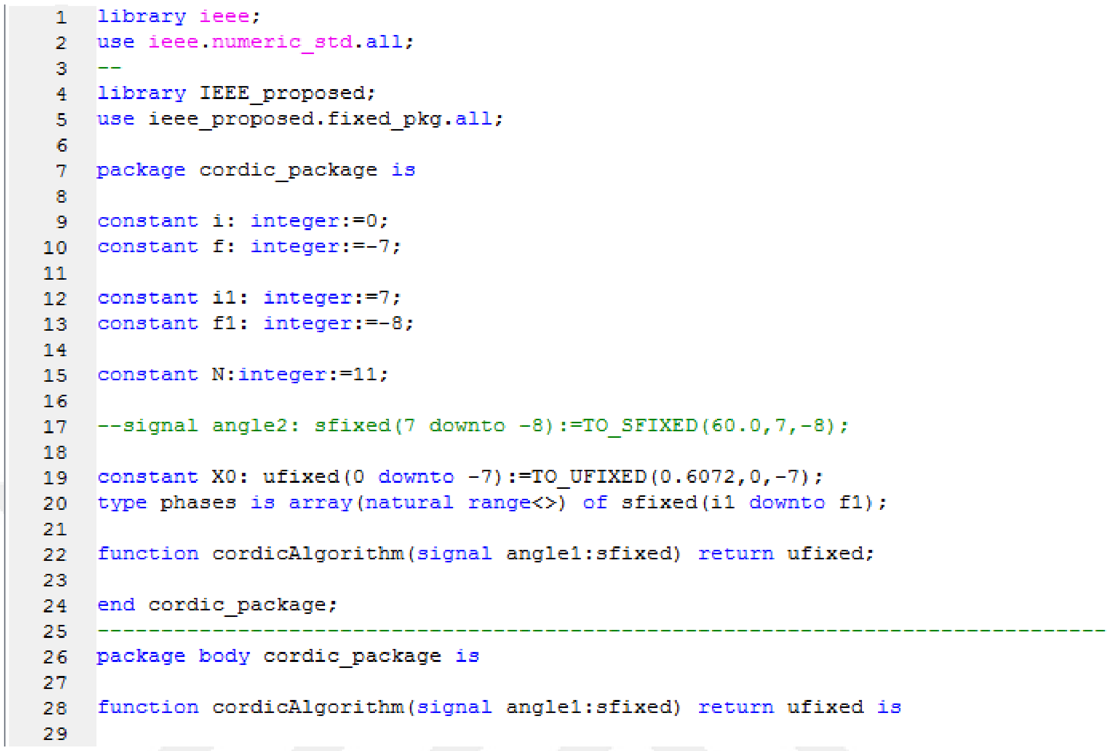
The VHDL code loaded into FPGA

Main program

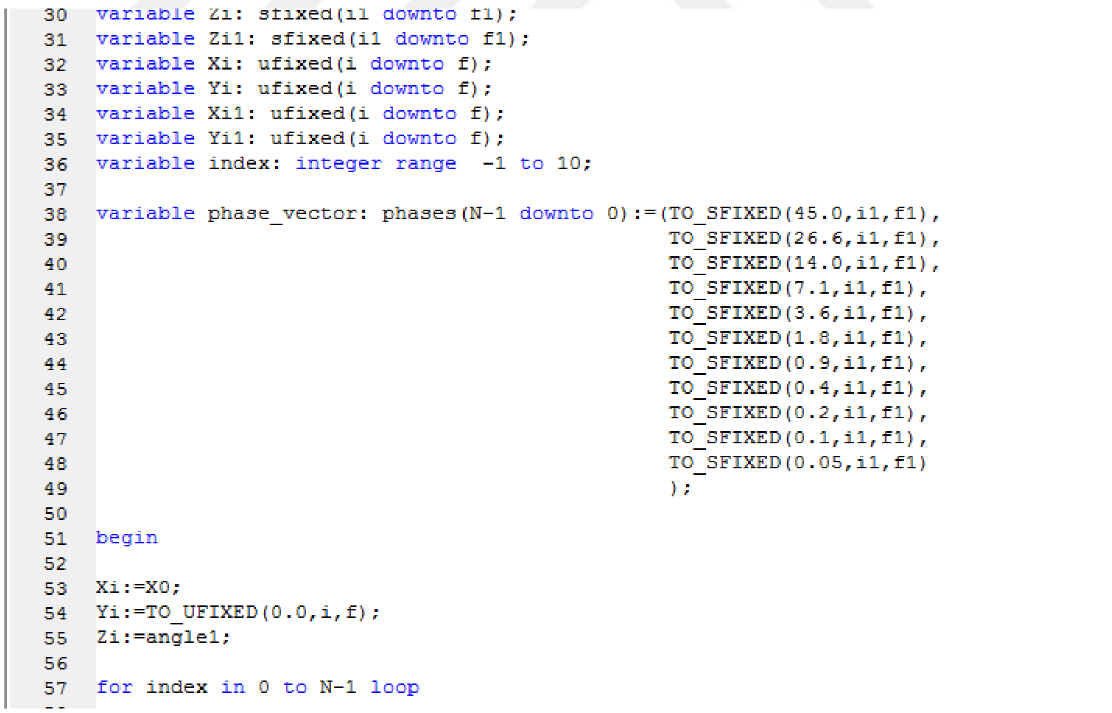


```
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use work.cordic_pakage.all;

entity cordicAlgorithm is
port(angle:in sfixed(i1 downto f1);
     cosine:out ufixed(i downto f)
     );

end cordicAlgorithm;

architecture Behavioral of cordicAlgorithm is

begin

cosine<=cordicAlgorithm(angle);

end Behavioral;
```

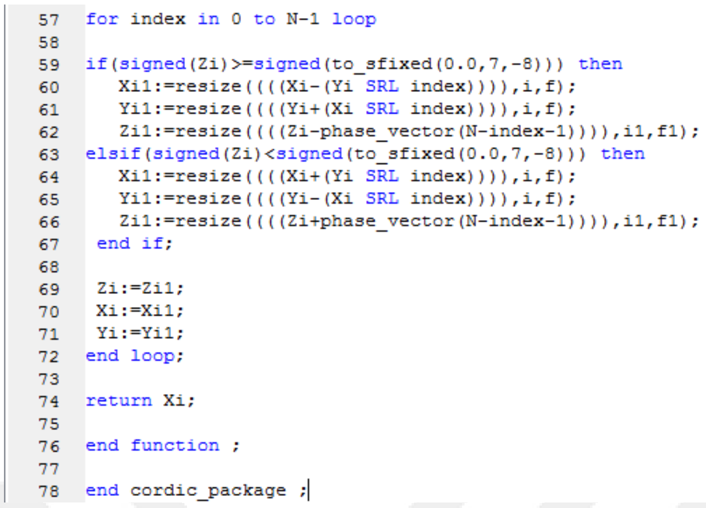
The package program



```vhdl
library ieee;
use ieee.numeric_std.all;
--
library IEEE_proposed;
use ieee_proposed.fixed_pkg.all;

package cordic_package is

constant i: integer:=0;
constant f: integer:=-7;

constant i1: integer:=7;
constant f1: integer:=-8;

constant N:integer:=11;

--signal angle2: sfixed(7 downto -8):=TO_SFIXED(60.0,7,-8);

constant X0: ufixed(0 downto -7):=TO_UFIXED(0.6072,0,-7);
type phases is array(natural range<>) of sfixed(i1 downto f1);

function cordicAlgorithm(signal angle1:sfixed) return ufixed;

end cordic_package;
-------------------------------------------------------------------------------
package body cordic_package is

function cordicAlgorithm(signal angle1:sfixed) return ufixed is

```

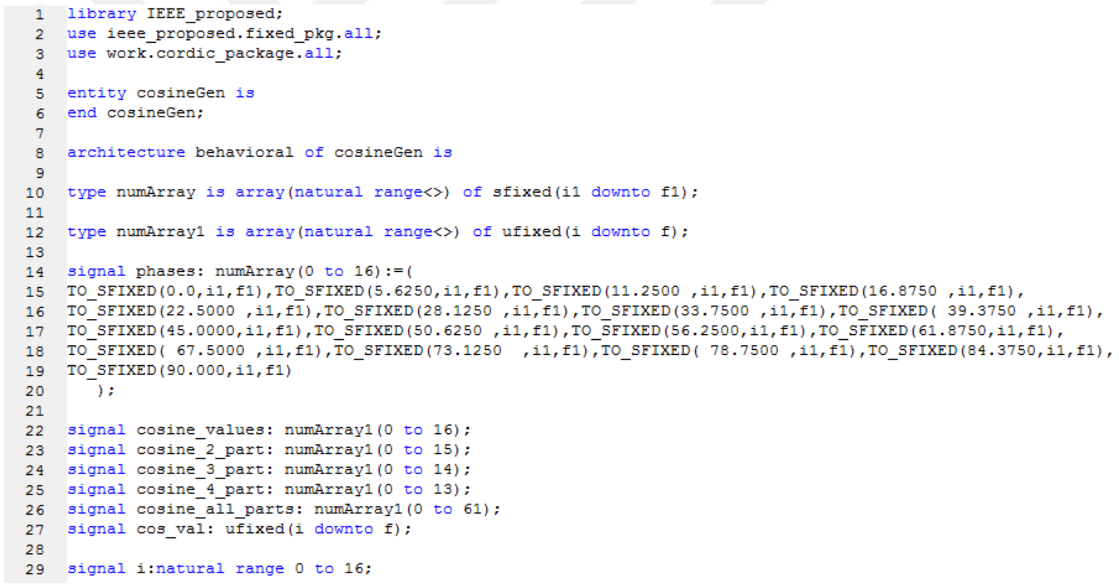


```vhdl
variable Zi: sfixed(i1 downto f1);
variable Zi1: sfixed(i1 downto f1);
variable Xi: ufixed(i downto f);
variable Yi: ufixed(i downto f);
variable Xi1: ufixed(i downto f);
variable Yi1: ufixed(i downto f);
variable index: integer range  -1 to 10;

variable phase_vector: phases(N-1 downto 0):=(TO_SFIXED(45.0,i1,f1),
                                              TO_SFIXED(26.6,i1,f1),
                                              TO_SFIXED(14.0,i1,f1),
                                              TO_SFIXED(7.1,i1,f1),
                                              TO_SFIXED(3.6,i1,f1),
                                              TO_SFIXED(1.8,i1,f1),
                                              TO_SFIXED(0.9,i1,f1),
                                              TO_SFIXED(0.4,i1,f1),
                                              TO_SFIXED(0.2,i1,f1),
                                              TO_SFIXED(0.1,i1,f1),
                                              TO_SFIXED(0.05,i1,f1)
                                              );

begin

Xi:=X0;
Yi:=TO_UFIXED(0.0,i,f);
Zi:=angle1;

for index in 0 to N-1 loop
```

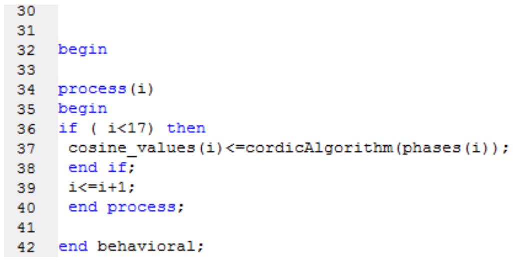


```vhdl
for index in 0 to N-1 loop

if(signed(Zi)>=signed(to_sfixed(0.0,7,-8))) then
   Xi1:=resize((((Xi-(Yi SRL index)))),i,f);
   Yi1:=resize((((Yi+(Xi SRL index)))),i,f);
   Zi1:=resize((((Zi-phase_vector(N-index-1)))),i1,f1);
elsif(signed(Zi)<signed(to_sfixed(0.0,7,-8))) then
   Xi1:=resize((((Xi+(Yi SRL index)))),i,f);
   Yi1:=resize((((Yi-(Xi SRL index)))),i,f);
   Zi1:=resize((((Zi+phase_vector(N-index-1)))),i1,f1);
 end if;

 Zi:=Zi1;
 Xi:=Xi1;
 Yi:=Yi1;
end loop;

return Xi;

end function ;

end cordic_package ;
```

Program to create a vector of sine values

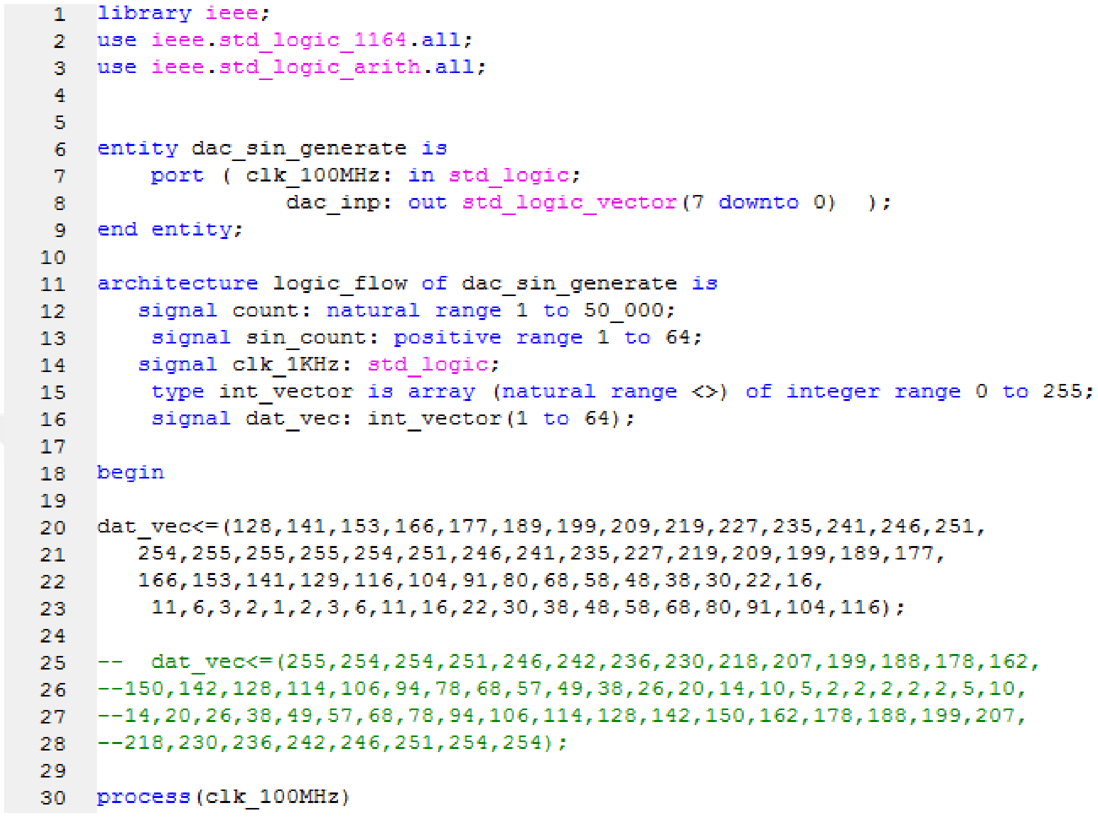


```vhdl
library IEEE_proposed;
use ieee_proposed.fixed_pkg.all;
use work.cordic_package.all;

entity cosineGen is
end cosineGen;

architecture behavioral of cosineGen is

type numArray is array(natural range<>) of sfixed(i1 downto f1);

type numArray1 is array(natural range<>) of ufixed(i downto f);

signal phases: numArray(0 to 16):=(
TO_SFIXED(0.0,i1,f1),TO_SFIXED(5.6250,i1,f1),TO_SFIXED(11.2500 ,i1,f1),TO_SFIXED(16.8750 ,i1,f1),
TO_SFIXED(22.5000 ,i1,f1),TO_SFIXED(28.1250 ,i1,f1),TO_SFIXED(33.7500 ,i1,f1),TO_SFIXED( 39.3750 ,i1,f1),
TO_SFIXED(45.0000,i1,f1),TO_SFIXED(50.6250 ,i1,f1),TO_SFIXED(56.2500,i1,f1),TO_SFIXED(61.8750,i1,f1),
TO_SFIXED( 67.5000 ,i1,f1),TO_SFIXED(73.1250  ,i1,f1),TO_SFIXED( 78.7500 ,i1,f1),TO_SFIXED(84.3750,i1,f1),
TO_SFIXED(90.000,i1,f1)
   );

signal cosine_values: numArray1(0 to 16);
signal cosine_2_part: numArray1(0 to 15);
signal cosine_3_part: numArray1(0 to 14);
signal cosine_4_part: numArray1(0 to 13);
signal cosine_all_parts: numArray1(0 to 61);
signal cos_val: ufixed(i downto f);

signal i:natural range 0 to 16;
```

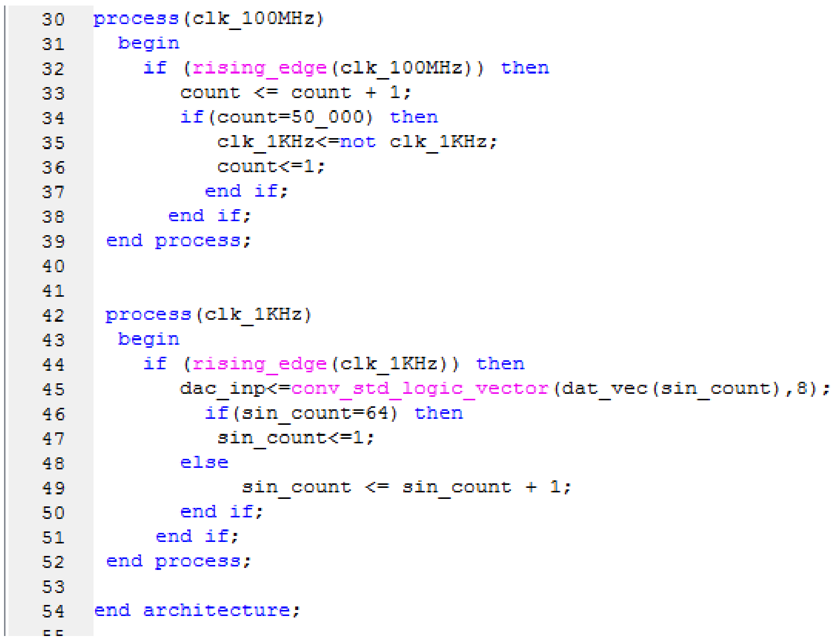


```vhdl


begin

process(i)
begin
if ( i<17) then
 cosine_values(i)<=cordicAlgorithm(phases(i));
 end if;
 i<=i+1;
 end process;

end behavioral;
```

Program to generate sine and cosine wave



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;


entity dac_sin_generate is
    port ( clk_100MHz: in std_logic;
              dac_inp: out std_logic_vector(7 downto 0)   );
end entity;

architecture logic_flow of dac_sin_generate is
   signal count: natural range 1 to 50_000;
    signal sin_count: positive range 1 to 64;
   signal clk_1KHz: std_logic;
    type int_vector is array (natural range <>) of integer range 0 to 255;
    signal dat_vec: int_vector(1 to 64);

begin

dat_vec<=(128,141,153,166,177,189,199,209,219,227,235,241,246,251,
   254,255,255,255,254,251,246,241,235,227,219,209,199,189,177,
   166,153,141,129,116,104,91,80,68,58,48,38,30,22,16,
    11,6,3,2,1,2,3,6,11,16,22,30,38,48,58,68,80,91,104,116);

--  dat_vec<=(255,254,254,251,246,242,236,230,218,207,199,188,178,162,
--150,142,128,114,106,94,78,68,57,49,38,26,20,14,10,5,2,2,2,2,2,5,10,
--14,20,26,38,49,57,68,78,94,106,114,128,142,150,162,178,188,199,207,
--218,230,236,242,246,251,254,254);

process(clk_100MHz)
```



```
process(clk_100MHz)
  begin
    if (rising_edge(clk_100MHz)) then
       count <= count + 1;
       if(count=50_000) then
          clk_1KHz<=not clk_1KHz;
          count<=1;
         end if;
      end if;
 end process;


 process(clk_1KHz)
  begin
    if (rising_edge(clk_1KHz)) then
       dac_inp<=conv_std_logic_vector(dat_vec(sin_count),8);
         if(sin_count=64) then
          sin_count<=1;
       else
            sin_count <= sin_count + 1;
       end if;
     end if;
 end process;

end architecture;
```

And then connect R-2R ladder and the output of R-2R ladder connect to oscilloscope.

We get the figures of waves.

We designed R-2R ladder in order to use it in our circuit as shown in Fig. 26.

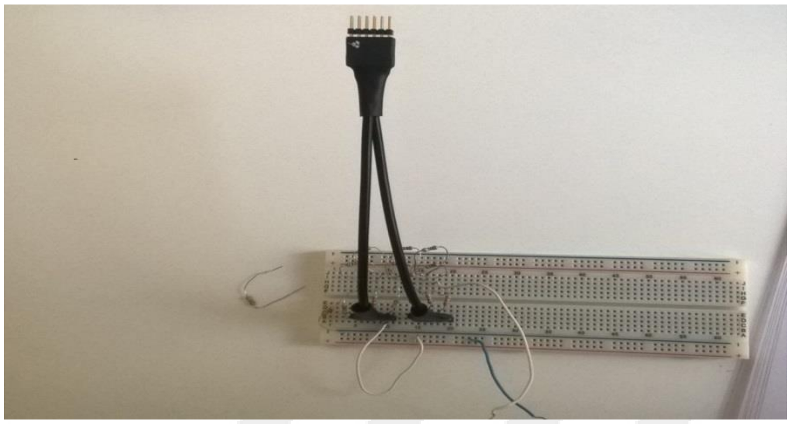


Figure 26 R-2R ladder

We connect all the parts of our circuit as shown in Fig. 27.

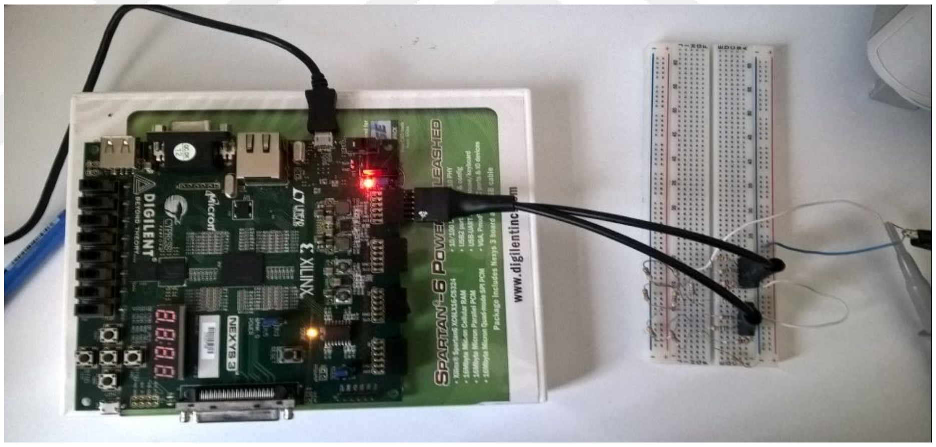


Figure 27 practical circuit

We get the waveform shown in Fig. 28.

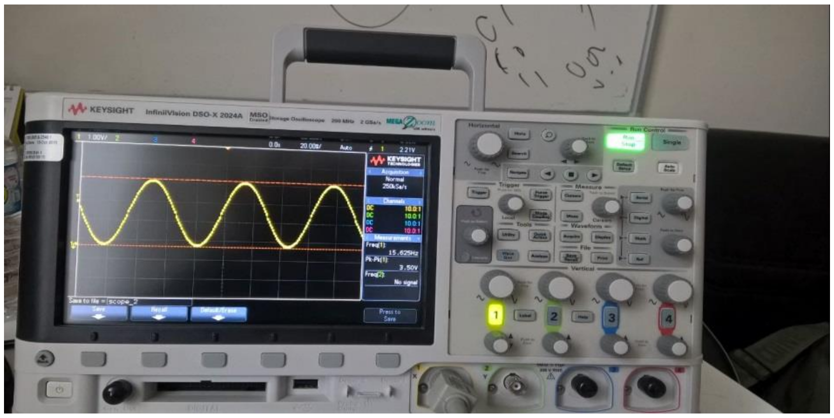


Figure 28 The waveform we generated it.

The Fig. 29 shows the sin wave we generated.

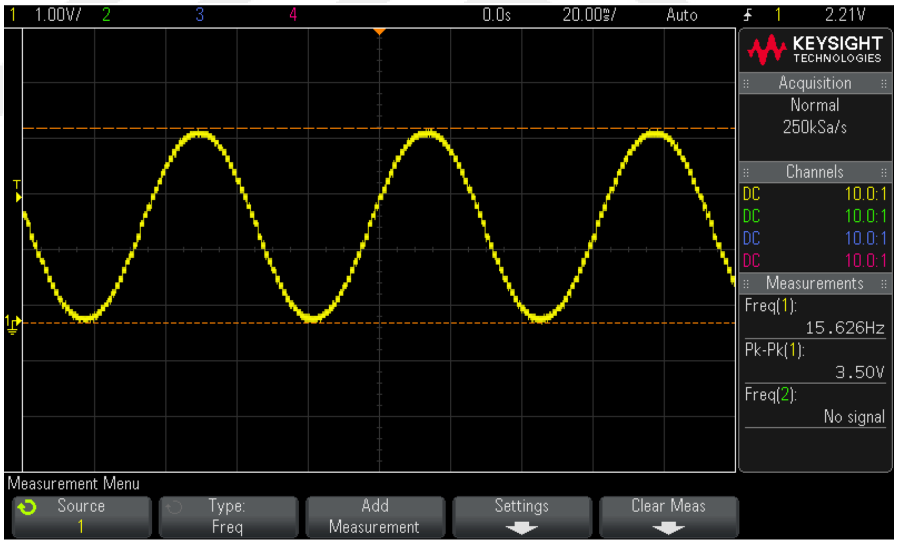


Figure 29 The generated sine wave

CHAPTER SIX

Conclusions and Future Works

In this thesis work, we studied the CORDIC algorithm in details. We implemented the algorithm in MATLAB platform and inspected its iteration number considering the accuracy of the calculated sine or cosine value. Later on, we implemented the algorithm on FPGA platform using the VHDL language.

Considering the hardware complexity of the CORDIC algorithm, we proposed a simpler technique for the generation of trigonometric waveforms. In this method, we first generate the signal in MATLAB platform and take samples from the generated signal and then represent each simple by an integer value. considering the ration between samples. Later, the obtained integer sequence is carried on to the VHDL program and using the clock frequency of the FPGA any sine signal with any frequency is generated. The proposed method is simple and effective to use, and very flexible. In fact, the method used for the generation of sine waves can be used for the generation of any waveform used in communication, such as square root raised cosine filter design, low pass filter design etc.

Future Work

1- Design and implementation of multistage band pass filter on FPGA.

2- FPGA implementation of special DSP processor.

3- Implementation of frequency analysis of real discrete signal.

4- Hardwar implementation of Ethernet signal analysis on FPGA.

REFERENCES

1. **ISE in depth tutorial “**www.xillinx.com'

2. **Volnei A. Pedroni “**Circuit Design with VHDL**”**, MIT Press Cambridge, Massachusetts London, England, 2004.

3. **Uwe Meyer-Baese “**Signals and Communication Technology', Springer-Verlag Berlin Heidelberg 2014.

4. **B. Lakshmi and A. S. Dhar “**CORDIC Architectures: A Survey**”** Hindawi Publishing Corporation, Volume 2010, Article ID 794891, 19 pages doi:10.1155/2010/794891.

5. **J. E. Volder, “**The birth of CORDIC,**”** Journal of VLSI Signal Processing, vol. 25, no. 2, pp. 101–105, 2000.

6. **M. D. Erecegovac and T. Lang, “**Digital Arithmetic”, Elsevier, Amsterdam, The Netherlands, 2004.

8. **Eva Murphy and Colm Slattery “**www.analog.com/dds**”**

9. **Antonius P. Renardy∗, Nur Ahmadi, Ashbir A. Fadila, Naufal Shidqi, Trio Adiono “**FPGA Implementation of CORDIC Algorithms for Sine and Cosine Generator**”** The 5th International Conference on Electrical Engineering and Informatics 2015 August 10-11, 2015, Bali, Indonesia

APPENDIX

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Shakir Salman Ahmad

Nationality: Iraqi

Date and Place of Birth: 23.Feb.1974, Mosul, Iraq

Marital status: Married

Phone: 009647701638934

E-mail: shakirsalman3@gmail.com

EDUCATION

Degree	**Institution**	**Year of Graduation**
M.Sc.	Çankaya University College of engineering Electronic and Communication Department	2017
B.Sc.	AL- Mosul University	1998

WORK EXPERIENCE

Year	**Place**	**Occupation**
1999- Present	Ministry of Electricity	Department Manager