

FPGA BASED SIGNAL GENERATION AND
ACQUISITION

*Project report submitted in partial fulfillment of the requirement for the
degree of*

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

ASHISH VERMA (161092)

YATHARTH GAUTAM (161098)

UNDER THE GUIDANCE OF

Dr. HARSH SOHAL



**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,
WAKNAGHAT**

MAY 2020

TABLE OF CONTENTS

CAPTION	PAGE NO.
DECLARATION	5
ACKNOWLEDGEMENT	6
LIST OF ACRONYMS AND ABBREVIATIONS	7
LIST OF FIGURE	9
BLOCK DIAGRAM	11
ABSTRACT	12
CHAPTER-1	13
INTRODUCTION	
1.1 WHAT IS ECG	13
1.2 HOW DOES IT WORK	13
1.3 ECG INTERPRETATION	15
1.1.3 WHAT ARE USE OF ECG	16
1.2 INTRODUCTION TO FPGA	17
1.2.1 THE PRINCIPLES OF FPGA	17
1.2.2 FPGA ARCHITECTURE	18
1.2.2 KINDS OF FPGA	19
1.2.3 APPLICATIONS OF FPGA	22
CHAPTER-2	23
LITERATURE REVIEW	
2.1 QRS DETECTION TECHINIQUES	23
2.1.1 By Shukla et	23
2.1.2 By Dokur et al	23
2.1.3 By Kesel brener et al	23

CHAPTER-3	25
SYSTEM COMPONENT	
3.1 ZYNQ ZBROAD 7000	25
3.1.1 FEATURES	26
3.2 ADC	29
3.3 ELECTROCARDIOGRAM(ECG)	30
3.3.1 FUNCTION OF ECG	31
3.4 LM25(TEMP SENSOR)	32
3.4.1 FEATURES	32
3.5 DDS	33
3.6 QUARTUS II	35
3.7 VIVADO	36
REFERENCES	37
APPENDIX	38
APPENDIX A	
A.1 DAC_AMP_CONT	38
A.1.1 RTL ANALYSIS	39
A.1.2 WAVEFORM	39
A.2 DAC_REG_CONT	40
A.2.1 RTL ANALYSIS	41
A.2.2 WAVEFORM	41
A.3 DAC_SPI	42
A.3.1 RTL ANALYSIS	44
A.3.2 WAVEFORM	45
A.4 DAC_TIMING_UNIT	46
A.4.1 RTL ANALYSIS	47
A.4.2 WAVEFORM	47

A.5 GENERATE_CONSTANT	48
A.5.1 RTL ANALYSIS	49
A.5.2 WAVEFORM	49
A.6 PSEUDO_GENERATE_CONSTANT	50
A.6.1 RTL ANALYSIS	51
A.6.2 WAVEFORM	51
A.7 PSEUDO_WG_CONTROL_UNIT	52
A.7.1 RTL ANALYSIS	53
A.7.2 WAVEFORM	53
A.8 TOP MODULE	54
A.9 SINE WAVE	55
APPENDIX B	56
APPENDIX C	
DATA SHEET OF ZBOARD	57
CONCLUSION	60

DECLARATION BY THE SCHOLAR

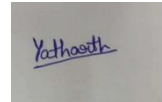
We hereby declare that the work reported in the B.tech project entitled **"FPGA BASED SIGNAL GENERATION AND ACQUISITION"** submitted at **Jaypee Univerity Of Information Technology,Waknaghat India**, in an authentic record of our work carried out under the supervision of **Dr.Harsh Sohal** (Assistant Professor, Department Electronics and Communication Engineering).We have not submitted this work elsewhere for any other dergree or diploma.



(student signature)

ASHISH VERMA

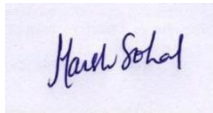
161092



(student signature)

YATHARTH GAUTAM

161098



Dr. HARSH SOHAL

Department of Electronics and Communication

Jaypee University of Information Technology, Waknaghat , India

Date :

ACKNOWLEDGEMENT

The fulfillment and rapture that go with the fruition of the venture would be inadequate without the notice of the individuals who made it conceivable.

We might want to accept the open door to thank and communicate their profound feeling of appreciation to our staff tutor Dr. Harsh Sohal for giving his significant direction at all phases of the examination, his recommendation, productive proposals, positive and steady mentality and consistent consolation, without which it would have not been conceivable to finish the venture. The gift, help and direction given by him an opportunity to time will convey us far in the excursion of life on which we are going to set out upon.

We are obliged to all the employees of JUIT, for the important data gave by them in their individual fields. We are appreciative for their participation during the time of our task

BY: ASHISH VERMA (161092)
YATHARTH GAUTAM (161098)

LIST OF ACRONYMS AND ABBREVIATIONS

FPGA: Field Programmable Gate Array.

DDS: Direct Digital Synthesis

DAC: Digital To Analog Converter

ADC: Analog to Digital Converter.

ECG: Electrocardiogram

VHDL: VHSIC Hardware Description Language

RTL: Register Transfer Level

PLD: Programmable logic devices

HDL: Hardware description language

I/P: Input

O/P: Output

ASIC: Application-specific integrated circuit

PRJM: Programmable read-just memory

SoC: system on a chip

ENOB: Effective number of bits

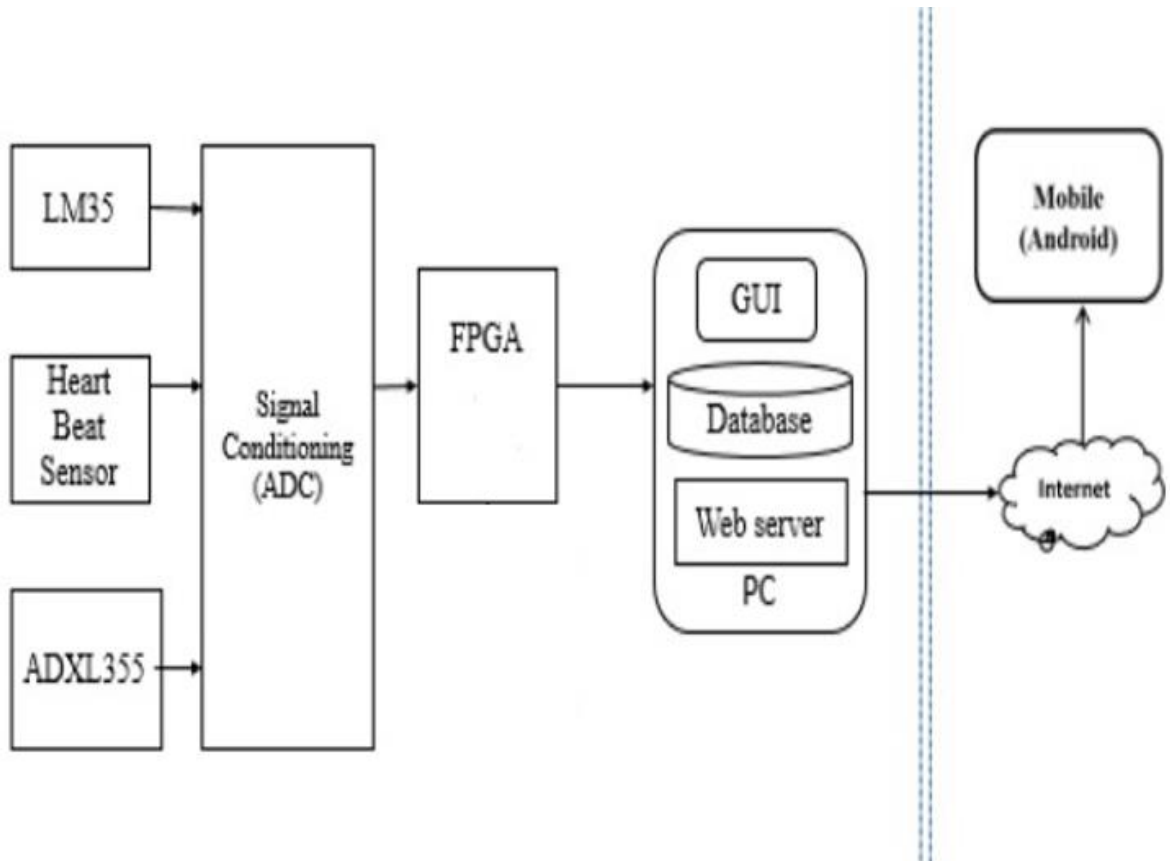
SNR: signal-to-noise ratio

LIST OF FIGURES

S.NO.	NAME OF FIGURE	PAGE NO.
1.	Figure 1.1 typical ECG signal	14
2.	Figure 1.2 ECG diagram recorded on Philips framework1	15
4.	Figure 1.3 symmetrical arrays architecture	19
5.	Figure 1.4 row_based architecture	20
6.	Figure 1.5 hierarchical PLDs	21
7.	Figure 3.1 zynq zboard 7000	25
8.	Figure 3.2 zynq zboard block diagram	27
9.	Figure 3.3 pin assignment og zboard	28
10.	Figure 3.4 Analog to digital converter	29
11.	Figure 3.5 ECG waveform	31
12.	Figure 3.4 LM25	32
13.	Figure 3.5 direct digital synthesis block diagram	34
14.	Figure 3.6 quartus II	35
15.	Figure 3.7 vivado	36
16.	Figure A.1 rtl analysis of DAC_AMP_cont	39
17.	Figure A.1.1 waveform of DAC_Amp_cont	39
18.	Figure A.2 RTL analysis of Dac_reg_cont	41
19.	Figure A.2.1 waveform of Dac_reg_cont	41
20.	Figure A.3.1 RTL analysis of DAC_SPI	44
21.	Figure A.3.2 waveform of DAC_SPI	45
21.	Figure A.4.1 RTL analysis of DAC_Timing_Unit	47
23.	Figure A.4.2 waveform of DAC_Timing_Unit	47
24.	Figure A.5.1 RTL analysis of Generate_Constant	49
25.	Figure A.5.2 wave form of Generate_Constant	49
26.	Figure A.6.1 RTL analysis of pseudo_generate_cont	51
27.	Figure A.6.2 waveform of pseudo_generate_cont	51

28.	Figure A.7.1 RTL analysis of pseudo_wg_control_unit	53
29.	Figure A.7.2 waveform of pseud_wg_control_unit	53
30.	Figure A.8 top module	54
31.	Figure A.8 sine wave	55

BLOCK DIAGRAM



ABSTRACT

This task is to structure a FPGA-based data acquisition system. This framework understands the physical sign securing, simple sign to computerized signal transformation and information stockpiling. FPGA, as the center of the information securing arrangement of ECG information, gathers and stores the information. This framework is isolated into three modules: the front-end signal handling module, FPGA information obtaining module, and information stockpiling module. The FPGA information procurement module is planned by the VHDL and reproduced by the Quartus II programming. This framework has the benefit of being a basic unit with low force utilization and being utilized to gather information from various sensors. The reenactment configuration utilizes Direct Digital Synthesis(DDS) classifier to group the info ECG signal. The framework is actualized on Zynq Zedboard7000 Field Programming Gate Array (FPGA) board. We are utilizing here FPGA rather than some other chip, for example, ARDINO as FPGA are reprogrammable and can later be changed by our need later on. Initially we are building up a ROM in which we will spare the ECG waveform then we will invigorate it and contrast it and other spared information.

CHAPTER 1

INTRODUCTION

This section gives a concise prologue to ECG. Clarifies what is ECG and the electrical action of the heart bringing about age of 5 unmistakable waves and how ECG information is deciphered to get the pulse from the electrical sign. This section also have the introduction about the FPGA and its principal of working and types of FPGA.

1.1 What is ECG

1.2

ECG is a type of medical test that shows the heart's electrical working which helps in understanding the plus of the heart and if any type of irregularities is present there. The outcome of this test is known as "electrocardiogram". The heart contract due to electrical signal received from the sinoatrial node. These electrical impulses are detected by ECG. It is the mostly used technique when a person having any trouble with breathing, chest pain etc.

1.1.1 HOW DOES IT WORKS

Electrical signal gets triggered due to heart muscle depolarization which happens with each and every heartbeat. Each cell of the heart muscle has a negative charge to it, these negative charge are along with the heart membrane when the heart is at rest. The depolarization is a technique of decreasing this negative charge with the help of cations like Na^{++} and Ca^{++} . These cations cause the heart muscle to contract. With every heart beat a healthy heart will show progression of a wave of depolarization that is triggered by the cells.

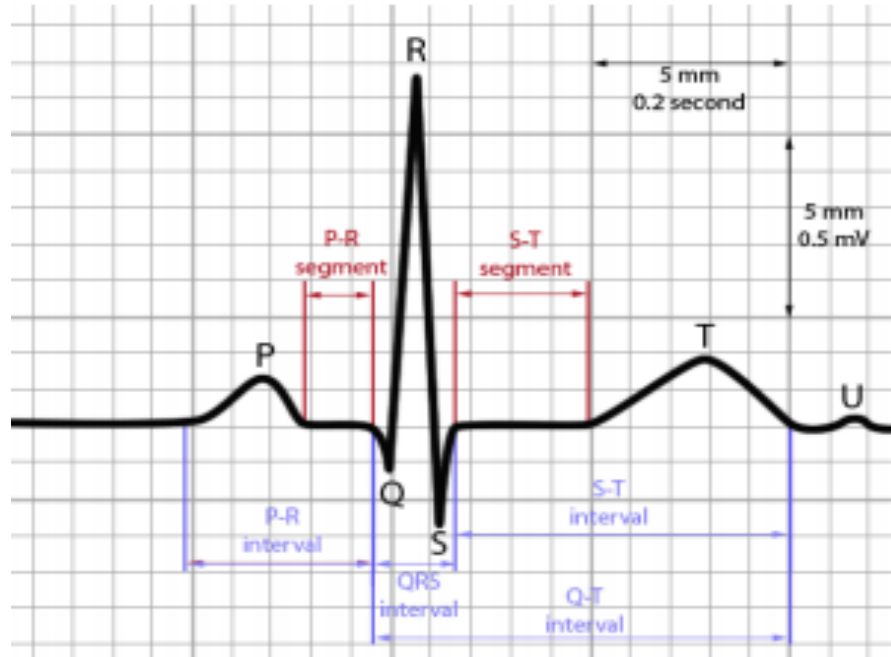


Figure1.1-Typical ECG signal

- P represents the atrial contractions
- QRS shows the ventricular contraction. R indicates the heartbeat.
- T is the last common wave in an ECG. It is produced when the ventricle are repolarizing.
- The letters utilized in the ECG signal portrayal don't have shortened forms in clinical phrasing.

1.1.2 ECG INTERPRETATION

The output which we get from ECG is in graphical format which is on the x-axis we have time and on y-axis we have voltage. The voltage is known as isoelectric line. When no signal is present this would be a flat line. QRS consist of a three graphical deflections. Overall ECG have five deflection P, Q, R, S, T.

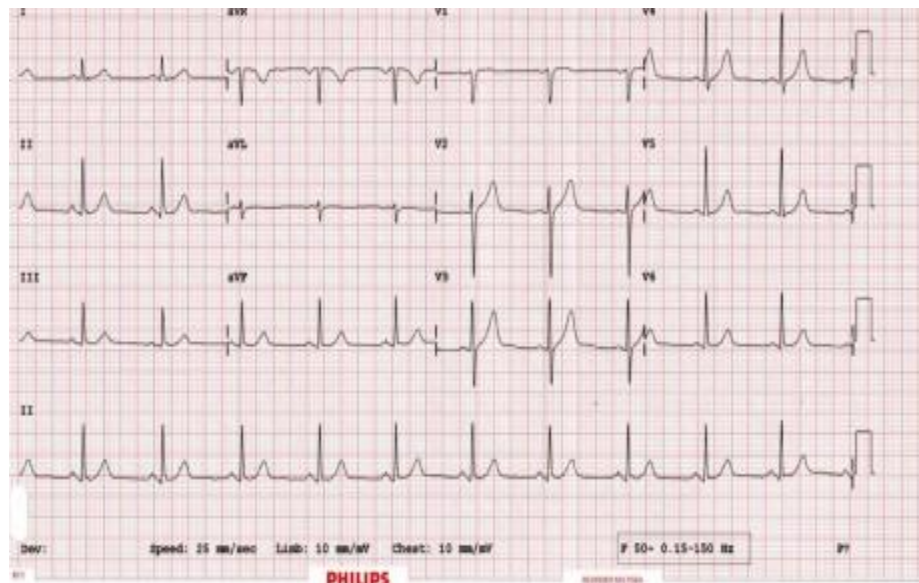


Figure1.2 - ECG diagram recorded on Philips framework 1

1.1.3 What are the employments of ECG?

- assess your heart mood.
- diagnose poor blood stream to the heart muscle (ischemia)
- diagnose a cardiovascular failure.
- diagnose variations from the norm of your heart, for example, heart chamber growth and strange. electrical conduction.

1.2 INTRODUCTION TO FPGA

A field-programmable gate array (FPGA) is an incorporated circuit intended to be arranged by a client or an originator in the wake of assembling – henceforth the expression "field-programmable". The FPGA arrangement is commonly determined utilizing a hardware description language (HDL), like that utilized for an application-specific integrated circuit (ASIC). Circuit outlines were recently used to indicate the arrangement, yet this is progressively uncommon because of the approach of electronic plan computerization instruments.

1.2.1 The Principles of FPGAs

FP gateway shows (FPGAs) are introduced more than 3 decades sooner, and starting now and into the not all that far off they have made, offering way to deal with oversee new events of FPGAs with better legitimization thickness and execution that can be used in an irrefutably far reaching level of employments.

The first FPGA was imagined by Ross Freeman (prime supporter of Xilinx) in 1985 and beginning now and into the not so distant their premise limit has refreshed incredibly and they have become an eminent decision considering the way that FPGA structures can be changed in the wake of gathering to execute the client's last required application. Some FPGAs can be reproduced tenacious occasions and some obliged occasions.

Everything contemplated term, FPGAs are arrange able on SC with a variety of organized guard squares included by I/p or o/p incapacitates which are amassed via programmable interconnect focal points for end up being any kind of modernized circuit or structure. FPGAs created utilizing (PROM) and method of reasoning device (PLDs).

Instead of processor, FPGA are extremely proportionate in character. Every free dealing with task is offered out to a submitted area of the chip. Accordingly, the presentation of the

slightest bit of the application isn't influenced when all the all the all the more preparing tries are consolidated.

1.2.2 FPGA ARCHITECTURE

A distinct arrangement of a **FPGA** contrasts from producer to maker. Now here, we show a nonexclusive FPGA from that have the going with parts:

- Programmable reason squares: squares may be shaped from a gigantic number of transistors to countless transistor. They execute as far as possible required by the structure and include support areas, for example, transistor sets, LUTs, Carry & Control Logic (flip lemon and multiplexers).
- Programmable I/O squares: Accomplish premise obstructs with outside bits by strategies for interfacing pins.
- Programmable resources: These are electrically programmable interconnection (pre-laid non horizontal and on a level plane) that give the directing course to the programmable strategy for thinking squares. Planning ways contains wire bits of fluctuating lengths which can be interconnected by strategies for switches that are electrically programmable. FPGA thickness depend on the measure of pieces within utilized for directing ways.

1.2.2 Kinds of FPGAs

The steering engineering influences thickness and execution of the FPGA. In light of interior course of action of squares, FPGAs may be grouped into three classifications:

1. Symmetrical clusters: This technique contains reason squares composed in lines and segments of a cross section as well as interconnected assets among them. This sensible structure is encircled by inputs/outputs obstructs that interface it to the outside world

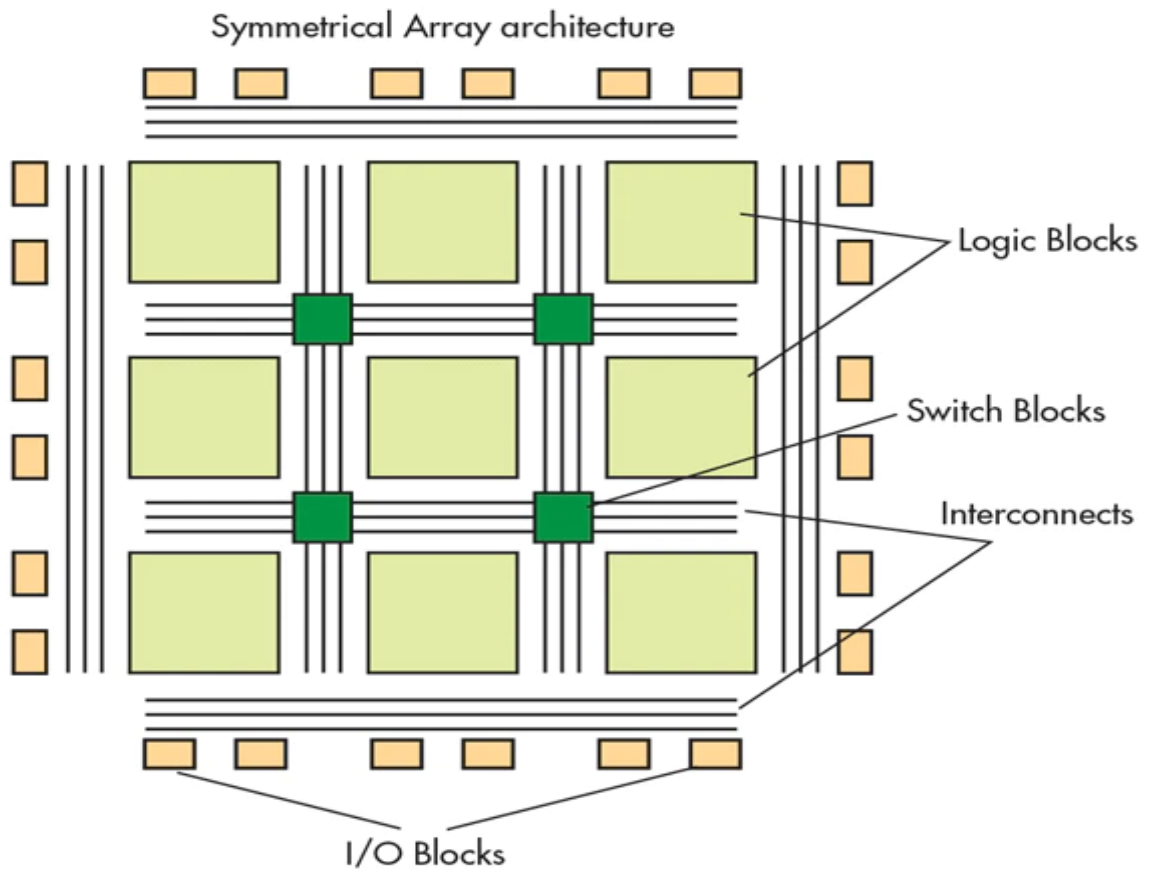


FIGURE1.3 Symmetrical clusters architecture

Adjusted groups include a two-dimensional demonstration of technique for thinking modules interconnected by vertical and even programmable interconnect assets.

2 Row-based architectures: It substitutes segments of programmable interconnect resources with lines of method of reasoning squares while the I/O blocks are arranged inside the edges of the lines by line which might be related with neighboring segments through vertical interconnect.

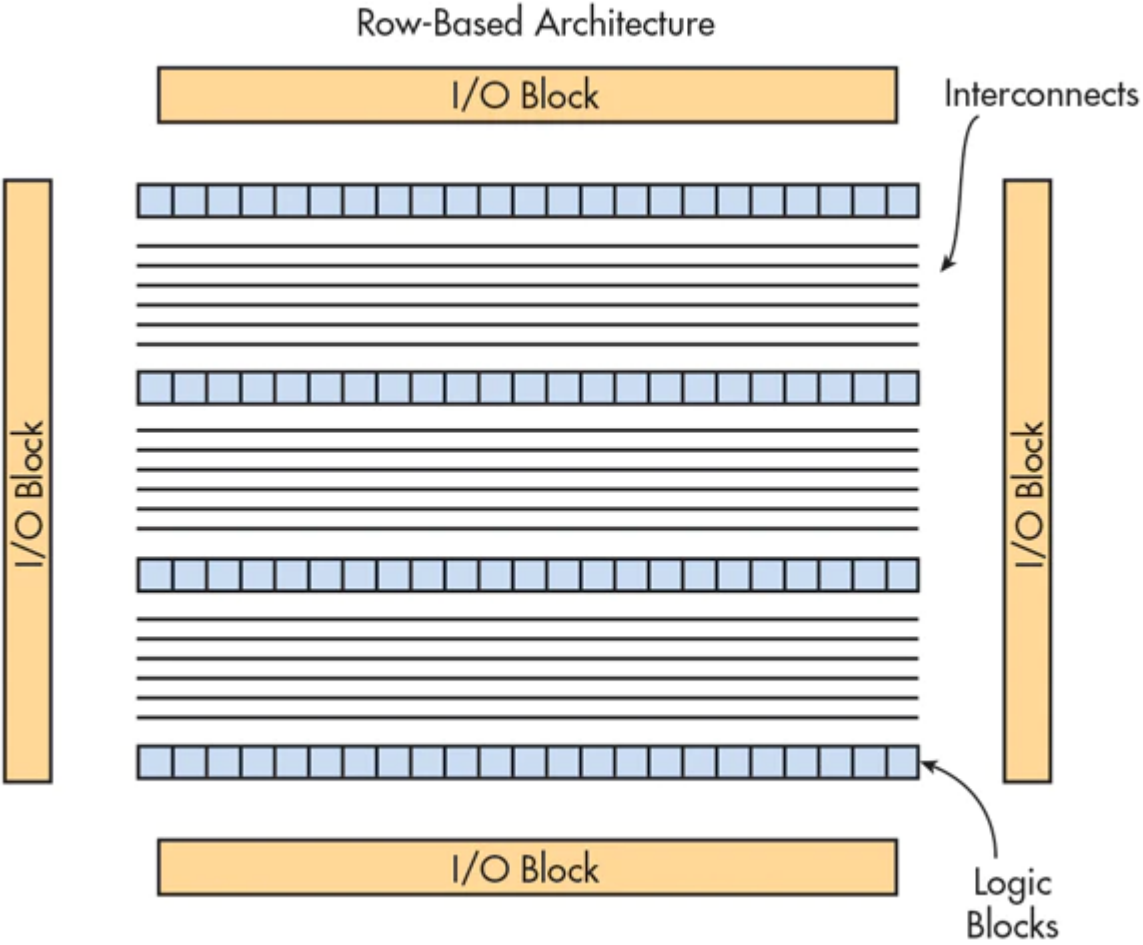


FIGURE 1.4 Row-based architecture

Row-based design involves lines of basis blocks that are detached by programmable interconnect resources

- **Hierarchical PLDs:** They are organized of different leveled route with the top level containing just basis squares and interconnect. Every logical square contain various logical modules. What's more, every logical module has combinatorial just as successive useful components.

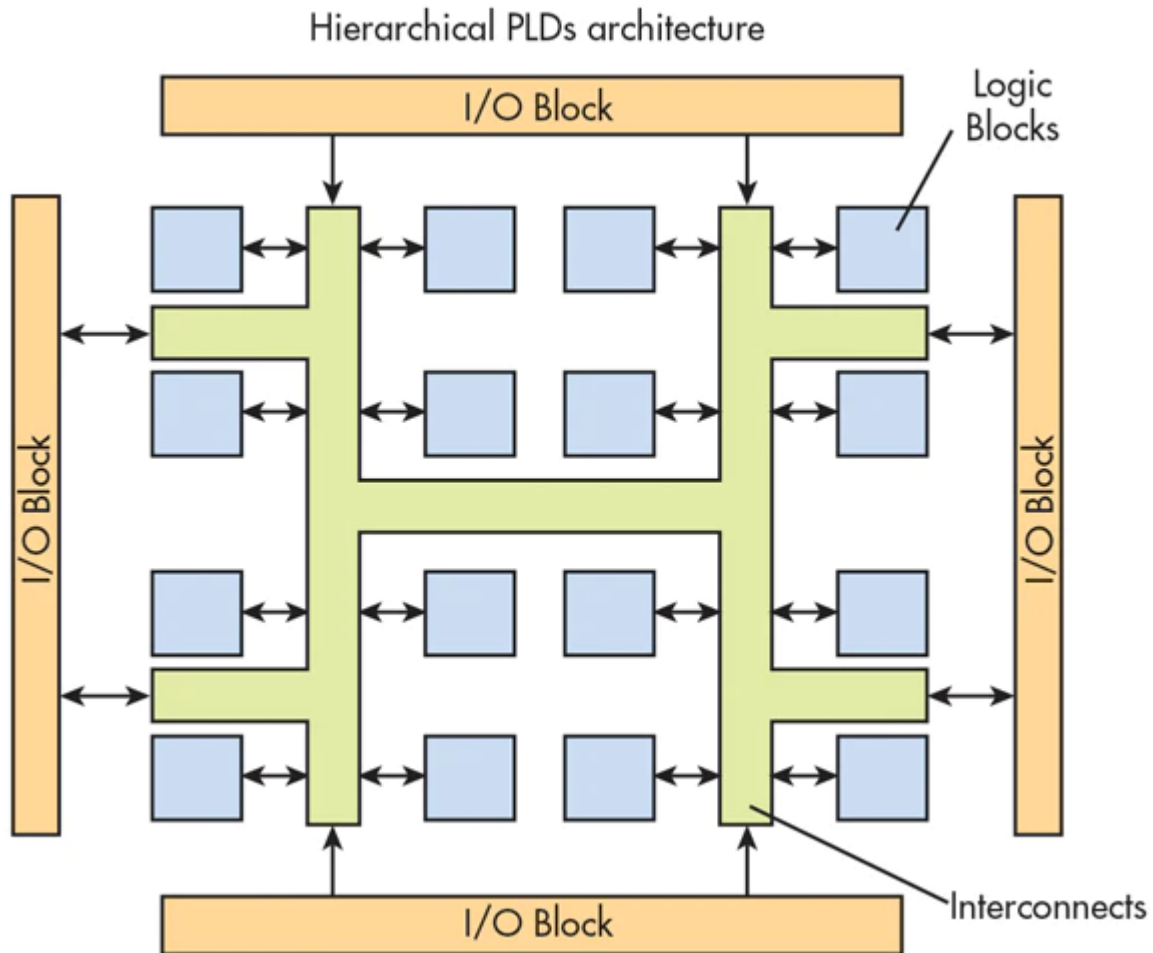


FIGURE1.5 • Hierarchical PLDs

In this type every rationale block contains number of rationale modules.

1.2.3 APPLICATION OF FPGA: -

Present day FPGAs are utilized over a few markets; the absolute most recent applications are:

1. **Energy:** Reusable noteworthy resources (e.g., sun energy and wind energy) are solid and are found considerably increasingly a significant part of the time as a component of a sharp framework where measures are as of not long ago making. An ideal control for stunning lattices foresees that start should finish correspondences and fruitful force systems, particularly in (T&D) substation. For robotization, gear have to screen, control & secure the structure endlessly for powerfully practical association of peak request loads.
2. **Automotive:** Micro semi FPGAs draw in vehicle extraordinary apparatus makers (OEMs) and providers to create inventive security applications, for example, adaptable journey control, influence dodging, and powerless side impugning.
3. **Aerospace and Defense:** Xilinx offers rad-hard and rad-liberal FPGAs that meet the showcase, suffering quality, and lifecycle requesting of over the top conditions, while connecting more basic adaptability than possible with standard ASIC executions.

CHAPTER 2

LITERATURE REVIEW

2.1 QRS DETECTION TECHNIQUES

2.1.1 By Shukla et

A Fast and Accurate FPGA based QRS discovery System" portrays a FPGA (Field Programmable Gate Array) framework for ECG investigation that joins a QRS location calculation that has a precision of around 96% that uses roughly 76% of the gadget being referred to. The structure comprises of two phases: a preprocessing stage followed by a pinnacle identification and middle based edge count stage. In their structure a postponed variant of the low passed signal is utilized to adequately acquire the area of the pinnacle of the R wave in the first signal.

2.1.2 By Dokur et al

Performed ECG waveform location by utilizing a counterfeit neural system. In their work the pinnacle of the R wave is first recognized and afterward highlight vectors are framed by utilizing the amplitudes of the huge recurrence parts of the DFT range. The idea of a "Develop and Learn" process is then applied to examine the ECG waveforms. The outcomes in this paper presents the precision of the framework, anyway the structure multifaceted nature as far as equipment or programming isn't referenced.

2.1.3 By Kesel brener et al

Nonlinear high pass channel for R-wave discovery in ECG signal" introduced a straightforward and effectively executed technique for R-wave location from ECG signals. Their technique depends on deduction of a sifted rendition of sign from the first sign. Edge location is then performed on the separated sign. Their outcomes are introduced for a

mimicked signal with sinusoidal and step benchmark floats just as ECG complex shape changes. The structure depicted in this paper fuses a channel as the underlying stage in FPGA usage. In the wake of going through the channel the sign is handled further in the time area to identify the interim between the pinnacles of the R waves.

A tale strategy for the recognition of QRS buildings in electrocardiographic signs that depends on a component got by checking the quantity of zero intersections for every time fragment is introduced by Kholer et al . This is by all accounts a fairly basic technique to identify R tops however brings about a calculation that gives an affectability of 99.70% and a positive predictivity of 99.57% when assessed against the MIT-BIH arrhythmia database.

CHAPTER 3

SYSTEM COMPONENT

3.1 ZYNQ ZBOARD 7000:

ZedBoard is a minimal effort improvement board for the Xilinx Zynq-7000 programmable SoC .This board contains all things needed to make a Linux, Android, Windows®, or different OS/RTOS based structure. Furthermore, a few development connectors uncover the handling framework and programmable rationale information and yield for simple client get to. Exploit the Zynq-7000 SoCs firmly coupled ARM handling framework and 7-arrangement programmable rationale to make one of a kind and incredible structures with the ZedBoard. Target applications incorporate video handling, engine control, programming increasing speed, Linux/Android/RTOS advancement, inserted ARM preparing, and general Zynq-7000 SoC prototyping.



Figure 3.1 ZYNQ ZBOARD 7000

3.1.1 Features:

- PS and PL I/O development (FMC, Pmod, XADC)
- Simple Devices ADV7511 High Performance 225 MHz HDMI Transmitter (1080p HDMI, 8-piece VGA, 128x32 OLED)
- Simple Devices ADAU1761 SigmaDSP® Stereo, Low Power, 96 kHz, 24-Bit Audio Codec
- USB OTG 2.0 and USB-UART
- 10/100/1000 Ethernet
- Installed USB-JTAG Programming
- 4 GB SD card
- 256 MB Quad-SPI Flash
- 512 MB DDR3
- Double center ARM Cortex™-A9
- Xilinx Zynq-7000 AP SoC XC7Z020-CLG484
- General Zynq-7000 AP SoC prototyping
- Installed ARM handling
- Linux/Android/RTOS improvement
- Programming speeding up
- Engine control
- Video preparing

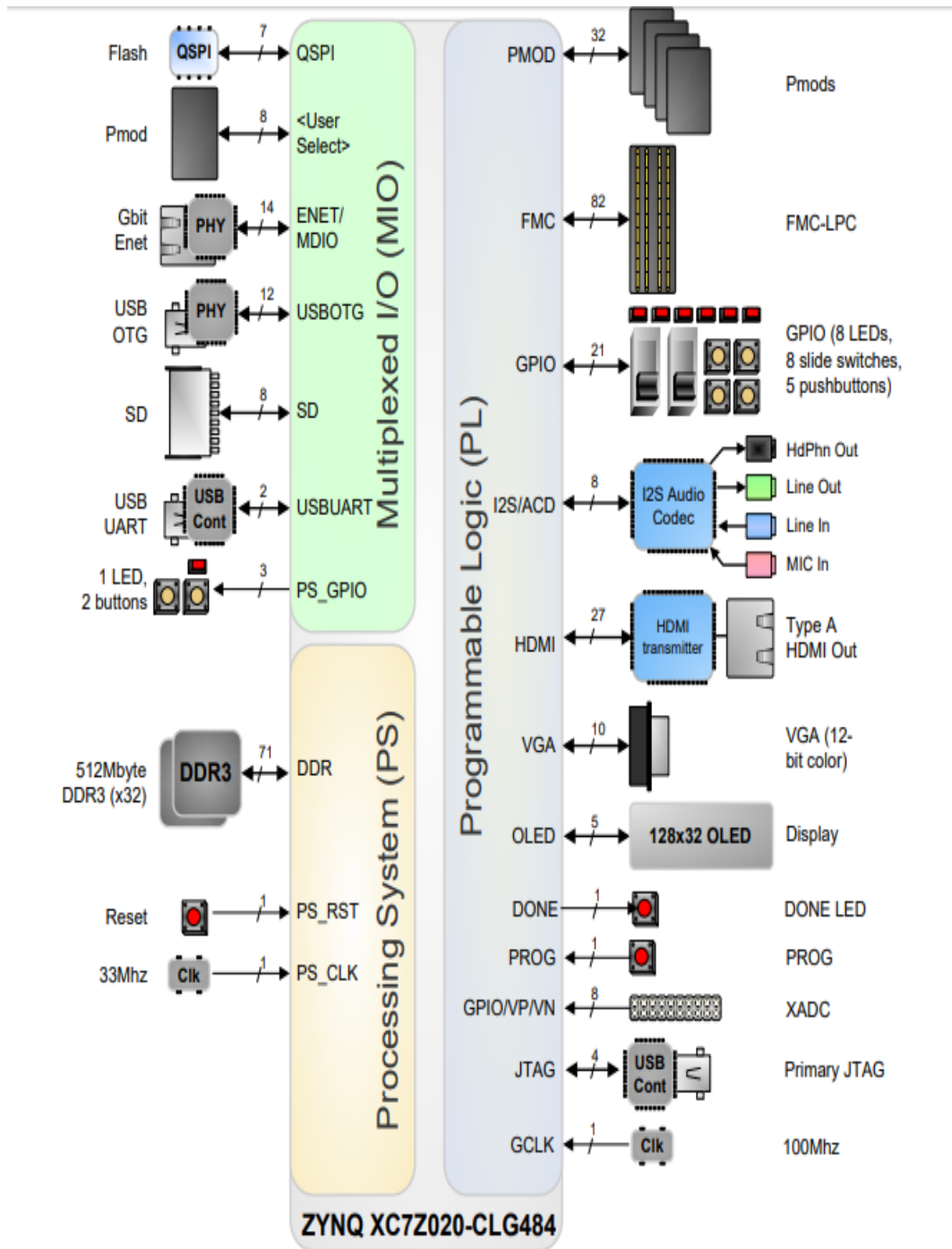


FIGURE 3.2 ZYNQ ZBOARD BLOCK DIAGRAM

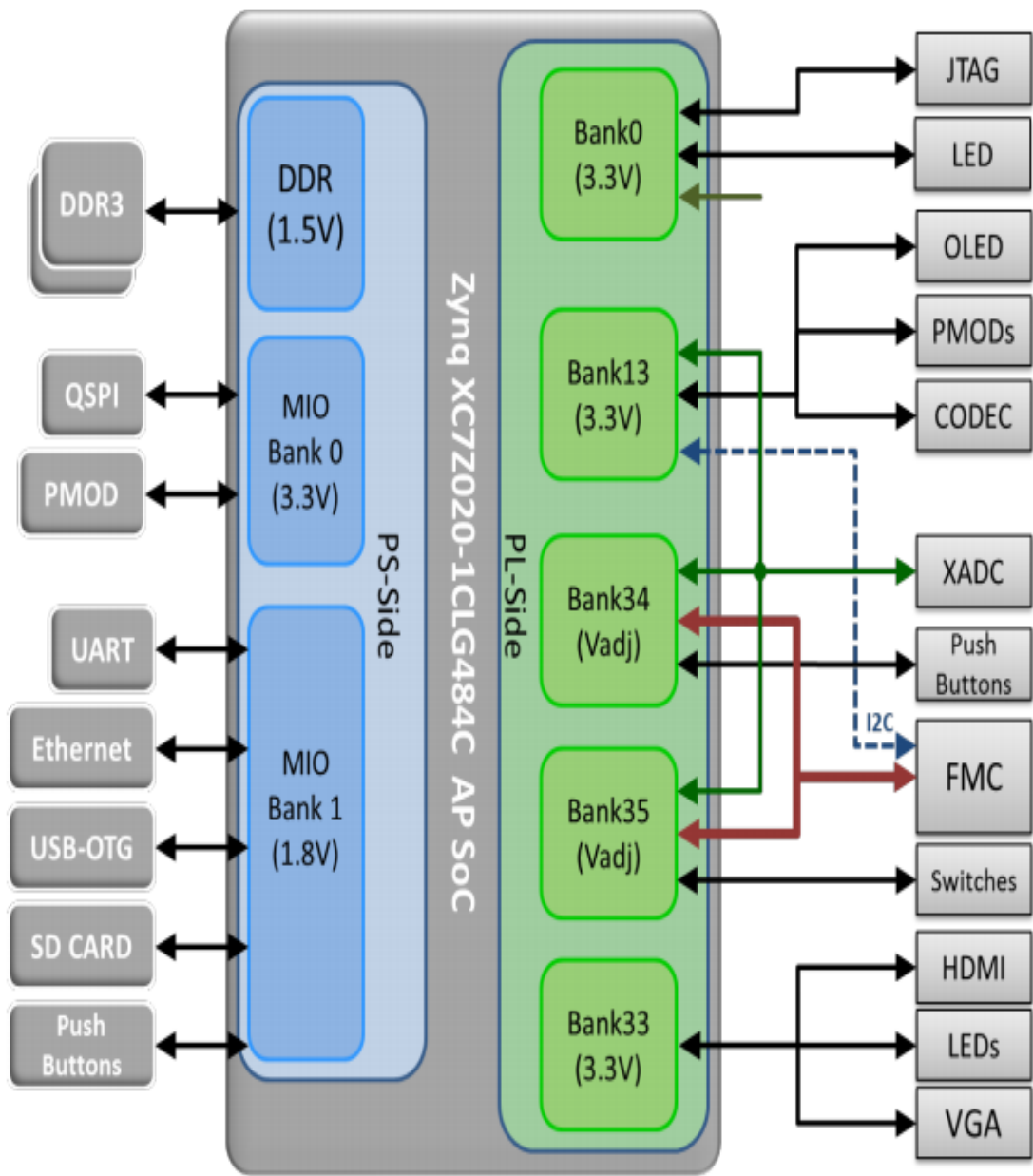


FIGURE 3.3 PIN ASSIGNMENT OF ZOARD

3.2 ADC:

An ADC changes over a steady time and constant sufficiency basic sign to a discrete-time and discrete-plentifulness electronic sign. The change incorporates quantization of the data, so it in a general sense presents a constrained amount of goof or upheaval. Plus, rather than constantly playing out the change, an ADC does the change every so often, testing the data, obliging the admissible data transmission of the information signal.

The presentation of an ADC is basically portrayed by its trade speed and sign to-commotion proportion (SNR). The transmission furthest reaches of an ADC is depicted on an extremely fundamental level by its breaking down rate. The SNR of an ADC is influenced by different elements, including the goals, linearity and exactness (how well the quantization levels orchestrate the bona fide essential sign), accomplice and jitter. The SNR of an ADC is from time to time summed up like its viable number of bits (ENOB), the measure of bits of each measure it returns that are on average not unsettling influence. A perfect ADC has an ENOB equivalent to its destinations.

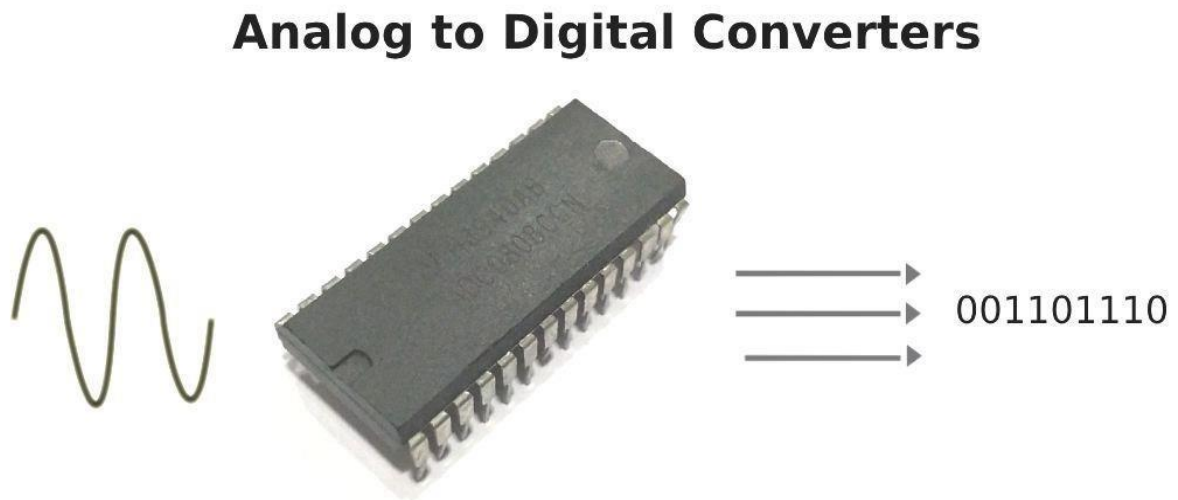


Figure. 3.4 Analog to digital converter

3.3 Electrocardiogram (ECG) and high blood pressure:

Electrocardiography is the path toward making an EKG, a record – an outline of voltage versus time of the electrical action of the heart utilizing terminals set on the skin. These cathodes see a little electrical changes that are an aftereffect of cardiovascular muscle depolarization followed by depolarization during each cardiovascular cycle (heartbeat). Changes in the common ECG design happen in various cardiovascular assortments from the norm, including heart beat disturbing effects , insufficient coronary corridor course framework , and electrolyte aggravations.

In a standard 12-lead ECG, ten anodes are resolved to the patient's limits and ostensibly of the chest. The general size of the heart's electrical potential is then assessed from twelve specific centers ("leads") and is recorded over some indistinct time slot (routinely ten seconds). In this manner, the general essentialness and heading of the heart's electrical depolarization is gotten at reliably all through the cardiovascular cycle.

There are three focal bits OF ECG: The P wave, which tends to the depolarization of the atria; the QRS complex, which tends to the depolarization of the ventricles; and the T wave, which tends to the repolarization of the ventricles.

An electrocardiogram (ECG) is a test which checks the electrical activity of your heart to show whether it is working generally.

An ECG records the heart's beat and action on a moving piece of paper or a line on a screen. Your fundamental thought specialist can look at and unravel the summits and plunges on paper or screen to check whether there is any capricious or impossible to miss action.

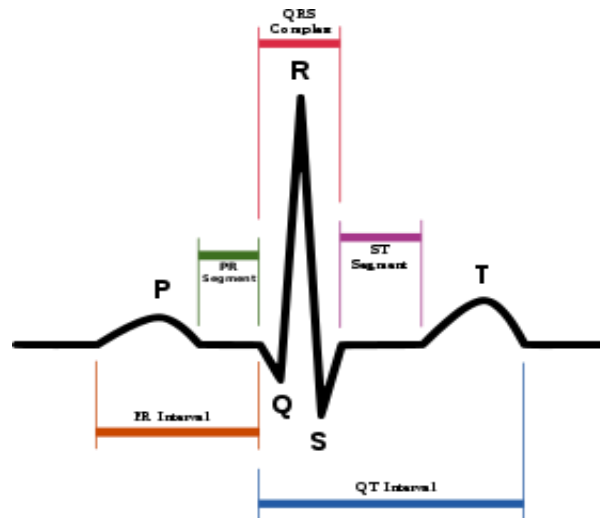


Figure 3.5 ECG waveform

3.3.1 What can an ECG (electrocardiogram) appear?

An electrocardiogram can be a significant strategy to check whether your hypertension has made any mischief your heart or veins. Thusly, you may be drawn nearer to have an ECG when you are first resolved to have hypertension.

A segment of the things an ECG can recognize are:

- cholesterol plugging up your heart's blood smoothly
- a respiratory disappointment beforehand
- expansion of one side of the heart
- unusual heart rhythms

3.5 DDS

Direct digital synthesis (DDS) is a strategy used to create a simple sign (like a sine wave or triangle wave) utilizing advanced strategies. The simple signs are incorporated from values put away in memory. A "format" containing the sign's plentifulness esteems for all waveform stages is put away in memory and used to reproduce the sign. With DDS, signs can be combined legitimately from the layout without requiring the stage bolted circles other aberrant techniques require. Various frequencies are delivered by changing the rate the stage esteems are handled and utilizing methods to include, duplicate and scale signals, different waveforms can be produced. The integrated signs are repeatable and the frequencies exact. Correspondence procedures like spread range recurrence jumping use DDS to rapidly change frequencies. It is likewise utilized for signal generators and empowers recurrence clears. DDS has had a major effect in testing.

A DDS circuit incorporates a stage aggregator, a stage sufficiency table (a query table for the most part in ROM - the "layout") and a digital to analog converter (DAC). The stage aggregator consolidates the reference recurrence and the incentive in the tuning word register. The yield from the DAC is typically applied to channels to smooth the waveform and evacuate any unessential yield.

The means to creating a sign are:

- The reference sign and tuning register update the stage aggregator, giving a stage esteem
- The relating adequacy for that stage is recovered from the stage abundancy table
- The DAC changes over the recovered abundancy esteems to a simple yield 4. The yield is sent through smoothing channel

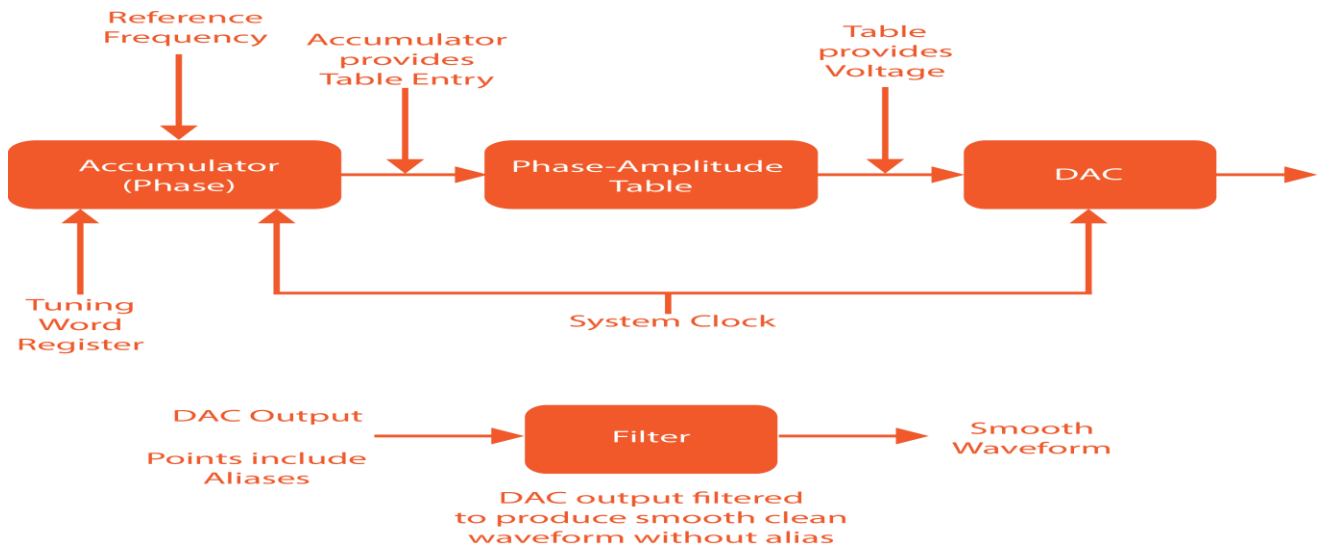


Fig 3.7 Direct digital synthesis block diagram

DDS gives an approach to produce simple signs from values put away in memory utilizing advanced strategies. By changing a tuning register esteem, frequencies can be changed rapidly without settling time, making it perfect for testing, interchanges and recurrence clear applications.

3.6 QUARTUS II



Figure 3.8 QUARTUS II

Intel Quartus Prime is programmable rationale gadget plan programming delivered by Intel; before obtaining of Altera the device was called Altera Quartus Prime, prior Altera quartus II. Quartus Prime empowers investigation and amalgamation of HDL structures, which empowers the designer to order their plans, perform timing examination, look at RTL charts, reproduce a plan's response to various upgrades, and arrange the objective gadget with the software engineer. Quartus Prime incorporates a usage of VHDL and HDL for equipment depiction, visual altering of rationale circuits., and vector waveform reproduction.

3.7 VIVADO



Figure 3.9 VIVADO

Vivado is a software made by Xilinx for mix and assessment of HDL plans, replacing Xilinx ISE with additional features for system on a chip improvement and huge level blend Vivado addresses a ground-up adjust and rethinking of the entire structure stream (diverged from ISE), and has been depicted by analysts as "adequately considered, immovably fused, impacting snappy, flexible, feasible, and regular"

CONCLUSION

With this project we can compare data of various patients by storing the ECG waveform in the ROM generated by us in the FPGA and then compare it with the data of patient which is healthy and diseased with the help of this application medical checkup will be easy and 24*7 available to the patient as doctor can work from home and prescribe medicines by analyzing the disease. This project mainly focusses on the health problem faced by the old age people and people living in remote area where medical facilities are very poor.

REFERENCES

- [1]. *FPGA implementation of Power-Efficient ECG pre-processing block* Kirti, Harsh Sohal, Shruti Jain.
- [2]. *Electrocardiogram (ECG/EKG) using FPGA* Vaibhav Desai San Jose State University.
- [3]. *FPGA implementation of electrocardiography(ECG)signal processing* Sunil Kumar,Manjit kaur.
- [4]. Subhas Chandra Mukhopadhyay, “*Wearable Sensors for Human Activity Monitoring: A Review*”, *IEEE SENSORS JOURNAL, VOL. 15, NO. 3, pp 1321-1330, MARCH 2015.*
- [5]. P. Nitin Jain, Preeti N. Jain, Trupti P. Agarkar, “*An Embedded, GSM based, Multiparameter, Realtime Patient Monitoring System and Control an Implementation for ICU Patients*”, IEEE, pp. 987- 992,2012
- [6]. Nitha V Panicker, Sukesh Kumar A, “*Development of a Blood Pressure Monitoring System for Home health Application*”, International Conference on Circuit, Power and Computing Technologies [ICCPCT], 2015.

APPENDIX

APPENDIX A

A.1 DAC_Amp_Cont

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

entity DAC_Amp_Cont is
    port ( Clk : in std_logic;
          Reset : in std_logic;
          Amp_Set1 : in std_logic;
          Amp_Set2 : in std_logic;
          Amp_Data1 : in std_logic_vector(9 downto 0);
          Amp_Data2 : in std_logic_vector(9 downto 0);
          Offset_Set : in std_logic;
          Offset_Data : in std_logic_vector(9 downto 0);
          SPI_AmpData : out std_logic_vector(15 downto 0);
          SPI_Send_Clear : in std_logic;
          SPI_Send_AmpFlag : out std_logic);
end DAC_Amp_Cont;

architecture a of DAC_Amp_Cont is
    signal Temp_SPI_AmpData: std_logic_vector(15 downto 0);
    begin
        process(Clk)
        begin
            if Clk'event and Clk = '1' then
                if Reset = '0' then
                    Temp_SPI_AmpData <= (others => '0');
                    SPI_Send_AmpFlag <= '0';
                elsif Amp_Set1 = '1' then
                    Temp_SPI_AmpData(15 downto 0) <= "00001011" & Amp_Data1(7 downto 0);
                    if SPI_Send_Clear = '1' then
                        SPI_AmpData <= Temp_SPI_AmpData;
                        SPI_Send_AmpFlag <= '1';
                    end if;
                elsif Amp_Set2 = '1' then
                    Temp_SPI_AmpData(15 downto 0) <= "00001111" & Amp_Data2(7 downto 0) ;
                    if SPI_Send_Clear = '1' then
                        SPI_AmpData <= Temp_SPI_AmpData;
                        SPI_Send_AmpFlag <= '1';
                    end if;
                elsif Offset_Set = '1' then
                    Temp_SPI_AmpData(15 downto 0) <= "000000" & Offset_Data ;
                    if SPI_Send_Clear = '1' then
                        SPI_AmpData <= Temp_SPI_AmpData;
                        SPI_Send_AmpFlag <= '1';
                    end if;
                elsif SPI_Send_Clear = '0' then
                    SPI_Send_AmpFlag <= '0';
                end if;
            end if;
        end process;
    end a;
```

A.1.1 RTL ANALYSIS

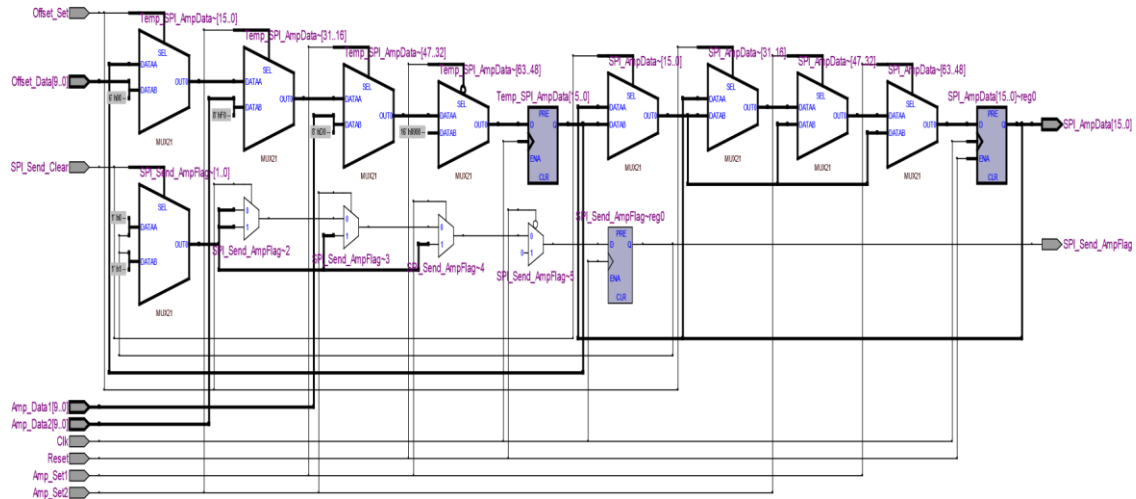


Figure A.1 RTL analysis of DAC_Amp_Cont

A.1.2 WAVEFORM

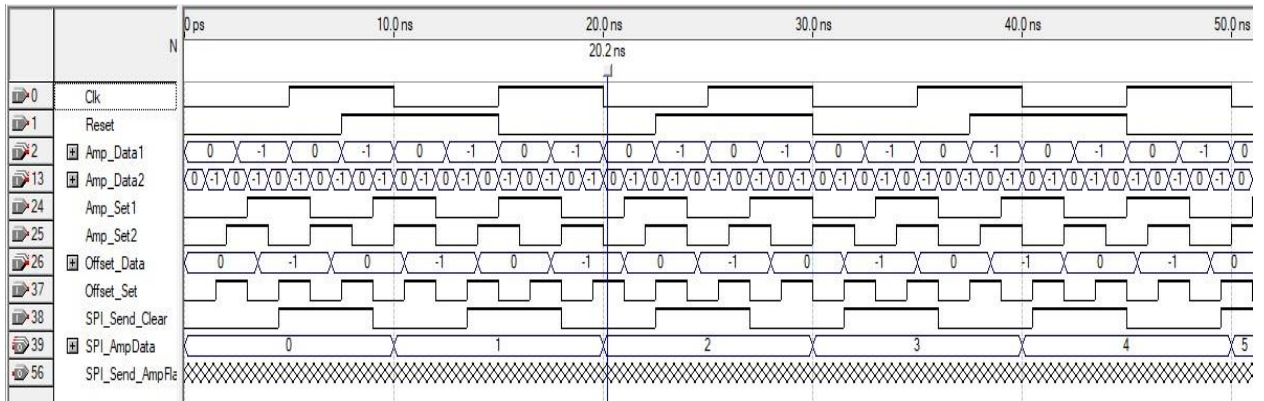


Figure A.1.1 waveform of DAC_Amp_Cont

A.2 Dac_reg_cont

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

entity DAC_Reg_Cont is
    port ( Clk : in std_logic;
          Reset : in std_logic;
          DAC_Inst_Set : in std_logic;
          DAC_Data_Set : in std_logic;
          DAC_Inst : in std_logic_vector(7 downto 0);
          DAC_Data : in std_logic_vector(7 downto 0);
          DAC_Comm_Send : in std_logic;
          SPI_RegData : out std_logic_vector(15 downto 0);
          SPI_Send_Clear : in std_logic;
          SPI_Send_RegFlag : out std_logic);
end DAC_Reg_Cont;

architecture a of DAC_Reg_Cont is
    signal Temp_SPI_RegData : std_logic_vector(15 downto 0);
begin
    process(Clk)
    begin
        if Clk'event and Clk = '1' then
            if Reset = '0' then
                Temp_SPI_RegData <= (others => '0');
                SPI_Send_RegFlag <= '0';
            elsif DAC_Inst_Set = '1' then
                Temp_SPI_RegData(15 downto 8) <= DAC_Inst;
                SPI_Send_RegFlag <= '0';
            elsif DAC_Data_Set = '1' then
                Temp_SPI_RegData(7 downto 0) <= DAC_Data;
                SPI_Send_RegFlag <= '0';
            elsif DAC_Comm_Send = '1' and SPI_Send_Clear = '1' then
                SPI_RegData <= Temp_SPI_RegData;
                SPI_Send_RegFlag <= '1';
            elsif SPI_Send_Clear = '0' then
                SPI_Send_RegFlag <= '0';
                SPI_RegData <= (others => '0');
            end if;
        end if;
    end process;

end a;
```

A.2.1 RTL ANALYSIS

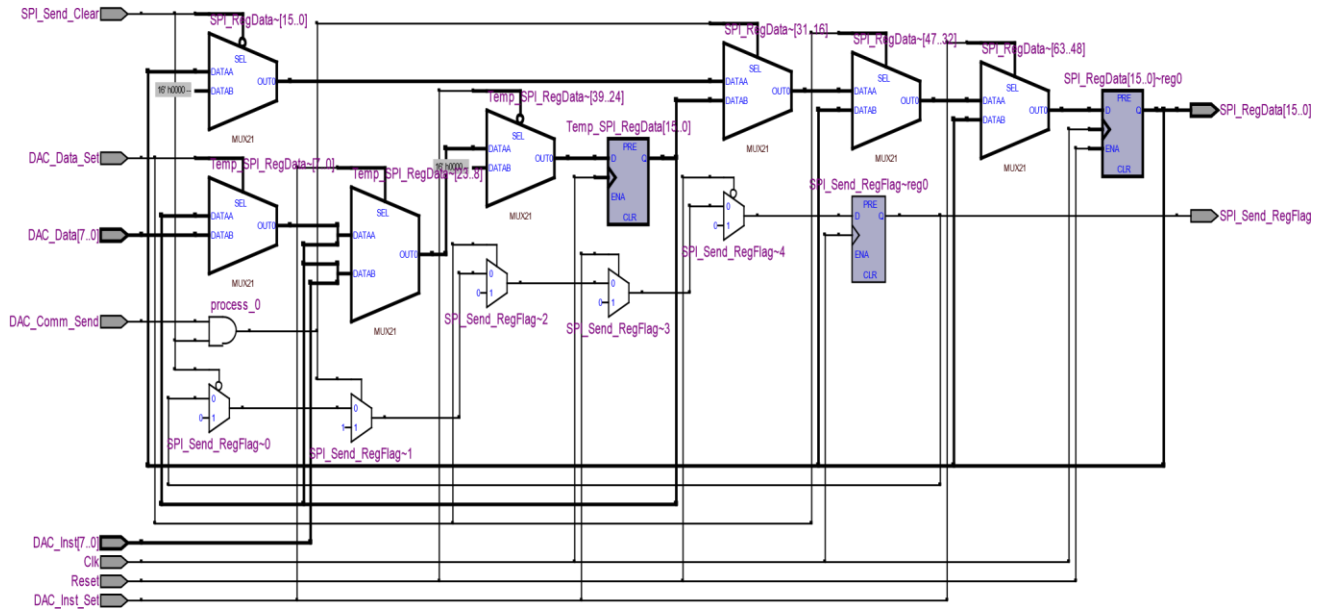


Figure A.2 RTL analysis of `Dac_reg_cont`

A.2.2 WAVEFORM

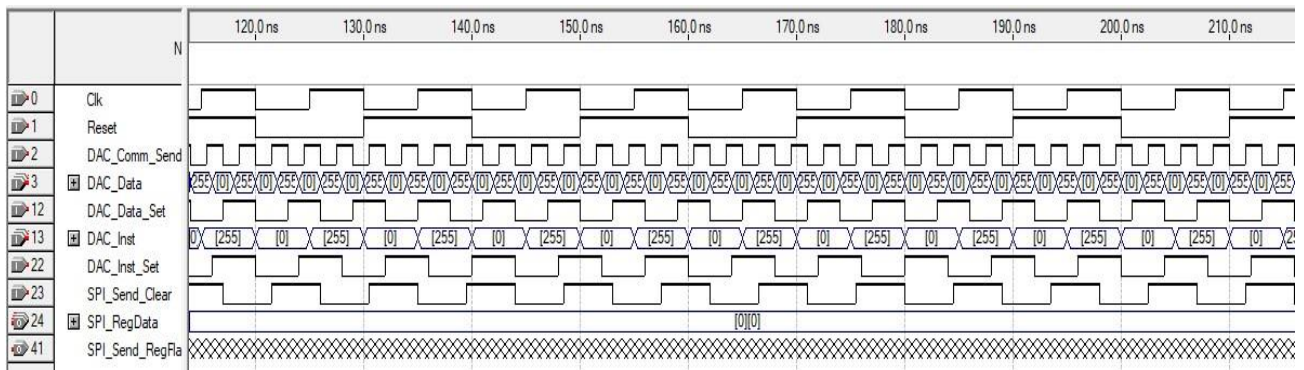


Figure A.2.1 waveform of `Dac_reg_cont`

A.3 DAC_SPI

```

architecture a of DAC_SPI is
signal State : std_logic_vector(5 downto 0);
signal Temp_Data : std_logic_vector(15 downto 0);
signal SPI_CSB_Temp : std_logic;
signal SPI_Ready : std_logic;
begin
    SPI_CSB <= SPI_CSB_Temp;
    SPI_Clk <= Div_Clk and NOT SPI_CSB_Temp;
    SPI_Send_Clear <= SPI_Ready;
    process(Clk)
    begin
        if Clk'event and Clk = '1' then
            if Reset = '0' then
                State <= "000000";
                Temp_Data <= "0000000000000000";
                SPI_CSB_Temp <= '1';
                SPI_DQ <= '0';
                SPI_End <= '0';
                SPI_Ready <= '1';
            elsif LE_Clk = '1' then
                case State is
                    when "000000" =>
                        SPI_End <= '0';
                        if Start_Reg_Cont = '1' and SPI_Ready = '1' then
                            State <= "000001";
                            Temp_Data <= SPI_RegData;
                            SPI_CSB_Temp <= '1';
                            SPI_DQ <= '0';
                            SPI_Ready <= '0';
                        elsif Start_Amp_Cont = '1' and SPI_Ready = '1' then
                            State <= "010010";
                            Temp_Data <= SPI_AmpData;
                            SPI_CSB_Temp <= '1';
                            SPI_DQ <= '0';
                            SPI_Ready <= '0';
                        else
                            State <= "000000";
                            SPI_CSB_Temp <= '1';
                            SPI_DQ <= '0';
                            SPI_Ready <= '1';
                        end if;
                    when "000001" | "000010" | "000011" | "000100" |
                        "000101" | "000110" | "000111" | "001000" |
                        "001001" | "001010" | "001011" | "001100" |
                        "001101" | "001110" | "001111" | "010000" =>
                        SPI_End <= '0';
                        SPI_DQ <= Temp_Data(15);
                        Temp_Data <= Temp_Data(14 downto 0) & '0';
                        SPI_CSB_Temp <= '0';
                        State <= State + 1;
                        SPI_Ready <= '0';
                    when "010001" =>
                        SPI_End <= '1';
                        State <= "000000";
                        Temp_Data <= "0000000000000000";
                end case;
            end if;
        end process;
    end architecture a;

```

```

        SPI_CSB_Temp <= '0';
        State <= State + 1;
        SPI_Ready <= '0';
    when "010001" =>
        SPI_End <= '1';
        State <= "000000";
        Temp_Data <= "0000000000000000";
        SPI_CSB_Temp <= '1';
        SPI_DQ <= '0';
        SPI_Ready <= '1';
    when "010010" | "010011" | "010100" | "010101" |
        "010110" | "010111" | "011000" | "011001" |
        "011010" | "011011" | "011100" | "011101" |
        "011110" | "011111" | "110000" | "110001" =>
        SPI_End <= '0';
        SPI_DQ <= Temp_Data(15);
        Temp_Data <= Temp_Data(14 downto 0) & '0';
        SPI_CSB_Temp <= '0';
        State <= State + 1;
        SPI_Ready <= '0';
    when "110010" =>
        SPI_End <= '1';
        State <= "000000";
        Temp_Data <= "0000000000000000";
        SPI_CSB_Temp <= '1';
        SPI_DQ <= '0';
        SPI_Ready <= '0';
    when others =>
        State <= "000000";
        Temp_Data <= "0000000000000000";
        SPI_CSB_Temp <= '1';
        SPI_DQ <= '0';
        SPI_Ready <= '1';
    end case;
end if;
end process;
end a;

```

A.3.1 RTL ANALYSIS



Figure A.3.1 RTL analysis of DAC_SPI

A.3.2 WAVEFORM

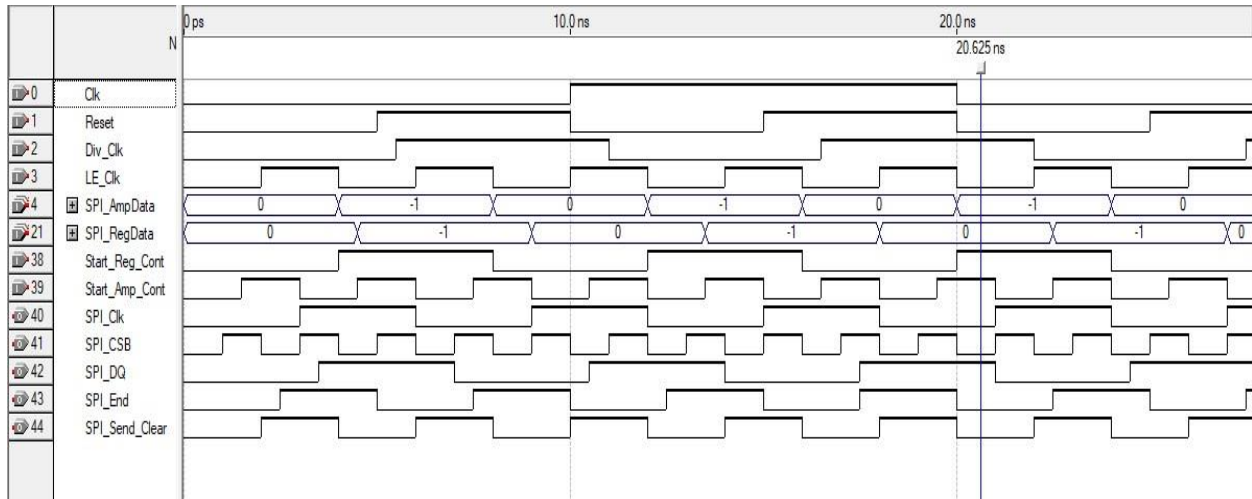


Figure A.3.2 waveform of DAC_SPI

A.4 DAC_Timing_Unit

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

entity DAC_Timing_Unit is
    port ( Clk : in std_logic;
          Reset : in std_logic;
          Div_Clk : out std_logic;
          LE_Clk : out std_logic);
end DAC_Timing_Unit;

architecture a of DAC_Timing_Unit is
    signal Cnt : std_logic_vector(2 downto 0);
begin

    process(Clk)
    begin
        if Clk'event and Clk = '1' then
            if Reset = '0' then
                cnt <= "000";
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;

    process(Clk)
    begin
        if Clk'event and Clk = '1' then
            if cnt = "111" then
                LE_Clk <= '1';
            else
                LE_Clk <= '0';
            end if;
        end if;
    end process;

    Div_Clk <= cnt(2);
end a;
```

A.4.1 RTL ANALYSIS

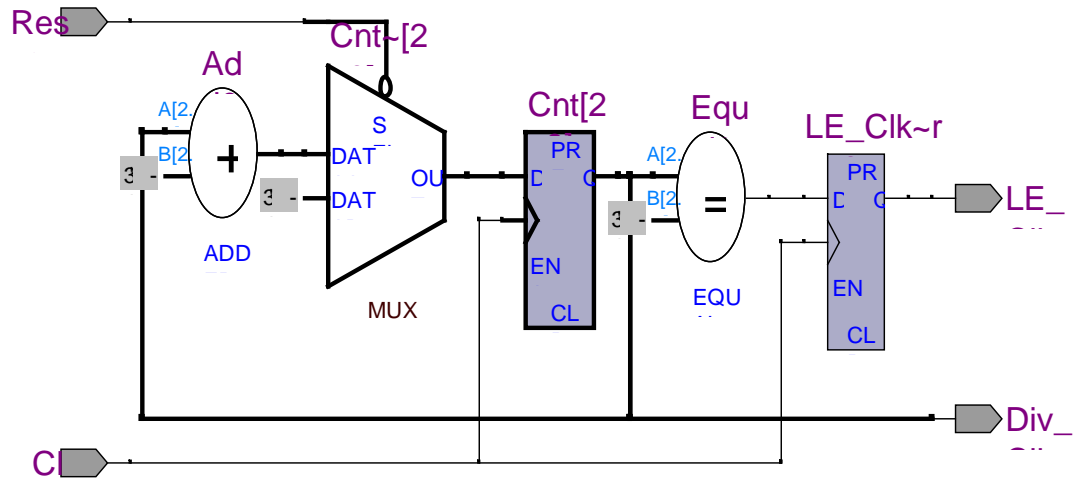


Figure A.4.1 RTL analysis of DAC_Timing_Unit

A.4.2 WAVEFORM

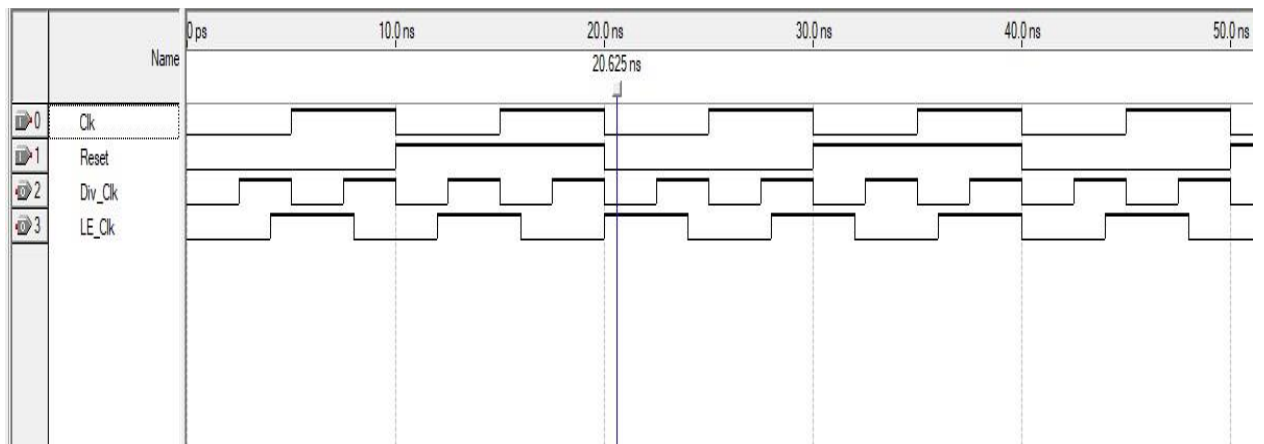


Figure A.4.2 waveform of DAC_Timing_Unit

A.5 Generate_Constant

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

entity Generate_Constant is
    port ( Data_Gap : in std_logic_vector(5 downto 0);
          Constant1 : out std_logic_vector(9 downto 0);
          Constant2 : out std_logic_vector(9 downto 0);
          Constant3 : out std_logic_vector(9 downto 0));
end Generate_Constant;

architecture a of Generate_Constant is
    signal Gap : std_logic_vector(9 downto 0);
begin
    Gap <= "0000" & Data_Gap;
    Constant1 <= Gap;
    Constant2 <= 1000 - Gap;
    Constant3 <= 999 - Gap;
end a;
```

A.5.1 RTL ANALYSIS

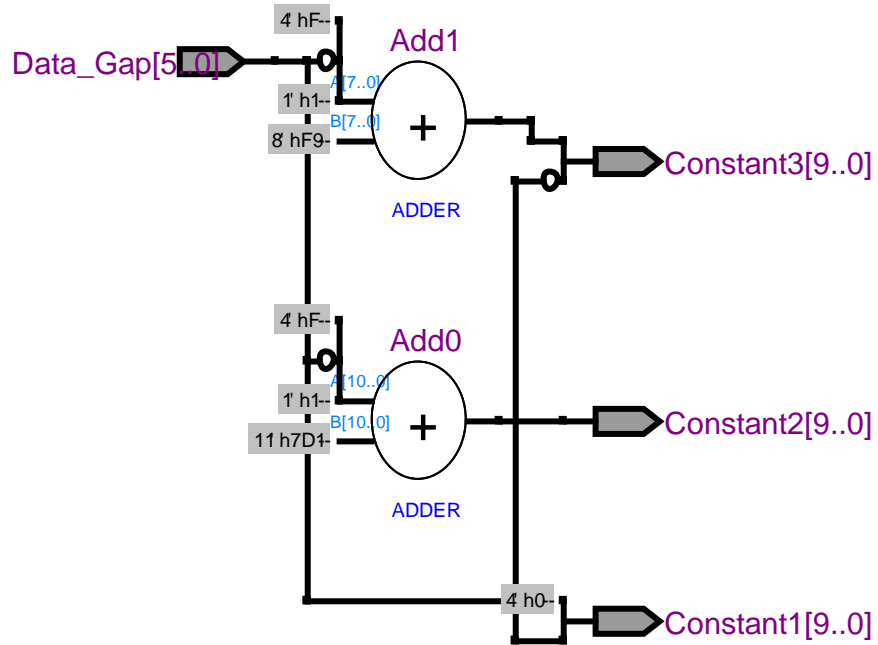


Figure A.5.1 RTL analysis of Generate_Constant

A.5.2 WAVEFORM

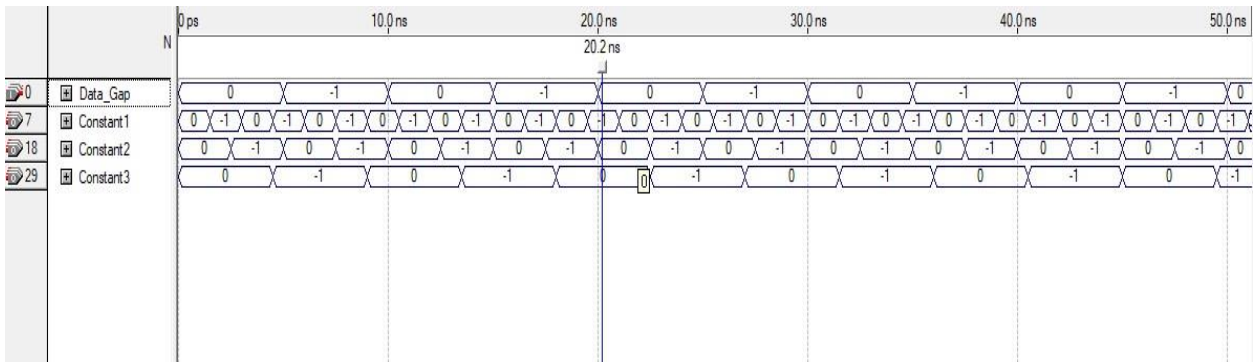


Figure A.5.2 wave form of Generate_Constant

A.6 Pseudo_Generate_Constant

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

entity Pseudo_Generate_Constant is
    port ( Pseudo_Data_Gap : in std_logic_vector(5 downto 0);
          Constant1 : out std_logic_vector(9 downto 0);
          Constant2 : out std_logic_vector(9 downto 0);
          Constant3 : out std_logic_vector(9 downto 0));
end Pseudo_Generate_Constant;

architecture a of Pseudo_Generate_Constant is
    signal Gap : std_logic_vector(9 downto 0);
begin
    Gap <= "0000" & Pseudo_Data_Gap;
    Constant1 <= Gap;
    Constant2 <= 1000 - Gap;
    Constant3 <= 999 - Gap;
end a;
```

4.6.1 RTL ANALYSIS

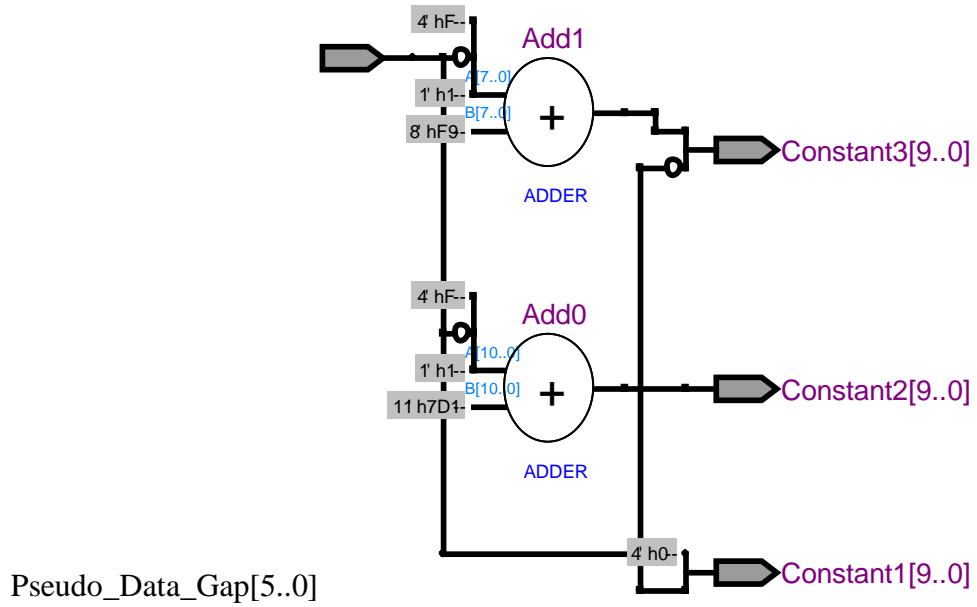


Figure A.6.1 RTL analysis of pseudo_Generate_constan

A.6.2 WAVEFORM

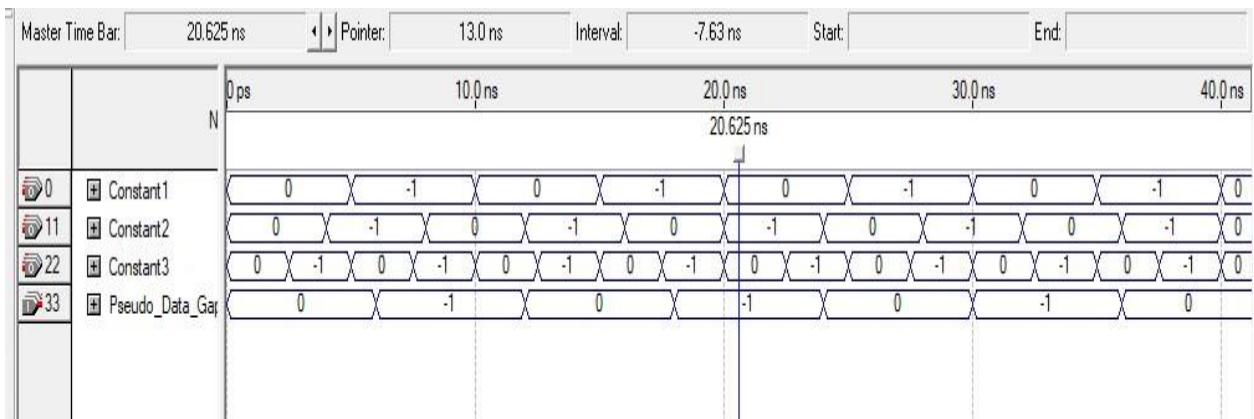


Figure A.6.2 waveform of Pseudo_Generate_Constant

A.7 Pseudo_WG_Control_Unit

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

entity Pseudo_WG_Control_Unit is
    port ( Clk : in std_logic;
          Reset : in std_logic;
          Clk_State : in std_logic_vector(1 downto 0);
          WG_On : in std_logic;
          WG_Off : in std_logic;
          Pseudo_Cnt_Limit : in std_logic_vector(9 downto 0);
          Address_On : out std_logic;
          Reset_Count : out std_logic);
end Pseudo_WG_Control_Unit;

architecture a of Pseudo_WG_Control_Unit is
    signal Cnt : std_logic_vector(9 downto 0);
    signal Reset_Count_Temp : std_logic;
    signal State : std_logic;
begin
    process(Clk)
    begin
        if Clk'event and Clk = '1' then
            if Reset = '0' or WG_Off = '1' then
                Cnt <= (others => '0');
                Reset_Count_Temp <= '1';
                State <= '0';
            else
                case State is
                    when '0' =>
                        Cnt <= (others => '1');
                        Reset_Count_Temp <= '0';
                        if WG_On = '1' then
                            State <= '1';
                        else
                            State <= '0';
                        end if;
                        Address_On <= '0';
                    when '1' =>
                        if Clk_State = "11" then
                            if Cnt = Pseudo_Cnt_Limit then
                                Cnt <= (others => '0');
                                Address_On <= '1';
                            else
                                Cnt <= Cnt + 1;
                                Address_On <= '0';
                            end if;
                        else
                            Address_On <= '0';
                        end if;
                        Reset_Count_Temp <= '0';
                        State <= '1';
                    when others =>
                        Cnt <= (others => '1');
                        Reset_Count_Temp <= '0';
                        State <= '0';
                    end case;
                end if;
            end if;
        end if;
    end process;
end a;
```

A.7.1 RTL ANALYSIS

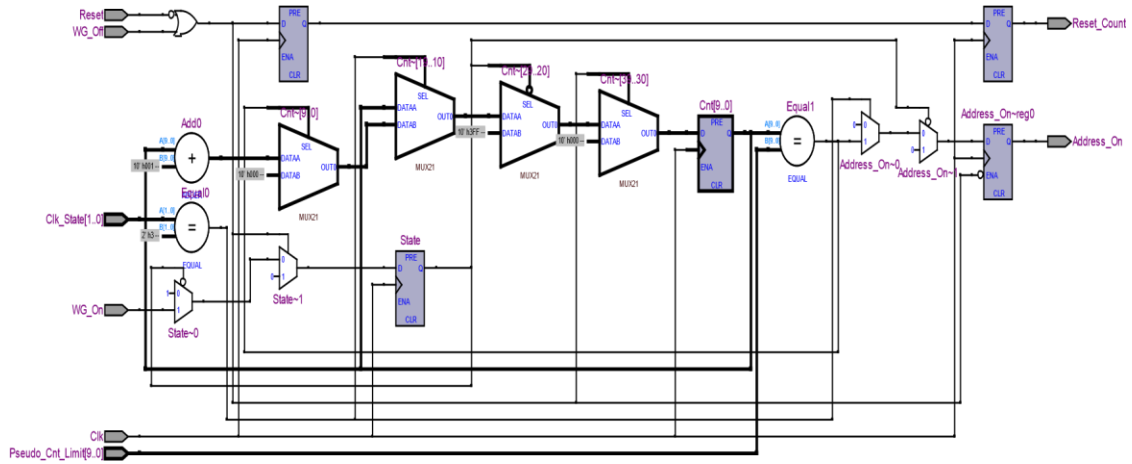


Figure A.7.1 RTL analysis of Pseudo_WG_Control_Unit

A.7.2 Waveform

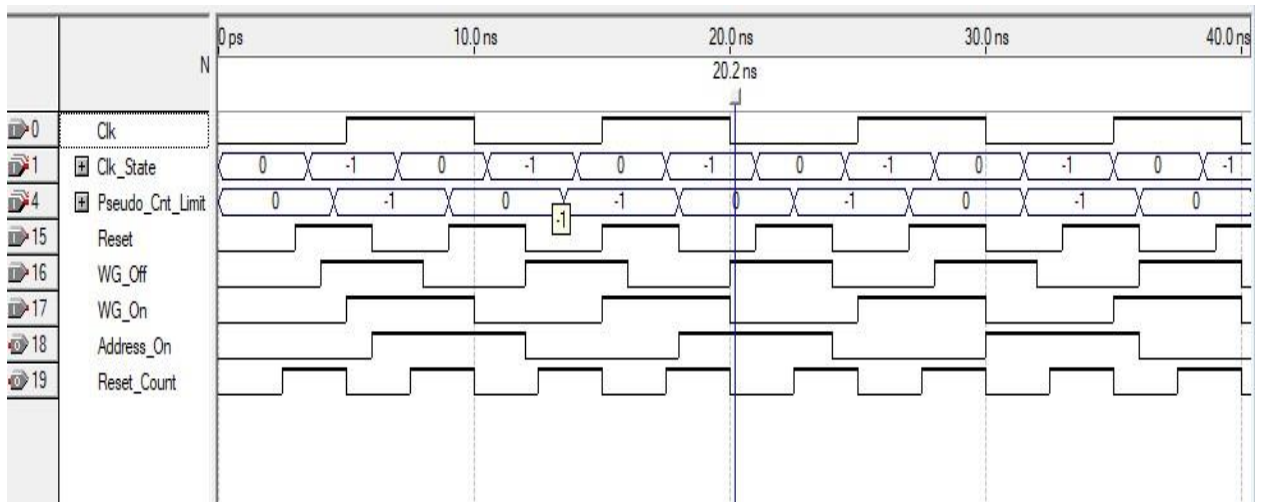


Figure A.7.2 waveform of Pseudo_WG_Control_Unit

A.8 TOP MODULE:- includes all the above modules

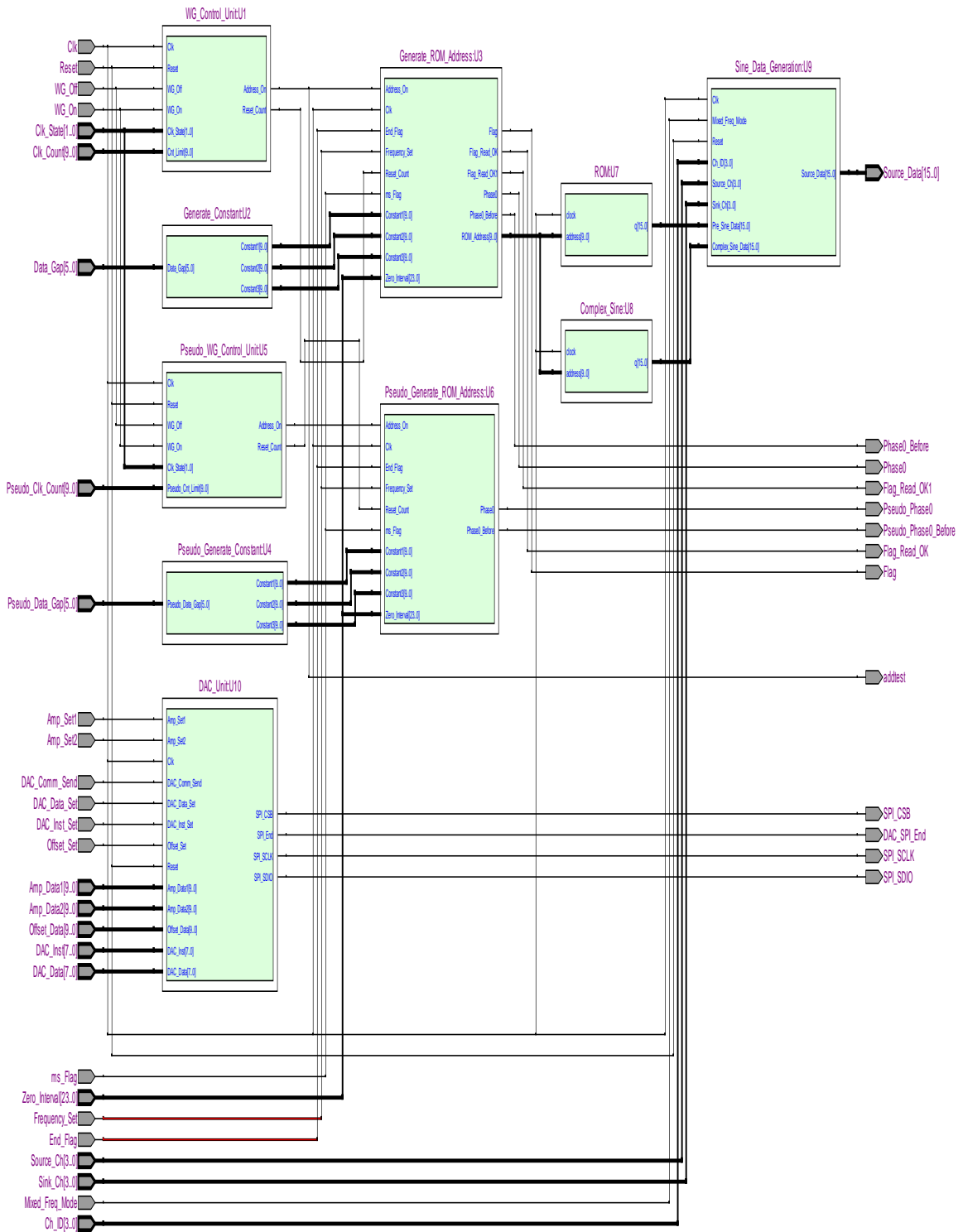


Figure A.8 top module

A.9 SINE WAVE AS AN OUTPUT

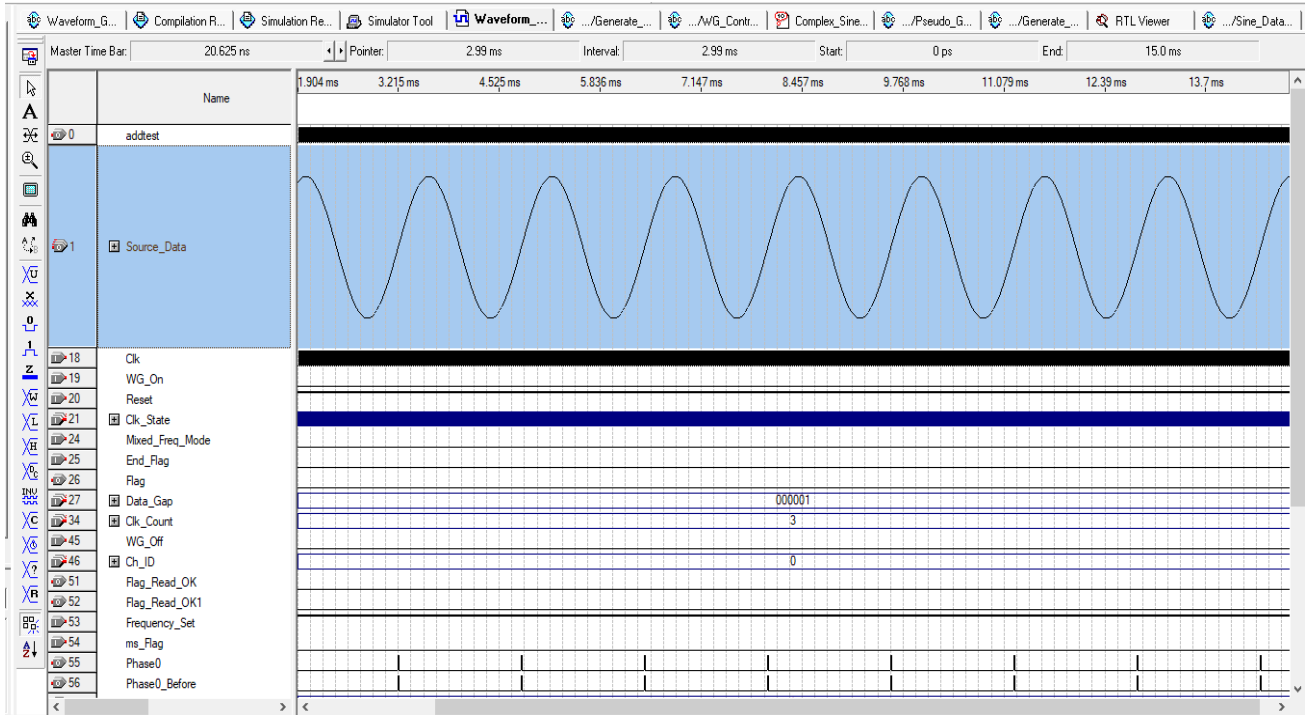


Figure A.9 sine wave

APPENDIX B

B.1 [https://category.alldatasheet.com/index.jsp?sSearchword=Adc%2520datasheet
&gclid=Cj0KCQiAt_PuBRDcARIsAMNIBdpkT94pjpJC_aLz1EvbLPeWdg9r9
LJi4quD8cdWi4rn-cxqiW5xf9kaAhMmEALw_wcB](https://category.alldatasheet.com/index.jsp?sSearchword=Adc%2520datasheet&gclid=Cj0KCQiAt_PuBRDcARIsAMNIBdpkT94pjpJC_aLz1EvbLPeWdg9r9LJi4quD8cdWi4rn-cxqiW5xf9kaAhMmEALw_wcB)

B.2 [https://www.mouser.in/new/xilinx/xilinx-zynq-
7000socs/?gclid=Cj0KCQiAt_PuBRDcARIsAMNIBdofKl5ITEvY1M6Lyfjzq3mmms
pretH1GnzxlO1gWesnTij8U1NKtREaAkCoEALw_wcB](https://www.mouser.in/new/xilinx/xilinx-zynq-7000socs/?gclid=Cj0KCQiAt_PuBRDcARIsAMNIBdofKl5ITEvY1M6Lyfjzq3mmmspretH1GnzxlO1gWesnTij8U1NKtREaAkCoEALw_wcB)

B.3 <https://ieeexplore.ieee.org/document/7978718>

B.4 [https://www.mouser.in/new/Intel/intelquartus/?gclid=Cj0KCQiAt_PuBRDcARIsAMN
IBdpvOYAbX7vu2pFY6mpsE16gVoRhgZLvuhdUMUMNLSP7Teoz7L5ngaAhNaEALw_
wcB](https://www.mouser.in/new/Intel/intelquartus/?gclid=Cj0KCQiAt_PuBRDcARIsAMNIBdpvOYAbX7vu2pFY6mpsE16gVoRhgZLvuhdUMUMNLSP7Teoz7L5ngaAhNaEALw_wcB)

APPENDIX C
DATA SHEET OF ZBOARD

Zynq-7000 SoC First Generation Architecture

The Zynq®-7000 family is based on the Xilinx SoC architecture. These products integrate a feature-rich dual-core or single-core ARM® Cortex™-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 CPUs are the heart of the PS and also include on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces.

Processing System (PS)

ARM Cortex-A9 Based

Application Processor Unit (APU)

- 2.5 DMIPS/MHz per CPU
- CPU frequency: Up to 1 GHz
- Coherent multiprocessor support
- ARMv7-A architecture
 - TrustZone® security
 - Thumb®-2 instruction set
- Jazelle® RCT execution Environment Architecture
- NEON™ media-processing engine
- Single and double precision Vector Floating Point Unit (VFPv3)
- CoreSight™ and Program Trace Macrocell (PTM)
- Timer and Interrupts
 - Three watchdog timers
 - One global timer
 - Two triple-timer counters

Caches

- 32 KB Level 1 4-way set-associative instruction and data caches (independent for each CPU)
- 512 KB 8-way set-associative Level 2 cache (shared between the CPUs)
- Byte-parity support

On-Chip Memory

- On-chip boot ROM
- 256 KB on-chip RAM (OCM)
- Byte-parity support

External Memory Interfaces

- Multiprotocol dynamic memory controller
- 16-bit or 32-bit interfaces to DDR3, DDR3L, DDR2, or LPDDR2 memories
- ECC support in 16-bit mode
- 1GB of address space using single rank of 8-, 16-, or 32-bit-wide memories
- Static memory interfaces
 - 8-bit SRAM data bus with up to 64 MB support
 - Parallel NOR flash support
 - ONFI1.0 NAND flash support (1-bit ECC)
 - 1-bit SPI, 2-bit SPI, 4-bit SPI (quad-SPI), or two quad-SPI (8-bit) serial NOR flash

8-Channel DMA Controller

- Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and scatter-gather transaction support

I/O Peripherals and Interfaces

- Two 10/100/1000 tri-speed Ethernet MAC peripherals with IEEE Std 802.3 and IEEE Std 1588 revision 2.0 support
 - Scatter-gather DMA capability
 - Recognition of 1588 rev. 2 PTP frames
 - GMII, RGMII, and SGMII interfaces
- Two USB 2.0 OTG peripherals, each supporting up to 12 Endpoints
 - USB 2.0 compliant device IP core
 - Supports on-the-go, high-speed, full-speed, and low-speed modes
 - Intel EHCI compliant USB host
 - 8-bit ULPI external PHY interface
- Two full CAN 2.0B compliant CAN bus interfaces
 - CAN 2.0-A and CAN 2.0-B and ISO 118981-1 standard compliant
 - External PHY interface
- Two SD/SDIO 2.0/MMC3.31 compliant controllers
- Two full-duplex SPI ports with three peripheral chip selects
- Two high-speed UARTs (up to 1 Mb/s)
- Two master and slave I2C interfaces
- GPIO with four 32-bit banks, of which up to 54 bits can be used with the PS I/O (one bank of 32b and one bank of 22b) and up to 64 bits (up to two banks of 32b) connected to the Programmable Logic
- Up to 54 flexible multiplexed I/O (MIO) for peripheral pin assignments

Interconnect

- High-bandwidth connectivity within PS and between PS and PL
- ARM AMBA® AXI based
- QoS support on critical masters for latency and bandwidth control

Programmable Logic (PL)

Configurable Logic Blocks (CLB)

- Look-up tables (LUT)
- Flip-flops
- Cascadeable adders

36 Kb Block RAM

- True Dual-Port
- Up to 72 bits wide
- Configurable as dual 18 Kb block RAM

DSP Blocks

- 18 x 25 signed multiply
- 48-bit adder/accumulator
- 25-bit pre-adder

Programmable I/O Blocks

- Supports LVCMOS, LVDS, and SSTL
- 1.2V to 3.3V I/O
- Programmable I/O delay and SerDes

JTAG Boundary-Scan

- IEEE Std 1149.1 Compatible Test Interface

PCI Express® Block

- Supports Root complex and End Point configurations
- Supports up to Gen2 speeds
- Supports up to 8 lanes

Serial Transceivers

- Up to 16 receivers and transmitters
- Supports up to 12.5 Gb/s data rates

Two 12-Bit Analog-to-Digital Converters

- On-chip voltage and temperature sensing
- Up to 17 external differential input channels
- One million samples per second maximum conversion rate

Feature Summary

Table 1: Zynq-7000 and Zynq-7000S SoCs

Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processor Core	Single-core ARM Cortex-A9 MPCore™ with CoreSight™			Dual-core ARM Cortex-A9 MPCore™ with CoreSight™						
Processor Extensions	NEON™ & Single / Double Precision Floating Point for each processor									
Maximum Frequency	667 MHz (-1); 766 MHz (-2)			667 MHz (-1); 766 MHz (-2); 866 MHz (-3)			667 MHz (-1); 800 MHz (-2); 1 GHz (-3)			667 MHz (-1) 800 MHz (-2)
L1 Cache	32 KB Instruction, 32 KB data per processor									
L2 Cache	512 KB									
On-Chip Memory	256 KB									
External Memory Support ⁽¹⁾	DDR3, DDR3L, DOR2, LPDDR2									
External Static Memory Support ⁽¹⁾	2x Quad-SPI, NAND, NOR									
DMA Channels	8 (4 dedicated to Programmable Logic)									
Peripherals ⁽¹⁾	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
Peripherals w/ built-in DMA ⁽¹⁾	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
Security ⁽²⁾	RSA Authentication, and AES and SHA 256-bit Decryption and Authentication for Secure Boot									
Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master 2x AXI 32-bit Slave 4x AXI 64-bit/32-bit Memory AXI 64-bit ACP 16 Interrupts									

Table 1: Zynq-7000 and Zynq-7000S SoCs (Cont'd)

Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100	
Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100	
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Kintax-7 FPGA	Kintax-7 FPGA	Kintax-7 FPGA	
	Programmable Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	444K	
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	
	Block RAM (# 36 Kb Blocks)	1.8 Mb (50)	2.5 Mb (72)	3.8 Mb (107)	2.1 Mb (60)	3.3 Mb (95)	4.9 Mb (140)	9.3 Mb (265)	17.6 Mb (500)	19.2 Mb (545)	26.5 Mb (755)
	DSP Slices (18x25 MACCs)	68	120	170	80	160	220	400	900	900	2,020
	Peak DSP Performance (Symmetric FIR)	73 GMACs	131 GMACs	187 GMACs	100 GMACs	200 GMACs	276 GMACs	588 GMACs	1,334 GMACs	1,334 GMACs	2,822 GMACs
	PCI Express (Root Complex or Endpoint) ⁽¹⁾		Gen2 x4			Gen2 x4		Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security ⁽²⁾	AES and SHA-256b for Boot Code and Programmable Logic Configuration, Decryption, and Authentication									