B. TECH. PROJECT REPORT

on

DEVELOPMENT OF DATA ACQUISITION SYSTEM FOR PHOTOACOUSTIC SETUP DEVELOPMENT USING MICRO-CONTROLLER AND FPGA

BY

Hemant Kumar (130002015)

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2016

DEVELOPMENT OF DATA ACQUISITION SYSTEM FOR PHOTOACOUSTIC SETUP DEVELOPMENT USING MICRO-CONTROLLER AND FPGA

A PROJECT REPORT

Submitted in partial fulfillment of the
Requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in
ELECTRICAL ENGINEERING

Submitted by:
Hemant Kumar (130002015)

Guided by:
Dr. Srivathsan Vasudevan

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2016

Table of Contents

CANDIDATE’S DECLARATION

I hereby declare that the project entitled **“DEVELOPMENT OF DATA ACQUISITION SYSTEM FOR PHOTOACOUSTIC SETUP DEVELOPMENT USING MICRO-CONTROLLER AND FPGA”** submittedin partial fulfillment for the award of the degree of Bachelor of Technology in ‘ELECTRICAL ENGINEERING’ completed under the supervision of **Dr. Srivathsan Vasudevan, Assistant Professor, Electrical Engineering,** IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student with date

CERTIFICATE by BTP Guide

It is certified that the above statement made by the studentis correct to the best of my knowledge.

Signature of BTP Guide with date and hisdesignation

Preface

This report on“Development of Data Acquisition System for Photoacoustic Setup Development using Micro-Controller And FPGA" is prepared under the guidance of Dr. Srivathsan Vasudevan.

Through this report I have tried to give an analysis of “High Frequency Square Wave Generation” and “Development of Data Acquisition System” using Micro-Controller and FPGA.

I have tried to the best of my abilities and knowledge to explain the content in a lucid manner.

Hemant Kumar
B.Tech. IV Year
Discipline of Electrical
IIT Indore

Acknowledgements

I would like to take this opportunity to thank my project guide Dr. Srivathsan Vasudevanfor enlightening me on my project and giving me the necessary guidelines to accomplish the project. His timely suggestions helped me very much.

I would like to thank Mr. Syed Sadaf Ali for helping me in the Arduino coding.

I would like to thank Mr. Raghvendra for helping me with the lab apparatus.

Hemant Kumar
B.Tech. IV Year
Discipline of Electrical
IIT Indore

Abstract

Arduino Atmega328P (Micro-controller) and Altera De0-Nano board (FPGA) are used in this project. Arduino is an open source electronics platform based on easy-to-use hardware and software. The De0-Nano board is a compact sized FPGA development platform suited for prototyping circuit designs such as robots and portable projects.

In this project, USB-to-UART bridge is used to provide serial communication between a PC and user logic within the FPGA, on the De0-Nano Board.

In this project, a variable high frequency square wave is generated that is very useful in laser modulation.

In this project, frequency counter for sine wave is developed that can measure frequency of sine wave at any cut-off point.

Introduction

Microcontroller:

A microcontroller is a self-contained system with peripherals, memory and a processor that can be used as an embedded system. A microcontroller is a solitary chip microcomputer fabricated from VLSI fabrication. Microcontroller's brain is named as CPU. CPU is the device which is employed to fetchdata, decode it and at the end complete the assigned task successfully. A micro controller is also known as embedded controller. The microcontroller is an embedded computer chip that controls most of the electronic gadgets and appliances people use on a daily basis, right from washing machines to anti-lock brakes in cars. Most programmable microcontrollers that are used today are embedded in other consumer products or machinery including phones, peripherals, automobiles and household appliances for computer systems. Even at a time when Intel presented the first microprocessor with the 4004 there was already a demand for microcontrollers: The contemporary TMS1802 from Texas Instruments, designed for usage in calculators was by the end of 1971 advertised for applications in cash registers, watches and measuring instruments. The TMS 1000, which was introduced in 1974, already included RAM, ROM, and I/O on-chip and can be seen as one of the first microcontrollers, even though it was called a microcomputer. The first controllers to gain really widespread use were the Intel 8048, which was integrated into PC keyboards, and its successor, the Intel 8051, as well as the 68HCxx series of microcontrollers from Motorola.

Microcontrollers are divided into categories according to their memory, architecture, bits and instruction sets. Types of microcontrollers are:

Bits-

8 bits microcontroller executes logic & arithmetic operations. Examples of 8 bits micro controller is Intel 8031/8051.

16 bits microcontroller executes with greater accuracy and performance in contrast to 8-bit. Example of 16 bit microcontroller is Intel 8096.

32 bits microcontroller is employed mainly in automatically controlled appliances such as office machines, implantable medical appliances, etc. It requires 32-bit instructions to carry out any logical or arithmetic function.

Memory-

External Memory Microcontroller - When an embedded structure is built with a microcontroller which does not comprise of all the functioning blocks existing on a chip it is named as external memory microcontroller. For illustration- 8031 microcontroller does not have program memory on the chip.

Embedded Memory Micro controller-When an embedded structure is built with a microcontroller which comprise of all the functioning blocks existing on a chip it is named as embedded memory microcontroller. For illustration- 8051 microcontroller has all program & data memory, counters & timers, interrupts, I/O ports and therefore its embedded memory microcontroller.

Instruction Set-

CISC- CISC means complex instruction set computer, it allows the user to apply 1 instruction as an alternative to many simple instructions.
RISC- RISC means Reduced Instruction Set Computers. RISC reduces the operation time by shortening the clock cycle per instruction.

Some microcontroller are: 8051 Microcontroller, PIC Microcontroller, AVR Microcontroller, ARM Microcontroller.

Following modules typically found in a microcontroller:

Processor Core: The CPU of the controller. It contains the arithmetic logic unit, the control unit, and the registers
Memory: The memory is sometimes split into program memory and data memory. In larger controllers, a DMA controller handles data transfers between peripheral components and the memory.
Interrupt Controller: Interrupts are useful for interrupting the normal program flow in case of external or internal events. In conjunction with sleep modes, they help to conserve power.
Timer/Counter: Most controllers have at least one and more likely 2-3 Timer/Counters, which can be used to timestamp events, measure intervals, or count events.
Digital I/O: Parallel digital I/O ports are one of the main features of microcontrollers. The number of I/O pins varies from 3-4 to over 90, depending on the controller family and the controller type.
Analog I/O: Apart from a few small controllers, most microcontrollers have integrated Analog/digital converters, which differ in the number of channels (2-16) and their resolution (8-12 bits). The Analog module also generally features an Analog comparator. In some cases, the micro controller includes digital/Analog converters.
Interfaces: Controllers generally have at least one serial interface which can be used to download the program and for communication with the development PC in general. Since serial interfaces can also be used to communicate with external peripheral devices, most controllers offer several and varied interfaces like SPI and SCI.

Arduino:

Arduino is a single-board microcontroller to make using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open source hardware board designed around an 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller.

Arduino is an open-source electronics prototyping platform based on flexible, easy-to use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language and the Arduino Development Environment.Arduino projects can be stand-alone, or they can communicate with software running on a computer.

Arduino Uno has Processor: 16 MHz ATmega328, Flash memory: 32 KB, RAM : 2kb, Operating Voltage: 5V, Input Voltage: 7-12 V, Number of Analog inputs: 6, Number of digital I/O: 14 (6 of them PWM).

FPGA:

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a designer after manufacturing hence field-programmable. The FPGA configuration is generally specified using a hardware description language (HDL).

The FPGA share a common history with most Programmable Logic Devices. The first of this kind of devices was the Programmable Read Only Memory. Further driven by need of specifically implementing logic circuits, Philips invented the Field-Programmable Logic Array (FPLA) in the 1970s. This consisted of two planes, a programmable wired AND-plane and the other as wired OR. It could implement functions in the Sum of Products form.

To overcome difficulties of cost and speed, Programmable Array Logics were developed which had only one programmable 'AND' plane fed into fixed OR gates. PALs and PLAs along with other variants are grouped as Simple Programmable Logic Devices (SPLDs). In order to cater to growing technological demands, SPLDs were integrated onto a single chip and interconnects were provided to programmably connect the SPLD blocks. These were called Complex PLDs and were first pioneered by Altera, the first in the family being Classic EPLDs and then, MAX series.

Programmable Logic-The purpose of programmable logic block in a FPGA is to provide the basic computation and storage elements used in digital systems. The basic logic element contains some form of programmable combinational logic, a flip-flop or latch and some fast carry logic to reduce area and delay cost to it could be entire processor. In addition to a basic logic block, many modern FPGAs contains a heterogeneous mixture of different blocks, some of which can only be used for specific functions, such as dedicated memory blocks, multipliers or multiplexers; of course, configuration memory is used throughout the logic block to control the specific function of each element within the block.

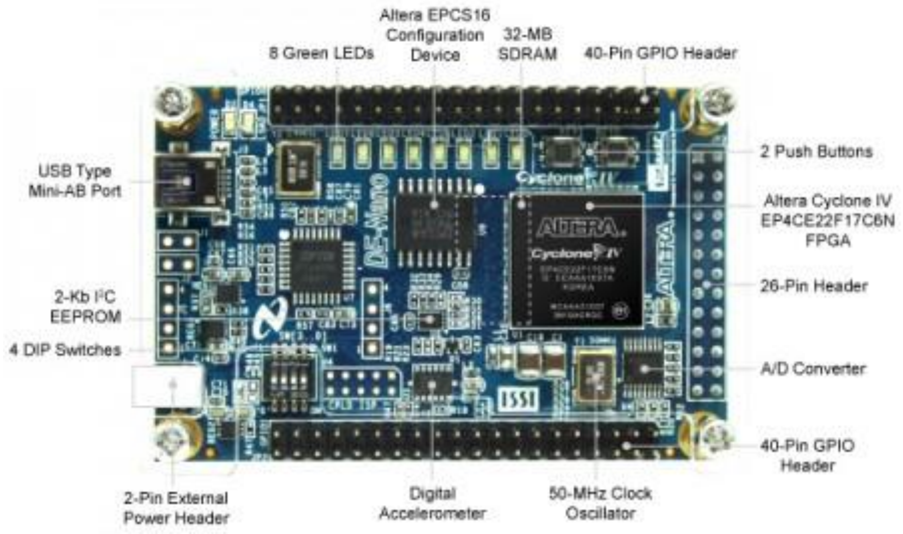
Programmable Interconnect-The programmable routing in an FPGA provides connections among logic blocks and I/O blocks to complete a user defined design. It consists of multiplexers, pass transistors and tri-state buffers, which forms the desired connection. Generally, pass transistors and multiplexers are used within a logic cluster to connect the logic elements together while all three are used for more global routing structures. Several global routing structures have been used in FPGAs as: island-style, cellular, bus-based and registered (pipelined) architectures.

Programming Technologies-

1. Antifuse Technology, which can be programmed only once. Devices manufactured by Quick Logic are examples of this type. Configuration is done by burning a set of fuses. These act as replacements for Application Specific ICs (ASIC) and used in places where protection of intellectual property is top priority.
2. Flash Technology based Programming, like devices from Actel. The FPGA may be reprogrammed several thousand times, taking a few minutes in the field itself for reprogramming and has non-volatile memory.
3. SRAM Technology based FPGAs, the currently dominating technology offering unlimited reprogramming and very fast reconfiguration and even partial reconfiguration during operation itself with little additional circuitry. Most companies like Altera, Actel, Atmel and Xilinx manufacture such devices.

Specific applications of FPGAs include digital signal processing, software-defined radio, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

Altera De0-Nano Board:



The De0-Nano board is a compact sized FPGA development platform suited for prototyping circuit designs such as robots and portable projects. The DE0-Nano board contains a Cyclone IV E FPGA which can be programmed using JTAG programming. This allows users to configure the FPGA with a specified design using Quartus II software. The programmed design will remain functional on the FPGA as long as the board is powered on, or until the device is reprogrammed.

Photoacoustic:

Photoacoustic is the study of vibrations induced in matter by light. Laser light causes localised heating when it is absorbed by a surface. This in turn makes the target area expand and sends a pressure wave through the rest of the material. This effect enables imaging of opaque systems, particularly biological samples. The photoacoustic effect is based on the sensitive detection of acoustic waves launched by the absorption of pulsed or modulated radiation by means of transient localized heating and expansion in a gas, liquid, or solid. This effect is due to the transformation of at least part of the excitation energy into kinetic (translational) energy by energy exchange processes between different degrees of freedom. Consequently, the absorption of modulated radiation generates an acoustic signal that is monitored by an ultrasonic piezoelectric transducer or optically with a contact-free method. Owing to the special nature of laser radiation, this light source plays an important role in photoacoustic.

DAQ:

Data acquisition (DAQ) is the process of measuring an electrical or physical phenomenon such as voltage, current, temperature, pressure, or sound with a computer. A DAQ system consists of sensors, DAQ measurement hardware, and a computer with programmable software. Data acquisition systems have many applications as Industrial applications, Medical applications and Scientific experiments etc.

Photoacoustic imaging (optoacoustic imaging) is a biomedical imaging modality based on the photoacoustic effect. In photoacoustic imaging, non-ionizing laser pulses are delivered into biological tissues (when radio frequency pulses are used, the technology is referred to as thermos acoustic imaging. Some of the delivered energy will be absorbed and converted into heat, leading to transient thermos elastic expansion and thus wideband (i.e. MHz) ultrasonic emission. The generated ultrasonic waves are detected by ultrasonic transducers and then analyzed to produce images.

In photoacoustic setup, very high frequency modulated square wave is given to the laser diode. Then modulated laser diode is passed through a sample, the acoustic signals are emitted from the sample and these acoustic signals are acquired by the sensor (analog). So for storing these signals in the computer (digital), ADC is required for converting analog data into digital.

FPGA, MCU and modulator are the parts of this project.

There are two objectives of this project:

(1) High Frequency Wave Generation

(2) Data Acquisition System Development

High Frequency Wave Generation:

(1) Generation of High Frequency (>=1 MHz) Square Wave using Altera De0-Nano Board and saving it in the memory
(2) Generation of a variable high frequency square wave (0.5MHz – 5MHz) using Arduino

- Data Acquisition System Development:
  (1) Development of a 32 bit counter using Altera De0-Nano board
  (2) Transferring data from external source to Arduino and save it in the Arduino memory
  (3) Development of a frequency counter for calculating the frequency of sine wave using Arduino
  (4) Transferring data from De0-Nano board to Computer and in the reverse way and saving it in Altera De0-Nano board's memory
  (5) Writing different data in the EEPROM memory at different addresses and read it back from the memory

Chapter 1: High Frequency Square Wave Generation

1.1Generation of High Frequency (>=1 MHz) Square Wave using Altera De0-Nano Board and saving it in the memory

Objective is to generate very high frequency (>=1 MHz) square waves for modulating laser.

Altera De0-Nano board and Quartus II 13.0 software are used in this project. Phase locked loop (PLL) circuit is used for generating high frequency square waves. A phase-locked loop or phase lock loop (PLL) is a control system that generates an output signal whose phase is related to the phase of an input signal.

The De0-Nano board includes a 50 MHz oscillator. The oscillator is connected directly to a dedicated clock input pin of the Cyclone IV E FPGA. The 50MHz clock input can be used as a source clock to drive the phase lock loops (PLL) circuit. The clock distribution on the De0-Nanoboard is shown in figure 1.1.1. Megafunctions such as the ones available in the LPM, are pre-designed modules that can be used in FPGA designs. These Altera-provided megafunctions are optimized for speed, area, and device family. We can increase efficiency by using a megafunction instead of writing the function. A PLL uses the on-board oscillator (DE0-Nano Board is 50 MHz) to create a constant clock frequency. To create the clock source, a pre-built LPM megafunction named ALTPLL is added.

Clock source is used as input of PLL circuit and clock division factor or clock multiplication factor is used for generating desired frequency square waves as output of PLL circuit. Figure 1.1.2 shows clock division factor or clock multiplication factor can be set for generating desired frequency square waves as output of PLL circuit.

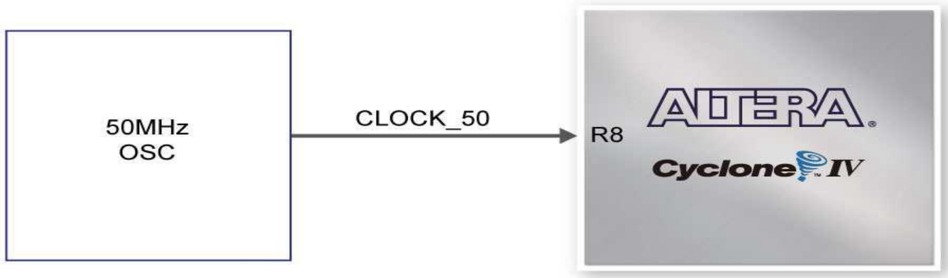
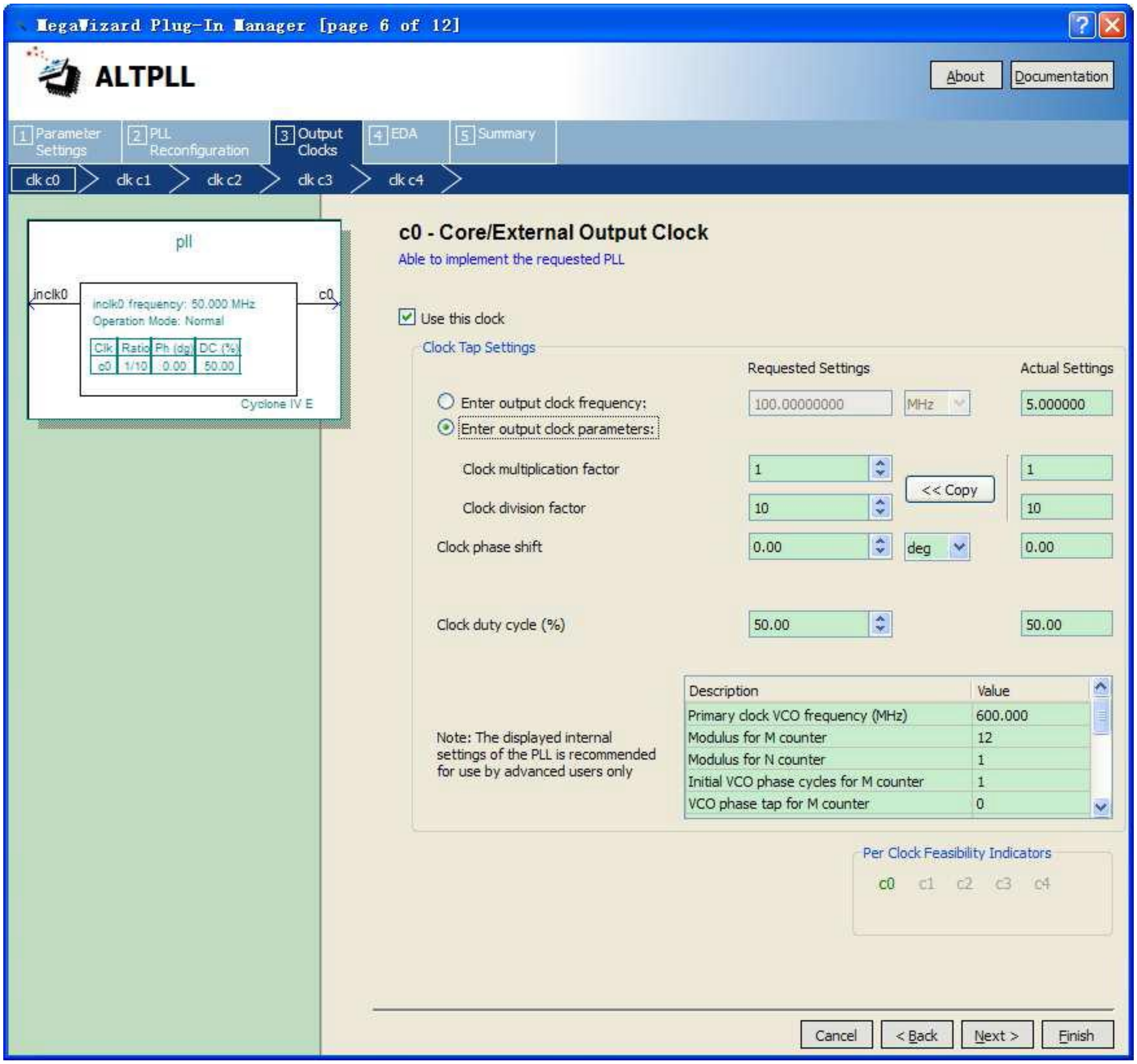


Figure 1.1.1 Block diagram of the clock distribution



inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/10	0.00	50.00

c0 - Core/External Output Clock

Able to implement the requested PLL

[✓] Use this clock

Clock Tap Settings

	Requested Settings		Actual Settings
○ Enter output clock frequency:	100.00000000	MHz	5.000000
◉ Enter output clock parameters:			
Clock multiplication factor	1		1
Clock division factor	10		10
Clock phase shift	0.00	deg	0.00
Clock duty cycle (%)	50.00		50.00

Note: The displayed internal settings of the PLL is recommended for use by advanced users only

Description	Value
Primary clock VCO frequency (MHz)	600.000
Modulus for M counter	12
Modulus for N counter	1
Initial VCO phase cycles for M counter	1
VCO phase tap for M counter	0

Per Clock Feasibility Indicators

c0 c1 c2 c3 c4

Figure 1.1.2 MegaWizard Plug-In Manager

In this, clock division factor is set as 50 and clock multiplication factor is set as 1 for obtaining 1 MHz square wave. Assigning input pin as R8 and output pin as D3, output 1 MHz square wave is observed on the CRO as shown in figure 1.1.3.

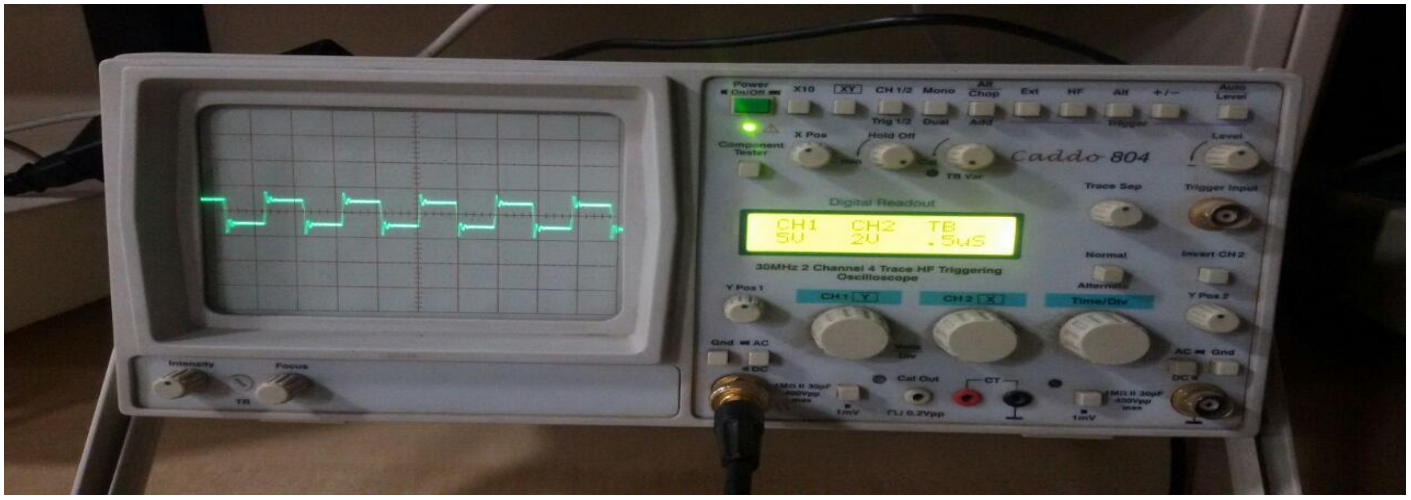


Figure 1.1.3

De0-Nano EPCS64 FLASH device is configured for saving the program in the chip.De0-Nano has a flash device named as EPCS64. It has 64Mbits capacity. Procedure that is used for configuring De0-Nano EPCS64 FLASH device:

- Generate the .sof file by compiling program.
- File > Convert Programming Files
- Under 'output programming file', select 'JTAG indirect configuration file' as the programming file type.
- Select 'EPCS64' as the configuration device.
- Give a file name and path.
- Under 'Input files to convert' click on 'Flash loader'.
- Then click on 'Add device' button at the right side of the window.
- Select 'EP4CE22' under 'Cyclone IV E' and click 'ok'.
- Under 'Input files to convert' click on 'SOF data'.
- Then click on 'Add file' button at the right side of the window.
- Select you generated '.sof' file which is generally it is located in the 'output_files' folder.
- Finally click on generate button.

After the configuration of EPCS64, on starting the device current program is seen in it instead of the default program.Thus using this process, very high frequency square waves (like 5MHz, 20MHz, 70MHz etc.) can be generated.

1.2 Generation of a variable high frequency square wave (0.5MHz – 5MHz) using Arduino

Objective is to generate a square wave which frequency becomes 0.5 MHz, 1 MHz, 1.5 MHz, 2 MHz, 2.5 MHz, 3 MHz, 3.5 MHz, 4 MHz, 4.5 MHz and 5 MHz after every 100 microsecond and after this, it stops for 1 millisecond and then same pattern repeats.

For this, Arduino code is developed as:

```
const int OutputPin = 11;
const int prescale  = 1;
float ocr2aval;
int i;
int j;
void setup(){
pinMode(OutputPin, OUTPUT);
Serial.begin(9600);
}
void loop(){
TCCR2A = ((1 << WGM21) | (1 << COM2A0));
   TCCR2B = (1 << CS20);
    TIMSK2 = 0;  //Compare-match register A interrupt for timer2 is disabled
    OCR2A = ocr2aval;  // This value determines the output frequency
if(j%11<=9){
  if(i%10==0){
                ocr2aval = 15;
 delayMicroseconds(100);}
    if(i%10==1){
                     ocr2aval = 7;
                    delayMicroseconds(100);}
 if(i%10==2) {
ocr2aval = 4.333334;
delayMicroseconds(100);}

            if(i%10==3){
            ocr2aval = 3;
            delayMicroseconds(100);  }
                    if(i%10==4) {
                                ocr2aval = 2.2;
                                delayMicroseconds(100); }
                   if(i%10==5){
                                ocr2aval = 1.666668;
                               delayMicroseconds(100); }
                   if(i%10==6){
                                 ocr2aval = 1.285716;
                                 delayMicroseconds(100); }
                    if(i%10==7){
                                     ocr2aval = 1;
                                    delayMicroseconds(100);  }
                     if(i%10==8){
                                     ocr2aval = 0.777788;
                                   delayMicroseconds(100);}
            if(i%10==9){
                                    ocr2aval = 0.6;
                                  delayMicroseconds(100); }
                        i=i+1;
             float period    = 2.0 * prescale * (ocr2aval+1) / (F_CPU/1.0e6);
            float freq      = 1 / period;
                      Serial.print("Period    = ");
                      Serial.print(period);
             Serial.println(" microseconds");
                      Serial.print("Frequency = ");
                      Serial.print(freq);
                      Serial.println(" MHz");}
```

```
  if(j%11==10){
  delay(1);  }
  j=j+1;
 }
```

<u>EXPLANATION OF CODE:</u>

- Timer2 is used to generate a signal for a particular frequency on pin 11.
- Constants are computed at compile time. If the prescale value is changed, it affects CS22, CS21, and CS20.
- For a given prescale value, the eight-bit number is loadedinto OCR2A determines the frequency according to thefollowing formulas:
  constant float period   = 2.0 * prescale * (ocr2aval+1) / (F_CPU/1.0e6)
  constant float frequency = 1 / period
- Timer2 CTC mode prescale division value (=1, no prescaling) is set and CPU Clock is used.
- OCR2A output pin at which output frequency can be checked is digital pin 11 for ATmega328 boardand digital pin 10 Mega boards. Arduino ATmega328 board is used in this project.
- It is necessary that Compare-match register A interrupt for timer2 is disabled.

<u>RESULT-</u>After running the above Arduino code and connecting digital pin 11 with CRO, the variable frequency square wave is observed on CRO as shown in figure 1.2.1 and figure 1.2.2.

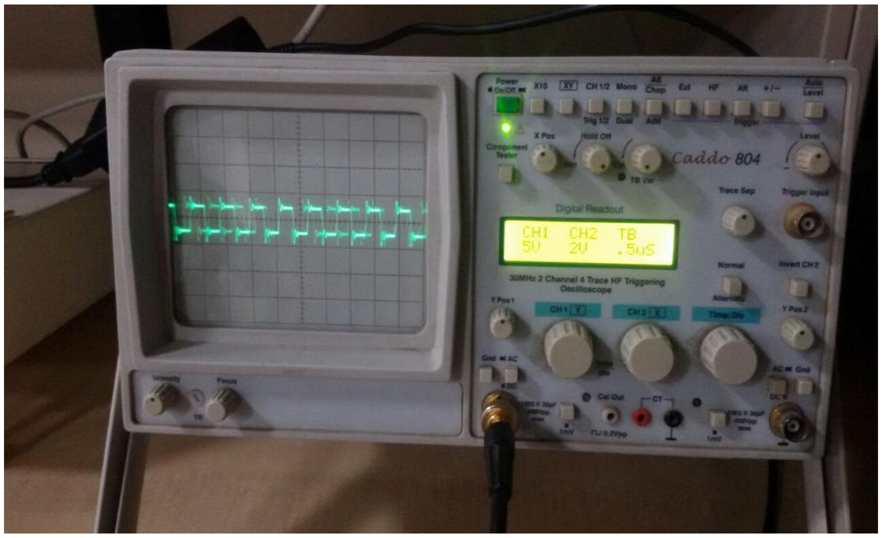


figure1.2.1

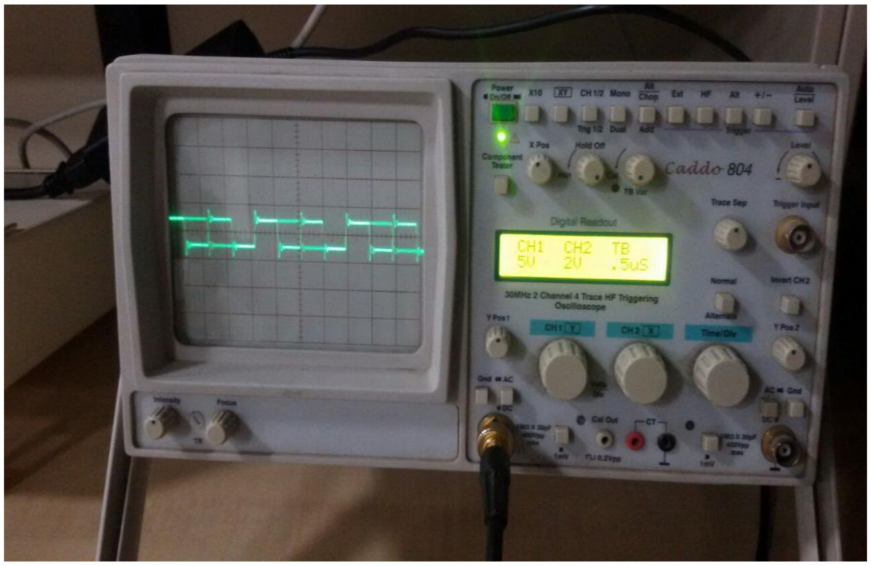

CH1 CH2 TB
5V 2V .5uS
Caddo 804
30MHz 2 Channel 4 Trace HF Triggering Oscilloscope

figure1.2.2

Data Acquisition System Development

2.1 Development of a 32 bit counter using Altera De0-Nano board

Objective is to develop a 32 bit counter using Altera De0-Nano board.

Figure 2.1.1 shows a block diagram of the FPGA design flow.

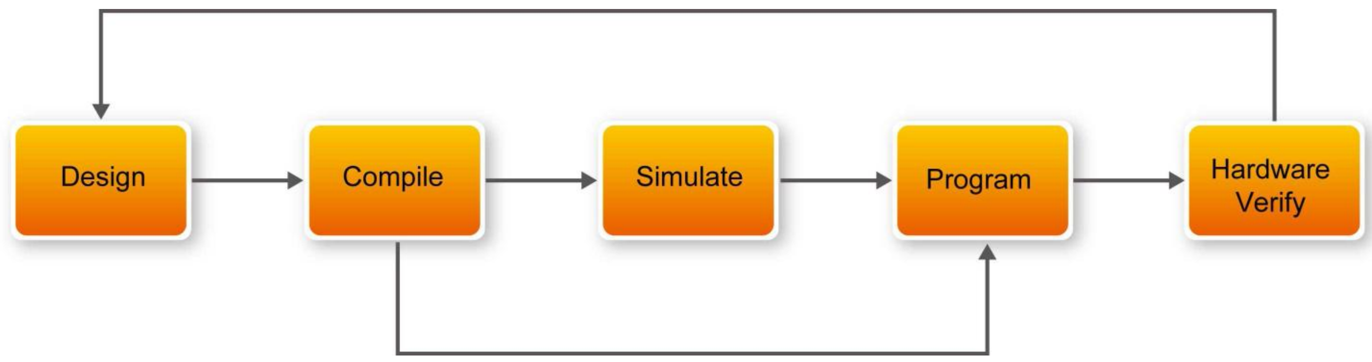


Figure 2.1.1

Computer and Altera De0-Nano Board hardware and Quartus II 13.0 software are used to develop 32 bit counter.

In this, following steps are used:

Creating an FPGA design

- Assigning the pins
- Compiling the FPGA design
- Programming the FPGA device
- Verifying the hardware

Verilog code for counter is:

```
module simple_counter (
                      CLOCK_5,
                       counter_out );
          input CLOCK_5;
          output [31:0] counter_out;
          reg [31:0] counter_out;
```

```
always @ (posedge CLOCK_5)
            begin
           counter_out <= counter_out + 1;
           end
           endmodule
```

- For developing a 32 bit counter, a symbol for 32 bit counter is created that has input 5 MHz clock and 32 bit counter output.
- PLL circuit is used for generating 5 MHz clock from 50 MHz clock source.
- A multiplexer is used to route the counter output to the LED pins on the De0-Nano board.
- The design multiplexes two portions of the counter bus to four LEDs on the De0-Nano board.

From the above Verilog code, a symbol for counter is created that is placed in BDF (Block Diagram File) file as shown in figure 2.1.2.

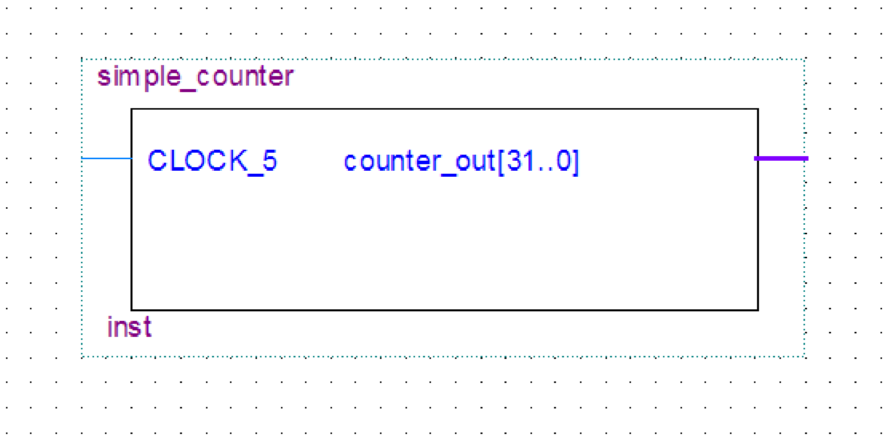


Figure 2.1.2

PLL circuit is used for generating 5 MHz frequency from 50 MHz clock source as shown in Figure 2.1.3. The output of the PLL circuit is connected to the input of the counter.

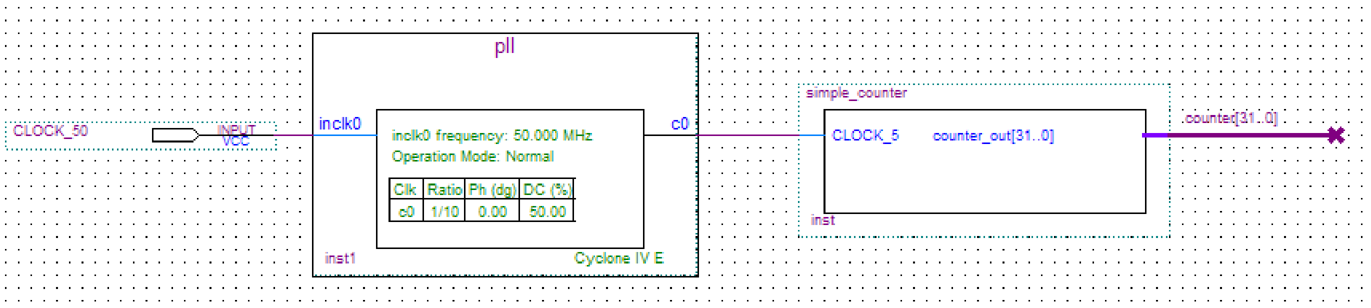


Figure 2.1.3

A multiplexer is used as shown in Figure 2.1.4. The design multiplexes two portions of the counter bus to four LEDs on the De0-Nano board.

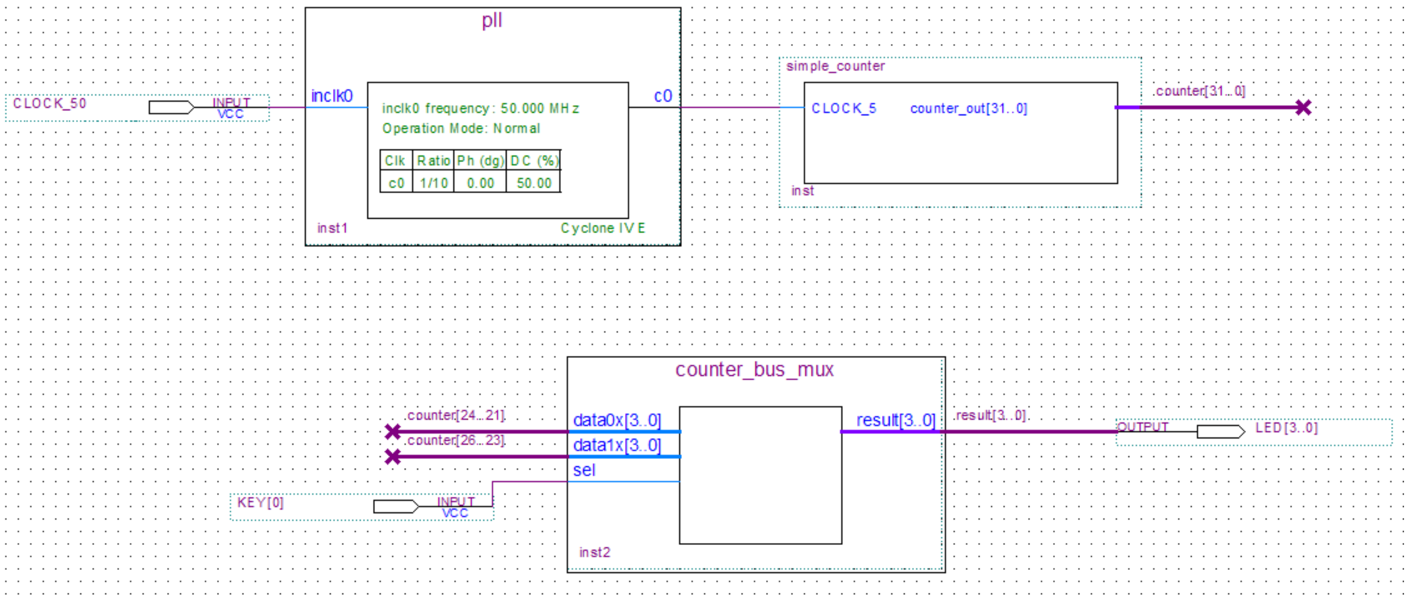


Figure 2.1.4

After that pins are assigned for Clock_50, KEY[0] and LED[3..0], the design is compiled and the FPGA device is programmed.

OBSERVATIONS-

- The four development board LEDs appear to be advancing slowly in a binary count pattern, which is driven by the simple_counter bits [26..23].
- The LEDs are active low, therefore, when counting begins all LEDs are turned on (the 0000 state).
- On pressing and holding KEY [0] on the development board the LEDs advance more quickly.
- Pressing this KEY causes the design to multiplex using the faster advancing part of the counter (bits [24..21]).

2.2Transferring data from external source to Arduino and save it in the Arduino memory

Objective is to transfer data from external source to Arduino and save it in the Arduino memory.

Computer, Arduino, 10 k ohm Potentiometer (Hardware) and Arduino software are used for this.

In this, Potentiometer is used as external source and data is stored in the EEPROM memory of the Arduino.An EEPROM is an Electrically Erasable Programmable Read-Only Memory. It is a form of non-volatile memory that can remember things with the power being turned off, or after resetting the Arduino.The beauty of this kind of memory is that we can store data generated within a sketch on a more permanent basis.

Internal EEPROM is useful for situations where data that is unique to a situation needs a more permanent home. For example, storing the unique serial number and manufacturing date of a commercial Arduino-based project – a function of the sketch could display the serial number on an LCD, or the data could be read by uploading a sketch.

Anything that can be represented as bytes of data, can be stored in internal EEPROM. One byte of data is made up of eight bits of data. A bit can be either on (value 1) or off (value 0), and are perfect for representing numbers in binary form.

ARDUINO CODE:

```
#include <EEPROM.h>
int k;
int EESize = 1024; //size in bytes of board's EEPROM
void setup(){
  Serial.begin(9600);
}
void loop(){
 int sensorValue = analogRead(A0);
Serial.println("Displaying numbers....");
for (int i =0; i < EESize; i++){
k=analogRead(A0) / 4;
```

```
    EEPROM.write(i , k);
  }
Serial.println();
  for(int a =0; a<EESize; a++){
k = EEPROM.read(a);
Serial.println("EEPROM position");
Serial.println(a);
Serial.println(" contains ");
Serial.println(k);
   delay(50);}
}
```

ARDUINO CODE EXPLANATION-

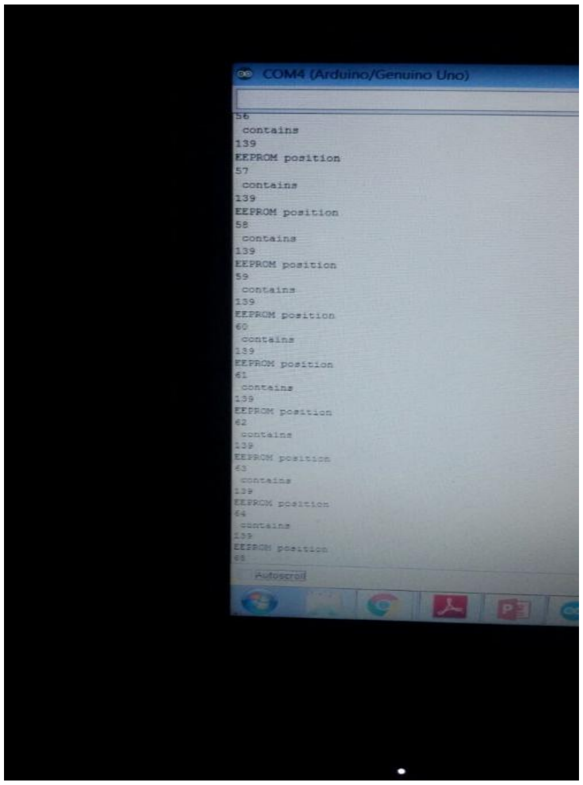
EEPROM.write(a,b): The parameter *a* is the position in the EEPROM to store the integer (0~255) of data *b*. There are 1024 bytes of memory storage, so the value of a is between 0 and 1023.

k=EEPROM.read(a): Where *k* is an integer to store the data from the EEPROMposition *a*.

The variable EEsize is the upper limit of your EEPROM size, so this would be 1024 for an Arduino Uno, or 4096 for a Mega. In this, Potentiometer is used as external source. Potentiometer is connected with analog input pin A0. Thus using potentiometer, different values are given to pin A0. These values are stored in the Arduino memory EEPROM with the locations.

OBSERVATION-

On making proper connections and running the Arduino code, values given from the potentiometer are observed with memory locations on the serial monitor as shown in Figure 2.2.1.



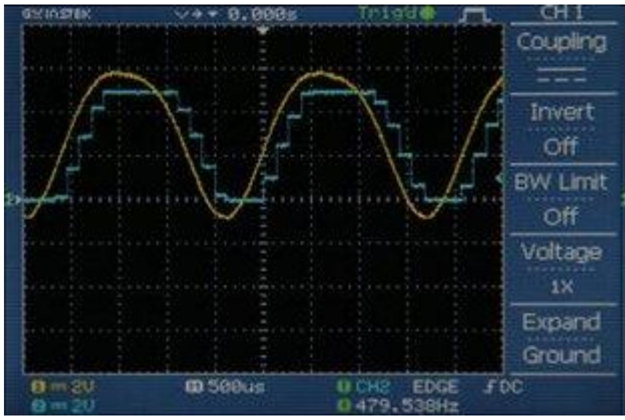
```
56
 contains
139
EEPROM position
57
 contains
139
EEPROM position
58
 contains
139
EEPROM position
59
 contains
139
EEPROM position
60
 contains
139
EEPROM position
61
 contains
139
EEPROM position
62
 contains
139
EEPROM position
63
 contains
139
EEPROM position
64
 contains
139
EEPROM position
65
```

Figure 2.2.1

2.3Development of a frequency counter for calculating the frequency of sine wave using Arduino

Objective is to develop a frequency counter for calculating frequency of sine wave.

<u>Principle used-</u>



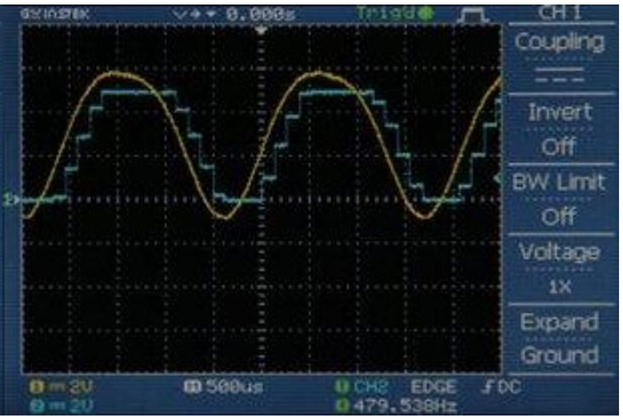


Figure 2.3.1 Figure 2.3.2

Figures 2.3.1 and 2.3.2 show the incoming signal (yellow) from the function generator and the data stored in the Arduino (blue) for both 8kHz and 38.5kHz sampling rates.

The sampling frequency (or sample rate) is the number of samples per second in a Sound.In this, sampling frequency is taken equal to 38.5 kHz.

In this, the period of an incoming sine wave is calculated. To do this a timer is set up in the ADC interrupt that increments each time the interrupt executes (a rate of 38462Hz/38.5kHz). Each time the incoming signal crosses 2.5V (or any selected cut off point) with a rising slope, the current value of the timer is sent to a variable called "period" and reset the timer to 0. All takes place within the ADC interrupt. Using the value of period, frequency of sine wave is calculated by following formula:

Frequency = 38462/period

There are different types of ADC architecture. All AVRs that contain an ADC have a successive approximation ADC. Figure 2.3.3 is a simplified diagram of the AVR ADC. The AVR has an 8 channel analog multiplexer. This multiplexes the 8 analog pins into the single 10 bit ADC. Only one ADC operation can be carried out at a time.

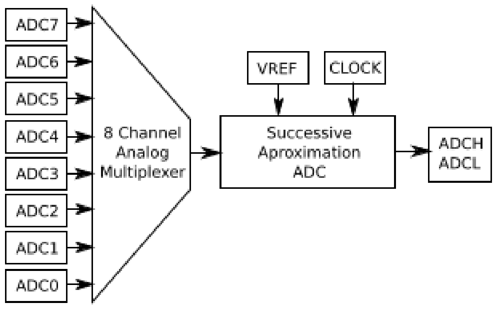


Figure 2.3.3

The AVR ADC has two operating modes. These are single conversion & free running.

In single conversion mode we have to initiate each conversion. When the conversion is complete, the results are placed into the ADCH & ADCL registers and ADIF is set. No other conversion is started.

In free running mode, we initiate the first conversion and then the ADC will automatically start the next conversions, as soon as the previous one is complete. We tell the ADC which channel to convert by setting the ADMUX register.

ADMUX - ADC Multiplexer Selection Register;

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

REFS1& REFS0 are used for selecting voltage reference for the ADC.

ADLAR (ADC Left Adjust Result) bit is used to set how the conversion result is stored in the two ADC data bytes. If the bit is set, the result is left-aligned, otherwise it is placed right-aligned.

Res: Reserved bit is reserved and is always read as 0.

MUX3, MUX2, MUX1, MUX0 (Analog Channel Selection Bits)These bits determine which analog input is connected to the ADC.

ADCSRA ADC Control and Status Register A:

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADEN: ADC Enable-With a 1 in this bit the ADC is released. Deleting the bit will disable the ADC. If the bit is deleted during an ongoing conversion, this conversion is terminated.

ADSC: ADC Start Conversion-In single-conversion mode, this bit must be set to start each conversion.

ADFR: ADC Free running Select-When this bit is set, the ADC operates in continuous conversion mode. In this mode, the ADC continuously records and updates the data registers. Deleting the bit stops further conversion.

ADIF: ADC Interrupt Flag-This bit is set when a conversion is completed and the data registers have been updated.

ADIE: ADC Interrupt Enable-If this bit is set and the I-bit in the SREG also releases the interrupts globally, then the ADC Conversion Complete Interrupt is enabled.

ADPS2, ADPS1, ADPS0 are used for determining the division factor.

Following ARDUINO CODE is developed for calculating the frequency of sine wave:

```
//data storage variables
 byte newData = 0;
  byte prevData = 0;
 //freq variables
unsigned int timer = 0;//counts period of wave
  unsigned int period;
   int frequency;
  void setup(){
  Serial.begin(9600);
  cli(); //disable interrupts
  ADCSRA = 0;
  ADCSRB = 0;
  ADMUX = 0x40;                        // analog pin selection(A0)
ADMUX |= (1 << REFS0);           // reference voltage
  ADMUX |= (1 << ADLAR);
ADCSRA |= (1<< ADPS2) | (1 << ADPS0);         // ADC clock with 32 prescaler-
16mHz/32=500kHzADCSRA |= (1 << ADATE);           //enable auto trigger
ADCSRA |= (1 << ADIE); //enable interrupts when measurement complete
ADCSRA |= (1 << ADEN); //enable ADC
ADCSRA |= (1 <<ADSC); //start ADC measurements
 sei();//enable interrupts}
ISR(ADC_vect) {
 prevData = newData;//store previous value
newData = ADCH;            //get value from selected analog pin

if (prevData < 127 && newData >=127){
period = timer;
timer = 0; }
 timer++;}
void loop(){
frequency = 38462/period;
  Serial.print(frequency);
 Serial.println(" Hz");
 delay(100);
}
```

<u>RESULT:</u>

Thus running the above Arduino code, frequency of the sine wave incoming at any analog pin is calculated for every cutoff point.

<u>CAUTION:</u>

This method fails when wave become more complicated (and cross cut off point more than twice in one cycle).

2.4 Transferring data from De0-Nano board to Computer and in the reverse way and saving it in Altera De0-Nano board's memory

Objective is to transfer data from De0-Nano board to Computer, from computer to De0-Nano board and saving the program in the memory.

A USB-to-UART converter module is used to provide serial communication between PC and user logic within the FPGA, on the De0-Nano board. Application software on the PC allows reading from user defined status registers and writing to user defined control registers in the FPGA. Typically, the application software communicates with the USB-to-UART bridge using virtual com port device drivers. Hence, the software treats the USB port as a serial one.

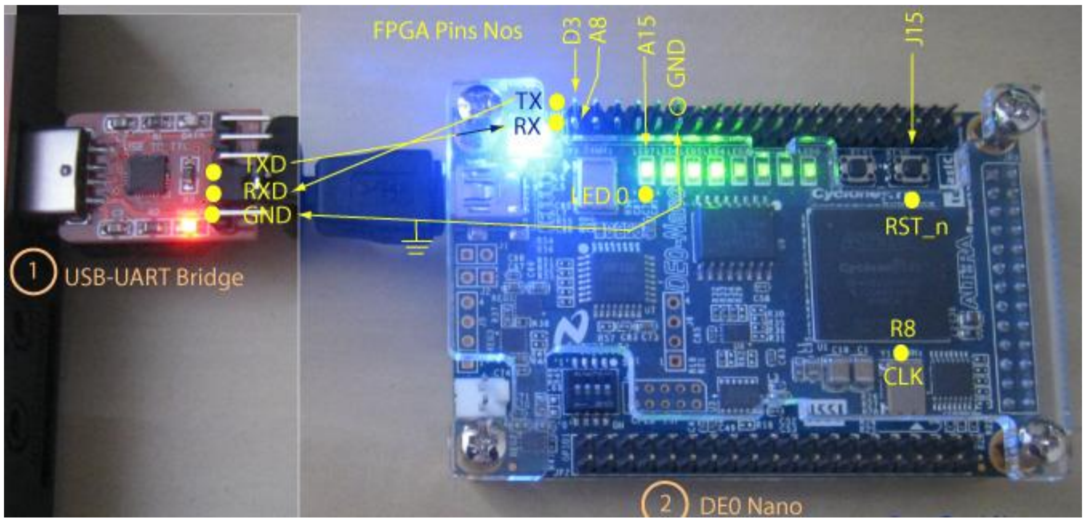


Figure 2.4.1

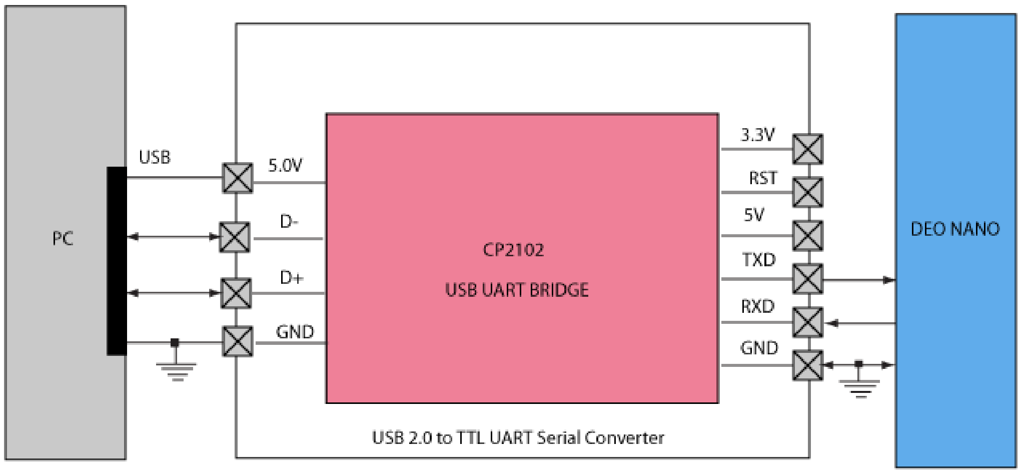


Figure 2.4.2

The characteristics of a serial data frame, of the protocol is defined to be the following. When the transceiver's lines are IDLE they are held at a high ('1') logic level. When active, the serial data stream consists of 1 start bit, which is at a fixed, low ('0') logic level and 8 data bits. The stream is terminated by 1 stop bit, which is at a high ('1') logic level and no parity bits.

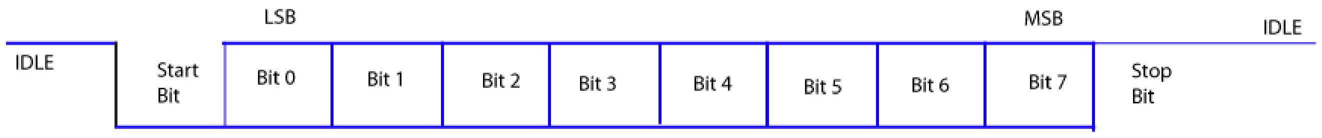


Figure 2.4.3

Serial communication is provided between application software on a PC and user logic on the De0-Nano board. The bidirectional communication protocol is developed in this project that allows to send/receive data, controlling FPGA through PC and so on.To correctly interpret the data sent and received between the two, a Universal Asynchronous Receiver/Transmitter, or UART is developed. The developed UART is full-duplex, allowing mutually exclusive data transmission and reception.

Logic used-

For the transmitter, 50 MHz clock source clk, a START signal which will start the transmission, the BUSY signal which will indicate transmission is running currently, 8 bit vector which is sent, transmission line TX_line which goes out of our FPGA board into PC, are used in this project.

In the architecture, some signals are used like prescalar (PRSCL) for main clock, INDEX to select which bit we are going to send and DATAFLL vector which consists of 8 bit data + start bit + stop bit.

New transmission starts only if no communication is currently running. TX_FLAG = '0' AND START SIGNAL is HIGH ('1') only if both conditions are true, TX_ FLAG is set to '1' read the data and place it into the DATAFLL vector between the start bit (DATAFLL (0)) and stop bit (DATAFLL (9)). Now, if TX_FLAG is set to '1' data transmission is started since the common data rate for UART is 9600 bit/sec.

It generated a simple prescalar which count to 5208 (50*10^6 / 9600 = 5208). Thus a desired bit rate is obtained.The bidirectional communication protocol is developed in this project that allows to send/receive data, controlling FPGA through PC and so on.

If prescalar hits 2600 the current bit is passed to TX_line &bit index is increased by 1 if it is less than 9. If it is already 9th bit, the last bit is sent and the transmission is complete.

At the end of the transmission, the TX_FLAG, INDEX and the busy indicator are reset.

Same logic is used for the receiver but in the reverse way. Both receiver and transmitter are used as components of the UART. (In appendices section, A.1 is the code for transmitter, A.2 is the code for UART only for the transmission, A.3 is the code for the receiver,A.4 is the code for UART for both reception and transmission).

For saving this program in the memory of De0-Nano board, De0-Nano EPCS64 FLASH device is configured (which is already discussed in section 1.1).

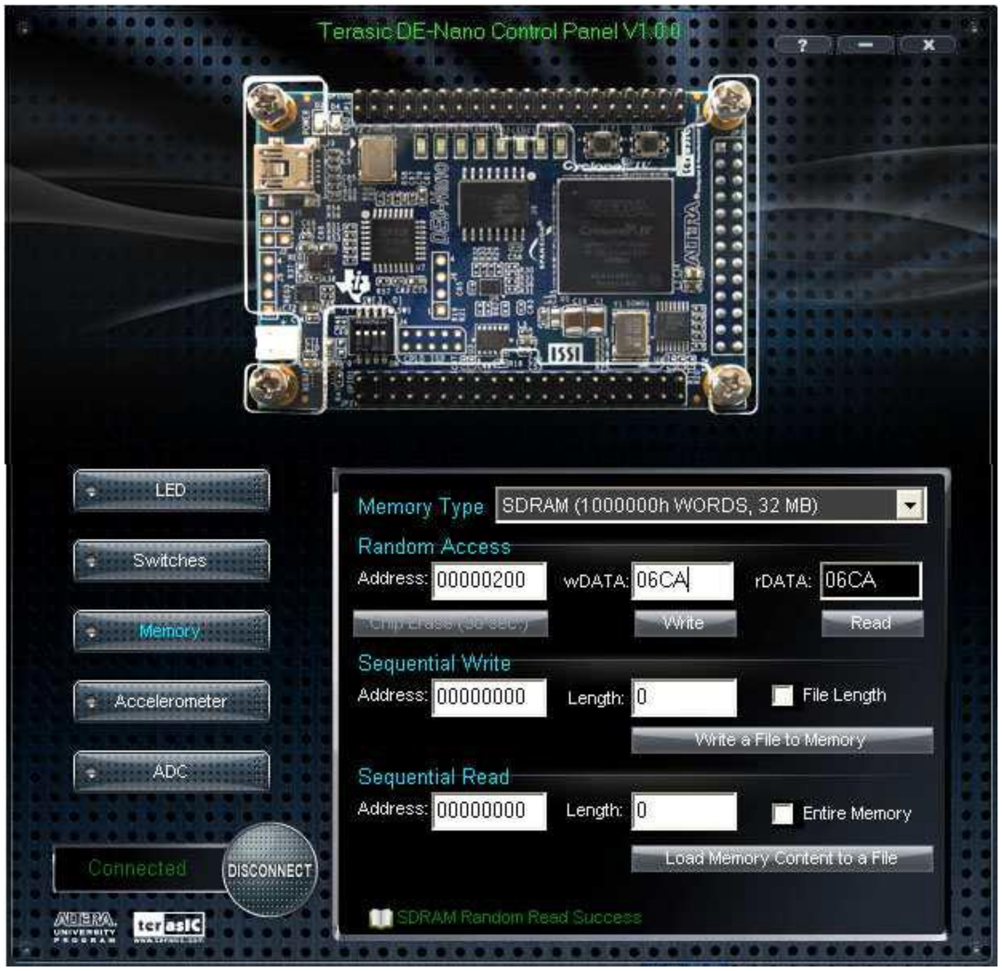
RESULTS:

On making proper connections, as shown in figure 2.4.1 results are obtained successfullyfor the transmitter using the Realterm software. But for the receiver, it has some problem. To solve this problem, I will work on this in future.

2.5 To write different data in the EEPROM memory at different addresses and read it back from the memory

Objective is to write different data (bitwise) in the EEPROM memory at different addresses and reading that data back from the memory.

The Control Panel is used to write/read data to/from the SDRAM / EEPROM /EPCS on the De0-Nano board. Memory Initialization File (.mif) and Hexadecimal (Intel-Format) File (.hex) is written/read to/from the SDRAM / EEPROM /EPCS on the DE0-Nano board.



Memory Type: SDRAM (1000000h WORDS, 32 MB)

Random Access
Address: 00000200 wDATA: 06CA rDATA: 06CA

Sequential Write
Address: 00000000 Length: 0 File Length

Sequential Read
Address: 00000000 Length: 0 Entire Memory

SDRAM Random Read Success

Figure 2.5.1

A 16-bit word can be written into the SDRAM by entering the address of the desired location, specifying the data to be written, and pressing the Write button. Contents of the location can be read by pressing the Read button. Figure 2.5.1shows the result of writing and reading the hexadecimal value 06CA into offset address 200.

Sequential Write function is used to write content of a file into SDRAM/ EEPROM/EPCS andSequential Read function is used to read content of a file from SDRAM/ EEPROM/EPCS.

<u>File that is saved in SDRAM / EEPROM / EPCS is:</u>

```
DEPTH = 32;                 -- The size of memory in words
WIDTH = 8;                  -- The size of data in bits
ADDRESS_RADIX = HEX;        -- The radix for address values
DATA_RADIX = BIN;           -- The radix for data values
CONTENT                     -- start of (address : data pairs)
BEGIN
00 : 00000000;              -- memory address : data
01 : 00000001;
02 : 00000010;
03 : 00000011;
04 : 00000100;
05 : 00000101;
06 : 00000110;
07 : 00000111;
08 : 00001000;
09 : 00001001;
0A : 00001010;
0B : 00001011;
0C : 00001100;
END;
```

I2C (inter integrated circuit) operates on slow clock speeds. In I2C communication, Altera Cyclone IV acts as master and memory device acts as slave. I2C clock is unidirectional from master to slave while I2C data bus is bidirectional.

I2C write Protocol: In the starting, a start bit, then control Byte, then Acknowledge from slave, then word address, then Acknowledge from slave, then Data, then Acknowledge from slave and then stop bit. START and STOP conditions are:

STARTcondition: When clock signal is high, Data line is high, Data line goes low then clock goes low.

STOP condition: When clock signal goes high then Data line also goes high.
If a value 8’AA is written into the memory address 8’ h00 then here Control byte is 8’hA0, word address is 8’h00, data is 8’AA. 8’hA0 is the EEPROM address.

A Verilog code (in Appendices section, A.5) is used for writing data in EEPROM that is configured through a 2-wire I2C serial interface. EEPROM can operate at maximum 400 kHz while Altera De0-Nano board has 50 MHz clock. So it is counted down in the Verilog code. In the code, data are written using I2C write protocol.

<u>RESULTS-</u>

Results can be checked on Signaltap II.

Conclusion & Future Work

In this project, very high frequency (>1 MHz) square waves are generated using Arduino and Altera De0-Nano, those very are useful for the process of laser modulation. Using Arduino, variable very high frequency square wave is generated.

In this project, frequency counter for calculating the frequency of sine wave is discussed for the purpose of calculating frequency difference between two up going levels.

Transferring data between computer and Arduino or Altera De0-Nano board,Saving data in the memory of Arduino or Altera De0-Nano board are discussed in this project.

In this project, I generated a frequency counter for sine wave that is calculating frequency of sine wave incoming at one analog pin of Arduino. Next, I want to generate a frequency counter that is calculating frequency difference of two sine waves incoming at same time at two analog pins of Arduino but it is not working properly. I will work in future on this.

I have worked with Microcontroller and FPGA. Now future work is working on advanced ADCs (Analog to digital convertors).

References

- Microcontroller: Internals, Instructions, Programming and Interfacing
  By- Subrata Ghosal, Publisher-Pearson, Reeased-2008
- Michael Mc Roberts – Beginning Arduino
- *1076-1987 – IEEE Standard VHDL Language Reference Manual*. 1988
- Peter J. Ashenden, "The Designer's Guide to VHDL, Third Edition (Systems on Silicon)", 2008,
- http://www.electronicshub.org/microcontrollers/
- https://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp
- https://embeddedmicro.com/tutorials/mojo-fpga-beginners-guide/what-is-an-fpga
- http://www.introtoarduino.com/downloads/IntroArduinoBook.pdf
- https://www.sparkfun.com/datasheets/IC/cp2102.pdf
- https://www.scribd.com/document/87292764/DE0-Nano-User-Manual-v1-5
- *1364-1995 — IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language*. 1996.
- http://www.hfremote.us/files/Arduino.pdf
- http://www.makeuseof.com/tag/getting-started-with-arduino-a-beginners-guide/
- http://ece-research.unm.edu/jimp/vhdl_fpgas/slides/UART.pdf
- http://idielectronica.blogspot.in/2015/10/creating-project-with-terasic-de0-nano.html

Appendices

A.1 VHDL code for transmitter:

```
library ieee ;
 use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
 entity TX is
  port(
         CLK: in std_logic;
         START:in std_logic;
         BUSY : out std_logic;
DATA: in std_logic_vector(7 downto 0);
         TX_LINE: out std_logic );
end TX;
  architecture MAIN of TX is
signal PRSCL : integer range 0 to 5208:=0;
signal index : integer range 0 to 9:=0;
signal DATAFLL : std_logic_vector (9 downto 0);
  signal TX_FLAG: STD_LOGIC:='0';
  begin
  process(CLK)
  begin
  if(CLK'EVENT and CLK='1') then
if(TX_FLAG='0' and START='1') then
             TX_FLAG<='1';
             BUSY<='1';
             DATAFLL(0)<='0';
             DATAFLL(9)<='1';
             DATAFLL(8 downto 1)<=DATA;
 TX_LINE<='1';

end if;
  IF(TX_FLAG='1') then
            IF (PRSCL <5207) then
            PRSCL<=PRSCL+1;
   else
          PRSCL<=0;
         end if;
      IF(PRSCL=2600)THEN
TX_LINE<=DATAFLL(INDEX);

           IF(INDEX<9)THEN
            INDEX<=INDEX+1;
       ELSE
               TX_FLAG<='0';
               BUSY<='0';
               INDEX<=0;
               END IF;
           END IF;
         END IF;
END IF;
end process ;
end MAIN;
```

A.2 VHDL code for UART (only transmission):

```
library ieee ;
     use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
    dentity UART is
    port (
    CLOCK_50: in std_logic;
    SW: in std_logic_vector(3 downto 0);
KEY: in std_logic_vector(1 downto 0)
LEDG : out std_logic_vector (7 downto 0);
    UART_TXD : out std_logic;
```

```
   UART_RXD: in std_logic;
 );
   end UART ;
ARCHITECTURE MAIN OF UART IS
SIGNAL TX_DATA: STD_LOGIC_VECTOR(7  downto 0);
SIGNAL TX_START: STD_LOGIC:='0';
SIGNAL TX_BUSY: STD_LOGIC:='0';
SIGNAL RX_DATA: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RX_BUSY: STD_LOGIC:='0';
COMPONENT TX
 PORT(
CLK:IN STD_LOGIC;
START:IN STD_LOGIC;
BUSY:OUT STD_LOGIC;
DATA: IN STD_LOGIC_VECTOR(7 downto 0);
TX_LINE:OUT STD_LOGIC
 );
END COMPONENT TX;
BEGIN
C1: TX PORT MAP (CLOCK_50,TX_START,TX_BUSY,TX_DATA,UART_TXD);
--C2: RX PORT MAP (CLOCK_50,UART_RXD,RX_DATA,RX_BUSY);
PROCESS(CLOCK_50)
BEGIN
IF(CLOCK_50'EVENT AND CLOCK_50='1')THEN
 IF(KEY(0)='1' AND TX_BUSY='0')THEN -- Key(0)='0' MEAN that the key is pressed
TX_DATA<="0000" & SW(3 DOWNTO 0);
TX_START<='1';
LEDG<=TX_DATA;
ELSE
 TX_START<='0';
TX_DATA<=(others=>'0') ;
END IF;
END IF;
END PROCESS;
END MAIN;
```

A.3 VHDL code for receiver :

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity RX is
port(
CLK: in std_logic;
 BUSY : out std_logic
 DATA: out std_logic_vector(7 downto 0);
RX_LINE: in std_logic
        );
end RX;
architecture MAIN of RX is
signal PRSCL : integer range 0 to 5208:=0;
signal index : integer range 0 to 9:=0;
signal DATAFLL : std_logic_vector (9 downto 0);
signal RX_FLAG: STD_LOGIC:='0';
begin
process(CLK)
begin
if(CLK'EVENT and CLK='1') then
 if(RX_FLAG='0' and RX_LINE='0') then
INDEX<=0;
 PRSCL<=0;
BUSY<='1';
RX_FLAG<='1';
 end if;
  IF(RX_FLAG='1') then
```

```
DATAFLL(INDEX)<=RX_LINE;
IF (PRSCL <5207) then
 PRSCL<=PRSCL+1;
else
PRSCL<=0;
end IF;
 IF(PRSCL=2600)THEN
IF(INDEX<9)THEN
INDEX<=INDEX+1;
   ELSE
IF (DATAFLL(0)='0' AND DATAFLL(9)='1') then
 DATA<= DATAFLL(8 downto 1);
ELSE
DATA<= (OTHERS=>'0');

END IF;

 RX_FLAG<='0';

   BUSY<='0';
END IF;
 END IF;
 END IF;
END IF;
end process ;
end MAIN;
```

A.4 VHDL code UART (both transmission & reception):

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity UART is
port (
CLOCK_50: in std_logic;
SW: in std_logic_vector(3 downto 0);
KEY: in std_logic_vector(1 downto 0)

LEDG : out std_logic_vector (7 downto 0);
LEDR : out std_logic_vector (7 downto 0);
UART_TXD : out std_logic;
UART_RXD: in std_logic
        );
end UART ;
ARCHITECTURE MAIN OF UART IS
SIGNAL TX_DATA: STD_LOGIC_VECTOR(7  downto 0);
SIGNAL TX_START: STD_LOGIC:='0';
SIGNAL TX_BUSY: STD_LOGIC:='0';
SIGNAL RX_DATA: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RX_BUSY: STD_LOGIC:='0';
COMPONENT TX
 PORT(
 CLK:IN STD_LOGIC;
START:IN STD_LOGIC;
BUSY:OUT STD_LOGIC;
DATA: IN STD_LOGIC_VECTOR(7 downto 0);
    TX_LINE:OUT STD_LOGIC
        );
END COMPONENT TX;
COMPONENT RX
 PORT(
  CLK: in std_logic;
  BUSY : out std_logic;
        DATA: out std_logic_vector(7 downto 0);
        RX_LINE: in std_logic

                );
END COMPONENT RX;
```

```
BEGIN
C1: TX PORT MAP (CLOCK_50,TX_START,TX_BUSY,TX_DATA,UART_TXD)C2: RX PORT MAP
(CLOCK_50,UART_RXD,RX_DATA,RX_BUSY);
PROCESS(RX_BUSY)
 IF (RX_BUSY'EVENT AND RX_BUSY='0') THEN
 LEDR(7 downto 0)<= RX_DATA;
 END IF;
END PROCESS;
PROCESS(CLOCK_50)
BEGIN
IF(CLOCK_50'EVENT AND CLOCK_50='1')THEN
   IF(KEY(0)='1' AND TX_BUSY='0')THEN -- Key(0)='0' MEAN that the key is pressed
TX_DATA<="0000" & SW(3 DOWNTO 0);
TX_START<='1';
            LEDG<=TX_DATA;
    ELSE
            TX_START<='0';
       TX_DATA<=(others=>'0') ;
  END IF;
END IF;
END PROCESS;
END MAIN;
```

A.5 Verilog code for storing data in EEPROM memory of Altera De0-Nano memory:

```
 module i2c_test  ( CLOCK_50, LED, KEY, SW, I2C_SCLK, I2C_SDAT, COUNT, SD_COUNTER
);
input CLOCK_50;
            output [7:0] LED;
            input [1:0] KEY;
           input [3:0] SW;
           output I2C_SCLK;
           input I2C_SDAT;
```

```
output [5:0] SD_COUNTER
output [9:0] COUNT;
 wire reset_n;
 reg GO;
reg [5:0] SD_COUNTER;
reg SDI;
reg SCLK;
reg [9:0] COUNT;
assign reset_n = KEY[0];
always@(posedge CLOCK_50)
COUNT <= COUNT + 1;
always @ (posedge COUNT[9] or negedge reset_n)
begin
        if (! reset_n)
        GO <=0;
        else
               if (! KEY[1])
              GO <=1;
end;
always @ (posedge COUNT[9] or negedge reset_n)
begin
 if (! reset_n)
SD_COUNTER <= 6'b0;
else
begin
if (!GO)  SD_COUNTER <=0;
              else
             if (SD_COUNTER < 33)
           SD_COUNTER <= SD_COUNTER+1;
             end
end
always @ (posedge COUNT[9] or negedge reset_n)
begin
```

```
 if (! reset_n)
begin
      SCLK <= 1;
      SDI <= 1;
 end
    else
           case (SD_COUNTER)
6'd0 : begin SDI<=1; SCLK<=1; end
          6'd1 : SDI <=0;
          6'd2 : SCLK <= 0;
 6'd3 : SDI <=1;
         6'd4 : SDI <=0;
          6'd5 : SDI <=1;
6'd6 : SDI <=0;
 6'd7 : SDI <=0;
            6'd8 : SDI <=0;
            6'd9 : SDI <=0;
           6'd10 : SDI <=0;
 6'd11 : SDI <=1'bz;
6'd12 : SDI <=0;
 6'd13 : SDI <=0;
             6'd14 : SDI <=0;
             6'd15 : SDI <=0;
            6'd16 : SDI <=0;
           6'd17 : SDI <=0;
           6'd18 : SDI <=0;
           6'd19 : SDI <=0;
          6'd20 : SDI <=1'bz;
          6'd21 : SDI <=1;
          6'd22 : SDI <=0;
          6'd23 : SDI <=1;
          6'd24 : SDI <=0;
```

```
6'd25 : SDI <=1;
         6'd26 : SDI <=0;
        6'd27 : SDI <=1;
        6'd28 : SDI <=0;
       6'd29 : SDI <=1'b1;
6'd30 : begin SDI <=1'b0; SCLK <=1'b1; end
           6'd31 : SDI <=1'b1;
 endcase
end
assign I2C_SCLK = ((SD_COUNTER >=4) & (SD_COUNTER <= 31))? -COUNT[9] : SCLK;
assign I2C_SDAT = SDI;
endmodule
```