University of Windsor Electrical and Computer Engineering Department

06-88-443 Embedded System Design

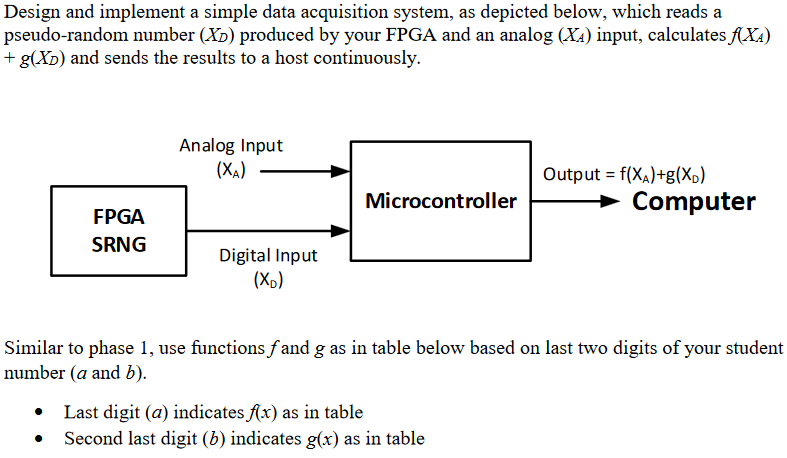
Project 3: Data Relay System: FPGA-Microcontroller

# Instructor: Arash Ahmadi

|  |  |
| --- | --- |
| Group Member: Daksh Patel | ID: 104 030 031 |

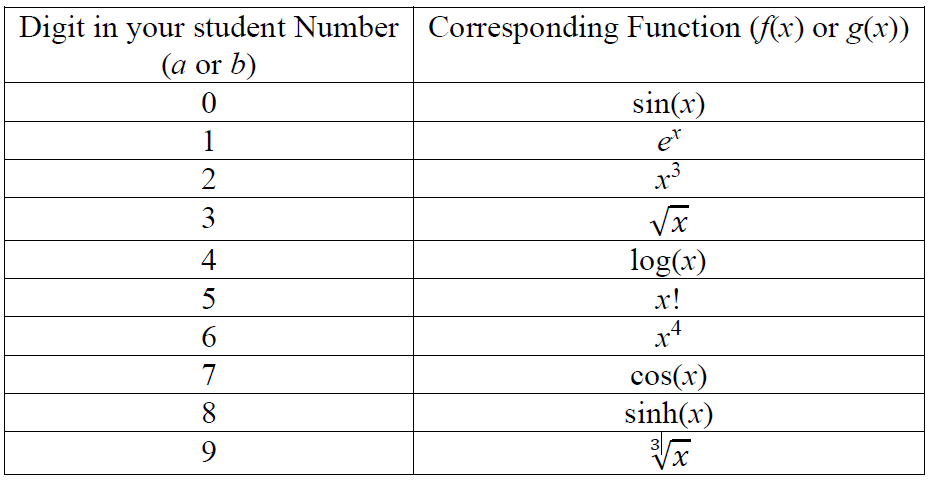
# Introduction

The objective of this project is to design and implement a data relay system between a free programmable gate array (FPGA) and an Arduino microcontroller. The instructions provided describe the goals as:



*Figure 1: Assignment Apparatus and Instructions*

As shown from this diagram, there are analog and digital inputs that are used to source the assignment data for calculations that are then carried out by the microcontroller. In this particular assignment, the digital input is generated by the FGPA. In addition, numerical functions f and g as in the table provided identify the functions to be used based on the last two digits of our student numbers (a and b). The last two digits being 6 and 6 resulting in both functions being x4. Which is represented in the microcontroller code as multiplying the variable onto itself 4 times. Showing the differences between Project 1 where this was done on FPGA at module level. It shows how much simpler higher-level languages and hardware are while still being able to do increasingly complex things.

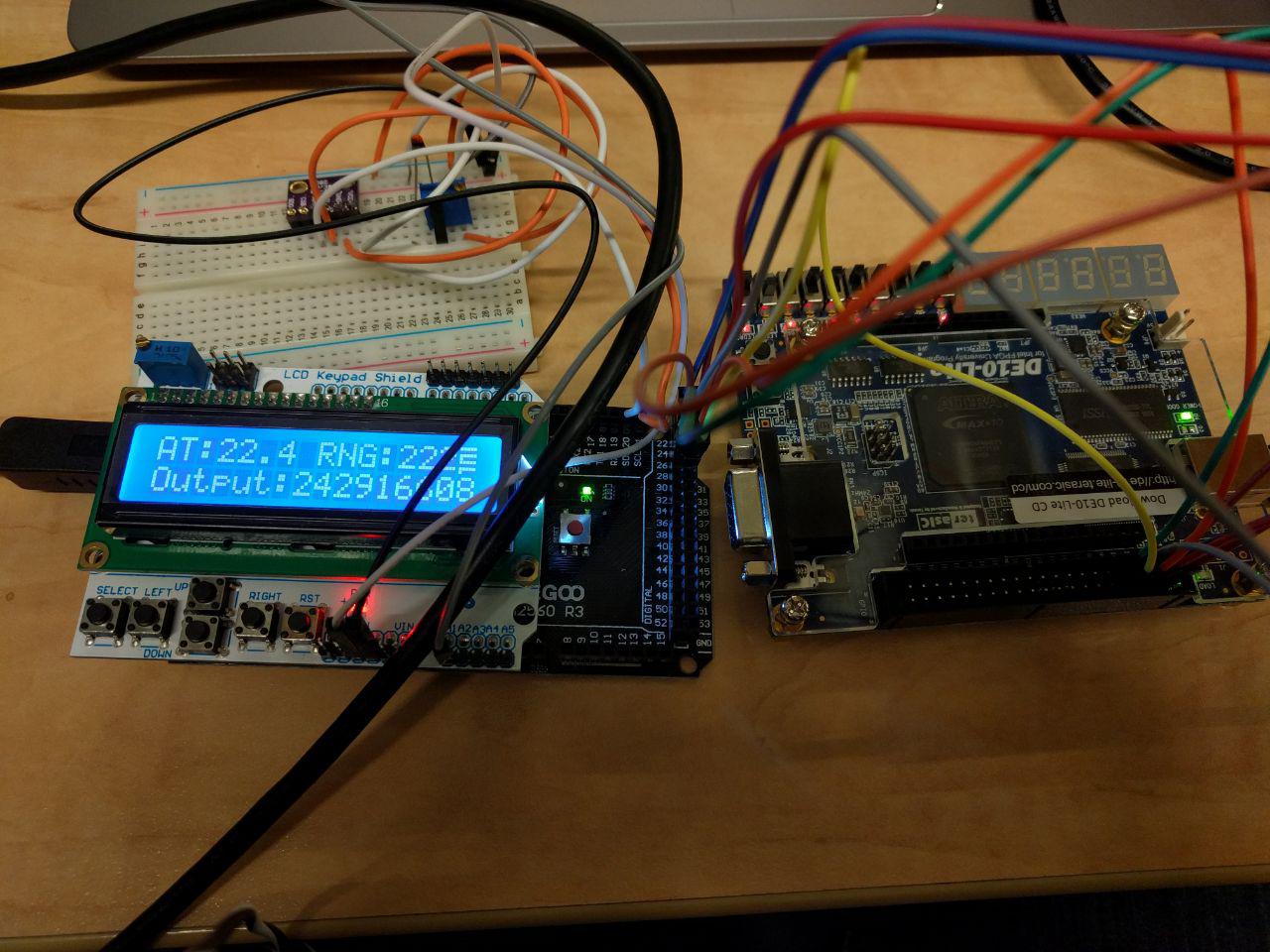


*Figure 2: Assignment Calculation Table*

* Last digit (a) indicates f(x) as in table
* Second last digit (b) indicates g(x) as in table

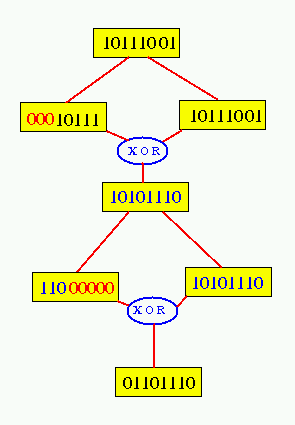
# Our Solution

We implement our solution using one of our group provided his MAX10 DE10 Lite FPGA board for generating the random number generator as well as implementing a UART solution for communication between the microcontroller and FPGA board.



*Figure 3: DE10 Development Board (on the right)*

* We implemented our FPGA random number generator using a VHDL programing environment.
* Pseudo-random number generation can be accomplished using a combination of bit shifting and ‘flipping’ binary strings using XOR gates. This operation is successively carried out on a seed (starter) value and the output is an acceptably random value. Below is an example structure of the operation flow:



*Figure 4: Pseudorandom Number Generation using XOR’s and Shifters (shifters not shown)*

* Relaying this set of values to the microcontroller is accomplished using serial communication driven by the system’s clock
* The subsequent stages of the assignment (calculations) are then carried out once the as the random value is relayed.

# Analog I/O Theory in Microcontrollers

This ESD course covers an entire chapter on ports and protocols and a number of our lectures covered how to properly select and implement I/O communication when using embedded systems. Several portions of the learning material applied to this assignment:

1. Analog and digital communication are the two main approaches considered when examining ports and communication protocols.
   1. Analog communication has infinite resolution in terms of its dependant values (amplitude, voltage, loudness, velocity etc.)
   2. Digital communication is made up of a finite number of digits or levels that represent the sampled dependant value. It has a given minimum resolution.
   3. Continuous and discrete values represent the characteristics mentions about the two properties of signals mentioned above. they can apply to both dependant values such as voltage as well as independent values such as time.
2. There are various communication protocols that are typically used in embedded systems. Selection of the appropriate communication protocol is crucial in system design because it affects
   1. Chapter 1 considerations such as costs, timelines, amount of labour input
   2. Compatibility considerations when selecting hardware, design constraints and software strategies that will affect performance, robustness and ease of use
   3. Protocol features that may be of consideration include
      1. Error correction (built-in/updatable/configurable)
      2. Physical properties associated with the technology (length, temperature, resistance, line-of-sight, noise and interference and other operational constraints
      3. Speed and bandwidth
      4. Protocol Architecture characteristics such as serial/parallel, wired/wireless, adaptability/retro-adaptability

The most commonly used protocols for microcontrollers is likely to be serial communication. Serial communication works by sending one bit at a time between senders and recipients. The serial protocol includes parity checking, a preset *Baud* rate (communication speed setup) and start and stop bits. Analog communication in these devices typically involves sending a set voltage, of a given value relative to a common ground as well as a particular frequency and/or duty cycle. Serial communication is great for such embedded systems because:

* It is relatively cost-effective (down to a wire, and necessary for our Capstone)
* It is a good bandwidth match for embedded systems, which are usually used for on-site, low data throughput tasks requiring short length connections that require no adapters or special connectors

# UART Communication Protocol Theory

UART, or Universal Asynchronous Serial Transmission is essentially the fully versed serial protocol functioning in an asynchronous manner. Asynchronous communication is not clocked. in other words, events such as sending, receiving and acknowledging data are not driven by any clocked circuits. A handshaking procedure occurs to establish all of the parameters required for serial communication and then data is sent and received. Some of these parameters include the Baud Rate (200, 4800, 9600, 19.2K), which specifies the transmission rate.

In addition, UART sends and receives data in serial, but the data starts off and ends up in a parallel form. Our course textbook elaborates:

*Internally, a simple UART may possess a baud-rate configuration register, and two*

*independently operating processors, one for receiving and the other for transmitting. The*

*transmitter may possess a register, often called a transmit buffer, that holds data to be*

*sent. This register is a shift register, so the data can be transmitted one bit at a time by*

*shifting at the appropriate rate. Likewise, the receiver receives data into a shift register, (Vahid, 36)*

# Procedure

The following steps were taken to implement the project goals

1. Implement a serial sender if VHDL, ensuring that the FPGA Board uses the correct ports as well as specifying the serial communication mechanism
2. Implement serial input in the micro-controller as carried out in the previous assignment
3. Also, identify the needed libraries to implement serial communication using Arduino
4. Carry out the serial-parallel and vice-versa conversions before and after data transmission respectively
5. Test the results to ensure that they match the expected computation outputs

# Project Code (VHDL on DECA Board)

Below is the VHDL code. Of particular interest it the pseudorandom number generation, which is in bold.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.std\_logic\_signed.all;

use IEEE.numeric\_std.all;

entity rando is

port(

clk : in std\_logic;

rst : in std\_logic;

number : out std\_logic\_vector(7 downto 0)

);

end rando;

architecture operation of rando is

signal temp : std\_logic\_vector(7 downto 0) := "00000001";

begin

process(clk)

variable pseudo : std\_logic := '0';

begin

if rising\_edge(clk) then

**pseudo := temp(4) XOR temp(3) XOR temp(2) XOR temp(0);**

**temp <= pseudo & temp(7 downto 1);**

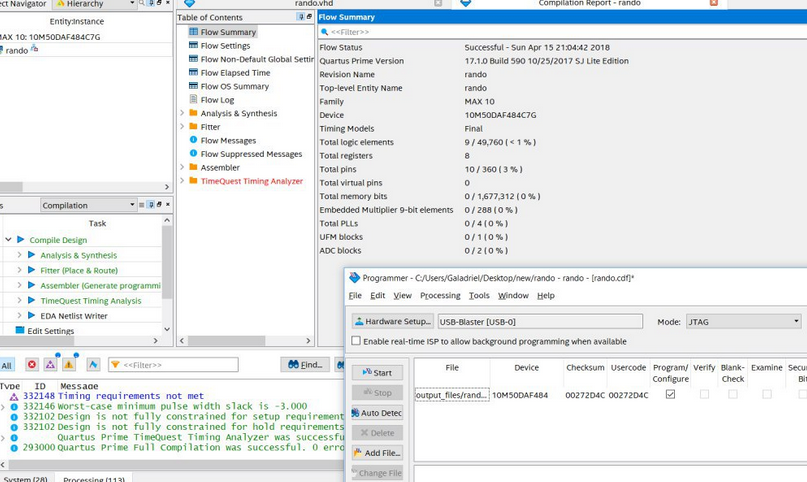
end if;

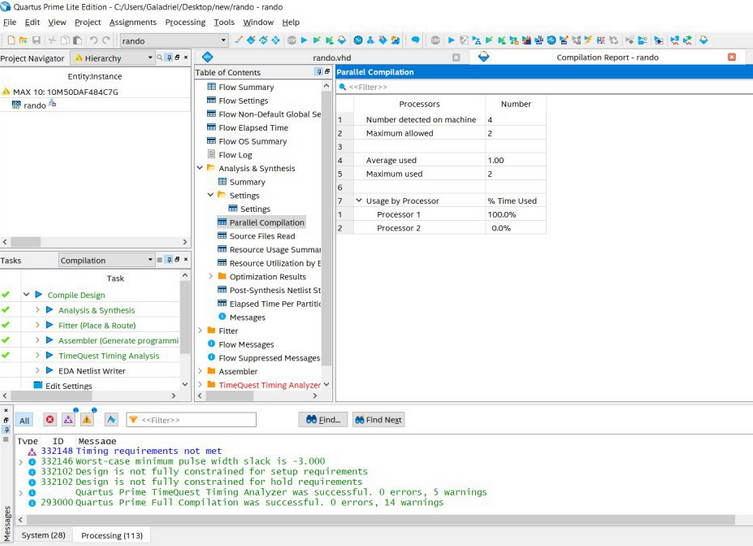
end process;

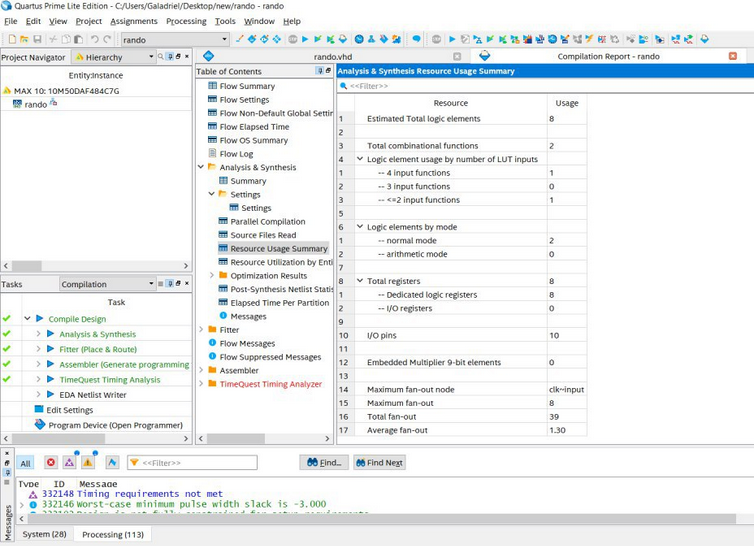
number <= temp;

end operation;

# Compilation Results



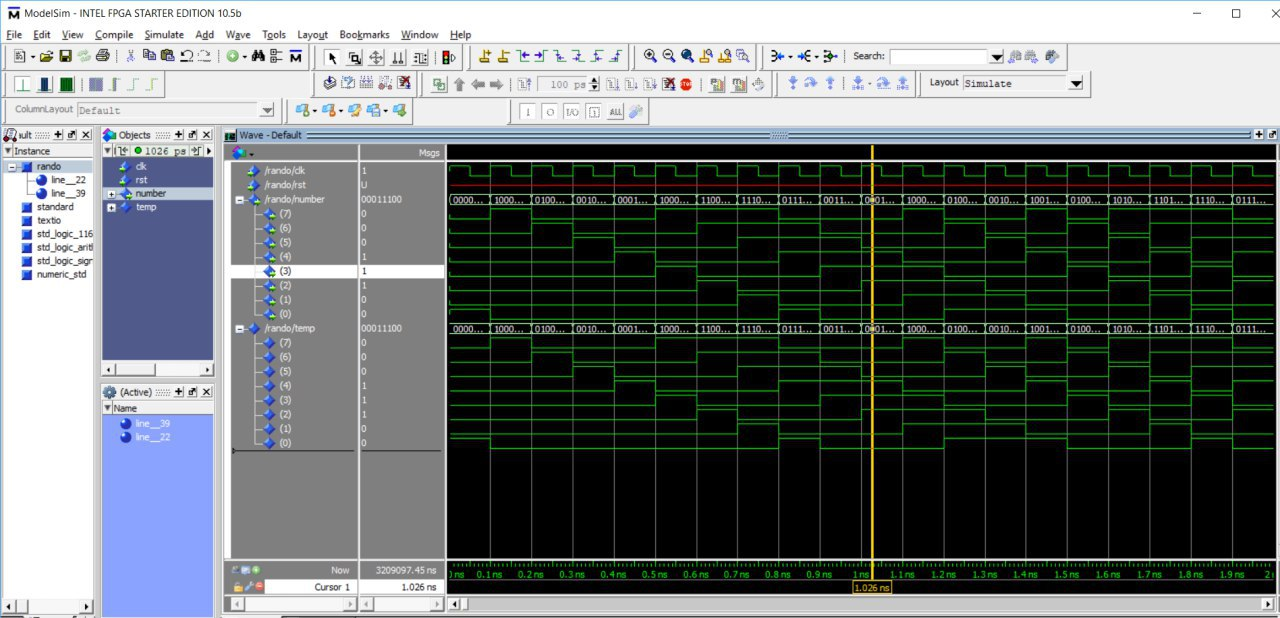


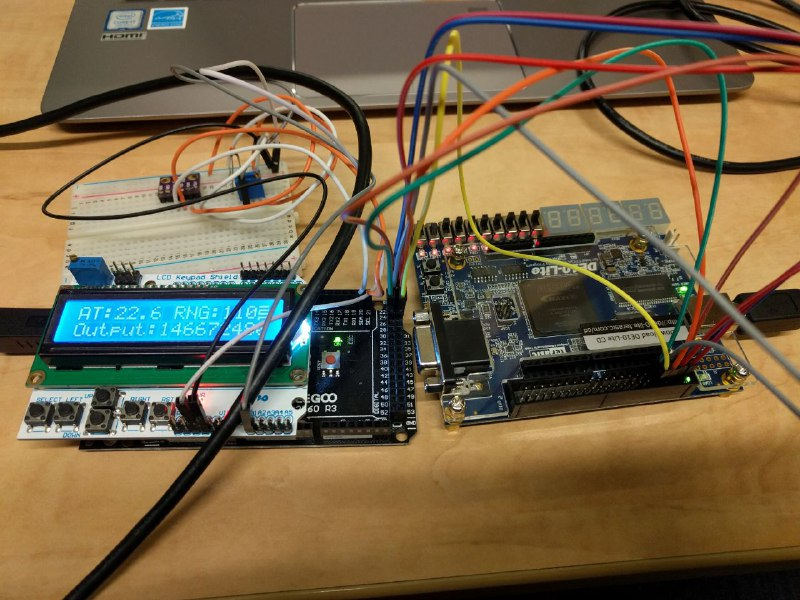


*Figures 5-8: Quartus VHDL Compilation Results*

# Simulation Results

The following is the VHDL simulation for our UART sender implementation

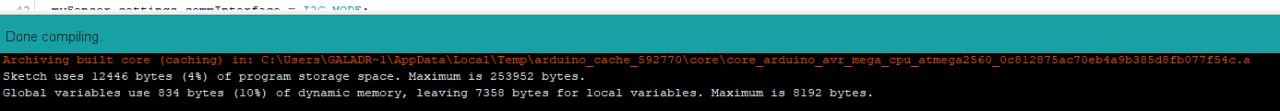


*Figure 3: Overview of the project, which provides a visual aid in the data Being Passed through I/O and displayed in the LCD*  


*Figure 6: Apparatus setup, including an LCD display to show the calculation output*

# Arduino Compiled Program Size

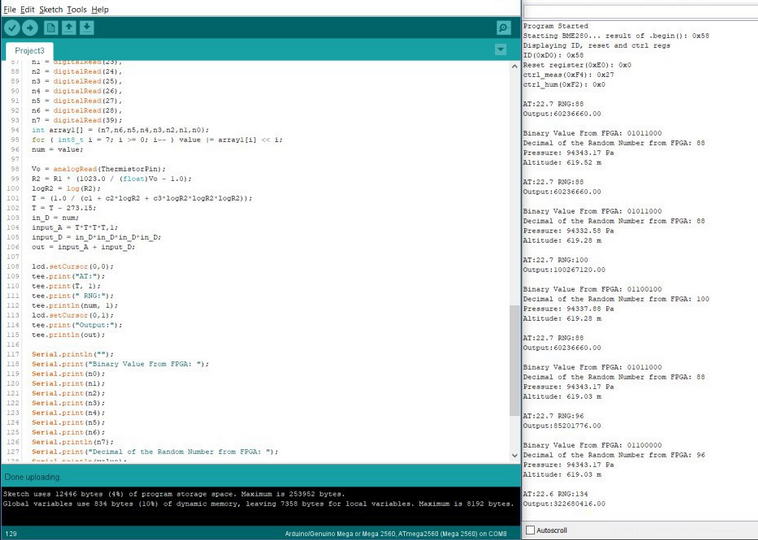
A key part of the assignment asked for the size of the program. Shown below is that data, with an accompanying image.



* Sketch uses 12446 bytes (4%) of total program storage
* Global variables use 834 bytes (10%) of dynamic memory

# Runtime Output from the Arduino Microcontroller

Shown on the left is the Arduino IDE and the runtime output from our implementation



*Figure 7: Runtime output from the microcontroller runtime*

# Arduino Microcontroller Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Project3.ino

Read values from FPGA and Thermistor to display on both Serial Window and LCD Screen with Arduino

Version: 0.5

Apr 15, 2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <LiquidCrystal.h>

#include <stdint.h>

#include "SparkFunBME280.h"

#include "Wire.h"

#include <NeoTee.h>

#include <Math.h>

// define some values used by the panel and buttons

int lcd\_key = 0;

int adc\_key\_in = 0;

int ThermistorPin = A1;

int Vo;

float R1 = 10000;

float logR2, R2, T;

float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 = 2.019202697e-07;

unsigned long delayTime;

double out, input\_A, input\_D, in\_D;

bool on = false;

// select the pins used on the LCD panel

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

//Global sensor object

BME280 mySensor;

//Print out Data cleanly

Print \*outputs[] = { &Serial, &lcd }; // <-- list all the output destinations here

NeoTee tee( outputs, sizeof(outputs)/sizeof(outputs[0]) );

void setup() {

Serial.begin(9600);

mySensor.settings.commInterface = I2C\_MODE;

mySensor.settings.I2CAddress = 0x76;

mySensor.settings.runMode = 3; //Normal mode

mySensor.settings.tStandby = 0;

mySensor.settings.filter = 0;

mySensor.settings.tempOverSample = 1;

mySensor.settings.pressOverSample = 1;

mySensor.settings.humidOverSample = 1;

Serial.print("Program Started\n");

Serial.print("Starting BME280... result of .begin(): 0x");

//Calling .begin() causes the settings to be loaded

delay(10); //Make sure sensor had enough time to turn on. BME280 requires 2ms to start up.

Serial.println(mySensor.begin(), HEX);

Serial.print("Displaying ID, reset and ctrl regs\n");

Serial.print("ID(0xD0): 0x");

Serial.println(mySensor.readRegister(BME280\_CHIP\_ID\_REG), HEX);

Serial.print("Reset register(0xE0): 0x");

Serial.println(mySensor.readRegister(BME280\_RST\_REG), HEX);

Serial.print("ctrl\_meas(0xF4): 0x");

Serial.println(mySensor.readRegister(BME280\_CTRL\_MEAS\_REG), HEX);

Serial.print("ctrl\_hum(0xF2): 0x");

Serial.println(mySensor.readRegister(BME280\_CTRL\_HUMIDITY\_REG), HEX);

uint8\_t memCounter = 0x80;

uint8\_t tempReadData;

lcd.begin(16, 2); // start the library

lcd.setCursor(0,0);

delayTime = 1000;

Serial.println();

}

void loop() {

printValues();

delay(delayTime);

lcd.clear();

} //Loop end

void printValues() {

int num;

int i;

byte value;

int n0 = digitalRead(22),

n1 = digitalRead(23),

n2 = digitalRead(24),

n3 = digitalRead(25),

n4 = digitalRead(26),

n5 = digitalRead(27),

n6 = digitalRead(28),

n7 = digitalRead(39);

int array1[] = {n7,n6,n5,n4,n3,n2,n1,n0};

for ( int8\_t i = 7; i >= 0; i-- ) value |= array1[i] << i;

num = value;

Vo = analogRead(ThermistorPin);

R2 = R1 \* (1023.0 / (float)Vo - 1.0);

logR2 = log(R2);

T = (1.0 / (c1 + c2\*logR2 + c3\*logR2\*logR2\*logR2));

T = T - 273.15;

in\_D = num;

input\_A = T\*T\*T\*T,1;

input\_D = in\_D\*in\_D\*in\_D\*in\_D;

out = input\_A + input\_D;

lcd.setCursor(0,0);

tee.print("AT:");

tee.print(T, 1);

tee.print(" RNG:");

tee.println(num, 1);

lcd.setCursor(0,1);

tee.print("Output:");

tee.println(out);

Serial.println("");

Serial.print("Binary Value From FPGA: ");

Serial.print(n0);

Serial.print(n1);

Serial.print(n2);

Serial.print(n3);

Serial.print(n4);

Serial.print(n5);

Serial.print(n6);

Serial.println(n7);

Serial.print("Decimal of the Random Number from FPGA: ");

Serial.println(value);

Serial.print("Digital Temperature: ");

Serial.print(mySensor.readTempC(), 1);

Serial.println(" C");

Serial.print("Pressure: ");

Serial.print(mySensor.readFloatPressure(), 2);

Serial.println(" Pa");

Serial.print("Altitude: ");

Serial.print(mySensor.readFloatAltitudeMeters(), 2);

Serial.println(" m");

Serial.println();

}

# Discussion

The FPGA code is fairly easy to explain. Input and Output declared and signals to hold values are created. Pseudo variable holds an XOR’d value from 4 different bits and it is concatenated with the previous temp value. The temp value is then outputted to number. Through pin planner the number is outputted at the bitwise state through 8 GPIO pins. A push button acts as a clock and random values can be sent to the Arduino to be digitally read.

The Arduino code is similar to that of project 2 to make it easier to show all values at once during presentation period. To explain the code used let’s start with going through the included libraries. Math is used for the various calculations performed such as the ones required for the Analog Temperature and the project requirements. LiquidCrystal and stdint are required for the LCD Screen functions. SparkFunBME280 is for the digital sensor along with Wire. NeoTee is to output the results to both Serial Terminal and LCD Screen at the same time. There is some additional output detailed in the Serial terminal such as Pressure but not shown on the LCD Screen due to wanting to show the basics all the time. It was chosen to show the Analog and Digital temperatures on the LCD Screen.

After the libraries variables are declared. Some of them are related to the Analog Temperature to calculate some specific values while other variables are there to be global so that all functions can access them. LCD screen pins are declared and sensor is declared as an object. Next the NeoTee code is organized and outputs selected.

Setup performs the required sensor setup and the LCD screen and baudrate declarations. The loop calls the printValues function and clears the LCD every loop to refresh. The sensors part of the code is the printValues function where the analog temperature values are calculated. The digital pins from the FPGA are read and calculations performed to convert the array of values into a decimal value. To clearly show the outputs the binary and decimal values are displayed. The required project calculations are performed and the analog, digital and FPGA values are neatly displayed.

Implementing a UART system for the FPGA board and microcontroller was quite challenging because there is no well-defined approach to troubleshooting the errors that we encountered that was known by our team. Consequently, we spent a significant amount of time simply trying to zero-in on the possible errors we had.

Theoretical understanding of UART was simple but practical implementation of this system using VHDL is not a commonly covered learning resource. Many learning materials that we examined used custom software tools and wizards to implement UART. In the end, we did understand how to implement this practically and worked through some of the bugs successfully. Capturing the data and carrying out calculations was simpler for us since this is a task that we accomplished in the last two assignments. In addition, the LCD display portion, which is a serial-out implementation, was a bit challenging as we needed to display more content than typically viewable on one screen. If we had additional time, e would have liked to spend more time trying out the various configurations associated with UART communication, but we have a somewhat limited understanding of the appropriate settings and desired parameters when using this protocol for our assignment beyond simply relaying the data.

# Conclusion

We managed to apply our theoretical understanding of utilizing protocol understanding to implement a fitting setup to relay data from a FPGA to a microcontroller. challenges encountered were largely related to identifying methods to troubleshoot issues with the FPGA to microcontroller communication. In addition to the assignment requirements, we added an LCD output to make testing and demonstration easier.

# References

1. Course Materials: Embedded Systems Design: A Unified Hardware and Software Introduction, Vahid, Frank, Givargis, Tony. John Wiley & Sons, 2002
2. DECA Development Board Datasheet, <http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=1021&FID=a13a2782811152b477e60203d34b1baa>, accessed, March 26, 2018
3. Arduino ATMEGA2560 Datasheet, Accessed March 24 2018, <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf>, www. microchip.com
4. Serial Communication Tutorial, Sparkfun Electronics, <https://learn.sparkfun.com/tutorials/serial-communication/all>, Accessed March 25 2018.
5. Random Number Generation, Donald H, W, Hyatt, <https://www.tjhsst.edu/~dhyatt/arch/random.html>, date accessed: April 2 2018.