MIAMI UNIVERSITY

Project 1

Ye Qiao

Austin Patrick

Instructor: Dr. Kimball

ECE 387

March 13th, 2020

Description:

The ultimate goal of this project is building a data logging system via FPGA and test it with Arduino ATmega328p microcontroller. It is useful when data collected is not needed in real time, but later than when collected. We implement logic with writing Verilog in Quartus software as two separate modules, one for AD/C reading and the other for EEPROM writing.

Once we successfully write data into the EEPROM, we pull it out and hook it up with Arduino microcontroller and read data out to serial monitor for demonstration.

Component used:

1. Arduino board, attached to a PC
2. EEPROM IC chip (Digi-Key Part # 24LC256-I/P-ND)
3. Analog to Digital converter IC chip (Digi-Key Part # MCP3002-I/P-ND)
4. Breadboard
5. Signal Generator
6. Jumper wire
7. DE-2 FPGA board
8. 2x 5k ohm resistors (Arduino part as pull high resistors)
9. 2x 10k ohm resistors (FPGA to EEPROM protection resistors)

Results and Discussion, and conclusion:

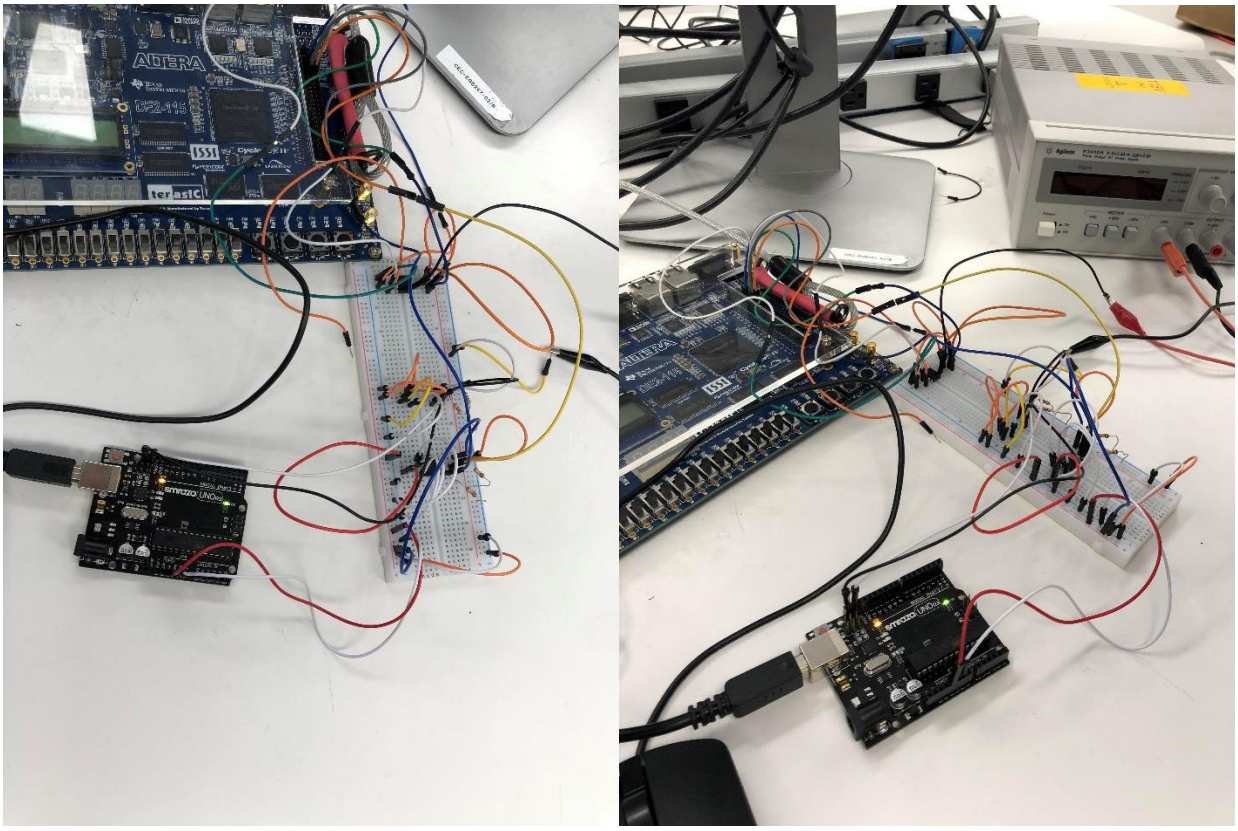
Referring from gist.github.com, we wrote A/DC module with alternating channel mode with 50Khz of sub clock. State machine was configured with total 16 state since the AD/C chip need 16 clock cycle to finish 10-bit of value grabbing. We set up a latch to alternate channel to config the system with lower delay rate. We also have a 10-bit shared register to pass value to EEPROM module and a notify signal come from the EEPROM submodule to report if the EEPROM finish writing values. Once reported notify signal was pulled to high. The AD/C module will resume to take next value in, otherwise it will stay in previous state. We also mapped the output of AD/C to ten red led to verify the values.

Our Verilog code for EEPROM submodule is relatively straight forward. We followed the data sheet and set up a total of 41 state machine controlling operation of every clock pulse to perform I2C protocol data transfer. The last state is using to notify AD/C module to resume and update data. Since our EEPROM module work with 50hz sub-clock rate, time period of one signal pulse is enough for AD/C module to update its data value. Moreover, we throw out two the list significant value from AD/C data because we implement our EEPROM by byte writing.

An Arduino code was been written to read the data out form EEPROM chip and print it out on serial monitor. We used wire.h library header file to implement this part of the project after asking permission from instructor.

As it turns out, we manage to take data from AD/C and successfully store it to EEPROM and read it out afterward. We spent a lot of time on timing control and I2C protocol implementation and finally learned how it work and finish the project task.

Circuit and hardware configuration:

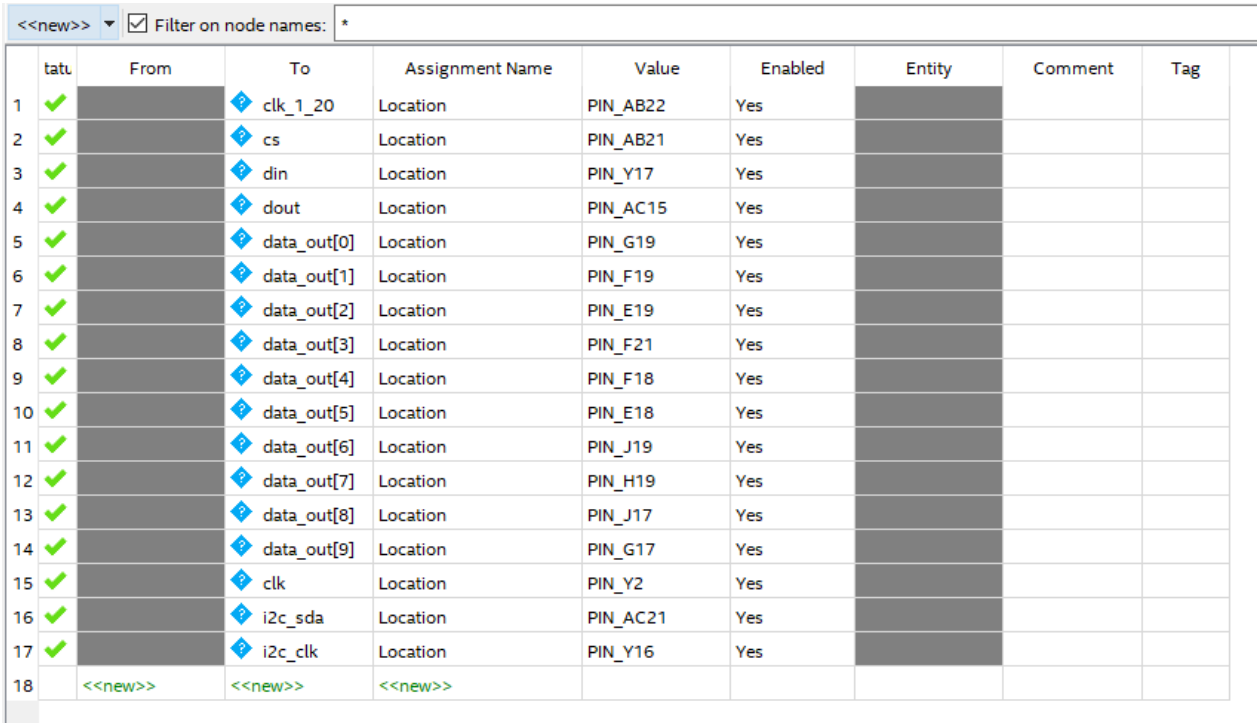


Screen capture and demonstration:

I will send a clear video demonstration with this report together, please check it.

Appendix:

FPGA ping assignment:



<<new>> Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✔		clk_1_20	Location	PIN_AB22	Yes			
2	✔		cs	Location	PIN_AB21	Yes			
3	✔		din	Location	PIN_Y17	Yes			
4	✔		dout	Location	PIN_AC15	Yes			
5	✔		data_out[0]	Location	PIN_G19	Yes			
6	✔		data_out[1]	Location	PIN_F19	Yes			
7	✔		data_out[2]	Location	PIN_E19	Yes			
8	✔		data_out[3]	Location	PIN_F21	Yes			
9	✔		data_out[4]	Location	PIN_F18	Yes			
10	✔		data_out[5]	Location	PIN_E18	Yes			
11	✔		data_out[6]	Location	PIN_J19	Yes			
12	✔		data_out[7]	Location	PIN_H19	Yes			
13	✔		data_out[8]	Location	PIN_J17	Yes			
14	✔		data_out[9]	Location	PIN_G17	Yes			
15	✔		clk	Location	PIN_Y2	Yes			
16	✔		i2c_sda	Location	PIN_AC21	Yes			
17	✔		i2c_clk	Location	PIN_Y16	Yes			
18		<<new>>	<<new>>	<<new>>					

Code for AD/C module:

```
`timescale 1ns / 1ps
//The following program is a modified version of one found on gist.github.com
module project1adc(clk, cs, clk_1_20, dout, din, channel, cnt16, data_out, i2c_clk, i2c_sda);
input    clk;                          //FPGA internal clock
output  cs;                            //ADC chip select pin
output  clk_1_20;                    //ADC clock
input    dout;                         //ADC data out
output  din;                           //ADC data in
output channel;                    //output channel
output data_out;                    //sampled data to FPGA
output cnt16;                       //state counter
```

```
wire clk;
wire dout;

reg [9:0] data_out;
reg clk_1_20;
reg din;
reg cs;

//Internal clock of FPGA = 50 MHz
//Since this is beyond the operating frequency of the ADC,
//a slower clock is required.
//For this project, a 50kHz clock was used.
reg [25:0] cnt50;
reg [25:0] cnt50_next;
always @(posedge clk) cnt50<=cnt50_next;

//generate  50kHz clock
always @(posedge clk)
begin
	if (cnt50==500)
	begin
		cnt50_next<=0; //Reset
		clk_1_50<=~clk_1_50; //Flip clk_1_50
	end
	else
	begin
		cnt50_next<=cnt50_next+1;
		clk_1_50<=clk_1_50;
	end
```

```
end


//A latch signal is needed since the positive and negative edges
//of the 50kHz clock, clk_1_50, are used.

reg init_latch;          //indicates when sample is being taken

wire notify;             // notify signal from epprom01() module to indicate that EEPROM is
//ready for next value, so when notify == 1, ADC module will resume.
always @(posedge clk_1_50) begin init_latch<=1;end

//counter to count to 16 states
reg [3:0] cnt16;
always @(posedge clk_1_50) if (init_latch==1 & notify) cnt16<=cnt16+1;

//alternating channel
//This channel switches after each sample.
//sample is taken after 16 clocks
reg channel;

always @(negedge clk_1_50) if (init_latch==1) if (cnt16==15) channel<=~channel; else
channel<=channel;

//Since, ADC samples on positive edge,
//SPI input signals (data to FPGA) are active on negative edges.
//This is done to improve timing and prevent communication
//errors between ADC and FPGA.
always @(negedge clk_1_50)
begin
```

```
if (init_latch==1)
begin
        case (cnt16)
        0://start sampling process
                begin
                        cs<=1;
                        din<=0;
                end
        1://start communication
                begin
                        cs<=0;
                        din<=1;
                end
        2://single-ended mode selected
                begin
                        cs<=0;
                        din<=1;
                end
        3://channel 0 selected for single-ended mode
                begin
                        cs<=0;
                        din<=channel;
                end
        4://MSB First output format
                begin
                        cs<=0;
                        din<=1;
                end
        default://cs low for rest of cnt16 clock cycle
                begin
```

```
                    cs<=0;
                    din<=0;
                end
            endcase
        end
end

reg [9:0] sample;           //ADC output data

//data sampled on counts 6-16 of cnt16 clock cycle
always @(posedge clk_1_50)
begin
        if (init_latch==1)
        begin
            if (cnt16>=6)
            begin
                sample[9:1]<=sample[8:0];
                sample[0]<=dout;
            end
            else sample<=sample;
        end
end

//clk and sda data are output after each process
always @(posedge clk_1_50) if (cnt16==0) data_out<=sample; else data_out<=data_out;

output i2c_clk;
 inout i2c_sda;

//initialize an instance of eepromtest1 named epprom01() and connect to EEPROM module
```

```
eepromtest1 epprom01(clk, i2c_clk, i2c_sda, data_out,notify);

endmodule
```

EEPROM module:

```
module eepromtest1(clk, i2c_clk, i2c_sda, datain, notifyout);
  input clk; // 50mhz internal clock
  output i2c_clk; // sclk
  inout i2c_sda; // cda
  input wire [9:0]datain; // data from ADC module
  reg [7:0] addr;
  reg [7:0] addrh;
  wire rst;
  reg en;
  reg [8:0] stateCounter;
  reg SDI;
  reg SCLK;
  reg [24:0] counter;
  reg ste_clk;
 // reg  dataBuffer [7:0];
  // get 500hz pulse
always @(posedge clk, posedge rst)
    begin
        // When rst is high set count and ste_clk to 0
        if (rst == 1'b1)
        begin
            counter <= 25'b0;
```

```
            ste_clk <= 1'b0;
        end
        else if (counter == 5000000)
        begin
            counter <= 25'b0;
            ste_clk <= ~ste_clk;
        end

        else
        begin
            counter <= counter + 1;
            ste_clk <= ste_clk;
        end
    end

 output reg notifyout; // signal to notify main ADC module
 // notify main module once finish up one byte of writing
always @(posedge ste_clk) begin
if (stateCounter > 0 & stateCounter < 39)
 notifyout<=0;
 else
  notifyout<=1;
end

// state counter
always @(posedge ste_clk)
begin
if (stateCounter < 42)
stateCounter <= stateCounter + 1;
else
```

```
 stateCounter <= 0;
end


// address counter
always @(posedge ste_clk)
begin
if (addr > 255) begin
addr <= 8'd0;
addrh <= addrh + 1;
end
else if (stateCounter > 20 & stateCounter < 29)
addr <= addr + 1;
else
addr <= addr;

if (addrh > 255) begin
addrh <= 8'd0;
end
end

// detial of clock state opration
always @(posedge ste_clk)
begin
case (stateCounter)
        7'd0 : begin SDI <= 1; SCLK <= 1; end
// start
        7'd1 : SDI <= 0;
        7'd2 : SCLK <= 1;
// control
```

```
        7'd3 : SDI <= 1;
        7'd4 : SDI <= 0;
        7'd5 : SDI <= 1;
        7'd6 : SDI <= 0;
        7'd7 : SDI <= 0;
        7'd8 : SDI <= 0;
        7'd9 : SDI <= 0;
        7'd10 : SDI <= 0;
        7'd11 : SDI <= 1'bz;

//add high
    7'd21 : SDI <= addrh[7:6];
    7'd22 : SDI <= addrh[6:5];
    7'd23 : SDI <= addrh[5:4];
        7'd24 : SDI <= addrh[4:3];
        7'd25 : SDI <= addrh[3:2];
        7'd26 : SDI <= addrh[2:1];
        7'd27 : SDI <= addrh[1:0];
        7'd28 : SDI <= addrh[0];
        7'd29 : SDI <= 1'bz;

//add low
    7'd21 : SDI <= addr[7:6];
    7'd22 : SDI <= addr[6:5];
    7'd23 : SDI <= addr[5:4];
        7'd24 : SDI <= addr[4:3];
        7'd25 : SDI <= addr[3:2];
        7'd26 : SDI <= addr[2:1];
        7'd27 : SDI <= addr[1:0];
        7'd28 : SDI <= addr[0];
```

```
    7'd29 : SDI <= 1'bz;


//data
    7'd30 : SDI <= datain[9:8];
    7'd31 : SDI <= datain[8:7];
    7'd32 : SDI <= datain[7:6];
    7'd33 : SDI <= datain[6:5];
          7'd34 : SDI <= datain[5:4];
          7'd35 : SDI <= datain[4:3];
          7'd36 : SDI <= datain[3:2];
          7'd37 : SDI <= datain[2:1];
    7'd38 : SDI <= 1'bz;

//stop conditon
7'd39 : begin SDI <= 0; SCLK <= 1; end
7'd40 : begin SDI <= 1;end
endcase
end

// clock and data assignment
assign i2c_clk = ((stateCounter >= 4) & (stateCounter <= 39)) ? ~ste_clk : SCLK;
assign i2c_sda = SDI;

endmodule
```

Arduino code for read data out:

```
#include <Wire.h>
```

```
#define eeprom 0x50 //defines the base address of the EEPROM

unsigned int address = 0; //first address of the EEPROM
void setup()  {
  Wire.begin(); //creates a Wire object
  Serial.begin(9600);
  }
void loop() {
Serial.print(readEEPROM(eeprom, address));
Serial.print('\n');
address ++;
}
//defines the writeEEPROM function
void writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data ) {
  Wire.beginTransmission(deviceaddress);
  Wire.write((int)(eeaddress >> 8));      //writes the MSB
  Wire.write((int)(eeaddress & 0xFF));    //writes the LSB
  Wire.write(data);
  Wire.endTransmission();
  }

//defines the readEEPROM function
byte readEEPROM(int deviceaddress, unsigned int eeaddress ) {
  byte rdata = 0xFF;
  Wire.beginTransmission(deviceaddress);
  Wire.write((int)(eeaddress >> 8));      //writes the MSB
  Wire.write((int)(eeaddress & 0xFF));    //writes the LSB
  Wire.endTransmission();
  Wire.requestFrom(deviceaddress,1);
  if (Wire.available())
```

```
  rdata = Wire.read();
return rdata;
}
```

Works Cited

MCP3002 ADC Module. (2020). Retrieved 14 March 2020, from https://gist.github.com/jjcarrier/2590386

Kit, E., & Kit, E. (2020). EEPROM Interfacing with Spartan-3 XC3S200 FPGA Development Kit. Retrieved 14 March 2020, from https://www.pantechsolutions.net/eeprom-interfacing-with-spartan-3-xc3s200-fpga-development-kit