MIAMI
UNIVERSITY

Project 1: Data Logger

Spring 2020

College of Engineering and Computing

Department of Electrical and Computer Engineering

ECE 387: Embedded Systems

March 13th, 2020

Tyler McGrew

Description

The aim of this project is to build a system that can read in high frequency analog signals and store samples of data for later retrieval. This is to be accomplished using a 10-bit analog-digital converter IC (MCP3002), an EEPROM IC (24LC256), an FPGA development board (DE2-115), and an MCU (Arduino Uno). The analog signal is fed into the ADC, which is sampled through the FPGA via Serial Peripheral Interface (SPI), buffered, and written to the EEPROM via Inter-Integrated Circuit (I^2C). Then, the data is to be read and displayed at a later point via the MCU without using any existing I^2C libraries. The data resolution is permitted to be only 8 bits/sample; however, the sampling/writing rate is to be maximized.

Results

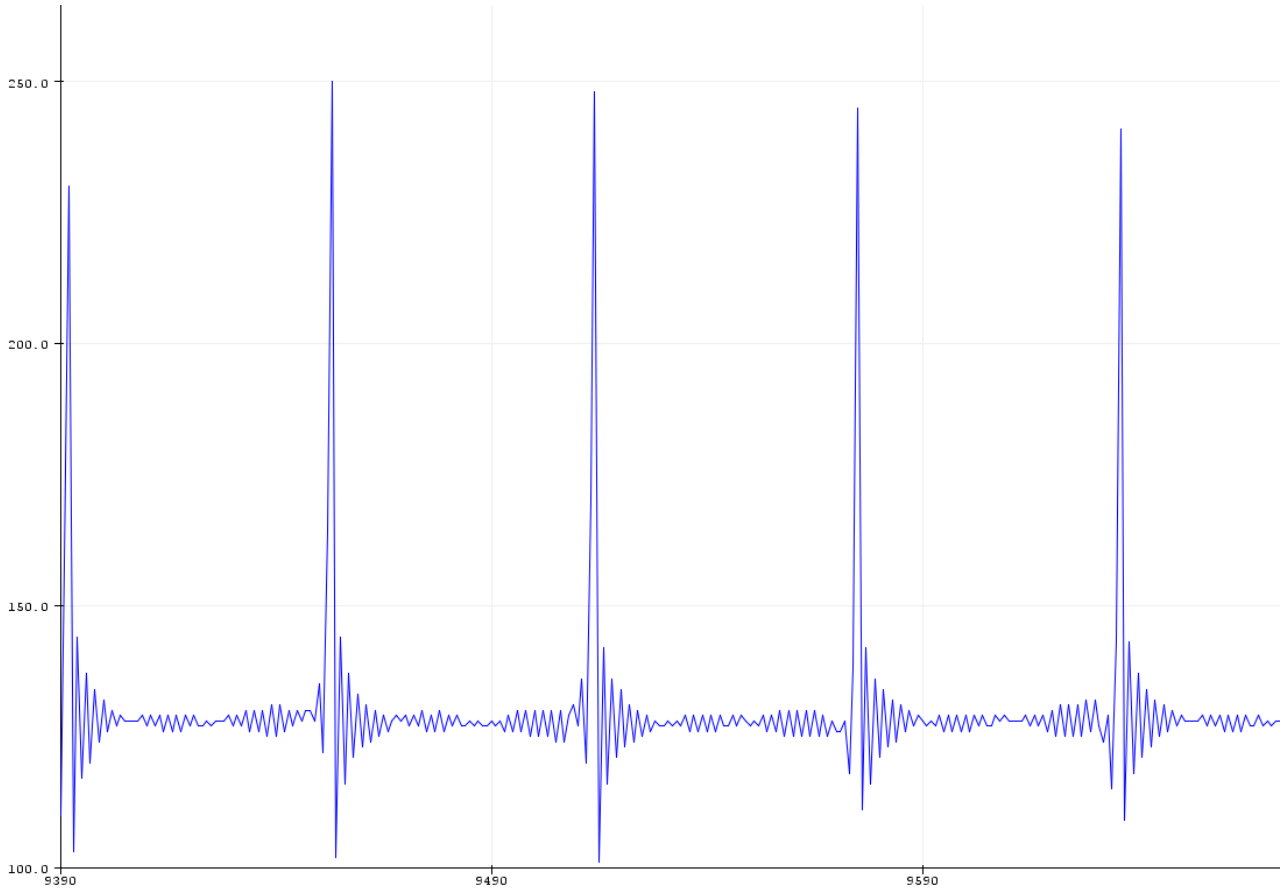

Figure 1. Arduino IDE plot of data from EEPROM (Original signal @ 100Hz)

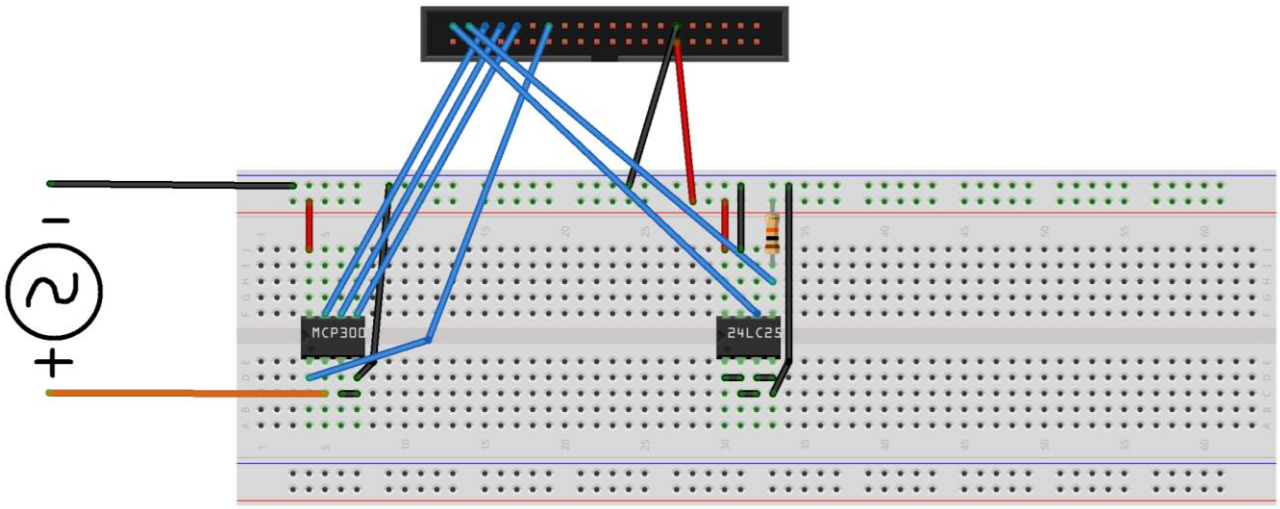

Figure 2. Circuit layout for sampling and writing data to the EEPROM using ADC and FPGA GPIO pins

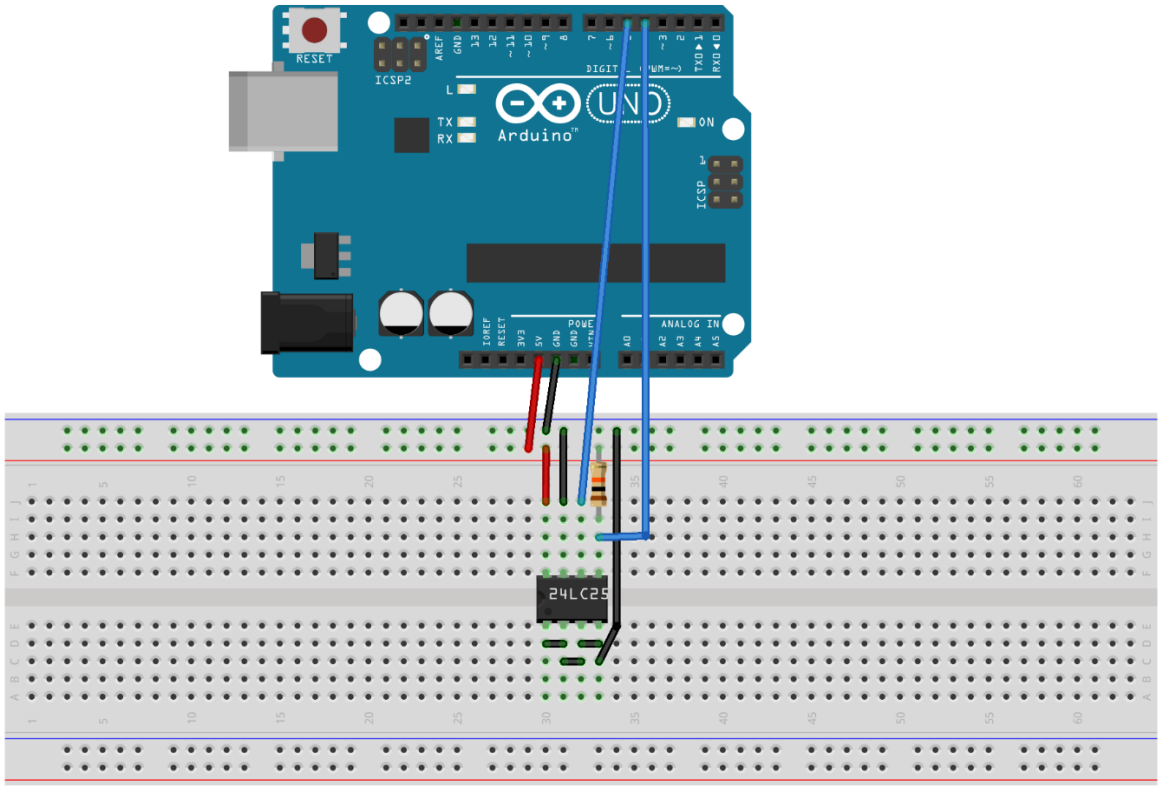

Figure 3. Circuit layout for reading and writing data to/from EEPROM with Arduino

Our completed system samples data from the ADC at 6.1ks/sec, buffers 64 eight-bit values, and writes a page (64 bytes) to the EEPROM while already buffering the next page from the ADC. The FPGA sends a “page-write” command to the EEPROM every 10.5ms using a 390kHz clock. The max clock frequency for the EEPROM is 400kHz and there is a minimum of 5ms between write operations permitted by the EEPROM. Additionally, our FPGA takes 1.6ms to send the control and data information for a page write operation. So, this system is near optimal and works without any noticeable issues for its purpose.

Similarly, the ADC allows for 75,000 samples/sec and a serial clock speed of 1.2MHz at 2.7V V_{DD}. Our system uses a 390kHz clock to sample at 6.1kHz because of the EEPROM bottleneck. Since each sample transfer takes only 14 clock cycles, this enables a sampling rate of 27ks/sec from the ADC- of which the FPGA uses only about one in four.

The FPGA layout is described in Verilog HDL within nine modules. There are two timer modules (“samplecountertimer.v”, “downclock.v”), four waveform generating modules (“startcon.v”, “stopcon.v”, “writehigh.v”, “writelow.v”), a module for obtaining samples from the ADC (“adcreader.v”), a module for writing a page of data to the EEPROM (“pagewriter.v”) and a high level control module to combine the other modules into a functional system (“eeprom.v”). Of these, only “adcreader.v”, “pagewriter.v”, “eeprom.v” can be considered state machines and “adcreader.v” is minimal in complexity. Most of the challenge and complexity comes in correctly implementing the I^2C page-write operation (“pagewriter.v”) and in determining proper timing, control, and data flow between ADC and EEPROM (“eeprom.v”).

The page-writer module is implemented using six high level states: CONTROL, ADDRESS_HIGH, ADDRESS_LOW, DATA, STOP, and WAIT. To conduct a write operation

to the EEPROM the master must send a start condition, a control byte, two address bytes, 64 data bytes, and a stop condition. These states correspond to these bytes with the start condition being conducted by the data byte and the wait state looping in between write operations. Each state has up to 11 sub-states corresponding to bits and changing/resetting of signals between bytes. This module was very difficult to create without a digital logic analyzer to debug. Fortunately, we were able to use the waveform simulation feature of Quartus to get this module working properly.

The high-level control module "eeprom.v" was similarly difficult to implement. The module is a state machine that cycles through 64 states corresponding to the 64 bytes in a page. The states are cycled through in a rate which was calculated to allow the page to be filled with 64 evenly spaced samples in time for another page-write operation to be executed in order to have no discontinuities in the sampled data. On the rising edge of each state, the next byte of the page buffer is updated from the ADC. Every 64 bytes, the page buffer is copied into another buffer, the writing address is increased by one page, and the page writer's enable signal is pulsed high. This is what allows the page writer to take multiple samples worth of time to write the page while the buffer is already being updated with new samples.

We wrote our own code for the Arduino which is able to read and write pages worth of information to/from the EEPROM over I^2C. This was the first thing we worked on because it was quicker and easier than doing it on the FPGA and allowed us to reliably evaluate the performance of the FPGA system as we developed it. The program uses straight-forward "bit-banging" to turn on and off the two I^2C lines in correct sequence using delay statements and reusable functions such as start() (start condition), write_high (writes one bit high), and read_bit().

Appendix 1 – Arduino Code

Can also be found here:
https://github.com/tymcgrew/EEPROM_Data_Logger/blob/master/Arduino/eeprom/eeprom.ino

```
const int s_data = 4;
const int s_clk = 5;
const int delay_time = 1;

void setup() {
  pinMode(s_clk, OUTPUT);
  pinMode(s_data, INPUT);
  Serial.begin(9600);
}

void loop() {
  for (int i = 2048; i < (2048)+(3*64); i += 64)
  {
    //page_write(i,0);
    page_read(i);
  }
  while(true);
}

void page_read(int start_address) {
  set_address(start_address);

  // Control Byte
  start();
  write_high();
  write_low();
  write_high();
  write_low();
  write_low();
  write_low();
  write_low();
  write_high();
  int ack = check_ack();
  //Serial.println(ack);

  for (int i = 0; i < 63; i++)
  {
    int val = 0;
    val += read_bit()*1;
    val += read_bit()*2;
    val += read_bit()*4;
    val += read_bit()*8;
    val += read_bit()*16;
    val += read_bit()*32;
    val += read_bit()*64;
    val += read_bit()*128;
    Serial.println(val);
    write_low();
  }

    int val = 0;
    val += read_bit()*1;
    val += read_bit()*2;
    val += read_bit()*4;
    val += read_bit()*8;
    val += read_bit()*16;
    val += read_bit()*32;
    val += read_bit()*64;
    val += read_bit()*128;
    Serial.println(val);
    stop1();

}

void page_write(int start_address, int write_value) {
```

```
// Find binary address
int start_address_binary [15] = {0};
int i1 = 0;
while (start_address > 0) {
    start_address_binary[i1] = start_address % 2;
    start_address = start_address / 2;
    i1++;
  }

// Find binary write_value
int write_value_binary [8] = {0};
int i2 = 0;
while (write_value > 0) {
    write_value_binary[i2] = write_value % 2;
    write_value = write_value / 2;
    i2++;
  }

  start();
  write_high();
  write_low();
  write_high();
  write_low();
  write_low();
  write_low();
  write_low();
  write_low();
  int ack = check_ack();
  //Serial.println(ack);

  // Address High Byte
  write_low();
  for (int j = 14; j >= 8; j--) {
    if (start_address_binary[j] == 1)
      write_high();
    else
      write_low();
  }
  ack = check_ack();
  //Serial.println(ack);

  // Address Low Byte
  for (int j = 7; j >= 0; j--) {
    if (start_address_binary[j] == 1)
      write_high();
    else
      write_low();
  }
  ack = check_ack();
  //Serial.println(ack);

  for (int i = 0; i < 63; i++)
  {
    // Write Byte
    for (int j = 0; j < 8; j++) {
      if (write_value_binary[j] == 1)
        write_high();
      else
        write_low();
    }
    ack = check_ack();
    //Serial.println(ack);
  }

  // Write Byte
  for (int j = 0; j < 8; j++) {
    if (write_value_binary[j] == 1)
      write_high();
    else
      write_low();
  }
```

```
    ack = check_ack();
    //Serial.println(ack);
    stop1();

}

void set_address(int start_address) {
  // Find binary address
  int start_address_binary [15] = {0};
  int i = 0;
  while (start_address > 0) {
      start_address_binary[i] = start_address % 2;
      start_address = start_address / 2;
      i++;
    }

  // Control Byte
  start();
  write_high();
  write_low();
  write_high();
  write_low();
  write_low();
  write_low();
  write_low();
  write_low();
  int ack = check_ack();
  //Serial.println(ack);

  // Address High Byte
  write_low();
  for (int i = 14; i >= 8; i--) {
    if (start_address_binary[i])
      write_high();
    else
      write_low();
  }
  ack = check_ack();
  //Serial.println(ack);

  // Address Low Byte
  for (int i = 7; i >= 0; i--) {
    if (start_address_binary[i])
      write_high();
    else
      write_low();
  }
  ack = check_ack();
  //Serial.println(ack);

}

int read_bit() {
  digitalWrite(s_clk, HIGH);
  delay(delay_time);
  int val = digitalRead(s_data);
  digitalWrite(s_clk, LOW);
  delay(delay_time);
  return val;
}

void start() {
  digitalWrite(s_data,HIGH);
  pinMode(s_data,OUTPUT);
  delay(delay_time);
  digitalWrite(s_clk, HIGH);
  delay(delay_time);
  digitalWrite(s_data,LOW);
  delay(delay_time);
  digitalWrite(s_clk,LOW);
  pinMode(s_data,INPUT);
```

```
  delay(delay_time);
}

void write_high() {
  digitalWrite(s_data,HIGH);
  pinMode(s_data,OUTPUT);
  delay(delay_time);
  digitalWrite(s_clk,HIGH);
  delay(delay_time);
  digitalWrite(s_clk,LOW);
  pinMode(s_data,INPUT);
  delay(delay_time);
}

void write_low() {
  digitalWrite(s_data,LOW);
  pinMode(s_data,OUTPUT);
  delay(delay_time);
  digitalWrite(s_clk,HIGH);
  delay(delay_time);
  digitalWrite(s_clk,LOW);
  pinMode(s_data,INPUT);
  delay(delay_time);
}

int check_ack() {
  digitalWrite(s_clk,HIGH);
  delay(delay_time);
  int ack = digitalRead(s_data);
  digitalWrite(s_clk,LOW);
  delay(delay_time);
  return ack;
}

void stop1(){
  digitalWrite(s_data,LOW);
  pinMode(s_data,OUTPUT);
  delay(delay_time);
  digitalWrite(s_clk, HIGH);
  delay(delay_time);
  digitalWrite(s_data,HIGH);
  delay(delay_time);
  digitalWrite(s_clk,LOW);
  delay(delay_time);
  pinMode(s_data,INPUT);
  delay(delay_time);
}
```

Appendix 2 – FPGA Code

Can also be found here:
https://github.com/tymcgrew/EEPROM_Data_Logger/tree/master/FPGA/eeprom

```
module eeprom(clk, rst, sdata, sclk, done, adcclock, din, dout, cs);


	input clk, rst;
	output sclk, done;
	inout sdata;
	output adcclock, dout, cs;
	input din;

	wire [7:0]adc_out;

   wire pagewriter_done;
	wire [31:0]samplecounter;

	reg [14:0]address = 15'd2048;
	reg [511:0]write_data;
   reg [511:0]page_buffer;
	reg [31:0]prev_samplecounter;
   reg [31:0]page_counter;
   reg firstpass = 1'b1;
   reg done;
	reg enable;


	pagewriter pagewriter1(clk, rst, sdata, sclk, write_data, address, pagewriter_done,
enable);
	adcreader adcreader1(clk, adcclock, din, dout, cs, rst, adc_out);
	samplecountertimer samplecountertimer1(clk, rst, samplecounter);


	always@(posedge clk or negedge rst)
	begin
		if (rst == 1'b0)
		begin
			address <= 15'd2048;
			write_data <=  512'd0;
	  page_buffer <= 512'd0;
			prev_samplecounter <= 32'd0;
	  page_counter <= 32'd0;
			firstpass <= 1'b1;
	  done <= 1'b0;
			enable <= 1'b0;
		end

		else
		begin
			if (page_counter < 32'd16)
	  begin
				if (prev_samplecounter == samplecounter)
					enable <= 1'b0;
				prev_samplecounter <= samplecounter;
				if (samplecounter != prev_samplecounter)
				begin
					case(samplecounter)
						default: page_buffer[511:504] <= adc_out;
						32'd62: page_buffer[503:496] <= adc_out;
						32'd61: page_buffer[495:488] <= adc_out;
						32'd60: page_buffer[487:480] <= adc_out;
						32'd59: page_buffer[479:472] <= adc_out;
						32'd58: page_buffer[471:464] <= adc_out;
						32'd57: page_buffer[463:456] <= adc_out;
						32'd56: page_buffer[455:448] <= adc_out;
						32'd55: page_buffer[447:440] <= adc_out;
						32'd54: page_buffer[439:432] <= adc_out;
						32'd53: page_buffer[431:424] <= adc_out;
```

```
                32'd52: page_buffer[423:416] <= adc_out;
                32'd51: page_buffer[415:408] <= adc_out;
                32'd50: page_buffer[407:400] <= adc_out;
                32'd49: page_buffer[399:392] <= adc_out;
                32'd48: page_buffer[391:384] <= adc_out;
                32'd47: page_buffer[383:376] <= adc_out;
                32'd46: page_buffer[375:368] <= adc_out;
                32'd45: page_buffer[367:360] <= adc_out;
                32'd44: page_buffer[359:352] <= adc_out;
                32'd43: page_buffer[351:344] <= adc_out;
                32'd42: page_buffer[343:336] <= adc_out;
                32'd41: page_buffer[335:328] <= adc_out;
                32'd40: page_buffer[327:320] <= adc_out;
                32'd39: page_buffer[319:312] <= adc_out;
                32'd38: page_buffer[311:304] <= adc_out;
                32'd37: page_buffer[303:296] <= adc_out;
                32'd36: page_buffer[295:288] <= adc_out;
                32'd35: page_buffer[287:280] <= adc_out;
                32'd34: page_buffer[279:272] <= adc_out;
                32'd33: page_buffer[271:264] <= adc_out;
                32'd32: page_buffer[263:256] <= adc_out;
                32'd31: page_buffer[255:248] <= adc_out;
                32'd30: page_buffer[247:240] <= adc_out;
                32'd29: page_buffer[239:232] <= adc_out;
                32'd28: page_buffer[231:224] <= adc_out;
                32'd27: page_buffer[223:216] <= adc_out;
                32'd26: page_buffer[215:208] <= adc_out;
                32'd25: page_buffer[207:200] <= adc_out;
                32'd24: page_buffer[199:192] <= adc_out;
                32'd23: page_buffer[191:184] <= adc_out;
                32'd22: page_buffer[183:176] <= adc_out;
                32'd21: page_buffer[175:168] <= adc_out;
                32'd20: page_buffer[167:160] <= adc_out;
                32'd19: page_buffer[159:152] <= adc_out;
                32'd18: page_buffer[151:144] <= adc_out;
                32'd17: page_buffer[143:136] <= adc_out;
                32'd16: page_buffer[135:128] <= adc_out;
                32'd15: page_buffer[127:120] <= adc_out;
                32'd14: page_buffer[119:112] <= adc_out;
                32'd13: page_buffer[111:104] <= adc_out;
                32'd12: page_buffer[103:96] <= adc_out;
                32'd11: page_buffer[95:88] <= adc_out;
                32'd10: page_buffer[87:80] <= adc_out;
                32'd9:  page_buffer[79:72] <= adc_out;
                32'd8:  page_buffer[71:64] <= adc_out;
                32'd7:  page_buffer[63:56] <= adc_out;
                32'd6:  page_buffer[55:48] <= adc_out;
                32'd5:  page_buffer[47:40] <= adc_out;
                32'd4:  page_buffer[39:32] <= adc_out;
                32'd3:  page_buffer[31:24] <= adc_out;
                32'd2:  page_buffer[23:16] <= adc_out;
                32'd1:  page_buffer[15:8] <= adc_out;
                32'd0:   page_buffer[7:0] <= adc_out;
        endcase

        case(samplecounter)
                32'd0:
                begin
                        //if (firstpass == 1'b0)
                        //begin
                                write_data <= page_buffer;
                                enable <= 1'b1;
                        //end

                end
                32'd60:
                begin
                        firstpass <= 1'b0;
                        if (firstpass == 1'b0)
                        begin
                                page_counter <= page_counter + 32'd1;
```

```
                                                        address <= address + 15'd64;
                                                end
                                        end
                                        default:
                                        begin
                                        end
                                endcase
                        end
                end
                else
                begin
                        done <= 1'b1;
                        enable <= 1'b0;
                end
        end
end

endmodule



module pagewriter(clk, rst, sdata, sclk, write_data, address, done, enable);


//----------------------------------------------------------------------
//                -- Input/Output Declarations --
//----------------------------------------------------------------------

input clk, rst;
output reg sclk;
inout reg sdata;

input [14:0]address;
input [511:0]write_data;

wire sclk_internal;
wire sdata_startcon,
     sdata_stopcon,
     sdata_writehigh,
     sdata_writelow;

input enable;
output reg done;


//----------------------------------------------------------------------
//                 -- State & Reg Declarations  --
//----------------------------------------------------------------------

parameter CONTROL = 3'd0,
          AHIGH = 3'd1,
          ALOW = 3'd2,
          DATA = 3'd3,
                  STOP = 3'd4,
                  WAIT = 3'd5;

parameter LOW_THRESH  = 32'd3,       // 3
                  HIGH_THRESH = 32'd60;       // 30

parameter SCLK_COUNTER_THRESH = 32'd128; // 64

reg [2:0]STATE, NEXT_STATE;
reg [31:0]counter;
reg [31:0]bit_counter;
reg [31:0]data_bit_counter;
reg secondpass;

downclock downclock1(clk, sclk_internal, rst);
startcon startcon1(clk, rst, sdata_startcon, counter);
```

```
stopcon stopcon1(clk, rst, sdata_stopcon, counter);
writehigh writehigh1(clk, rst, sdata_writehigh, counter);
writelow writelow1(clk, rst, sdata_writelow, counter);

//----------------------------------------------------------------------
//                 -- Begin Declarations & Coding --
//----------------------------------------------------------------------

always@(posedge clk or negedge rst)     // Determine STATE
begin

        if (rst == 1'b0)
                STATE <= DATA;
        else
                STATE <= NEXT_STATE;

end


always@(posedge clk or negedge rst)     // Determine outputs
begin

        if (rst == 1'b0)
        begin
                sclk <= 1'b0;
                counter <= 32'd0;
                bit_counter <= 32'd0;
                data_bit_counter <= 32'd0;
                sdata <= 1'bz;
                NEXT_STATE <= (enable == 1'b1)? CONTROL : WAIT;
                secondpass <= 1'b0;
                done <= 1'b0;
        end

        else
        begin

                sclk <= sclk_internal;
                counter <= (counter >= SCLK_COUNTER_THRESH)? 32'd0 : counter + 32'd1;

                case(STATE)

                        CONTROL:
                        begin
                                case (bit_counter)
                                        32'd0:
                                        begin
                                                sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_startcon : 1'bz;
                                                if (counter == LOW_THRESH)
                                                        secondpass <= 1'b1;
                                                bit_counter <= (counter == LOW_THRESH && secondpass
== 1'b1)? bit_counter + 32'd1 : bit_counter;
                                        end
                                        32'd1:
                                        begin
                                                sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writehigh : 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                        end
                                        32'd2:
                                        begin
                                                sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writelow : 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                        end
                                        32'd3:
                                        begin
```

```
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writehigh : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd4:
                                begin
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writelow : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd5:
                                begin
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writelow : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd6:
                                begin
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writelow : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd7:
                                begin
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writelow : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd8:
                                begin
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writelow : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd9:
                                begin
                                        sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                default:
                                begin
                                        NEXT_STATE <= AHIGH;
                                        bit_counter <= 32'd0;
                                end
                        endcase
                end

                AHIGH:
                begin
                        case (bit_counter)
                                32'd0:
                                begin
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_writelow : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd1:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[14] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
```

```
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd2:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[13] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                3'd3:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[12] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd4:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[11] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd5:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[10] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd6:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[9] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd7:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[8] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd8:
                                begin
                                        sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                default:
                                begin
```

```
                                        NEXT_STATE <= ALOW;
                                        bit_counter <= 32'd0;
                                end
                        endcase
                end

                ALOW:
                begin
                        case (bit_counter)
                                32'd0:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[7] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd1:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[6] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd2:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[5] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd3:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[4] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd4:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[3] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd5:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[2] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd6:
                                begin
```

```
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[1] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd7:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (address[0] == 1'b1)?
sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                32'd8:
                                begin
                                        sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                default:
                                begin
                                        NEXT_STATE <= DATA;
                                        bit_counter <= 32'd0;
                                end
                        endcase
                end

                DATA:
                begin
                        case (bit_counter)
                                32'd0:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                        data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                end
                                32'd1:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                        data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                end
                                32'd2:
                                begin
                                        if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                        else
                                                sdata <= 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                        data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                end
```

```
                                        32'd3:
                                        begin
                                                if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                        sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                                else
                                                        sdata <= 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                                data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                        end
                                        32'd4:
                                        begin
                                                if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                        sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                                else
                                                        sdata <= 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                                data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                        end
                                        32'd5:
                                        begin
                                                if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                        sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                                else
                                                        sdata <= 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                                data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                        end
                                        32'd6:
                                        begin
                                                if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                        sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                                else
                                                        sdata <= 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                                data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                        end
                                        32'd7:
                                        begin
                                                if (counter >= HIGH_THRESH || counter <= LOW_THRESH)
                                                        sdata <= (write_data[data_bit_counter] ==
1'b1)? sdata_writehigh : sdata_writelow;
                                                else
                                                        sdata <= 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                                data_bit_counter <= (counter == LOW_THRESH)?
data_bit_counter + 32'd1 : data_bit_counter;
                                        end
                                        32'd8:
                                        begin
                                                sdata <= 1'bz;
                                                bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                        end
                                        default:
                                        begin
                                                if (data_bit_counter >= 32'd511)
                                                begin
                                                        NEXT_STATE <= STOP;
```

```
                                                data_bit_counter <= 32'd0;
                                        end
                                        bit_counter <= 32'd0;
                                end
                        endcase
                end

                STOP:
                begin
                        case(bit_counter)
                                32'd0:
                                begin
                                        sdata <= (counter >= HIGH_THRESH || counter <=
LOW_THRESH)? sdata_stopcon : 1'bz;
                                        bit_counter <= (counter == LOW_THRESH)? bit_counter
+ 32'd1 : bit_counter;
                                end
                                default:
                                begin
                                        bit_counter <= 32'd0;
                                        NEXT_STATE <= WAIT;
                                end
                        endcase
                end

                default: //WAIT
                begin
                        if (enable == 1'b1)
                        begin
                                NEXT_STATE <= CONTROL;
                                done <= 1'b0;
                        end
                        done <= 1'b1;
                        sdata <= 1'bz;
                end

        endcase
    end

end


endmodule



module adcreader(clk, adcclock, din, dout, cs, rst, out);

    input clk, rst, din;
    output adcclock;
    output reg dout, cs;
    downclock downclock1(clk, adcclock, rst);
    output reg [7:0]out;


    reg [31:0]state;
    reg [7:0]buffer;

    always@(posedge adcclock or negedge rst)
    begin

        if (rst == 1'b0)
        begin
                dout <= 1'b0;
                cs <= 1'b1;
                state <= 32'd0;
                buffer <= 8'd0;
                out <= 8'd0;
        end
```

```
else
begin

        state <= (state > 32'd12)? 32'd0 : state + 32'd1;

        case(state)
                32'd0:
                begin
                        cs <= 1'b0;
                        dout <= 1'b1;
                        buffer <= 8'd0;
                end
                32'd1:
                begin
                        cs <= 1'b0;
                        dout <= 1'b0;
                end
                32'd2:
                begin
                        cs <= 1'b0;
                        dout <= 1'b0;
                end
                32'd3:
                begin
                        cs <= 1'b0;
                        dout <= 1'b1;
                end
                32'd4:
                begin

                end
                32'd5:
                begin
                        buffer[7] <= din;
                end
                32'd6:
                begin
                        buffer[6] <= din;
                end
                32'd7:
                begin
                        buffer[5] <= din;
                end
                32'd8:
                begin
                        buffer[4] <= din;
                end
                32'd9:
                begin
                        buffer[3] <= din;
                end
                32'd10:
                begin
                        buffer[2] <= din;
                end
                32'd11:
                begin
                        buffer[1] <= din;
                end
                32'd12:
                begin
                        buffer[0] <= din;
                        cs <= 1'b1;
                end
                default:
                begin
                        cs <= 1'b0;
                        dout <= 1'b1;
                        out <= buffer;
                end
        endcase
```

```
                end
        end


endmodule



module downclock(clk, adcclock, rst);

input clk, rst;
output reg adcclock;


reg [31:0]counter;
parameter threshold = 32'd128;     // 64
parameter half = 32'd64;          // 32

always@(posedge clk or negedge rst)
begin
        if (rst == 1'b0)
        begin
                counter <= 32'd0;
                adcclock = 1'b0;
        end
        else
        begin
                if (counter >= threshold)
                        counter <= 32'd0;
                else
                        counter <= counter + 32'd1;
                adcclock = (counter > half)? 1'b1 : 1'b0;
        end
end
endmodule



module samplecountertimer(clk, rst, samplecounter);

        input clk, rst;
        output reg [31:0]samplecounter;

        reg [31:0] counter;

        parameter THRESHOLD = 32'd8192;   //2^15 // 64 of these needs to be at least 7
milliseconds (8192), 128*64

        always@(posedge clk or negedge rst)
        begin
                if (rst == 1'b0)
                begin
                        counter <= 32'd0;
                        samplecounter <= 32'd0;
                end
                else
                begin
                        if (counter >= THRESHOLD)
                        begin
                                counter <= 32'd0;
                                samplecounter <= (samplecounter >= 32'd63)? 32'd0 : samplecounter +
32'd1;
                        end
                        else
                        begin
                                counter <= counter + 32'd1;
                        end
                end
        end
```

```
endmodule
module startcon(clk, rst, sdata, counter);

	input clk, rst;
	inout reg sdata;
	input [31:0]counter;

	always@(*)
	begin
		if (rst == 1'b0)
			sdata <= 1'bz;
		else
			 sdata = (counter < 32'd3 || counter >= 32'd100)? 1'b0 : 1'b1;
	end

endmodule


module stopcon(clk, rst, sdata, counter);

	input clk, rst;
	inout reg sdata;
	input [31:0]counter;

	always@(*)
	begin
		if (rst == 1'b0)
			sdata <= 1'bz;
		else
			 sdata = (counter < 32'd3 || counter >= 32'd100)? 1'b1 : 1'b0;
	end

endmodule


module writehigh(clk, rst, sdata, counter);

	input clk, rst;
	inout sdata;
	input [31:0]counter;

	assign sdata = (rst == 1'b1)? 1'bz : 1'b1;

endmodule


module writelow(clk, rst, sdata, counter);

	input clk, rst;
	inout sdata;
	input [31:0]counter;

	assign sdata = (rst == 1'b1)? 1'bz : 1'b0;

endmodule
```