Microprocessor Final Report

Michael Jang and George Wang

12/10/21

Abstract:

A breathalyzer is a device that measures and displays the breath alcohol content (BrAC) of a person. It samples the air that it detects for ethanol, and some models use this to change the voltage output of a printed circuit board (PCB). A simple circuit was constructed to fit the constraints of the analog-digital converter (ADC) on the STM32F401RE microcontroller unit (MCU). The ADC reading is then processed on the MCU and transformed into the appropriate two-digit reading, which is 5 bits long. The MCU then sends this reading over SPI to the MAX10 FPGA on the Arrow MAX1000 PCB. The FPGA then takes this information and time multiplexes rows of LEDs to display the two digits on an 8-by-8 LED matrix. Calibration was done by taking various measurements with a store-bought breathalyzer. This was successfully implemented, and the system as a whole behaves as desired.

Introduction:

The final project of E155 directed students to use the principles taught in the class to design and build a complex system that implements novel hardware not previously used in the labs. A volatile organic compound (VOC) sensor's resistance changes based on the level of a compound it detects in the air around it. Certain ones can detect ethanol, which appears in the breath of a person who has consumed alcohol. Alcohol consumption can have serious consequences, especially when it comes to the operation of motor vehicles, which is why laws are in place based on the blood alcohol (BAC) of a person. Because of this, using a sensor to display the BAC of the user is highly valuable, as it can prevent a person from participating in dangerous activities or getting arrested. This was the motivation behind this project.

The block diagram shown below in Figure 1 shows an overview of the entire breathalyzer system.

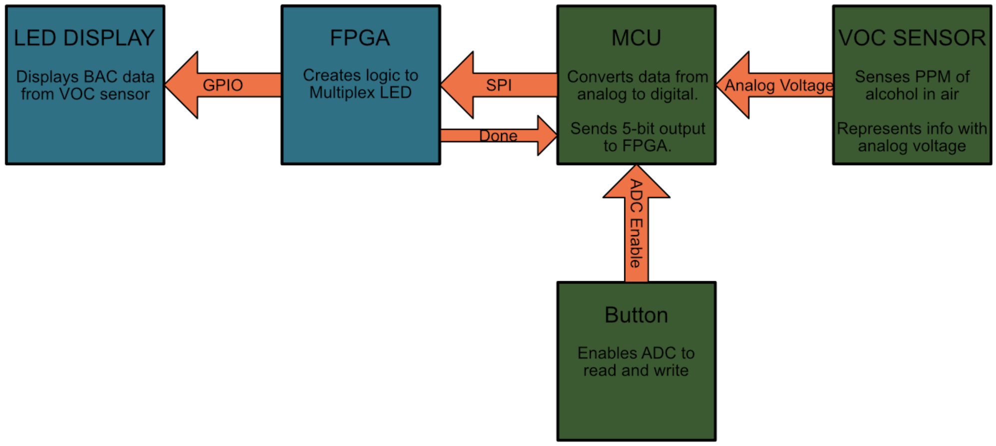


Figure 1: Overall System Block Diagram

As seen in the block diagram, the MCU is taking in the input from the VOC sensor and sending converted data to the FPGA, which runs the 8 by 8 LED matrix. The other human input is a button that tells the MCU to take a measurement.

VOC Sensor and Circuit:

The VOC sensor used in this project is the MiCS 5524. This sensor is capable of detecting concentrations of up to 500 ppm of ethanol. The PCB that includes the sensor takes in a 5V and GND input from the MCU, and outputs a voltage that increases with as the concentration of alcohol it detects increases. However, due to reasons that will be explained later, the ADC can only read values between 0V and 3.3V, while the output of the PCB can reach 5V. To combat this, the ADC will read the voltage in between two identical resistors linking the output to ground, which makes the maximum readable voltage 2.5V, which the ADC is able to read.

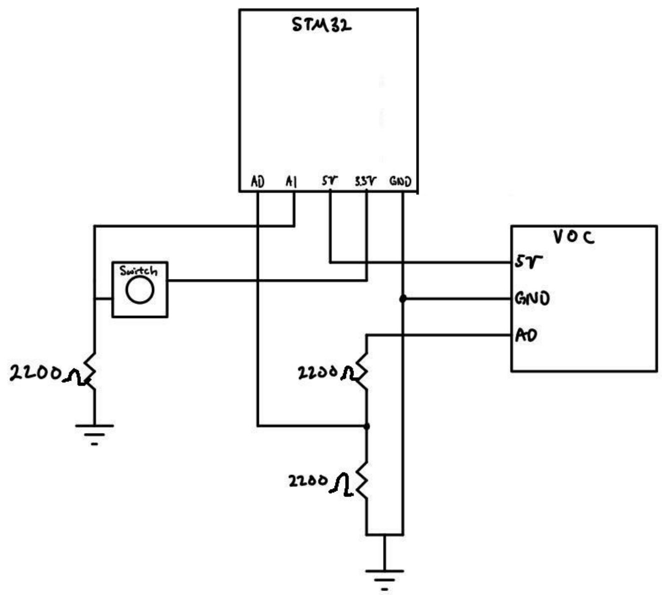


Figure 2: VOC to MCU Connections

MCU ADC Usage:

The ADC on the MCU converts the analog voltage output of the VOC sensor circuit, which is connected to pin A0 on the MCU, into a 12 bit number, which is the highest resolution possible for this unit. The 12 bit number is output by the ADC according to the Equation 1, with analog voltage on pin A0 as an input:

$$OUT = IN \cdot ((V_{REF+} - V_{REF-})/2^{N}) \quad (1)$$

In this case, V_{REF+} is 3.3V, V_{REF-} is 0V, and N is 12, meaning that the ADC outputs a number from 0 to 4095.

The ADC has the option of aligning the number to the left or the right, and the right-aligned option was chosen because when storing the number in a variable, the code will convert the binary number to a number in decimal format. Right shifting allows this number to be read correctly, rather than being shifted and multiplied by a certain amount.

The ADC has 16 channels to sample the voltages of, and any sequence of channels can be sampled in any order. For this project, the only channel used was ADC_IN0, which connected to pin A0 when it was set as an analog pin. The ADC also has single conversion mode, continuous conversion mode, and scan mode. Single conversion mode was used for this project, as it was desired to take only one measurement at the push of a button, shown in Figure 2. There are also sequence registers that keep track of the amount of channels to sample and what order to do so in. These registers were set to 1 channel and ADC_IN0, respectively.

Transformation to BAC on MCU:

As stated in a previous section, the maximum ethanol concentration that the BAC can detect is 500ppm, which roughly translates to 0.1974% BAC. To represent this fact, the code on the MCU converts the ADC reading to a 5 bit binary unsigned integer between 0 and 19. A calibration curve was found by blowing into a store-bought breathalyzer and then reading the voltage on pin A0 with an oscilloscope. Equation 2 below shows the polynomial curve found to be the calibration curve, where x is the BAC in hundreths of a percentage point.

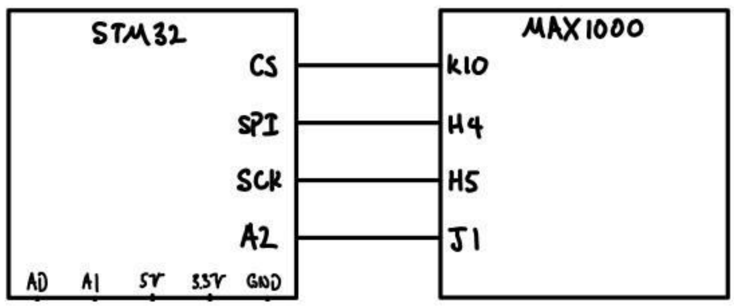
$$Voltage\ =\ 44.7352 \cdot x^2 - .5527 \cdot x\ + .207\ (2)$$

The code takes into account what ADC outputs these voltages translate to, and properly maps them onto a binary unsigned integer between 0 and 19. This value is then stored onto a variable, which is now ready to be sent to the FPGA via SPI.

MCU SPI Communication:

The MCU communicates the 5-bit value calculated during the prior step to the FPGA. This value is placed into the SPI data register, which is set to 8 bits and most significant bit first, and is shifted out on the rising edge of the SPI clock. The MCU acts as the SPI master while the FPGA acts as the client, except that it does not send data back to the MCU. The clock speed is set by a prescaler in the RCC and the SPI prescaler. The prescaler of the RCC was 16, and the SPI prescaler was set to 256. This caused the SPI clock to have a frequency of 10253 Hz. This low frequency was chosen so that the FPGA was able to sample the incoming bits while also running slowly enough that digits on the LED matrix did not bleed together. Figure 3 below shows the connections between the FPGA and MCU.

Figure 3: MCU to FPGA SPI Connections



MCU Algorithm:

After initializing and enabling the clock and necessary peripherals, the code enters a while loop and then waits for a button press. Once the button is pressed, pin A1 is driven high which tells the ADC to take a single measurement of the voltage output of the VOC circuit. It stores this measurement in a variable, and then puts that value through the voltage to BAC conversion mentioned in a previous section. A timer on the MCU activates to count to 1 second, which prevents any debouncing issues from causing the ADC to read multiple measurements with one button press. The MCU then sends the 5-bit value over SPI, which ends up being an 8-bit value with 3 leading zeroes. The MCU then waits for the FPGA to respond with a high signal that goes to pin A2, indicating that it has received and implemented the 5-bit number in the way that will be described later in the FPGA section of the report.

FPGA SPI Reception:

The FPGA hardware was programmed to be able to receive the 8-bit serially transmitted data from the MCU by causing it to act as a shift register. Essentially, 8 parallel bits were designated, the 5 least significant of which would be the input to the FPGA LED Multiplexing system that will be described in the next section of the report. Using a flop, at every positive edge of the SCK input from the microcontroller, the value on the H4 pin at that time will be shifted in as the least significant bit. This happens 8 times per transmission, so that the 5-bit number will be used as the input to the multiplexing module. Using a counter, the FPGA's clock runs at 20507 Hertz, twice as fast as SCK. This is to ensure that each of the shifts are detected and that, as mentioned previously, the digits on the LED matrix don't bleed together.

FPGA LED Multiplexing:

The FPGA takes in the 5 least significant bits of the SPI data from the MCU, decodes the data to a percentage (0.00, 0.01, 0.02 . . . 0.20) and displays the data on an LED matrix via multiplexing. A seven state FSM is used to determine which row of the LED is turned on for multiplexing. Combinational logic is used to determine which LEDs to assert based on which row is powered and which number needs to be displayed. Transistors are used to provide power to each row of the LED matrix.

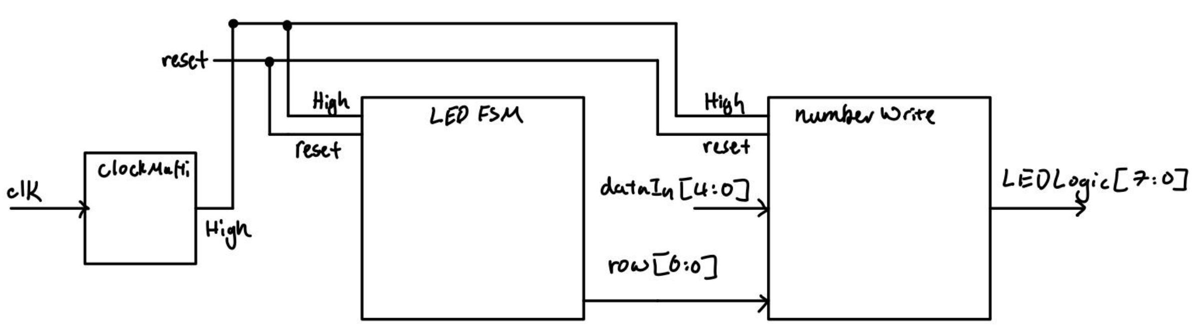


Figure 4: FPGA Block Diagram Without SPI

Results:

The system works as intended. The voltages read on the oscilloscope match the ADC and BAC readings that are read in the debugger in VS Code. The system reacts to the presence of alcohol in many forms. It reacts to the presence of vodka on one's breath, the presence of mouthwash containing alcohol on one's breath, and even the presence of mouthwash on a cotton ball held closely to the sensor. This means that the breathalyzer can be demonstrated in a professional setting while also adhering to COVID regulations, which is what was desired from the system when it was first thought of.

References:

lady ada. Adafruit. https://learn.adafruit.com/adafruit-mics5524-gas-sensor-breakout July 2016

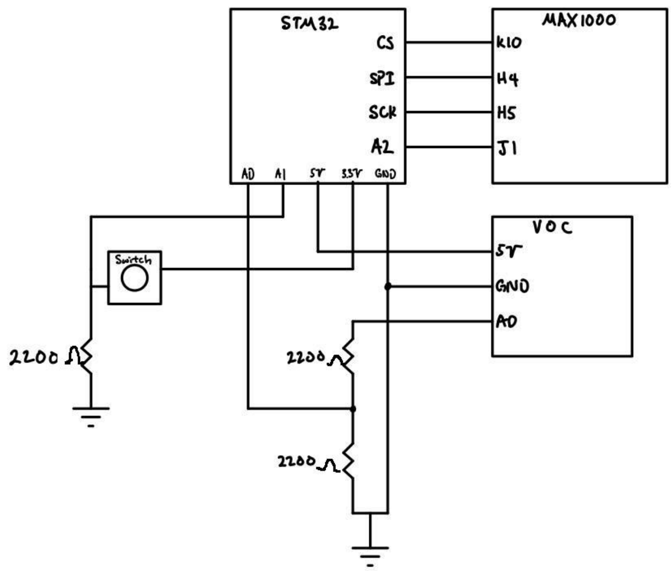
http://www.mecinca.net/ALCOHOLIMETROS_Alcosim/BAC%20BrAC%20conversion%20table[1].pdf

https://senseair.com/knowledge/information-and-education/gases/c-h-oh-ethanol/

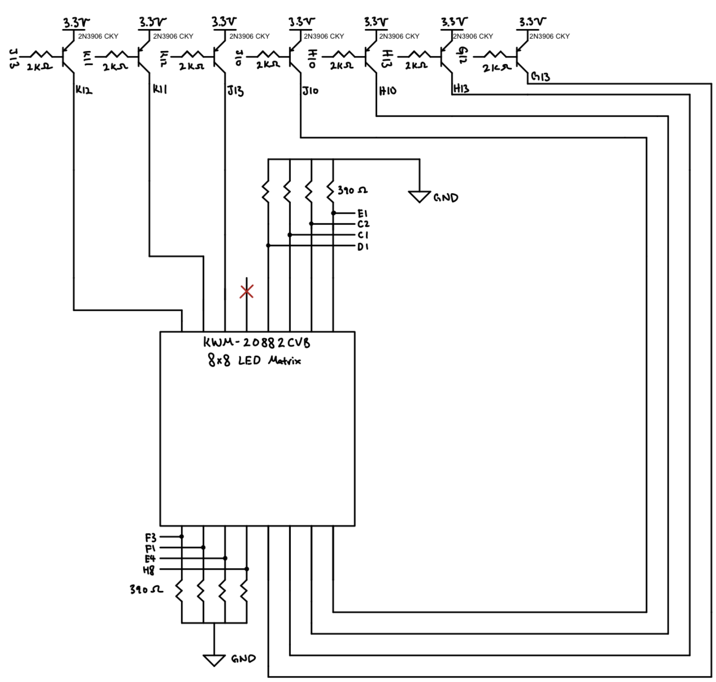
Bill of Materials:

Material	Quantity	Cost	Total Cost
STM32F401RE MCU	1	0	0
MAX1000	1	0	0
MiCS 5524 Sensor + PCB + Shipping	1	$62.00	$62.00
2N3906 Transistor	7	0	0
2200 Ohm Resistor	3	0	0
390 Ohm Resistor	8	0	0
Push Button	1	0	0
8x8 LED Matrix	1	0	0
2000 Ohm Resistor	7	0	0
2700 Ohm Resistor	4	0	0
Total			$62.00

Appendix A: Breadboard Schematics



MCU and VOC Sensor



FPGA and LED

Appendix B: MCU Code

```
final_project.c

#include "STM32F401RE_GPIO.h"
#include "STM32F401RE_FLASH.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_ADC.h"
#include "STM32F401RE_SPI.h"
#include "STM32F401RE_TIM.h"
#include <string.h> // for strstr()
#include <stdint.h> // for integer types (i.e., uint32_t)
#include <stdio.h>  // for sprintf()

uint8_t ADCto5bits(uint16_t adc){
  uint8_t a;
  if (adc > (.207*4096/(3.3*2))){
    if(adc > (.209*4096/(3.3*2))){
      if(adc > (.2138*4096/(3.3*2))){
        if(adc > (.2307*4096/(3.3*2))){
          if (adc > (.2565*4096/(3.3*2))){
            if(adc > (.2912*4096/(3.3*2))){
              if(adc > (.3349*4096/(3.3*2))){
                if(adc > (.3875*4096/(3.3*2))){
                  if(adc > (.4491*4096/(3.3*2))){
                    if(adc > (.5196*4096/(3.3*2))){
                      if(adc > (.5991*4096/(3.3*2))){
                        if(adc > (.6875*4096/(3.3*2))){
                          if(adc > (.7849*4096/(3.3*2))){
                            if(adc > (.8912*4096/(3.3*2))){
                              if(adc > (1.0064*4096/(3.3*2))){
                                if(adc > (1.1306*4096/(3.3*2))){
                                  if(adc > (1.2638*4096/(3.3*2))){
                                    if(adc > (1.4059*4096/(3.3*2))){
                                      if(adc > (1.5569*4096/(3.3*2))){
                                        a = 19;}
                                      else {a = 18;}
                                    }
                                    else {a = 17;}
                                  }
                                  else {a = 16;}
                                }
                                else {a = 15;}
                              }
                              else {a = 14;}
                            }
                            else {a = 13;}
```

```
                          }
                          else {a = 12;}
                        }
                        else {a = 11;}
                      }
                      else {a = 10;}
                    }
                    else {a = 9;}
                  }
                  else {a = 8;}
                }
                else {a = 7;}
              }
              else {a = 6;}
            }
            else {a = 5;}
          }
          else {a = 4;}
        }
        else {a = 3;}
      }
      else {a = 2;}
    }
    else {a = 1;}
  }
  else {a = 0;}
  return a;
}


int main(void) {
  // Configure the flash and then set clock to 84 MHz from PLL
  configureFlash();
  configureClock();
  RCC->APB2ENR.ADC1EN = 1;
  // Turn on GPIOA
  RCC->AHB1ENR.GPIOAEN = 1;

  //Enable timer
  RCC->CFGR.HPRE = 0b0001;
  RCC->APB2ENR.TIM11EN = 1;

  // Set PA0 as an input for the ADC, another pin as input for reading and
another for signaling ready
  pinMode(GPIOA, 0, GPIO_ANALOG);
  pinMode(GPIOA, 1, GPIO_INPUT);
  pinMode(GPIOA, 4, GPIO_INPUT);
```

```
durationTimer();
// Set MISO, MOSI, SCK, and CE
spiInit(0b111, 0, 0);

// Initialize the ADC
ADCinit();
//digitalWrite(GPIOB, 6, 1);
uint16_t b;
//b = measure();
uint8_t a;
//a = ADCto5bits(b);
//b = measure();
//a = ADCto5bits(b);
//SPI Stuff
//spiSendReceive(a);
while(1) {
  while(digitalRead(GPIOA, 1)!= 1); //wait for button push
  TIMERD->ARR.ARR = (1000 * 2)-1;  // Wait for amount of time to pass
  TIMERD->CCR1.CCR1 = (1000 * 2);
  TIMERD->EGR.UG = 1;
  while(TIMERD->SR.CC1IF == 0){}
  TIMERD->SR.CC1IF = 0;
  durationTimer(); // Reset Timer
  b = measure(); // Measure with ADC
  a = ADCto5bits(b); //Convert measurement
  //SPI Stuff
  spiSendReceive(a); //Send 5 bit number to FPGA
  while(digitalRead(GPIOA, 4)!= 1); // Wait for done signal from FPGA
}
return b+a;
```

Appendix C: SystemVerilog Code

```
//11/22/2021
//George Wang
//gewang@g.hmc.edu
//module determining the row we are in
module Breathylizer
   (input logic clk,
    input logic load,
    input logic sck,
    input logic reset,
    input logic sdi, //switches
    output logic [6:0] rowTrans, //transistors
    output logic [7:0] LEDLogic,
    //output logic [7:0] LEDLogic2,
    output logic done);

    logic [4:0] dataIn;
    logic [6:0] row;
    logic High;

    spi SPI(sck, sdi, dataIn);

    clockMulti CM0(clk, High);

    LEDFSM FSM0(High, reset, load, row, done);

    numberWrite Write0(row, dataIn, LEDLogic);

   //transistor logic
    assign rowTrans[0] = ~row[0];
    assign rowTrans[1] = ~row[1];
    assign rowTrans[2] = ~row[2];
    assign rowTrans[3] = ~row[3];
    assign rowTrans[4] = ~row[4];
    assign rowTrans[5] = ~row[5];
    assign rowTrans[6] = ~row[6];

endmodule


module LEDFSM
    (input  logic clk,
     input  logic reset,
     input  logic load,
     output logic [6:0] row,
     output logic done);

     typedef enum logic [4:0] {R0, R1, R2, R3, R4, R5, R6,R7,R8} statetype;   //we wont use the top LED matrix
     statetype state, nextState;

     //state register
     always_ff @(posedge clk, posedge reset)
        if (reset)  state <= R0;
        else            state <= nextState;

     //next state logic
     //5 state FSM. 4 record row information
     always_comb
        case(state)
            R0: if(load) nextState <= R1;
                else nextState <= R0;
            R1: if(load) nextState <= R1;
                else nextState <= R2;
            R2: if(load) nextState <= R1;
                else nextState <= R3;
            R3: if(load) nextState <= R1;
                else nextState <= R4;
            R4: if(load) nextState <= R1;
                else nextState <= R5;
            R5: if(load) nextState <= R1;
                else nextState <= R6;
            R6: if(load) nextState <= R1;
                else nextState <= R7;
            R7: if(load) nextState <= R1;
                else nextState <= R8;
            R8: if(load) nextState <= R1;
```

```
                    else nextState <= R2;
                default: nextState <= R0;
            endcase

        assign row[0] = (state == R2);
        assign row[1] = (state == R3);
        assign row[2] = (state == R4);
        assign row[3] = (state == R5);
        assign row[4] = (state == R6);
        assign row[5] = (state == R7);
        assign row[6] = (state == R8);

        assign done = ~(state == R0 | state == R1);
endmodule




module numberWrite
(input logic [6:0] row,
input logic [4:0] dataIn,  //switches
output logic [7:0] LEDLogic);

 logic [7:0] zero0, zero1, zero2, zero3, zero4, zero5, zero6;
 logic [7:0] one0, one1, one2, one3, one4, one5, one6;
 logic [7:0] two0, two1, two2, two3, two4, two5, two6;
 logic [7:0] three0, three1, three2, three3, three4, three5, three6;
 logic [7:0] four0, four1, four2, four3, four4, four5, four6;
 logic [7:0] five0, five1, five2, five3, five4, five5, five6;
 logic [7:0] six0, six1, six2, six3, six4, six5, six6;
 logic [7:0] seven0, seven1, seven2, seven3, seven4, seven5, seven6;
 logic [7:0] eight0, eight1, eight2, eight3, eight4, eight5, eight6;
 logic [7:0] nine0, nine1, nine2, nine3, nine4, nine5, nine6;
 logic [7:0] ten0, ten1, ten2, ten3, ten4, ten5, ten6;
 logic [7:0] onet0, onet1, onet2, onet3, onet4, onet5, onet6;
 logic [7:0] twot0, twot1, twot2, twot3, twot4, twot5, twot6;
 logic [7:0] threet0, threet1, threet2, threet3, threet4, threet5, threet6;
 logic [7:0] fourt0, fourt1, fourt2, fourt3, fourt4, fourt5, fourt6;
 logic [7:0] fivet0, fivet1, fivet2, fivet3, fivet4, fivet5, fivet6;
 logic [7:0] sixt0, sixt1, sixt2, sixt3, sixt4, sixt5, sixt6;
 logic [7:0] sevent0, sevent1, sevent2, sevent3, sevent4, sevent5, sevent6;
 logic [7:0] eightt0, eightt1, eightt2, eightt3, eightt4, eightt5, eightt6;
 logic [7:0] ninet0, ninet1, ninet2, ninet3, ninet4, ninet5, ninet6;


        assign zero0 = 8'b00000000;
        assign zero1 = 8'b01100110;
        assign zero2 = 8'b01100110;
        assign zero3 = 8'b01100110;
        assign zero4 = 8'b01100110;
        assign zero5 = 8'b01100110;
        assign zero6 = 8'b00000000;

        assign one0 = 8'b00000111;
        assign one1 = 8'b01100111;
        assign one2 = 8'b01100111;
        assign one3 = 8'b01100111;
        assign one4 = 8'b01100111;
        assign one5 = 8'b01100111;
        assign one6 = 8'b00000111;

        assign two0 = 8'b00000000;
        assign two1 = 8'b01101110;
        assign two2 = 8'b01101110;
        assign two3 = 8'b01100000;
        assign two4 = 8'b01100111;
        assign two5 = 8'b01100111;
        assign two6 = 8'b00000000;

        assign three0 = 8'b00000000;
        assign three1 = 8'b01101110;
        assign three2 = 8'b01101110;
        assign three3 = 8'b01100000;
        assign three4 = 8'b01101110;
        assign three5 = 8'b01101110;
        assign three6 = 8'b00000000;
```

```
        assign four0 = 8'b00000110;
        assign four1 = 8'b01100110;
        assign four2 = 8'b01100110;
        assign four3 = 8'b01100000;
        assign four4 = 8'b01101110;
        assign four5 = 8'b01101110;
        assign four6 = 8'b00001110;

        assign five0 = 8'b00000000;
        assign five1 = 8'b01100111;
        assign five2 = 8'b01100111;
        assign five3 = 8'b01100000;
        assign five4 = 8'b01101110;
        assign five5 = 8'b01101110;
        assign five6 = 8'b00000000;

        assign six0 = 8'b00000000;
        assign six1 = 8'b01100111;
        assign six2 = 8'b01100111;
        assign six3 = 8'b01100000;
        assign six4 = 8'b01100110;
        assign six5 = 8'b01100110;
        assign six6 = 8'b00000000;

        assign seven0 = 8'b00000000;
        assign seven1 = 8'b01101110;
        assign seven2 = 8'b01101110;
        assign seven3 = 8'b01101110;
        assign seven4 = 8'b01101110;
        assign seven5 = 8'b01101110;
        assign seven6 = 8'b00001110;

        assign eight0 = 8'b00000000;
        assign eight1 = 8'b01100110;
        assign eight2 = 8'b01100110;
        assign eight3 = 8'b01100000;
        assign eight4 = 8'b01100110;
        assign eight5 = 8'b01100110;
        assign eight6 = 8'b00000000;

        assign nine0 = 8'b00000000;
        assign nine1 = 8'b01100110;
        assign nine2 = 8'b01100110;
        assign nine3 = 8'b01100000;
        assign nine4 = 8'b01101110;
        assign nine5 = 8'b01101110;
        assign nine6 = 8'b00001110;

        assign ten0 = 8'b01110000;
        assign ten1 = 8'b01110110;
        assign ten2 = 8'b01110110;
        assign ten3 = 8'b01110110;
        assign ten4 = 8'b01110110;
        assign ten5 = 8'b01110110;
        assign ten6 = 8'b01110000;

        assign onet0 = 8'b01110111;
        assign onet1 = 8'b01110111;
        assign onet2 = 8'b01110111;
        assign onet3 = 8'b01110111;
        assign onet4 = 8'b01110111;
        assign onet5 = 8'b01110111;
        assign onet6 = 8'b01110111;

        assign twot0 = 8'b01110000;
        assign twot1 = 8'b01111110;
        assign twot2 = 8'b01111110;
        assign twot3 = 8'b01110000;
        assign twot4 = 8'b01110111;
        assign twot5 = 8'b01110111;
        assign twot6 = 8'b01110000;

        assign threet0 = 8'b01110000;
        assign threet1 = 8'b01111110;
        assign threet2 = 8'b01111110;
```

```
        assign threet3 = 8'b01110000;
        assign threet4 = 8'b01111110;
        assign threet5 = 8'b01111110;
        assign threet6 = 8'b01110000;

        assign fourt0 = 8'b01110110;
        assign fourt1 = 8'b01110110;
        assign fourt2 = 8'b01110110;
        assign fourt3 = 8'b01110000;
        assign fourt4 = 8'b01111110;
        assign fourt5 = 8'b01111110;
        assign fourt6 = 8'b01111110;

        assign fivet0 = 8'b01110000;
        assign fivet1 = 8'b01110111;
        assign fivet2 = 8'b01110111;
        assign fivet3 = 8'b01110000;
        assign fivet4 = 8'b01111110;
        assign fivet5 = 8'b01111110;
        assign fivet6 = 8'b01110000;

        assign sixt0 = 8'b01110000;
        assign sixt1 = 8'b01110111;
        assign sixt2 = 8'b01110111;
        assign sixt3 = 8'b01110000;
        assign sixt4 = 8'b01110110;
        assign sixt5 = 8'b01110110;
        assign sixt6 = 8'b01110000;

        assign sevent0 = 8'b01110000;
        assign sevent1 = 8'b01111110;
        assign sevent2 = 8'b01111110;
        assign sevent3 = 8'b01111110;
        assign sevent4 = 8'b01111110;
        assign sevent5 = 8'b01111110;
        assign sevent6 = 8'b01111110;

        assign eightt0 = 8'b01110000;
        assign eightt1 = 8'b01110110;
        assign eightt2 = 8'b01110110;
        assign eightt3 = 8'b01110000;
        assign eightt4 = 8'b01110110;
        assign eightt5 = 8'b01110110;
        assign eightt6 = 8'b01110000;

        assign ninet0 = 8'b01110000;
        assign ninet1 = 8'b01110110;
        assign ninet2 = 8'b01110110;
        assign ninet3 = 8'b01110000;
        assign ninet4 = 8'b01111110;
        assign ninet5 = 8'b01111110;
        assign ninet6 = 8'b01111110;



    always_comb
        case(row)
            7'b0000001: if      (dataIn == 5'b00000) LEDLogic = zero0;          //zero0
                        else if (dataIn == 5'b00001) LEDLogic = one0;           //one0
                        else if (dataIn == 5'b00010) LEDLogic = two0;                 //two0
                        else if (dataIn == 5'b00011) LEDLogic = three0;       //three0
                        else if (dataIn == 5'b00100) LEDLogic = four0;                //four0
                        else if (dataIn == 5'b00101) LEDLogic = five0;                //five0
                        else if (dataIn == 5'b00110) LEDLogic = six0;           //six0
                        else if (dataIn == 5'b00111) LEDLogic = seven0;       //seven0
                        else if (dataIn == 5'b01000) LEDLogic = eight0;               //eight0
                        else if (dataIn == 5'b01001) LEDLogic = nine0;                //nine0

                        else if (dataIn == 5'b01010) LEDLogic = ten0;           //10
                        else if (dataIn == 5'b01011) LEDLogic = onet0;          //11
                        else if (dataIn == 5'b01100) LEDLogic = twot0;          //12
                        else if (dataIn == 5'b01101) LEDLogic = threet0;          //13
                        else if (dataIn == 5'b01110) LEDLogic = fourt0;           //14
                        else if (dataIn == 5'b01111) LEDLogic = fivet0;           //15
                        else if (dataIn == 5'b10000) LEDLogic = sixt0;          //16
                        else if (dataIn == 5'b10001) LEDLogic = sevent0;          //17
```

```
                    else if (dataIn == 5'b10010) LEDLogic = eightt0;            //18
                    else if (dataIn == 5'b10011) LEDLogic = ninet0;             //19
                    else LEDLogic = 8'b11111111;

        7'b0000010: if      (dataIn == 5'b00000) LEDLogic = zero1;          //zero0
                    else if (dataIn == 5'b00001) LEDLogic = one1;           //one0
                    else if (dataIn == 5'b00010) LEDLogic = two1;               //two0
                    else if (dataIn == 5'b00011) LEDLogic = three1;       //three0
                    else if (dataIn == 5'b00100) LEDLogic = four1;              //four0
                    else if (dataIn == 5'b00101) LEDLogic = five1;              //five0
                    else if (dataIn == 5'b00110) LEDLogic = six1;           //six0
                    else if (dataIn == 5'b00111) LEDLogic = seven1;       //seven0
                    else if (dataIn == 5'b01000) LEDLogic = eight1;             //eight0
                    else if (dataIn == 5'b01001) LEDLogic = nine1;              //nine0

                    else if (dataIn == 5'b01010) LEDLogic = ten1;           //10
                    else if (dataIn == 5'b01011) LEDLogic = onet1;          //11
                    else if (dataIn == 5'b01100) LEDLogic = twot1;          //12
                    else if (dataIn == 5'b01101) LEDLogic = threet1;          //13
                    else if (dataIn == 5'b01110) LEDLogic = fourt1;           //14
                    else if (dataIn == 5'b01111) LEDLogic = fivet1;           //15
                    else if (dataIn == 5'b10000) LEDLogic = sixt1;          //16
                    else if (dataIn == 5'b10001) LEDLogic = sevent1;          //17
                    else if (dataIn == 5'b10010) LEDLogic = eightt1;          //18
                    else if (dataIn == 5'b10011) LEDLogic = ninet1;           //19
                    else LEDLogic = 8'b11111111;


        7'b0000100: if      (dataIn == 5'b00000) LEDLogic = zero2;          //zero0
                    else if (dataIn == 5'b00001) LEDLogic = one2;           //one0
                    else if (dataIn == 5'b00010) LEDLogic = two2;               //two0
                    else if (dataIn == 5'b00011) LEDLogic = three2;       //three0
                    else if (dataIn == 5'b00100) LEDLogic = four2;              //four0
                    else if (dataIn == 5'b00101) LEDLogic = five2;              //five0
                    else if (dataIn == 5'b00110) LEDLogic = six2;           //six0
                    else if (dataIn == 5'b00111) LEDLogic = seven2;       //seven0
                    else if (dataIn == 5'b01000) LEDLogic = eight2;             //eight0
                    else if (dataIn == 5'b01001) LEDLogic = nine2;              //nine0

                    else if (dataIn == 5'b01010) LEDLogic = ten2;           //10
                    else if (dataIn == 5'b01011) LEDLogic = onet2;          //11
                    else if (dataIn == 5'b01100) LEDLogic = twot2;          //12
                    else if (dataIn == 5'b01101) LEDLogic = threet2;          //13
                    else if (dataIn == 5'b01110) LEDLogic = fourt2;           //14
                    else if (dataIn == 5'b01111) LEDLogic = fivet2;           //15
                    else if (dataIn == 5'b10000) LEDLogic = sixt2;          //16
                    else if (dataIn == 5'b10001) LEDLogic = sevent2;          //17
                    else if (dataIn == 5'b10010) LEDLogic = eightt2;          //18
                    else if (dataIn == 5'b10011) LEDLogic = ninet2;             //19
                    else LEDLogic = 8'b11111111;


        7'b0001000: if      (dataIn == 5'b00000) LEDLogic = zero3;          //zero0
                    else if (dataIn == 5'b00001) LEDLogic = one3;           //one0
                    else if (dataIn == 5'b00010) LEDLogic = two3;               //two0
                    else if (dataIn == 5'b00011) LEDLogic = three3;       //three0
                    else if (dataIn == 5'b00100) LEDLogic = four3;              //four0
                    else if (dataIn == 5'b00101) LEDLogic = five3;              //five0
                    else if (dataIn == 5'b00110) LEDLogic = six3;           //six0
                    else if (dataIn == 5'b00111) LEDLogic = seven3;       //seven0
                    else if (dataIn == 5'b01000) LEDLogic = eight3;             //eight0
                    else if (dataIn == 5'b01001) LEDLogic = nine3;              //nine0

                    else if (dataIn == 5'b01010) LEDLogic = ten3;           //10
                    else if (dataIn == 5'b01011) LEDLogic = onet3;          //11
                    else if (dataIn == 5'b01100) LEDLogic = twot3;          //12
                    else if (dataIn == 5'b01101) LEDLogic = threet3;          //13
                    else if (dataIn == 5'b01110) LEDLogic = fourt3;           //14
                    else if (dataIn == 5'b01111) LEDLogic = fivet3;           //15
                    else if (dataIn == 5'b10000) LEDLogic = sixt3;          //16
                    else if (dataIn == 5'b10001) LEDLogic = sevent3;          //17
                    else if (dataIn == 5'b10010) LEDLogic = eightt3;          //18
                    else if (dataIn == 5'b10011) LEDLogic = ninet3;           //19
                    else LEDLogic = 8'b11111111;
```

```


            7'b0010000: if      (dataIn == 5'b00000) LEDLogic = zero4;          //zero0
                        else if (dataIn == 5'b00001) LEDLogic = one4;           //one0
                        else if (dataIn == 5'b00010) LEDLogic = two4;                 //two0
                        else if (dataIn == 5'b00011) LEDLogic = three4;      //three0
                        else if (dataIn == 5'b00100) LEDLogic = four4;                //four0
                        else if (dataIn == 5'b00101) LEDLogic = five4;                //five0
                        else if (dataIn == 5'b00110) LEDLogic = six4;           //six0
                        else if (dataIn == 5'b00111) LEDLogic = seven4;      //seven0
                        else if (dataIn == 5'b01000) LEDLogic = eight4;               //eight0
                        else if (dataIn == 5'b01001) LEDLogic = nine4;                //nine0

                        else if (dataIn == 5'b01010) LEDLogic = ten4;           //10
                        else if (dataIn == 5'b01011) LEDLogic = onet4;          //11
                        else if (dataIn == 5'b01100) LEDLogic = twot4;          //12
                        else if (dataIn == 5'b01101) LEDLogic = threet4;           //13
                        else if (dataIn == 5'b01110) LEDLogic = fourt4;            //14
                        else if (dataIn == 5'b01111) LEDLogic = fivet4;            //15
                        else if (dataIn == 5'b10000) LEDLogic = sixt4;          //16
                        else if (dataIn == 5'b10001) LEDLogic = sevent4;           //17
                        else if (dataIn == 5'b10010) LEDLogic = eightt4;           //18
                        else if (dataIn == 5'b10011) LEDLogic = ninet4;
      //19
                        else LEDLogic = 8'b11111111;


            7'b0100000: if      (dataIn == 5'b00000) LEDLogic = zero5;          //zero0
                        else if (dataIn == 5'b00001) LEDLogic = one5;           //one0
                        else if (dataIn == 5'b00010) LEDLogic = two5;                 //two0
                        else if (dataIn == 5'b00011) LEDLogic = three5;      //three0
                        else if (dataIn == 5'b00100) LEDLogic = four5;                //four0
                        else if (dataIn == 5'b00101) LEDLogic = five5;                //five0
                        else if (dataIn == 5'b00110) LEDLogic = six5;           //six0
                        else if (dataIn == 5'b00111) LEDLogic = seven5;      //seven0
                        else if (dataIn == 5'b01000) LEDLogic = eight5;               //eight0
                        else if (dataIn == 5'b01001) LEDLogic = nine5;                //nine0

                        else if (dataIn == 5'b01010) LEDLogic = ten5;           //10
                        else if (dataIn == 5'b01011) LEDLogic = onet5;          //11
                        else if (dataIn == 5'b01100) LEDLogic = twot5;          //12
                        else if (dataIn == 5'b01101) LEDLogic = threet5;           //13
                        else if (dataIn == 5'b01110) LEDLogic = fourt5;            //14
                        else if (dataIn == 5'b01111) LEDLogic = fivet5;            //15
                        else if (dataIn == 5'b10000) LEDLogic = sixt5;          //16
                        else if (dataIn == 5'b10001) LEDLogic = sevent5;           //17
                        else if (dataIn == 5'b10010) LEDLogic = eightt5;           //18
                        else if (dataIn == 5'b10011) LEDLogic = ninet5;
      //19
                        else LEDLogic = 8'b11111111;


            7'b1000000: if      (dataIn == 5'b00000) LEDLogic = zero6;          //zero0
                        else if (dataIn == 5'b00001) LEDLogic = one6;           //one0
                        else if (dataIn == 5'b00010) LEDLogic = two6;                 //two0
                        else if (dataIn == 5'b00011) LEDLogic = three6;
                        else if (dataIn == 5'b00100) LEDLogic = four6;
                        else if (dataIn == 5'b00101) LEDLogic = five6;
                        else if (dataIn == 5'b00110) LEDLogic = six6;
                        else if (dataIn == 5'b00111) LEDLogic = seven6;
                        else if (dataIn == 5'b01000) LEDLogic = eight6;
                        else if (dataIn == 5'b01001) LEDLogic = nine6;

                        else if (dataIn == 5'b01010) LEDLogic = ten6;           //10
                        else if (dataIn == 5'b01011) LEDLogic = onet6;          //11
                        else if (dataIn == 5'b01100) LEDLogic = twot6;          //12
                        else if (dataIn == 5'b01101) LEDLogic = threet6;           //13
                        else if (dataIn == 5'b01110) LEDLogic = fourt6;            //14
                        else if (dataIn == 5'b01111) LEDLogic = fivet6;            //15
                        else if (dataIn == 5'b10000) LEDLogic = sixt6;          //16
                        else if (dataIn == 5'b10001) LEDLogic = sevent6;           //17
                        else if (dataIn == 5'b10010) LEDLogic = eightt6;           //18
                        else if (dataIn == 5'b10011) LEDLogic = ninet6;
      //19
                        else LEDLogic = 8'b11111111;

```

```
            default: LEDLogic = 8'b11111111;
        endcase
endmodule




//9/8/21
//George Wang
//gewang@g.hmc.edu
//module for generating clock signal at around 2.4Hz
//Referenced Better Verilog Counter Idiom
module clockMulti
    (input logic clk,
     output logic High);

    logic [15-1:0] LEDHigh;

    always_ff @(posedge clk)
         LEDHigh <= LEDHigh + 56;
                assign High = LEDHigh[14];

endmodule


module spi(input  logic sck,
           input  logic sdi,
           output logic [4:0] ADC);

    logic [7:0] ADC8;
    always_ff @(posedge sck)
        ADC8 = {ADC8[6:0],sdi};

    assign ADC = ADC8[4:0];


endmodule
```