Final Project: 8-bit Depth Scanner and 21-bit Dual Servo Assembly
Luis Gomez, Kattia Chang-Kam and Jorben Russel Ablang

Introduction

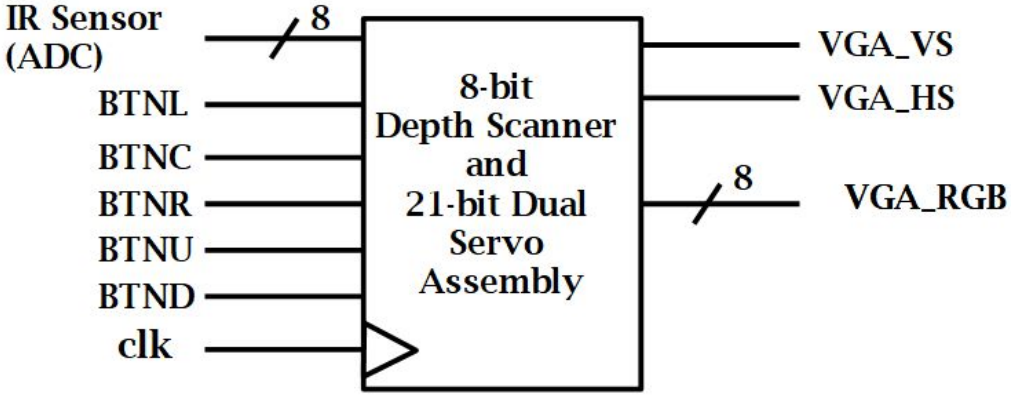


Figure 1. 8-bit Depth Scanner and 21-bit Dual Servo Assembly Block Box Diagram

This project consisted of a depth scanner that is able to scan across a two dimensional area and output a rough image of the objects present in that area. The color of each pixel depends on the distance of the sensor to the actual object. The input to this device would be receive from and distance measuring sensor and the output would be shown as an image at a VGA monitor. The sensor used is the SHARP GP2Y0A21YK0F which has an effective range between 10 cm to 80 cm. This analog input is then converted to an 8-bit value through an ADC implemented using an Arduino UNO, so that this data can be sent to the 8-bit RAT computer. The 8-bit value inputs for the output color to take 256 different values. The output to the VGA is 8-bits for the value of the color that should be painted at a particular pixel, and also 1-bit output each to the VGA to control the vertical and horizontal sync. Additionally, two HS-485HB servo motors are used to move the sensor in the XY plane. The motion of the motors are driven by the interrupt signal created by the sensor reading.

Operation Manual

Initially, the components of the device should properly hooked up to their respective port. Once the device is correctly set up, the user is free to pick which control they want to press. Button right (BTNR) and button left (BTNL) are responsible for controlling the servo horizontally while button down (BTND) and button up(BTNU) are responsible for controlling the servo vertically. Here, each time that the user presses any button, the pixel in the screen will be painted with a color that corresponds to the particular digital value read from the sensor. This process will continue as long as the user is driving the servo to move either right, left, down, or up. The pixel will be painted with dark colors if the object is extremely near from the sensor, otherwise, it will be light colors. If the user is finished scanning the object, the device can be restarted by pressing the center button (BTNC). This will allow the user to scan another object of their choice.

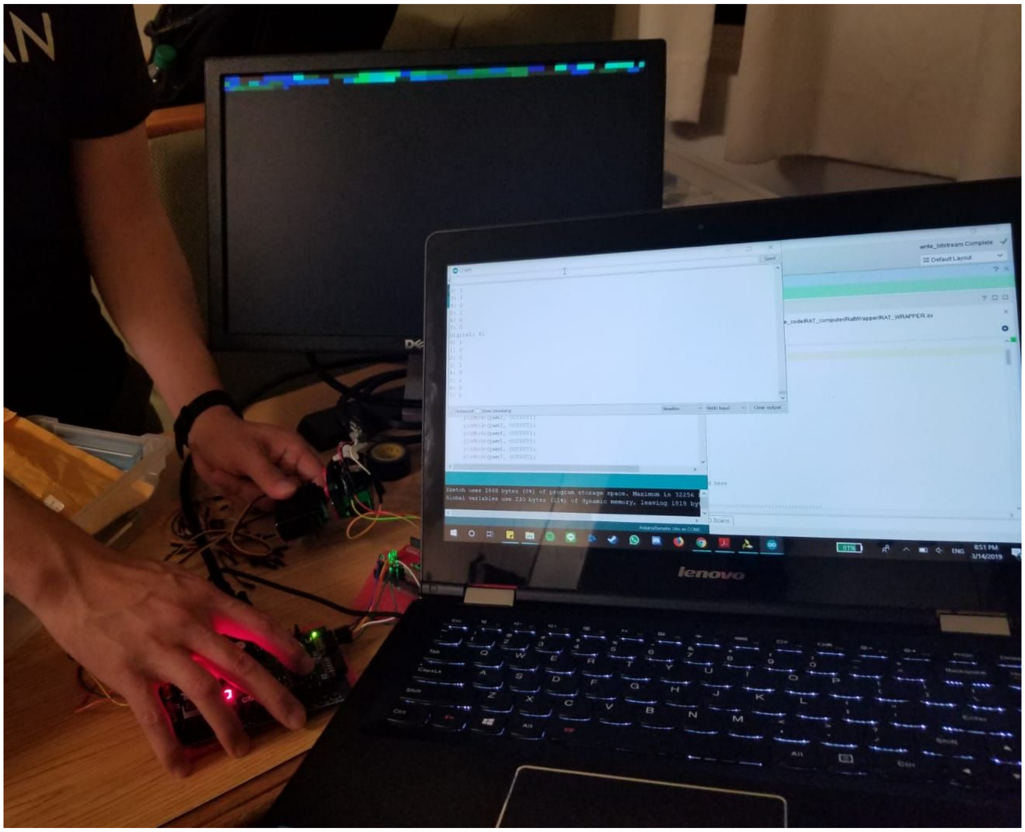
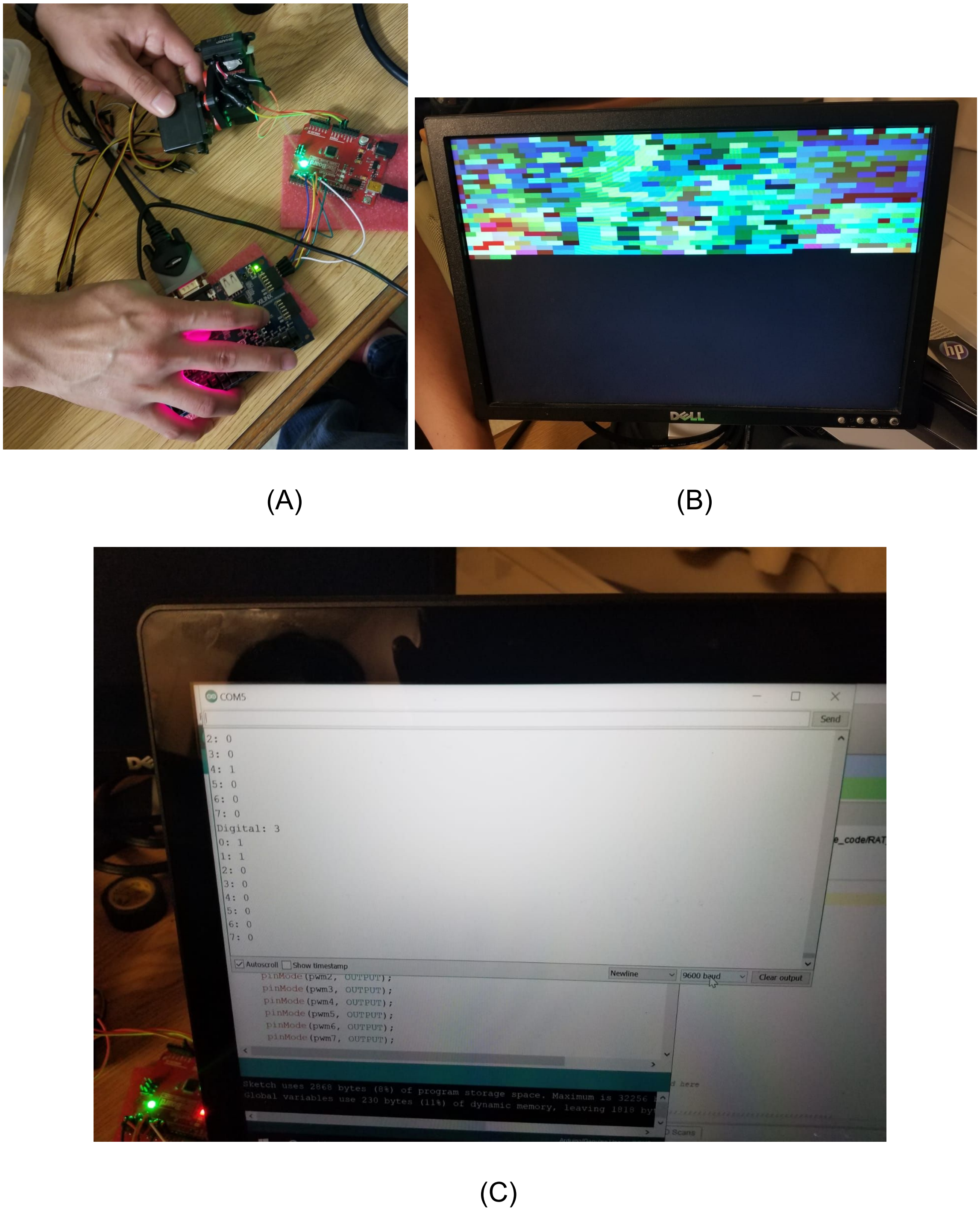


Figure 2. Complete setup of the project



(A)

(B)

COM5

```
2: 0
3: 0
4: 1
5: 0
6: 0
7: 0
Digital: 3
0: 1
1: 1
2: 0
3: 0
4: 0
5: 0
6: 0
7: 0
```

☑ Autoscroll □ Show timestamp — Newline — 9600 baud — Clear output

```
pinMode(pwm2, OUTPUT);
pinMode(pwm3, OUTPUT);
pinMode(pwm4, OUTPUT);
pinMode(pwm5, OUTPUT);
pinMode(pwm6, OUTPUT);
pinMode(pwm7, OUTPUT);
```

```
Sketch uses 2868 bytes (8%) of program storage space. Maximum is 32256 
Global variables use 230 bytes (11%) of dynamic memory, leaving 1818 by
```

(C)

Figure 3. Individual parts of the project setup. Figure 3A shows the two servo motors mounted together with the IR sensor and connected to both the Arduino Uno and the Basys Board 3. Figure 3B shows the output to the VGA monitor as the sensor is scanning through the area. Figure 3C shows the digital voltage being sent in the Basys Board 3 from the ADC

Peripheral Details

IR Sensor

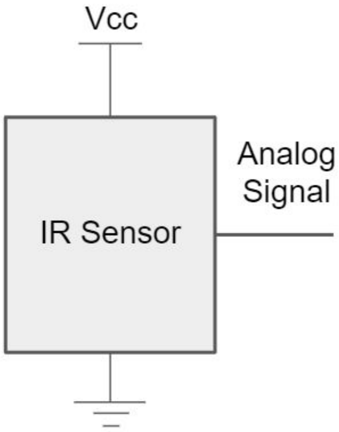


Figure 4. IR Sensor block diagram

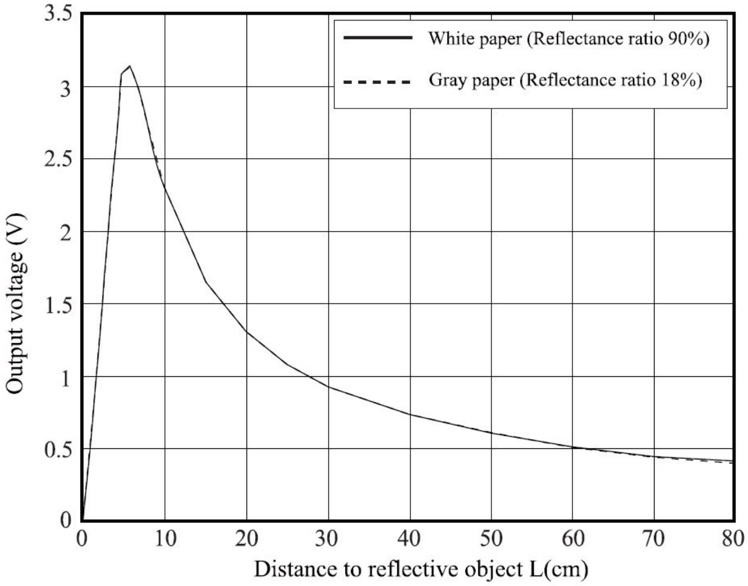


Figure 5. Distance measuring characteristic (output)

A distance measuring sensor is used to obtain an analog signal that will correspond a particular voltage to a particular distance. The effective operating range of the SHARP GP2Y0A21YK0F sensor used is from 10 cm to 80 cm. This sensor receives an input voltage Vcc between 4.5v to 5v. The output voltage ranges from -0.3v to Vcc + 0.3v.

Analog to Digital Converter

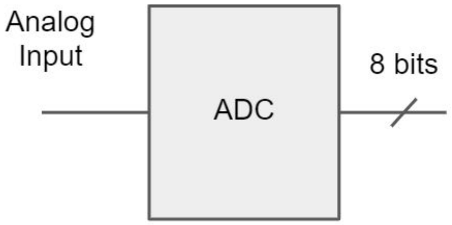


Figure 6. Analog to Digital Converter block diagram

The function of the ADC is to convert the analog signal obtained from the distance measuring sensor in to a discrete voltage value that is 8-bit long. In order to achieve this, an arduino uno is programmed and used as the ADC. Normally, the Arduino uno outputs a digital signal of 10-bits; however, the included mapping function of the arduino IDE is used to change this output into an 8-bit value. Additionally, the arduino board will be used to provide the Vcc = 5v needed to power the IR sensor.

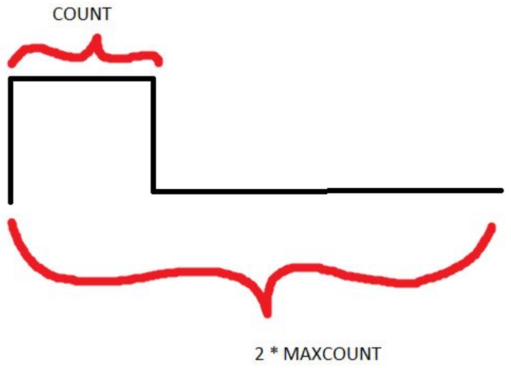
Servo Motor

We developed two system-verilog modules to control the HS-485hb servo motors. At the heart of both modules are 21-bit pwm-generators built using our calculations further below. The two modules can be distinctly described as follows:

1) **21-bit, HS-485hb Servo Driver, w/ Button Control:** provides the user of the RAT computer with direct control of a servo. Input BR allows the user to increment the position of the motor by ~1 degree in the clockwise direction, BL provides a counter clockwise reset, and BC provides a system reset of the position and internal counter.
2) **21-bit, HS-485hb Servo Driver FSM** : provides the user of the RAT computer with control of the servo via Rat Assembly OUT instructions to SERVO_PORT. The input to the FSM is a 3-bit DIN, whos bits from MSB to LSB, map to Left, Neutral, and Right states respectively. The FSM provides 5 states: Start, Hold, Neutral, Left, Right. The design of the FSM intended the following:
    a) Start: Initializes internal count and position to 0 degree. Next state is always Hold.
    b) Hold: Executes the move instruction from Left, Right, Neutral, and Start states, by enabling the pwm generator. The [2:0] DIN input determines the next state according to the enabled bits.
    c) Left: Sets position of servo to 0 degree. Next state is Hold.

d) Neutral: Sets position of servo to 90 degrees. Next state is Hold.
e) Right: Sets position of servo to ~180 degrees. Next state is Hold.

Deriving equations governing the servo motor, given an input [7:0] SELECT, consider the following:



Let us say that MAXCOUNT equals the # of 100Mhz clock cycles that occur during a single cycle of the wave above, such that the divided clock signal frequency $f_{slow\ clock\ 50} = \frac{100Mhz}{2*MAXCOUNT}$. Thus if we wanted to divide our slow clock into as many discrete intervals as possible, we would find that the interval between discrete values or $\Delta = \frac{SELECT}{MAXCOUNT} = \frac{[0,2^n]}{MAXCOUNT}$, where n is the bit width of our input [7:0] SELECT, such that n = 8. Thus $0 \leq \Delta \leq 1$.

If we wanted to create a PWM signal, any one of the possible discrete pulses would be described by $count = \Delta * 2 * MAXCOUNT = 2 * SELECT$.

Now if we wanted to describe the duration of our PWM pulse, we can say that $T_{pulse} = COUNT * T_{100MHZ} = \frac{2*SELECT}{fclock}$. Such that to generate a pulse with any given duration using a given clock signal, our input to the PWM generator would be $SELECT = \frac{Tpulse*f_{clock}}{2}$.

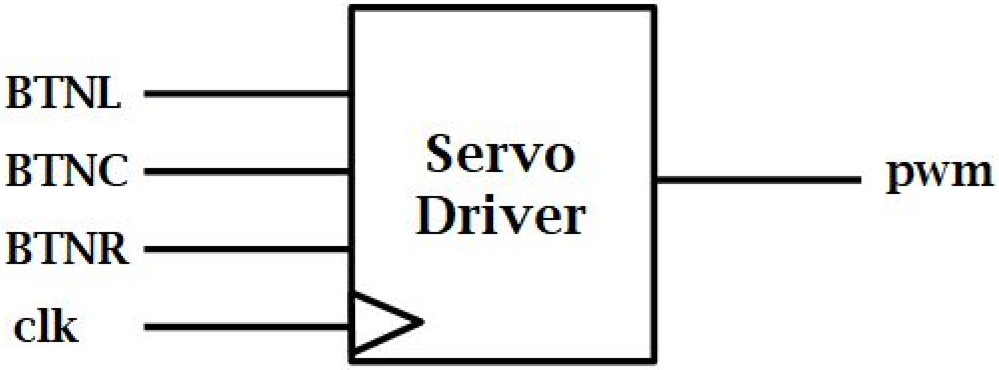
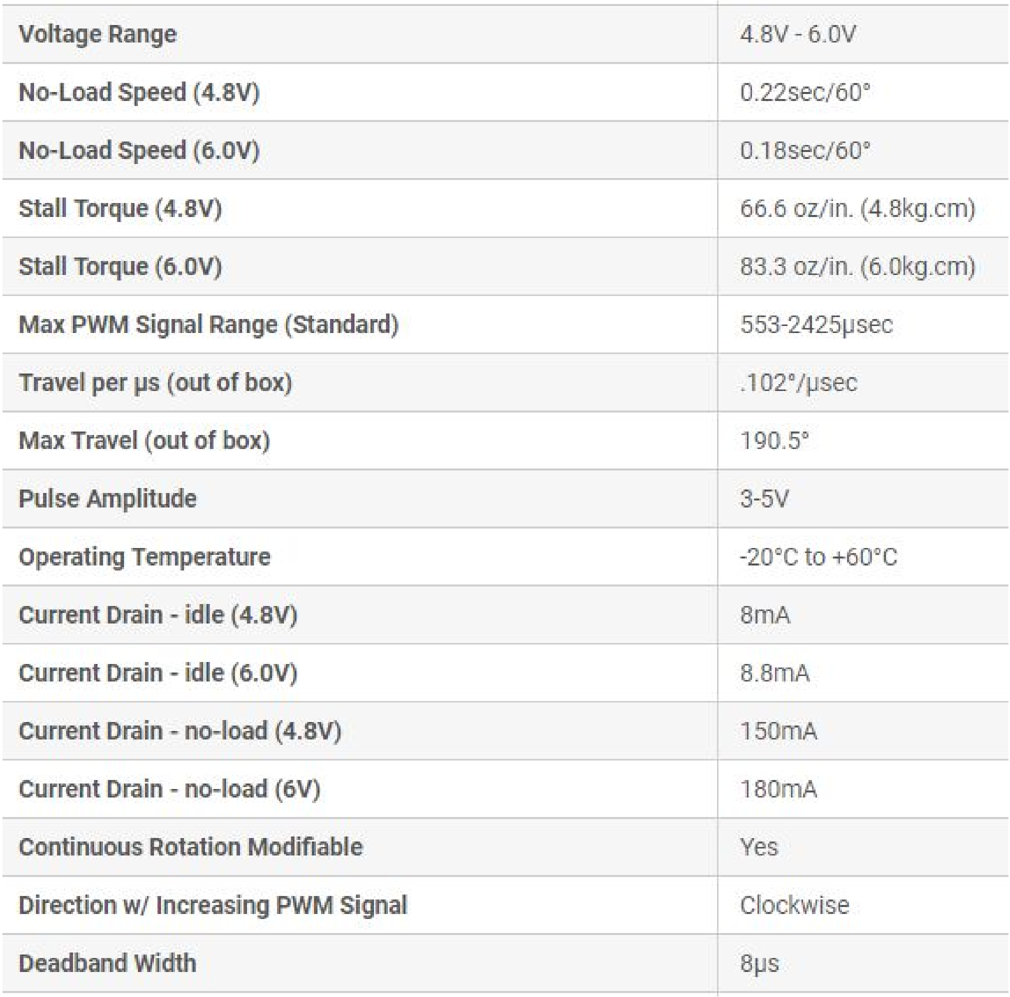


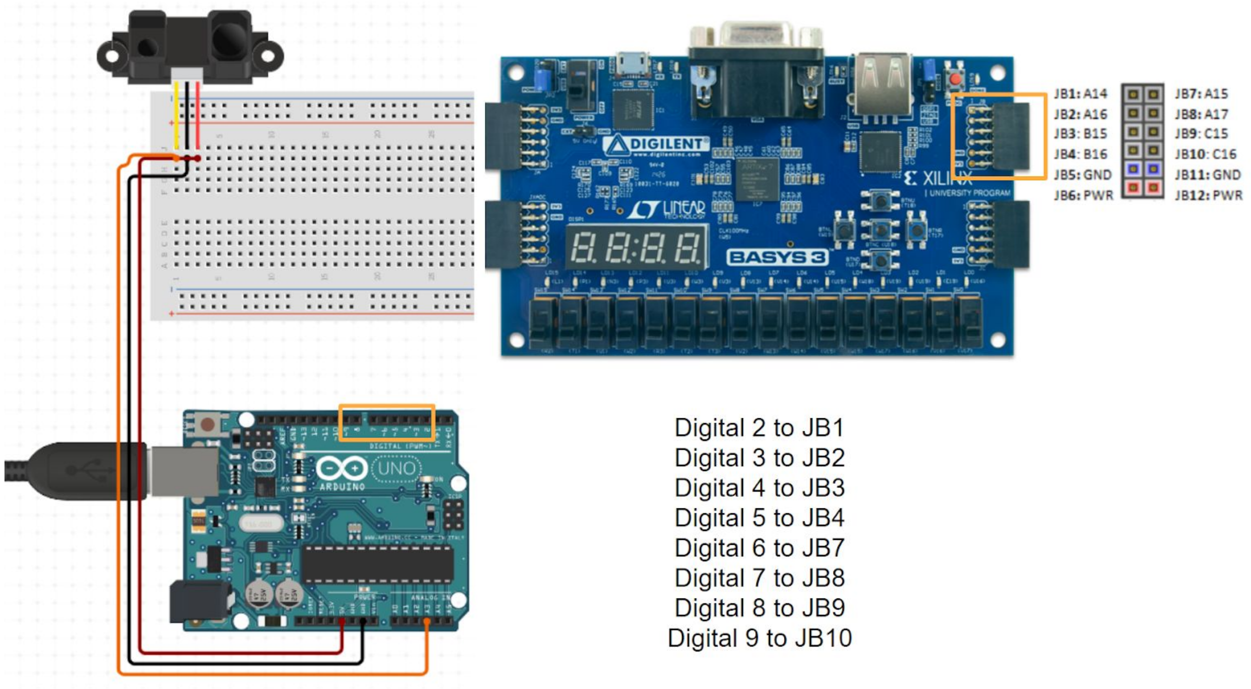
Figure 7. Servo Driver Block Diagram

Table 1. Servo Driver Specifications



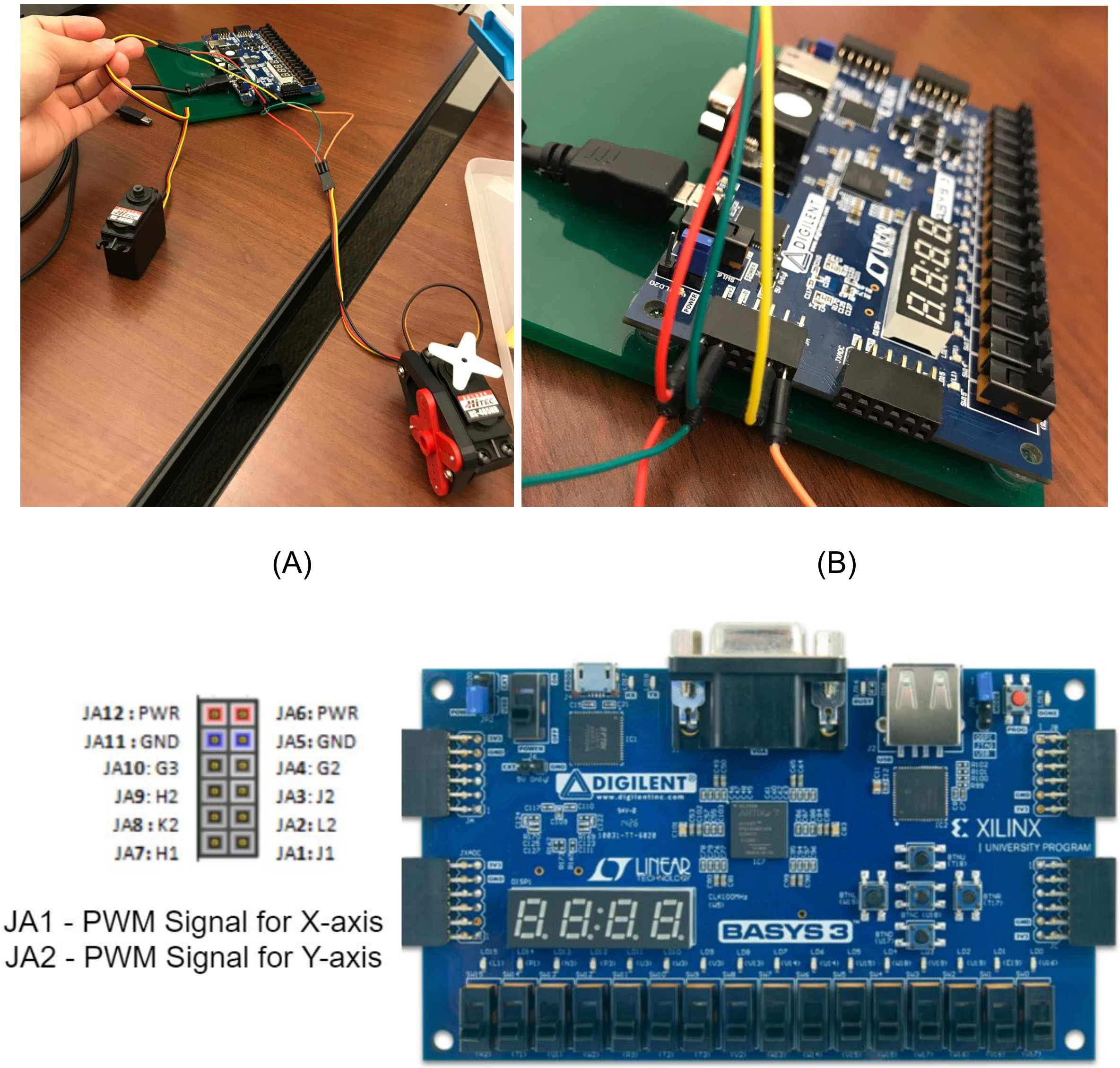
Voltage Range	4.8V - 6.0V
No-Load Speed (4.8V)	0.22sec/60°
No-Load Speed (6.0V)	0.18sec/60°
Stall Torque (4.8V)	66.6 oz/in. (4.8kg.cm)
Stall Torque (6.0V)	83.3 oz/in. (6.0kg.cm)
Max PWM Signal Range (Standard)	553-2425μsec
Travel per μs (out of box)	.102°/μsec
Max Travel (out of box)	190.5°
Pulse Amplitude	3-5V
Operating Temperature	-20°C to +60°C
Current Drain - idle (4.8V)	8mA
Current Drain - idle (6.0V)	8.8mA
Current Drain - no-load (4.8V)	150mA
Current Drain - no-load (6V)	180mA
Continuous Rotation Modifiable	Yes
Direction w/ Increasing PWM Signal	Clockwise
Deadband Width	8μs

External Circuit Peripheral



Digital 2 to JB1
Digital 3 to JB2
Digital 4 to JB3
Digital 5 to JB4
Digital 6 to JB7
Digital 7 to JB8
Digital 8 to JB9
Digital 9 to JB10

Figure 8. Analog to Digital converter connected to the IR sensor and Basys Board 3



(A)

(B)

(C)

Figure 9. Connection of the servo motors HS-485HB to Basys Board 3. Figure 9A shows the wiring of two servo motors, one used to move in the horizontal and axis and the second one is used to move in the vertical axis. Figure 9B shows the connection of both servo motors to the JA pmods of the board. Both servos connect to 3.3v and ground. Figure 9C shows the mapping of the signals to the corresponding pmod.

Software Design

An interrupt driven program was written. The objective of this program is to output to screen the pixel with the color that corresponds to the particular digital value read from the sensor once that an interrupt is pressed. On successive interrupts, the new value will be output to the pixel adjacent to the previous one. This will populate the VGA screen from left to right and will move to the next row if sufficient values are generated with successive interrupts.

In order to do this, the draw background function will be called first to populate the screen in black. After this initial call, the main program will loop continuously waiting an interrupt. Once an interrupt is triggered, the color corresponding to the current 8-bit value read from the IR sensor will be displayed in the upper left corner of the screen. Successive interrupts will paint the pixel adjacent to the right of the current pixel unless it reaches the maximum number of columns. Then, it will paint the pixel on the next row.

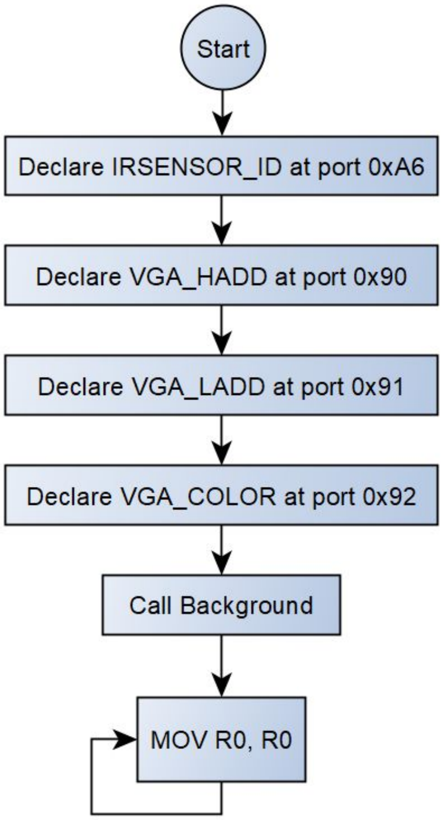

Start
Declare IRSENSOR_ID at port 0xA6
Declare VGA_HADD at port 0x90
Declare VGA_LADD at port 0x91
Declare VGA_COLOR at port 0x92
Call Background
MOV R0, R0

Figure 10. Flowchart for the main program

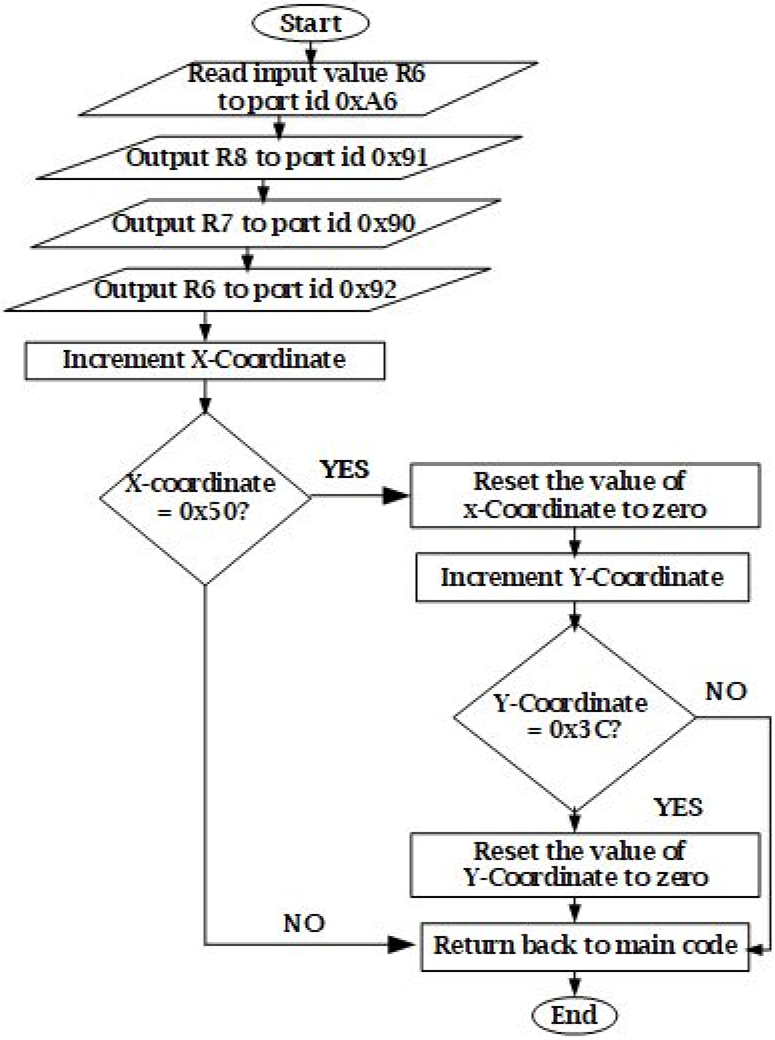

Start
Read input value R6 to port id 0xA6
Output R8 to port id 0x91
Output R7 to port id 0x90
Output R6 to port id 0x92
Increment X-Coordinate
X-coordinate = 0x50?
YES
Reset the value of x-Coordinate to zero
Increment Y-Coordinate
Y-Coordinate = 0x3C?
NO
YES
Reset the value of Y-Coordinate to zero
NO
Return back to main code
End

Figure 11. Flowchart for ISR

Appendix

Full assembly code listing with comments

IR sensor Driver

```
-------------------------------------------------------------------------------------------------------------------
----------
.CSEG
.ORG 0x10

.EQU IRSENSOR_ID = 0xA6
.EQU COUNT = 0xFF
.EQU COUNT_3 = 0x5E     ; 94
.EQU COUNT_5 = 0x9A     ; 154
.EQU COUNT_6 = 0x4B     ; 74

.EQU VGA_HADD = 0x90
.EQU VGA_LADD = 0x91
.EQU VGA_COLOR = 0x92
.EQU BG_COLOR = 0xFF            ; Background:  white
.EQU BG_COLOR_2 = 0x00          ; Background:  black
;r6 is used for color
;r7 is used for Y
;r8 is used for X

;------------------------------------------------------------------


                CALL  draw_background
                MOV   R7, 0x00
                MOV   R8, 0x00 ;column

main:           SEI
                MOV R0, R0 ;line for infinite loop
                BRN   main

;------------------------------------------------------------------
```

```
;-  Subroutine: draw_horizontal_line
;-
;-  Draws a horizontal line from (r8,r7) to (r9,r7) using color in r6
;-
;-  Parameters:
;-   r8  = starting x-coordinate
;-   r7  = y-coordinate
;-   r9  = ending x-coordinate
;-   r6  = color used for line
;-
;- Tweaked registers: r8,r9
;---------------------------------------------------------------------
draw_horizontal_line:
                    ADD    r9,0x01          ; go from r8 to r15 inclusive

draw_horiz1:
                    CALL  draw_dot
                    ADD   r8,0x01
                    CMP   r8,r9
                    BRNE draw_horiz1
                    RET
;---------------------------------------------------------------------


;----------------------------------------------------------------------
;-  Subroutine: draw_background
;-
;-  Fills the 80x60 grid with one color using successive calls to
;-  draw_horizontal_line subroutine.
;-
;-  Tweaked registers: r10,r7,r8,r9
;-----------------------------------------------------------------------
draw_background:
                    CMP            r0,0x00
                    BRNE           color2
                    MOV            r6,BG_COLOR_2            ; use default color
                    BRN            setup
Color2:                       MOV            r6,BG_COLOR_2
setup:              MOV            r10,0x00                 ; r10 keeps track of rows
start:              MOV            r7,r10                           ; load current row count
```

```
            MOV         r8,0x00                 ; restart x coordinates
            MOV         r9,0x4F                     ; set to total number of columns

            CALL        draw_horizontal_line
            ADD         r10,0x01               ; increment row count
            CMP         r10,0x3C               ; see if more rows to draw
            BRNE        start                  ; branch to draw more rows
            RET
;-----------------------------------------------------------------

;-----------------------------------------------------------------
;- Subrountine: draw_dot
;-
;- This subroutine draws a dot on the display the given coordinates:
;-
;- (X,Y) = (r8,r7)  with a color stored in r6

;-----------------------------------------------------------------
draw_dot:
            OUT         r8,VGA_LADD                 ; write bot 8 address bits to
register
            OUT         r7,VGA_HADD      ; write top 5 address bits to register
            OUT         r6,VGA_COLOR     ; write color data to frame buffer
            RET
; ----------------------------------------------------------------



ISR:        IN          R4, IRSENSOR_ID
            MOV         r6,R4            ; color from sensor
output:           OUT         r8,VGA_LADD                 ; write bot 8 address bits
to register
            OUT         r7,VGA_HADD      ; write top 5 address bits to register
            OUT         r6,VGA_COLOR     ; write color data to frame buffer


incr:       ADD         R8, 0x01         ;increment x - coordinate
            CMP         R8, 0x50
            BRNE              return                  ; if equal to 0 go to next
```

```
row
          CLC
          ADD       R7, 0x01            ;increment y - coordinate
          MOV       r8, 0x00            ;restart the value of x-coordinate by
0x00
          CMP       r7,0x3C
          BRNE           return
          MOV       r7, 0x00             ;restart the value of y-coordinate by
0x00
          CLC
return:        RETIE                          ;go back to the main code


.CSEG
.ORG 0x3FF
VECTOR:   BRN       ISR
```

Peripheral SystemVerilog & Arduino Code

21-bit, HS-485hb Servo Driver, w/ Button Control

```
--------------------------------------------------------------------------------------------------------------------
----------
`timescale 1ns / 1ps
module pwm_gen_btn(
  input CLK,        // internal clock
  input SCLK,
  input BC,         // System Reset
  input BL,         // Counter Clockwise Reset
  input BR,         // Clockwise Increment
  output logic pwm    // output duty cycle signal
  );

  localparam maxcount = 20'hFFFFF;    // numerical rep of half wave length, a
  localparam R = 20'h1D9A2;          // ~180 angular position
  localparam L = 20'h06C02;          // 0 angular position
  localparam N = 20'h122D2;          // Neutral pos 90 degrees
  localparam DELTA = 20'H001F4;      // ~1 deg increment
```

```
    logic [20:0] count = 0;
    logic [19:0] select = L;           // By default, servo set to 0 degrees

    always_ff @(posedge SCLK)
    begin
    if (BC) begin       // System reset to 0 position
        select <= L;
        end
    else if (BL) begin  // Module reset to 0 position, doesnt affect system
        select <= L;
        end
    else if (BR) begin  // Increment by 1 degree
        if(select < R)  // Limit check
            select <= select + DELTA;
        else select <= R;
        end
    end
    ////////////////////////////////////////GOOD
    always_ff @ (posedge CLK)
    begin
           if (select == 0)        // 0% duty cycle
               pwm <= 0;
           else if (select == maxcount) // % 100 duty cycle
               pwm <= 1;
           else begin
               count <= count + 1;
               if (count == 2*maxcount)
               begin
                   count <= 0;
                   pwm <= 1;
               end
               else if (count == 2*select)  // generates duty cycle
               begin
                   pwm <= 0;
               end
           end // line # 64
    end
endmodule
```

21-bit, HS-485hb Servo Driver FSM

```
`timescale 1ns / 1ps

module pwm_generator (
    input CLK,              // internal clock
    input SCLK,
    input RESET,            // System Reset
    input [2:0] DIN,        // Control Signals
    output logic [2:0] LEDS // Control Indicator Lights
    output logic pwm = 0    // output duty cycle signal
    );

    localparam maxcount = 20'hFFFFF;    // numerical rep of half wave length, a
    localparam R = 20'h1D9A2;           // ~180 angular position
    localparam L = 20'h06C02;           // 0 angular position
    localparam N = 20'h122D2;           // ~90 pos
    localparam DELTA = 20'h001F4;       // ~1 deg increment

    logic [20:0] count;
    logic [19:0] select;
    logic ENABLE = 0;   // PWM enable signal

    typedef enum        // States
    {START,
    LEFT,
    NEUTRAL,
    RIGHT,
    HOLD} STATE;

    STATE NS = HOLD;
    STATE PS = START;

    always_ff @ (posedge SCLK) // State Selection
    begin
        if (RESET == 1)
        begin
            PS <= START;
        end
        else
            PS <= NS;
    end

    always_comb // evaluate states
    begin
        case(PS)
            START:
            begin
                ENABLE = 1;
                LEDS = 0;
                select = 0;
                NS = HOLD;
            end
            HOLD:
            begin
                ENABLE = 1;
                LEDS = LEDS;
                select = select;
                case(DIN)
                    4: NS = LEFT;
                    2: NS = NEUTRAL;
                    1: NS = RIGHT;
                    default: NS = START;
                endcase
            end
            LEFT:
            begin
                ENABLE = 0;
                if(select > L)
                    select = select - DELTA;
                else select = L;
                LEDS = 3'b100;
                NS = HOLD;
            end
            NEUTRAL:
            begin
                ENABLE = 0;
```

```
         select = N;
         LEDS = 3'b010;
         NS = HOLD;
      end
      RIGHT:
      begin
         ENABLE = 0;
         if(select < R)
            select = select + DELTA;
         else select = R;
         LEDS = 3'b001;
         NS = HOLD;
      end
      default: NS = START;
   endcase // PS case
end // evaluate states

/PWM Generator
always_ff @ (posedge CLK)
begin
if(ENABLE)
  begin
     if (select == 0)        // 0% duty cycle
        pwm <= 0;
     else if (select == maxcount) // % 100 duty cycle
        pwm <= 1;
     else begin
        count <= count + 1;
        if (count == 2*maxcount)
        begin
           count <= 0;
           pwm <= 1;
        end
        else if (count == 2*select)  // generates duty cycle
        begin
           pwm <= 0;
        end
     end // line # 64
  end
```

```
    end
endmodule
```

8-bit, Analog to Digital Converter

```
--------------------------------------------------------------------------------------------------------------------
const int pwm0 = 2 ;    //Mapping digital pins 2 to 9 as pwm variables 0 (lsb) to 7 (msb)
const int pwm1 = 3;
const int pwm2 = 4;
const int pwm3 = 5;
const int pwm4 = 6;
const int pwm5 = 7;
const int pwm6 = 8;
const int pwm7 = 9;

const int adc = 0 ;   //naming pin 0 of analog input side as ‘adc’
void setup()
{
    pinMode(pwm0, OUTPUT);
    pinMode(pwm1, OUTPUT);
    pinMode(pwm2, OUTPUT);
    pinMode(pwm3, OUTPUT);
    pinMode(pwm4, OUTPUT);
    pinMode(pwm5, OUTPUT);
    pinMode(pwm6, OUTPUT);
    pinMode(pwm7, OUTPUT);

    Serial.begin(9600);
}
void loop()
{
    int adc  = analogRead(0) ;    //reading analog voltage and storing it in an integer
    adc = map(adc, 0, 1023, 0, 255);
    int bit0 = bitRead(adc, 0);
    int bit1 = bitRead(adc, 1);
    int bit2 = bitRead(adc, 2);
    int bit3 = bitRead(adc, 3);
    int bit4 = bitRead(adc, 4);
    int bit5 = bitRead(adc, 5);
    int bit6 = bitRead(adc, 6);
    int bit7 = bitRead(adc, 7);

/*
----------map funtion------------
the above function scales the output of adc,
which is 10 bit and gives values btw 0 to 1023,
in values btw 0 to 255 form analogWrite function
which only receives  values btw this range
*/
   //digital value sent to pin called pwm
    digitalWrite(pwm0, bit0);
    digitalWrite(pwm1, bit1);
    digitalWrite(pwm2, bit2);
    digitalWrite(pwm3, bit3);
    digitalWrite(pwm4, bit4);
    digitalWrite(pwm5, bit5);
    digitalWrite(pwm6, bit6);
    digitalWrite(pwm7, bit7);

    Serial.print("Digital: ");
    Serial.println(adc);

    Serial.print("0: ");
    Serial.println(bit0);

    Serial.print("1: ");
    Serial.println(bit1);

    Serial.print("2: ");
    Serial.println(bit2);

    Serial.print("3: ");
    Serial.println(bit3);

    Serial.print("4: ");
    Serial.println(bit4);

    Serial.print("5: ");
    Serial.println(bit5);
    Serial.print("6: ");
    Serial.println(bit6);

    Serial.print("7: ");
    Serial.println(bit7);
    delay(500);  //half a second delay
}
```

RAT wrapper

```
------------------------------------------------------------------------------------------------------------------
`timescale 1ns / 1ps
module RAT_WRAPPER(
   input CLK,
   input BTNC,            // System RESET
   input BTNU,            // Vert Servo Reset
   input BTND,
   input BTNL,            // Horiz Servo Reset
   input BTNR,
   input [7:0] IRSENSOR,   // IR feedback
   input [7:0] SWITCHES,
   output [7:0] LEDS,
   //output [2:0] FSM_LED,
   output [6:0] SEG,
   output [3:0] an,
   output [7:0] VGA_RGB,
   output VGA_HS,
   output VGA_VS,
   output PWM_H, PWM_V
   //output PWM_FSM
   // seven seg display annodes can be added here
   );

   // INPUT PORT IDS //////////////////////////////////////////////////////
   // Right now, the only possible inputs are the switches
   // In future labs you can add more port IDs, and you'll have
   // to add constants here for the mux below
   localparam SWITCHES_ID = 8'h20;
   localparam VGA_READ_ID = 8'h93;
   localparam IRSENSOR_ID = 8'hA6;     // Infrared Sensor

   // OUTPUT PORT IDS ///////////////////////////////////////////////////////
   // In future labs you can add more port IDs
   localparam LEDS_ID      = 8'h40;
   localparam SEG_ID       = 8'h81;
   localparam VGA_HADDR_ID = 8'h90;
   localparam VGA_LADDR_ID = 8'h91;
   localparam VGA_COLOR_ID = 8'h92;
   localparam SERVO_H_ID   = 8'h49;    // Servo FSM port

   // Signals for connecting RAT_MCU to RAT_wrapper ////////////////////////
   logic [7:0] s_output_port;
   logic [7:0] s_port_id;
   logic s_load;
   logic s_interrupt;
   logic s_reset;
   logic s_clk_50 = 1'b0;     // 50 MHz clock

   // Register definitions for output devices //////////////////////////////
   logic [7:0]     s_input_port;
   logic [7:0]     r_leds     = 8'h00;
   logic [3:0]     r_seg      = 4'h0;
   logic [2:0]     r_servo_h = 0;

   // signals for connecting VGA frambuffer Driver
   logic r_vga_we;         // write enable
   logic [12:0] r_vga_wa;  // address of framebuffer to read and write
   logic [7:0] r_vga_wd;   // pixel color data to write to framebuffer
   logic [7:0] r_vga_rd;   // pixel color data read from framebuffer

   // Declare VGA Frame Buffer//////////////////////////////////////////////
   vga_fb_driver RAT_VGA(
       .CLK(s_clk_50),
       .WA(r_vga_wa),
       .WD(r_vga_wd),
       .WE(r_vga_we),
       .RD(r_vga_rd),
       .ROUT(VGA_RGB[7:5]),
       .GOUT(VGA_RGB[4:2]),
       .BOUT(VGA_RGB[1:0]),
       .HS(VGA_HS),
       .VS(VGA_VS));

   // Declare Servo Fsm//////////////////////////////////////////////////////
   /*pwm_generator servo_fsm(
       .CLK(CLK),
       .SCLK(s_clk_50),
       .RESET(s_reset),
       .DIN(r_servo_h),
       .LEDS(FSM_LED),
       .pwm(PWM_FSM) // output duty cycle signal
       );*/

   // Declare Servos w/ Button Control///////////////////////////////////////
   logic H_BL, H_BR, V_BU, V_BD;

   debounce_one_shot pwm_L( // Reset CCW
           s_clk_50,
           BTNL,
           H_BL);

   debounce_one_shot pwm_R( // Increment CW
           s_clk_50,
           BTNR,
           H_BR);

   pwm_gen_btn H_servo(     // Horizontal servo
       .CLK(CLK),
       .SCLK(s_clk_50),
       .BC(s_reset),
       .BL(H_BL),
       .BR(H_BR),
       .pwm(PWM_H));

   debounce_one_shot pwm_U( // Reset CCW
       s_clk_50,
       BTNU,
       V_BU);

   debounce_one_shot pwm_D( // Reset CW
       s_clk_50,
       BTND,
       V_BD);

   pwm_gen_btn V_servo(    // Vertical Servo
       .CLK(CLK),
       .SCLK(s_clk_50),
       .BC(s_reset),
       .BL(V_BU),
       .BR(V_BD),
       .pwm(PWM_V));

   // Declare RAT_CPU ///////////////////////////////////////////////////////
   MCU RAT_MCU(
       .CLK(s_clk_50),
       .INTERRUPT(s_interrupt),
       .RESET(s_reset),
       .IN_PORT(s_input_port),
       .OUT_PORT(s_output_port),
       .PORT_ID(s_port_id),
       .IO_STRB(s_load));

   // Declare Sev Seg Display
   BinSseg RAT_SEV_SEG(
       r_seg,
       SEG,
       an);

   // Declare Debouncer
   debounce_one_shot RAT_reset(
       s_clk_50,
       BTNC,
       s_reset);

   // Clock Divider to create 50 MHz Clock //////////////////////////////////
```

```
    always_ff @(posedge CLK) begin
        s_clk_50 <= ~s_clk_50;
    end


    // MUX for selecting what input to read //////////////////////////////////
    // add else-if statements to read additional inputs
    always_comb begin
        if (s_port_id == SWITCHES_ID)
            s_input_port = SWITCHES;
        else if (s_port_id == VGA_READ_ID)
            s_input_port = r_vga_rd;
        else if (s_port_id == IRSENSOR_ID)
            s_input_port = IRSENSOR;
        else
            s_input_port = 8'h00;
    end

    // MUX for updating output registers /////////////////////////////////////
    // Register updates depend on rising clock edge and asserted load signal
    // add additional if statements to read the Port id and see if if matches your new
INPUT_ID
    // then assign r_input <= s_output_port
    always_ff @ (posedge CLK) begin
        r_vga_we <= 0;
        //r_pwm_en <= 0;

        if (s_load == 1'b1) begin
            if (s_port_id == LEDS_ID) begin
                r_leds <= s_output_port;
            end
            else if (s_port_id == SEG_ID) begin
                r_seg <= s_output_port;
            end
            /*else if (s_port_id == SERVO_H_ID) begin
                r_servo_h <= s_output_port[2:0];
                //r_pwm_en <= 1'b1;
            end*/
            else if (s_port_id == VGA_HADDR_ID) begin   // Y coord
            r_vga_wa[12:7] <= s_output_port[5:0];
        end
        else if (s_port_id == VGA_LADDR_ID) begin   // X coord
            r_vga_wa[6:0] <= s_output_port[6:0];
        end
        else if (s_port_id == VGA_COLOR_ID) begin   // Y coord
            r_vga_wd <= s_output_port;
            r_vga_we <= 1'b1;                       // write enable to save data to framebuffer
        end
    end
end

// Connect Signals ////////////////////////////////////////////////////////
assign s_interrupt = H_BR | V_BD; // Servo Interrupt
// Output Assignments ////////////////////////////////////////////////////
assign LEDS = r_leds;
endmodule
```