California University of PA

Dept. of Computer Science, Info Systems, and Engineering Technology

CET335 Microprocessor Interfacing

Fall 2021

= Lab Report =

Project: 180° Servo Rangefinder

Member 1: Andrew D. Bissell

Member 2: Charles Krug

Date Submitted: 12/02/2021

Project: 180° Servo Rangefinder

Date Performed: 11/11/2021 – 12/02/2021

I. OBJECTIVES

Main objectives are to use the user manuals and data sheets for the ultrasonic range finder to construct a working 180° rangefinder. Secondary objectives are to properly code, troubleshoot, and operate the code and circuit to complete the main objective.

II. PROCEDURE

First either create a flow chart, pseudo code, or take proper notes on what ports to use for the project layout. Since multiple ports need to be used for the code a proper setup is necessary to help construct the code without to many errors. Once the layout has been set, start by correctly initializing all the used ports and make sure SysTick and PLL is being used. The best way to work through the code is to set the major processes in the code as functions that can be used in blocks to help troubleshoot the code. Start with the 10µs delay for the trigger input, then work through driving the sensor output to the board. Create a function that reads the high trigger output, then set a count until it hits the low trigger output of the sensors echo. Using count from the echo's output, which is the pulse width, multiply it by a number starting at `0.01`. Now the sensor must be calibrated to get the correct centimeter output, get a measuring device, and measure out a flat surface (20cm is a good start) that leads to a large perpendicular wall or screen. Using a `printf` and the UART to output the calculated distance, keep changing the number till the sensor reads the length measured. Once the right measurement is achieved try it with different measured distances to check its accuracy. Next create an FSM that uses the data from the sensor to change its outputs. Once the sensor and code are working, set up the sensor on the servo, make sure to have the servo properly set to 0° before changing the degree. A mount for the servo would be helpful for trouble shooting since the sensor can cause it to be off balance. Set up two different sections of code for the CW and CCW movement, the code will be pretty similar, but the duty be set to different end points either +90° or -90°.

a. Equipment

- breadboard, jumper cables, red, yellow, green, and white LED
- power supply module, 9V 1A Power Supply, HC-SR04 ultrasonic ranging module
- resistors: 220 Ω x4

III. DISCUSSION

When constructing a project with no layout it is very important to create a layout to not cause confusion or issue when it comes to programming. Since there are many ports being used it can get confusing what goes where and what needs to be set to input or output. Another issue when using multiple ports some special functions like pulse width modulation are only used with port A and B pin seven and will not work properly with other pins. Another issue that was found while constructing this project is some ports can only be used for one function or it can cause issues with the function. Such as the ultrasonic ranging module, if other registers are used with the port that do not need to be set for the ultrasonic ranging module it will cause issues with sensor. Sometimes the trigger won't set, or the output won't be ran, only simple input and outputs can be put ran on the same port of the sensor.

IV. CONCLUSION

When constructing standalone projects, it is good to have a plan and a layout set in mind, the use of pseudo code, and flowcharts add the project. Port layouts and notes will also aid the construction of the projects, make sure to research what is necessary for the ports so that there are no compatibility issues such as pulse width modulation.

V. APPENDIX

A. Screen Captures

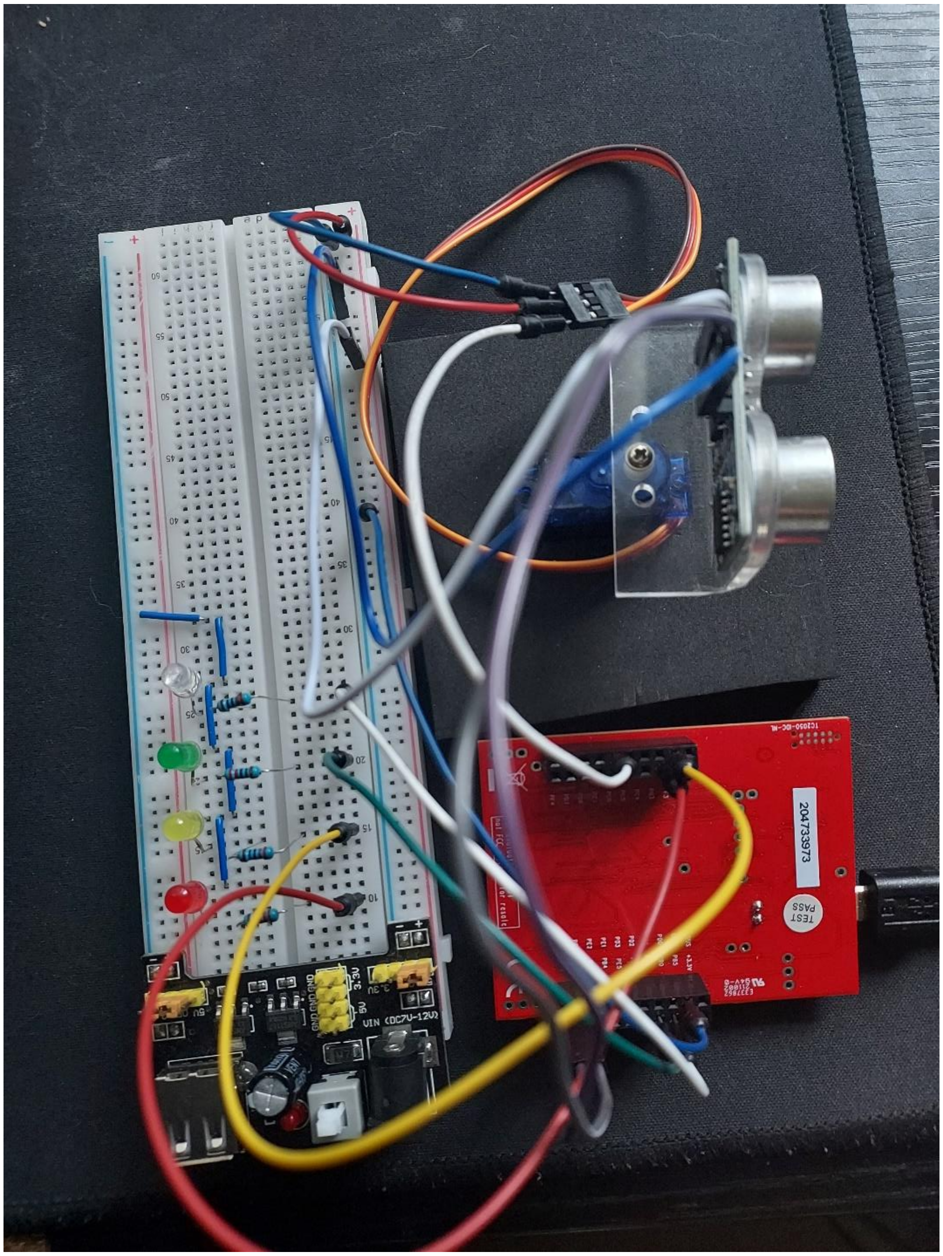

Figure 1: Circuit layout for ultrasonic ranging module, servo, and FSM output.

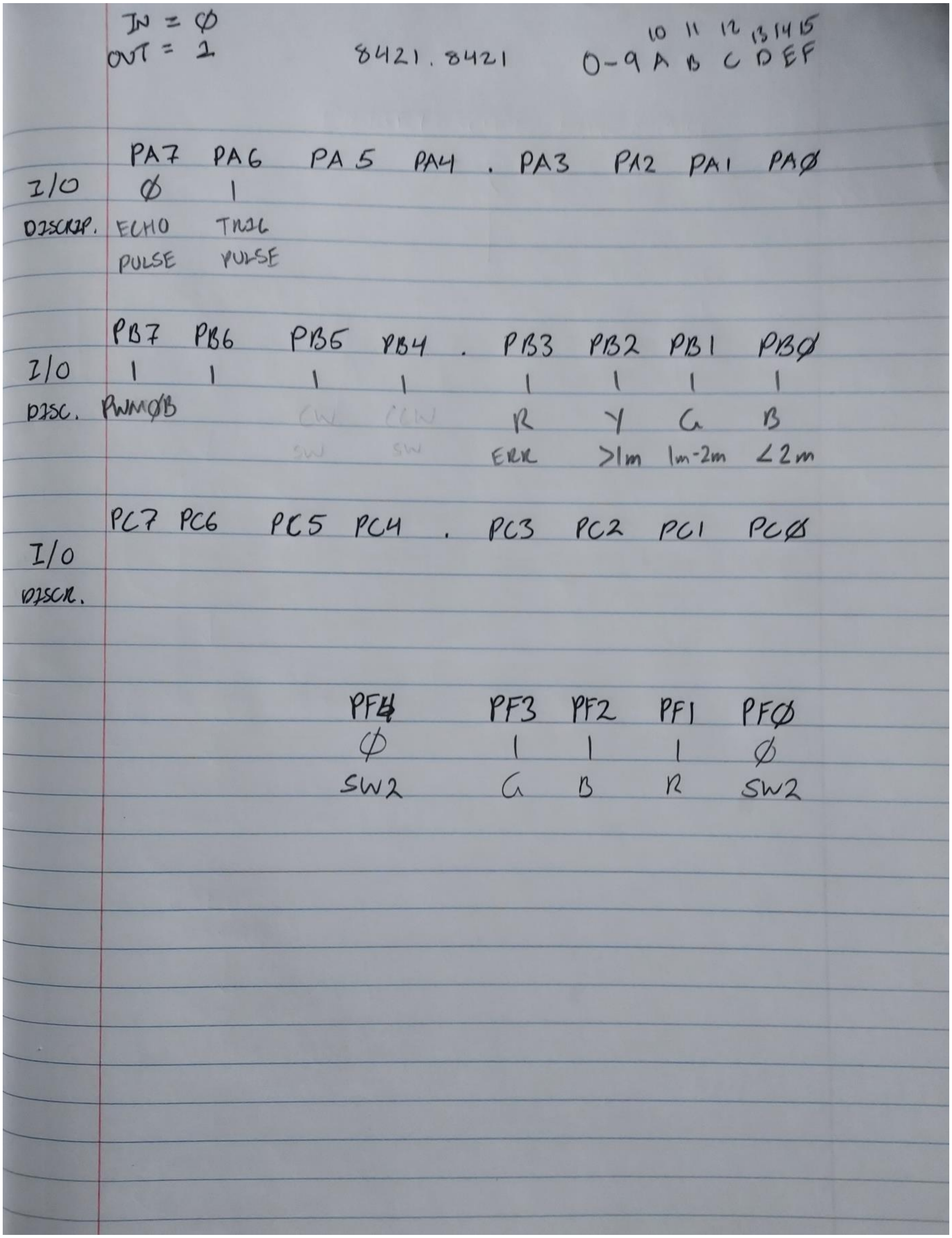

Figure 2: Notes on port layouts for input/output

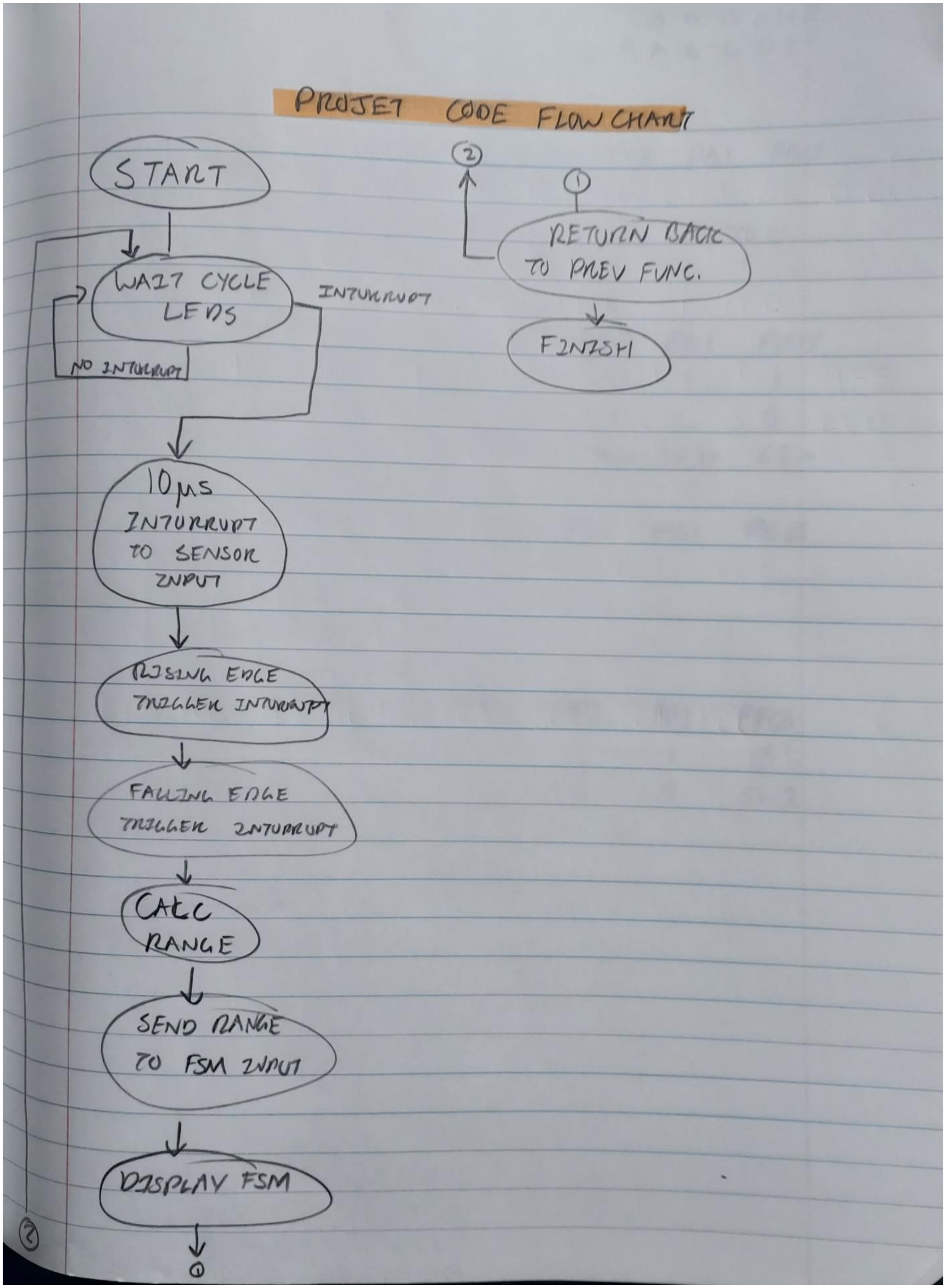

Figure 3: Project Flowchart

```c
// ******************************************** 0. Documentation Section ********************************
// main.c
// Runs on TM4C123
// Project: Range Finder using Interrupts/Timers
// Authors: Andrew D. Bissell, Charles Krug
// Date: November 11/2021 - November 28/2021

// Off Board hardware:
// Breadboard, jumper cables, red led, yellow led, green led, white led
// Power Supply Module, 9V 1A Power Supply, HC-SR04 Ultrasonic Ranging Module
// Resistors: 220ohm x4
// ---------------------------------------------------------------------------------------
// ****************************** 1. Pre-processor Directives Section ********************
// ---------------------------------------------------------------------------------------
//#include <stdio.h>              // standard C library
//#include "uart.h"               // Uart for outputing to user
#include "tm4c123gh6pm.h"       // Magical library of amazing #define's

#define Trigger                  (*((volatile unsigned long *)0x40004100)) // PortA Trigger Output
#define Echo                     (*((volatile unsigned long *)0x40004200)) // PortA Echo Input
#define LED                      (*((volatile unsigned long *)0x40025038)) // PortF LED's (on-board)
#define SW1                      (*((volatile unsigned long *)0x40025040)) // PortF SW1 (on-board)
#define SW2                      (*((volatile unsigned long *)0x40025004)) // PortF SW2 (on-board)
#define LIGHT                    (*((volatile unsigned long *)0x4000503C)) // PortA for LEDs all outputs
// ********************************** a. STATES *******************************************
#define RLED  0 // Red LED if error or out of range
#define YLED  1 // Yellow LED if between 1 cm and 50 cm (0.01 m - 0.5 m)
#define GLED  2 // Green LED if between 50 cm and 75 cm (1 m - 2 m)
#define WLED  3 // White LED if above 75 cm
// ---------------------------------------------------------------------------------------
// ****************************** 2. Declarations Section *********************************
// ---------------------------------------------------------------------------------------
// ******************************** a. FSM Declarations ***********************************
typedef struct StateStructure {
  unsigned long Out;          // 4-bit pattern to output
  unsigned long Time;         // delay in 1ms units
  unsigned long Next[4];
}State;
typedef const struct State STyp;

State FSM[4] =
{//Outp,D, {  00,   01,   10,   11,}},
  {0x08,5, {RLED, YLED, GLED, WLED,}}, // RLED 0
  {0x04,5, {RLED, YLED, GLED, WLED,}}, // YLED 1
  {0x02,5, {RLED, YLED, GLED, WLED,}}, // GLED 2
  {0x01,5, {RLED, YLED, GLED, WLED,}}, // WLED 3
//          ERR,>50cm,>75cm,<75cm,}},
};
```

Figure 4: Documentation, Pre-processor directives, declarations sections

```c
// ****************************** b. Function Prototypes ********************************
void PortA_Init(void);                            // Initializes PortA
void PortB_Init(void);                            // Initializes PortB
void PortF_Init(void);                            // Initializes PortF
void PWM0B_Init(unsigned int period, unsigned int duty);  // Pulse Width Modulation for the motor
void EnableInterrupts(void);                      // Allow interrupts in the program to occur
void SysTick_Init(void);                          // Initializes Systick Timer
void SysTick_Wait(unsigned long delay);           // Initializes Systick Delay using busy wait
                                                  // The delay parameter is in units of the core clock.
                                                  // (units of 20 nsec for 50 MHz clock)
void SysTick_Wait1ms(unsigned long delay);        // Initializes Systick Delay using busy wait (1ms delay)
                                                  // This assumes 50 MHz system clock.
void SysTick_Wait10us(unsigned long delay);       // Initializes Systick Delay using busy wait (1us delay)
                                                  // This assumes 50 MHz system clock.
void PLL_INIT(void);                              // Initializes the phase lock loop.
void PWM0A_Init(unsigned int period, unsigned int duty); // PWM for motor
void Timer0A_DelayMicroSec(int time);             // Timer with interrupt for a 10 µs delay
int Timer0A_periodCapture(void);                  // Captures the rising edge then falling edge of a
                                                  //  pulse to get is pulse width.
unsigned int PulseWidth (void);                   // Captures pulse width with software programming
unsigned int Distance_Input(unsigned int);        // Used set the input based on the measured distance
// ****************************** c. Global Variables ***********************************
unsigned long S;          // For current state
unsigned long Input;      // For next state
unsigned long j;          // Used for moving the motor by user input
unsigned int Distance;    // Final Calc of distance
unsigned int echo_time;   // Initial data from sensor
unsigned int average;     // To take a 3 data average of data
unsigned int i;           // Loop count
```

Figure 5: Function prototypes and Global variable sections

```c
// ---------------------------------------------------------------------------------------
// ****************************** 3. Subroutines Section ********************************
// ---------------------------------------------------------------------------------------
int main(void){
//  UART_Init();                          // Initialize UART for printing
  SysTick_Init();                         // Initialize SysTick
  PLL_INIT();                             // Initialize PLL
  PortA_Init();                           // Initialize PortA
  PortB_Init();                           // Initialize PortB
  PortF_Init();                           // Initialize PortF
  EnableInterrupts();                     // Allow interrupts
  PWM0B_Init(50000,3150);                 // Initialize the PWM and send the servo to 0°
  while(1){
    LIGHT = FSM[S].Out;                   // Set output for FSM before new cycle
    LED = 0x08;              // GREEN LED FLAG
// ****************************** a. Clockwise Movement **********************************
    if((GPIO_PORTF_DATA_R&0x11)==0x01){
      SysTick_Wait1ms(25);
      LED = 0x0E;
      while((GPIO_PORTF_DATA_R&0x11)==0x01){
        if(PWM0_0_CMPB_R < 5550){
          while((PWM0_0_CMPB_R < 5550) && ((GPIO_PORTF_DATA_R&0x11)==0x01)){
            j += 1;
            SysTick_Wait1ms(50);
            PWM0_0_CMPB_R += j;
          }
        }
        else{
          while((PWM0_0_CMPB_R > 5550) && ((GPIO_PORTF_DATA_R&0x11)==0x01)){
            j += 1;
            SysTick_Wait1ms(50);
            PWM0_0_CMPB_R -= j;
          }
        }
      }
    }
    LED = 0x00;
```

Figure 6: Subroutine and clockwise movement sections

```c
// *************************** b. Counter-Clockwise Movement ******************************
    if(((GPIO_PORTF_DATA_R&0x11)==0x10) && ((GPIO_PORTF_DATA_R&0x11)==0x10)){
      SysTick_Wait1ms(25);
      LED = 0x0C;
      while((GPIO_PORTF_DATA_R&0x11)==0x10){
        if(PWM0_0_CMPB_R < 1275){
          while(PWM0_0_CMPB_R < 1275){
            j += 1;
            SysTick_Wait1ms(50);
            PWM0_0_CMPB_R += j;
          }
        }
        else{
          while((PWM0_0_CMPB_R > 1275) && ((GPIO_PORTF_DATA_R&0x11)==0x10)){
            j += 1;
            SysTick_Wait1ms(50);
            PWM0_0_CMPB_R -= j;
          }
        }
      }
    }
    LED = 0x00;
// ****************************** c. Toggle Range Finder **********************************
    if((GPIO_PORTF_DATA_R&0x11)==0x00){
      average = 0;                            // Reset average

      for(i=0;i<3;i++){                       // Three samples, gets average to have better data.
        LED = 0x00;       // CLEAR LED FLAG
        echo_time = 0;                        // Reset echo
        Trigger |= 0x40;                      // Set trigger high
        SysTick_Wait10us(1);                  // 10 µs delay using systick
//        Timer0A_DelayMicroSec(1);             // 10 µs delay using timer/interrupt
        Trigger &= ~0x40;                     // Set trigger low
//        average += Timer0A_periodCapture(); // Get pulse width from sensor using timer/interrupts.
        average += PulseWidth();              // Get pulse width from sensor using software programming.
      }
      average /= 3;                           // After three samples take the average
      LED = 0x02;         // RED LED FLAG
// ****************************** d. Calculate Range(cm) **********************************
      Distance = average*0.01175;             // Average is then multipled to get correct cm output
// ************************************* e. FSM *******************************************
//      printf("\n Distance: %d cm\n",Distance);  // Output to display of the distance in cm
      Input = Distance_Input(Distance);       // Sets the input using the distance
      SysTick_Wait1ms(500);                   // Delay
      SysTick_Wait1ms(FSM[S].Time);           // Get delay of the FSM
      S = FSM[S].Next[Input];                 // Gets the next state
    }
  }
}
```

Figure 7: Counter-Clockwise movement, Rangefinder toggle, calculate range, and FSM sections

```c
// ---------------------------------------------------------------------------------------
// ************************** 4. Port Initializations Functions **************************
// ---------------------------------------------------------------------------------------
void PortF_Init(void){ volatile unsigned long delay;
  SYSCTL_RCGC2_R |=      0x00000020; // 1) F clock
  delay = SYSCTL_RCGC2_R;            //    delay
  GPIO_PORTF_LOCK_R =    0x4C4F434B; // 2) unlock PortF PF0(SW2)
  GPIO_PORTF_CR_R =      0x1F;       //    allow changes to PF4-0
  GPIO_PORTF_AMSEL_R =   0x00;       // 3) disable analog function
  GPIO_PORTF_PCTL_R =    0x00000000; // 4) GPIO clear bit PCTL
  GPIO_PORTF_DIR_R =     0x0E;       // 5) Inputs PF4(SW1),PF0(SW2)
                                     //    Outputs PF3(Green),PF2(White),PF1(Red) LEDs
  GPIO_PORTF_AFSEL_R =   0x00;       // 6) no alternate function
  GPIO_PORTF_PUR_R =     0x11;       //    enable pullup resistors on PF4,PF0
  GPIO_PORTF_DEN_R =     0x1F;       // 7) enable digital pins PF4-PF0
}

void PortB_Init(void){ volatile unsigned long delay;
  SYSCTL_RCGC2_R |=      0x00000002; // 1) B clock
  delay = SYSCTL_RCGC2_R;            //    delay
  GPIO_PORTB_LOCK_R =    0x4C4F434B; // 2) unlock PortF PF0(SW2)
  GPIO_PORTB_CR_R =      0xFF;       //    allow changes to PB7 - PB0
  GPIO_PORTB_AMSEL_R =   0x00;       // 3) disable analog function
  GPIO_PORTB_PCTL_R =    0x00000000; // 4) GPIO clear bit PCTL
  GPIO_PORTB_DIR_R =     0xFF;       // 5) PB7 - PB0 Outputs
  GPIO_PORTB_AFSEL_R =   0x00;       // 6) no alternate function
  GPIO_PORTB_PUR_R =     0x30;       //    enable pullup resistors on PB5,PB4
  GPIO_PORTB_DEN_R =     0xFF;       // 7) enable digital pins PB7-PB0
}

void PortA_Init(void){
  volatile unsigned long delay;
  SYSCTL_RCGC2_R |=      0x00000001;   // 1) A clock
  delay = SYSCTL_RCGC2_R;              //    delay
  GPIO_PORTA_DIR_R =       0x7F;       // 5) 0 = output, 1 = input
                                       //     0     1     1     1  .  1     1     1     1
                                       //    PA7   PA6   PA5   PA4 . PA3   PA2   PA1   PA0
  GPIO_PORTA_DEN_R =       0xFF;       // 7) enable digital pins PB7-PB0
// ********************************** a. Unused Registers **********************************
//  GPIO_PORTB_LOCK_R =    0x4C4F434B; // 2) unlock PortA
//  GPIO_PORTB_CR_R =      0xFF;       //    allow changes to PA7-PA0
//  GPIO_PORTB_AMSEL_R =   0x00;       // 3) disable analog function
//  GPIO_PORTB_PCTL_R =    0x00000000; // 4) GPIO clear bit PCTL
//  GPIO_PORTB_AFSEL_R |=  0xC0;       // 6) PA6 use alternate function
//  GPIO_PORTB_PUR_R =     0x00;       //    disable pullup resistor
//  GPIO_PORTB_DEN_R =     0xFF;       // 7) enable digital pins PA7-PA0
}
```

Figure 8: Port init functions

```c
// ---------------------------------------------------------------------------------------
// ******************************** 5. Systick/PLL Functions ******************************
// ---------------------------------------------------------------------------------------
// Initialize SysTick with busy wait running at bus clock.
void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;                    // disable SysTick during setup
  NVIC_ST_RELOAD_R = NVIC_ST_RELOAD_M;  // maximum reload value
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
                                        // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC;
}

// Time delay using busy wait.
// The delay parameter is in units of the core clock. (units of 20 nsec for 50 MHz clock)
void SysTick_Wait(unsigned long delay){
  volatile unsigned long elapsedTime;
  unsigned long startTime = NVIC_ST_CURRENT_R;
  do{
    elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
  }
  while(elapsedTime <= delay);
}

// Time delay using busy wait.
// This assumes 50 MHz system clock.
void SysTick_Wait10us(unsigned long delay){
  unsigned long i;
  for(i=0; i<delay; i++){
    SysTick_Wait(750);                     // wait 10µs (assumes 50 MHz clock)
  }
}

// Time delay using busy wait.
// This assumes 50 MHz system clock.
void SysTick_Wait1ms(unsigned long delay){
  unsigned long i;
  for(i=0; i<delay; i++){
    SysTick_Wait(80000);                   // wait 1ms (assumes 50 MHz clock)
  }
}

void PLL_INIT(void){
SYSCTL_RCC2_R |=  0x80000000;                // 0) Use RCC2, USERCC2
SYSCTL_RCC2_R |=  0x00000800;                // 1) bypass PLL while initializing, BYPASS2, PLL bypass
SYSCTL_RCC_R = (SYSCTL_RCC_R &~0x000007C0)   // 2) select the crystal value and oscillator source
                                             //    clear XTAL field, bits 10-6
+ 0x00000540;                                // 10101, configure for 16 MHz crystal
SYSCTL_RCC2_R &= ~0x00000070;                // configure for main oscillator source
SYSCTL_RCC2_R &= ~0x00002000;                // 3) activate PLL by clearing PWRDN
SYSCTL_RCC2_R |= 0x40000000;                 // 4) set the desired system divider, use 400 MHz PLL
SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~ 0x1FC00000) // clear system clock divider
+ (4<<22);                                   // configure for 80 MHz clock
while((SYSCTL_RIS_R&0x00000040)==0){};       // 5) wait for the PLL to lock by polling PLLLRIS, wait for PLLRIS bit
SYSCTL_RCC2_R &= ~0x00000800;                // 6) enable use of PLL by clearing BYPASS
}
```

Figure 9: SysTick and PLL functions

```
// ---------------------------------------------------------------------------------------
// ********************************** 6. Timer/Interrupts ********************************
// ---------------------------------------------------------------------------------------
// Microsecond delay using one-shot mode and prescalar
void Timer0A_DelayMicroSec(int time){
//   int ms = 4000;                                 // Millisecond
  int us = 4;                                     // Microsecond
  SYSCTL_RCGCTIMER_R |=               2;          // 1. Enable clock to Timer Block 0
  TIMER1_CTL_R =                      0;          // 2. Disable Timer before initialization
  TIMER1_CFG_R =                   0x04;          // 3. 16-bit option
  TIMER1_TAMR_R =                  0x01;          // 4. One-shot mode and down counter
  TIMER1_TAILR_R =        us * time - 1;          // 5. Timer A interval load value register
  TIMER1_TAPR_R =                 4 - 1;          //    Timer A prescaler 16MHz/4 = 4MHz
  TIMER1_ICR_R =                    0x1;          // 6. Clear the TimerA timeout flag
  TIMER1_CTL_R |=                  0x01;          // 7. Enable Timer A after initalization
  while((TIMER1_RIS_R & 0x1)==0);                 // 8. Wait for TimerA timeout flag to set
}

// Initialize Timer0A in edge-time mode to caputer rising edges.
// Input pin of Timer0A is PB6.
void Timer0Capture_Init(void){
  SYSCTL_RCGCTIMER_R |=            1; // 1. Enable clock to Timer Block 0
//   SYSCTL_RCGC2_R  |=                2; // 2. Enable clock to PortB
//   GPIO_PORTB_DIR_R &=         ~0x40; // 3. Make PB6 an input pin
//   GPIO_PORTB_DEN_R |=          0x40; // 4. Make PB6 as digital pin
  GPIO_PORTB_AFSEL_R |=       0x40; // 5. Use PB6 alternate funciton'
  GPIO_PORTB_PCTL_R &= ~0x0F000000; // 6. Configure PB6 for T0CCP0
  GPIO_PORTB_PCTL_R |=  0x07000000;
  TIMER0_CTL_R &=                ~1; // 7. Disable Timer0A during setup
  TIMER0_CFG_R =                  4; // 8. 16-bit timer mode
  TIMER0_TAMR_R =              0x17; // 9. Up-count, edge-time, capture mode
  TIMER0_CTL_R |=              0x0C; // 10. Capture either edge
  TIMER0_CTL_R |=                 1; // 11. Enable Timer0A
}

// Captures two consecutive rising edges of a periodic signal from Timer Block 0
//   Timer A and returns the time difference
int Timer0A_periodCapture(void){
  // capture the first rising edge
  int lastEdge, thisEdge;
  TIMER0_ICR_R = 4;                    // Clear Timer0A capture flag
  while((TIMER0_RIS_R & 4) == 0); // Wait till capture
  lastEdge = TIMER0_TAR_R;             // Save the timestamp

  // capture the second rising edge
  TIMER0_ICR_R = 4;                    //Clear Timer0A capture flag
  while((TIMER0_RIS_R & 4) == 0); // Wait till capture
  thisEdge = TIMER0_TAR_R;             // Save the timestamp

  return(thisEdge - lastEdge) & 0x00FFFFFF; // Return the time difference
}
```

Figure 10: Timer and Interrupts functions (UNUSED)

```c
// ---------------------------------------------------------------------------------------
// ******************************** 7. Pulse Width Modulation *****************************
// ---------------------------------------------------------------------------------------
void PWM0B_Init(unsigned int period, unsigned int duty){  // Output on PB7/M0PWM1
  volatile unsigned long delay;
  SYSCTL_RCGC0_R |=       0x00100000;             // 1) Activate PWM0
  SYSCTL_RCGCGPIO_R |=    0x02;                   // 2) Activate port B
  delay = SYSCTL_RCGCGPIO_R;                      // Allow time to finish activating
  GPIO_PORTB_AFSEL_R |=   0x80;                   // 3) Enable alt funtion on PB7
  GPIO_PORTB_PCTL_R &=   ~0xF0000000;             // 4) Configure PB7 as M0PWM1
  GPIO_PORTB_PCTL_R |=    0x40000000;
  GPIO_PORTB_AMSEL_R &=  ~0x80;                   // Disable analog funtionality on PB7
  GPIO_PORTB_DEN_R |=     0x80;                   // Enable digital I/O on PB7
  SYSCTL_RCC_R &=        ~(0x000E0000);           // 5) Clearing first
  SYSCTL_RCC_R |=         (0x00080000);           // The writing in the 19:17 with PWMDIV 0x3 configure for /32 divider
  SYSCTL_RCC_R |=         0x00100000;             // Use PWM divider
  PWM0_0_CTL_R =          0x00;                   // 6) Re-loading down-counting mode
  PWM0_0_GENB_R =         (0x00000C00|0x00000008); // 0xC8 in lesson   // PB7 goes low on LOAD, PB7 goes high on CMPB down
  PWM0_0_LOAD_R =         period - 1;             // 7) Cycles needed to count down to 0
  PWM0_0_CMPB_R =         duty - 1;               // 9) Count value when output rises
  PWM0_0_CTL_R |=         0x00000001;             // 10) Start PWM0
  PWM0_ENABLE_R |=        0x00000002;             // 11) Enable PB7/M0PWM1
}
// ---------------------------------------------------------------------------------------
// ******************************** 8. Main Program Functions *****************************
// ---------------------------------------------------------------------------------------
// Gets the pulse width by waiting on the L2H/H2L transitions of the input port.
// Returns the time it takes for the entire L2H/H2L transtition.
unsigned int PulseWidth (void){
  while((GPIO_PORTA_DATA_R&0x80)==0);       // Search for high transition
  while((GPIO_PORTA_DATA_R&0x80)==0x80){    // Search for low transition
    SysTick_Wait(3);
    echo_time++;                            // Count echo_time
  }
  return echo_time;
}

// Sets the right output for the FSM using the distance from the sensor.
// Takes in the Distance and returns the state the FSM needs to be.
unsigned int Distance_Input(unsigned int Distance){
  if((Distance > 400) || (Distance < 1)){  // Red LED
    return 0;
  }
  else if(Distance <= 50){                 // Yellow LED
    return 1;
  }
  else if(Distance <= 75){                 // Green LED
    return 2;
  }
  else if(Distance > 75){                  // White LED
    return 3;
  }
    return 0;
}
```

Figure 11: Pulse Width Modulation and other functions used in the main program