CECS 346 Fall 2020 Project 2

A Simple Smart House

By

Len Quach

12/18/2020

Design a simplified smart house use a stepper motor to simulate a garage door, an onboard push button to simulate a garage door button, three onboard LEDs to indicate garage door status, and an obstacle avoidance sensor to detect any object approaching/leaving the house.

Introduction

The purpose of this project is to learn how to use a stepper motor, obstacle avoidance sensor and how to build an embedded system with timer interrupt and external interrupt.

We will use the three LED to indicate garage door status: green indicates that the garage door is closed, blue indicates that the garage door is open, and flashing red indicates the garage door is moving. If stepper motor pointing downward indicates that the garage door is closed, pointing upward indicates that the garage door is open.

The system starts with the green LED on, stepper motor pointing downwards. For open garage door operation: the green LED will be turned off, the red LED will start flashing with a frequency of 4Hz (0.25s on and 0.25s off), and the stepper motor will start rotating to the opposite direction. The red LED will keep flashing until the stepper motor finishes 180 degree rotating pointing upward. After that, the blue LED will be turned on.

For close garage door operation: the blue LED will be turned off, the red LED will start flashing with a frequency of 4Hz (0.25s on and 0.25s off), and the stepper motor will start rotating to the opposite direction. The red LED will keep flashing until the stepper motor finishes 180 degree rotating pointing downward. After that, the green LED will be turned on.

When the obstacle avoidance sensor detects an object moving inside a 15cm distance, an open garage door operation will be triggered if the door is closed. If the door has already been open, no operation is needed for both the LED and the garage door. When the obstacle avoidance sensor detects an obstacle moving out of 15cm distance, a close garage door operation will be

triggered if the door is open. If the door has already been closed, no operation is needed for both the LED and the garage door.

The onboard push button sw1 will be used to toggle garage door open and close operation. When the garage door is open, touch the push button will trigger a close garage door operation; when the garage door is closed, touch the push button will trigger an open garage door operation.

Operation

https://youtu.be/294qz3u3mEM

Theory

1) When the garage door is closed, move an object from out of range to within range to observe an open garage door operation.

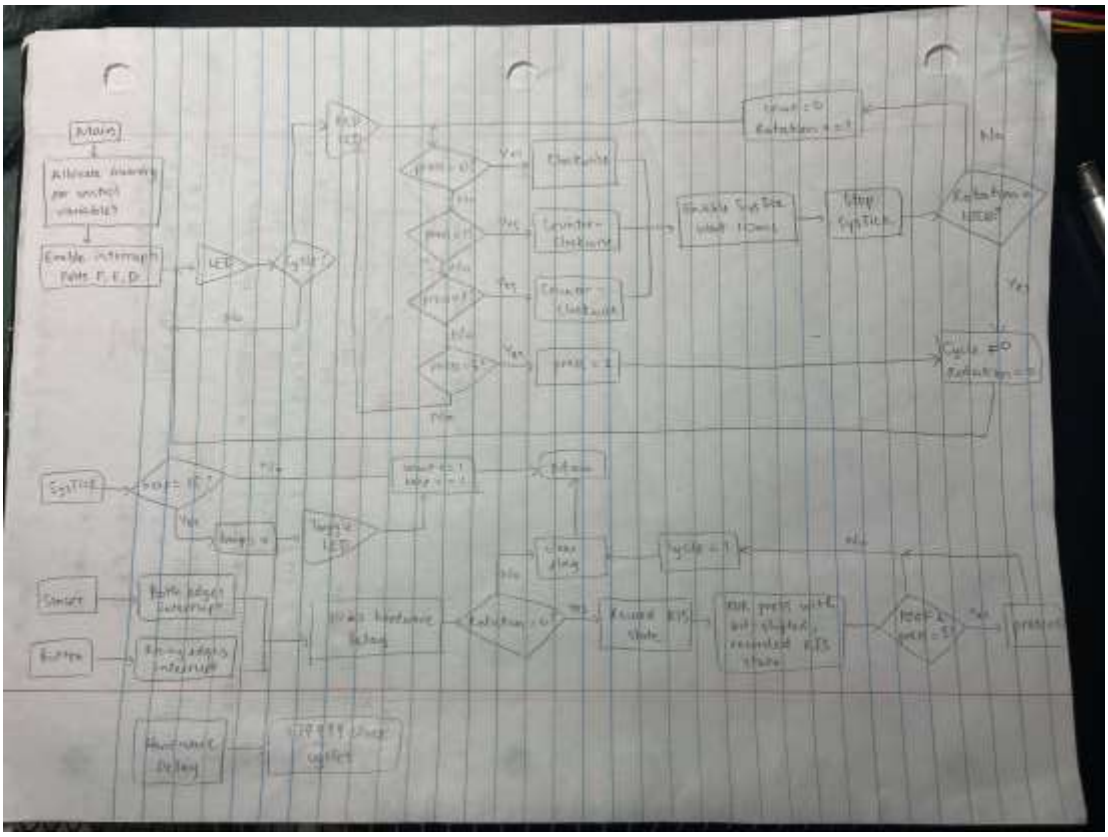
2) When the garage door is open, move an object from within range to out of range to observe a close garage door operation.

3) When the garage door is closed, press sw1 to observe an open garage door operation.

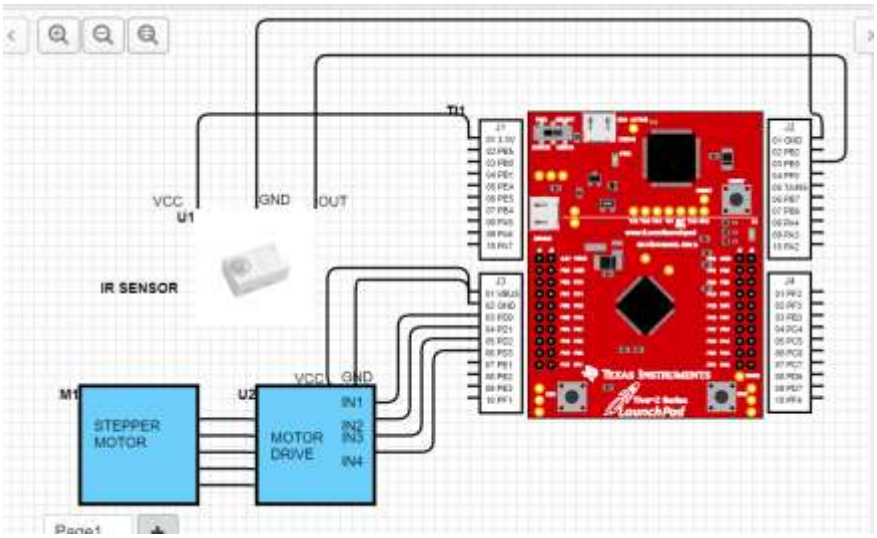
4) When the garage door is open, press sw1 to observe a close garage door operation.

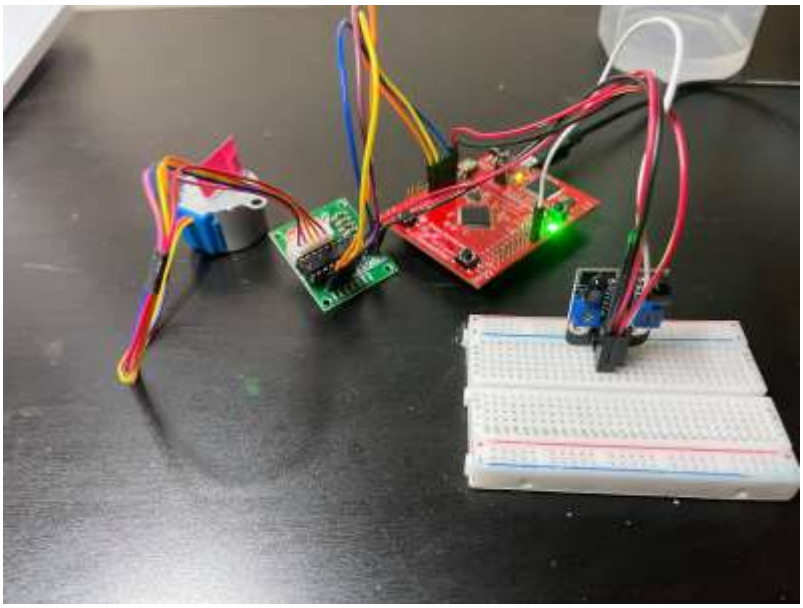
5) When the garage door is closed, press sw1 to open garage door, then move an object close to the sensor to observe nothing happen to the stepper motor and onboard LED. After that, move the object away from within range to out of range to observe a close garage door operation.

6) When the garage door is closed, move an object from out of range to within range to open garage door, then press sw1 to observe a close garage door operation.



Hardware design





Software design

```
 7
 8  // LaunchPad built-in hardware
 9  // SW1 left switch is negative logic PF4 on the Launchpad
10  // red LED connected to PF1 on the Launchpad
11  // blue LED connected to PF2 on the Launchpad
12  // green LED connected to PF3 on the Launchpad
13  // sensor is connected to PE0 on the Launchpad
14  // PD3 connected to driver for stepper motor coil A/In1
15  // PD2 connected to driver for stepper motor coil A'/In2
16  // PD1 connected to driver for stepper motor coil B/In3
17  // PD0 connected to driver for stepper motor coil B'/In4
18
19  // 1. Pre-processor Directives Section
20  // Constant declarations to access port registers using
21  // symbolic names instead of addresses
22  #define LIGHT                   (*((volatile unsigned long *)0x40025038)) // bits 3-1
23  #define GPIO_PORTF_DIR_R        (*((volatile unsigned long *)0x40025400))
24  #define GPIO_PORTF_AFSEL_R      (*((volatile unsigned long *)0x40025420))
25  #define GPIO_PORTF_PUR_R        (*((volatile unsigned long *)0x40025510))
26  #define GPIO_PORTF_DEN_R        (*((volatile unsigned long *)0x4002551C))
27  #define GPIO_PORTF_CR_R         (*((volatile unsigned long *)0x40025524))
28  #define GPIO_PORTF_AMSEL_R      (*((volatile unsigned long *)0x40025528))
29  #define GPIO_PORTF_PCTL_R       (*((volatile unsigned long *)0x4002552C))
30
31  #define GPIO_PORTF_RIS_R        (*((volatile unsigned long *)0x40025414))
32  #define GPIO_PORTF_IS_R         (*((volatile unsigned long *)0x40025404))
33  #define GPIO_PORTF_IBE_R        (*((volatile unsigned long *)0x40025408))
34  #define GPIO_PORTF_IEV_R        (*((volatile unsigned long *)0x4002540C))
35  #define GPIO_PORTF_IM_R         (*((volatile unsigned long *)0x40025410))
36  #define GPIO_PORTF_ICR_R        (*((volatile unsigned long *)0x4002541C))
37
38  #define SENSOR                  (*((volatile unsigned long *)0x400243FC)) // bit 0
```

```
38  #define SENSOR                  (*((volatile unsigned long *)0x400243FC)) // bit 0
39  #define GPIO_PORTE_DIR_R        (*((volatile unsigned long *)0x40024400))
40  #define GPIO_PORTE_AFSEL_R      (*((volatile unsigned long *)0x40024420))
41  #define GPIO_PORTE_DEN_R        (*((volatile unsigned long *)0x4002451C))
42  #define GPIO_PORTE_AMSEL_R      (*((volatile unsigned long *)0x40024528))
43  #define GPIO_PORTE_PCTL_R       (*((volatile unsigned long *)0x4002452C))
44  #define GPIO_PORTE_RIS_R        (*((volatile unsigned long *)0x40024414))
45  #define GPIO_PORTE_PUR_R        (*((volatile unsigned long *)0x40024510))
46  #define GPIO_PORTE_IS_R         (*((volatile unsigned long *)0x40024404))
47  #define GPIO_PORTE_IBE_R        (*((volatile unsigned long *)0x40024408))
48  #define GPIO_PORTE_IEV_R        (*((volatile unsigned long *)0x4002440C))
49  #define GPIO_PORTE_ICR_R        (*((volatile unsigned long *)0x4002441C))
50  #define GPIO_PORTE_IM_R         (*((volatile unsigned long *)0x40024410))
51
52  #define STEPPER                  (*((volatile unsigned long *)0x400073FC)) //bits 3-0
53  #define GPIO_PORTD_DIR_R         (*((volatile unsigned long *)0x40007400))
54  #define GPIO_PORTD_DEN_R         (*((volatile unsigned long *)0x4000751C))
55  #define GPIO_PORTD_AMSEL_R       (*((volatile unsigned long *)0x40007528))
56  #define GPIO_PORTD_AFSEL_R       (*((volatile unsigned long *)0x40007420))
57  #define GPIO_PORTD_PCTL_R        (*((volatile unsigned long *)0x4000752C))
58  #define GPIO_PORTD_DR8R_R        (*((volatile unsigned long *)0x40007040))
59
60  #define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))
61  #define SYSCTL_RCGC2_GPIOE      0x00000010  // port E Clock Gating Control
62  #define SYSCTL_RCGC2_GPIOF      0x00000020  // port F Clock Gating Control
63  #define SYSCTL_RCGC2_GPIOD      0x00000008  // port D Clock Gating Control
64
65  #define NVIC_EN0_R              (*((volatile unsigned long *)0xE000E100))  // IRQ 0 to 31 Set Enable Register
66  #define NVIC_PRI1_R             (*((volatile unsigned long *)0xE000E404))  // IRQ 5 to 7 Priority Register
67  #define NVIC_PRI7_R             (*((volatile unsigned long *)0xE000E41C))  // IRQ 28 to 31 Priority Register
68  #define NVIC_SYS_PRI3_R         (*((volatile unsigned long *)0xE000ED20))  // Sys. Handlers 12 to 15 Priority
69
```

```
70  #define NVIC_ST_CTRL_R          (*((volatile unsigned long *)0xE000E010))
71  #define NVIC_ST_RELOAD_R        (*((volatile unsigned long *)0xE000E014))
72  #define NVIC_ST_CURRENT_R       (*((volatile unsigned long *)0xE000E018))
73  #define NVIC_ST_CTRL_COUNT      0x00010000  // Count flag
74  #define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
75  #define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
76  #define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode
77  #define NVIC_ST_RELOAD_M        0x00FFFFFF  // Counter load value
78
79  #define GREEN   0x08
80  #define RED     0x02
81  #define BLUE    0x04
82
83
84  // 2. Declarations Section
85
86  //   Function Prototypes
87  void PortF_Init(void);
88  void PortE_Init(void);
89  void Stepper(void);
90  void Stepper_Init(void);
91  void SysTick_Init(unsigned long period);
92  extern void EnableInterrupts(void);  // Enable interrupts
93
94  unsigned char s; // current state
95  unsigned press_sw = 0;
96  unsigned sensor_out = 0;
97  volatile unsigned long FallingEdges = 0;
98  volatile unsigned long BothEdges = 0;
99  volatile unsigned long Counts = 1;
100 volatile unsigned long open = 0;
101 volatile unsigned long close = 0;
```

```
101 volatile unsigned long close = 0;
102
103 struct State{
104   unsigned int Out;     // Output
105   unsigned int Next[2]; // CW/CCW
106 };
107 typedef const struct State StateType;
108
109 #define clockwise 0         // Next index
110 #define counterclockwise 1 // Next index
111 StateType fsm[4]={
112   // index 0: state 0, state goes form 0 to 3, output 1100,
113   // if next state index is 0: move clockwise, next state for clockwise movement is 1
114   // CW state transition is: 0->1->2->3 then repeat
115   // CCW state transtion is: 0->3->2->1 then repeat
116   {12,{1,3}}, // state 0, PD3-0:1100
117   { 6,{2,0}}, // state 1, PD3-0:0110
118   { 3,{3,1}}, // state 2, PD3-0:0011
119   { 9,{0,2}}  // state 3, PD3-0:1001
120 };
121
122
123 // Interrupt service routine
124 // Executed every 62.5ns*(period)
125 void SysTick_Handler(void){
126   Stepper();
127   Counts = Counts + 1;
128   if (Counts % 100 == 0) // LED flash every 0.25s at 100Hz
129     LIGHT ^= RED;       //toggle PF1
130   if (Counts == 1000)    // turn 180 degrees: 0.18 degree for each step
131     Counts = 0;
132 }
```

```
void GPIOPortF_Handler(void){ //PF0
    for(unsigned long time=0; time<727240;time++){}
    GPIO_PORTF_ICR_R = 0x10;   // acknowledge flag4
    press_sw = 1;              //sw1
    FallingEdges = FallingEdges + 1; //falling edges interrupt
}
void GPIOPortE_Handler(void){ //PE0
    for(unsigned long time=0; time<727240;time++){}
    GPIO_PORTE_ICR_R |= 0x01;  // acknowledge flag0
    sensor_out = 1;            // sensor detects
    BothEdges = BothEdges + 1; //both edges interrupt
}

// 3. Subroutines Section
// MAIN: Mandatory for a C Program to be executable
int main(void){

  PortE_Init();        // Call initialization of port E
  PortF_Init();        // Call initialization of port F
  Stepper_Init();      // Call initialization of port D
  EnableInterrupts();

  LIGHT = GREEN;

  while(1){

      if (sensor_out == 1) {        //sensor detects
          sensor_out = 0;
          if ((SENSOR == 0x00) && (LIGHT == GREEN)) { //obtacles moving into
```

```
  while(1){

      if (sensor_out == 1) {        //sensor detects
          sensor_out = 0;
          if ((SENSOR == 0x00) && (LIGHT == GREEN)) { //obtacles moving into
              LIGHT = 0;            //turn off LED
              open = 1;             //door is open
              SysTick_Init(40000);  //waiting 0.25s -> 0.25s/62.5ns = 4,000,000
              while (Counts) {}     //does nothing
              NVIC_ST_CTRL_R = 0x00;//turn off SysTic timer
              LIGHT = BLUE;         //LED is blue
              Counts = 1;           //reset count
              open = 0;             //clear
              sensor_out= 0;        //clear sensor
          }

          if((SENSOR == 0x01) && (LIGHT == BLUE)) {  //obtacles moving away
              LIGHT = 0;
              close = 1;            //door is closed
              SysTick_Init(40000);
              while (Counts) {}
              NVIC_ST_CTRL_R = 0x00;//turn off SysTic timer
              LIGHT = GREEN;        //LED is greeen
              Counts = 1;
              close = 0;
              sensor_out = 0;
          }
      }

      if ((LIGHT == GREEN) && press_sw) { //sw1 is pressed
          LIGHT = 0;
          open = 1;             //door is open
```

```
    }

    if ((LIGHT == GREEN) && press_sw) { //sw1 is pressed
        LIGHT = 0;
        open = 1;               //door is open
        SysTick_Init(40000);
        while (Counts) {}
        NVIC_ST_CTRL_R = 0x00; //turn off SysTic timer
        LIGHT = BLUE;           //LED is blue
        Counts = 1;
        open = 0;
        press_sw = 0;
    }

    if ((LIGHT == BLUE) && press_sw) { //sw1 is pressed
        LIGHT = 0;
        close = 1;              //door is closed
        SysTick_Init(40000);
        while (Counts) {}
        NVIC_ST_CTRL_R = 0x00; //turn off SysTic timer
        LIGHT = GREEN;          //LED is green
        Counts = 1;
        open = 0;
        press_sw = 0;
    }
  }
}
```

```
// Subroutine to initialize port F pins for input and output
// Inputs: PF4 for SW1
// Outputs: PF3,PF2,PF1 to the LEDs
// Notes: These four pins are connected to hardware on the LaunchPad
void PortF_Init(void){
  SYSCTL_RCGC2_R |= 0x00000020;       // activate F clock
  while ((SYSCTL_RCGC2_R&0x00000020)!=0x00000020){} // wait for the clock to be ready

  GPIO_PORTF_CR_R |= 0x1E;            // allow changes to PF4
  GPIO_PORTF_AMSEL_R &= ~0x1E;        // disable analog function
  GPIO_PORTF_PCTL_R &= ~0x000FFFF0;   // GPIO clear bit PCTL
  GPIO_PORTF_DIR_R &= ~0x10;          // PF4 input
  GPIO_PORTF_DIR_R |= 0x0E;           // PF3,PF2,PF1 output
  GPIO_PORTF_AFSEL_R &= ~0x1E;        // no alternate function
  GPIO_PORTF_PUR_R |= 0x10;           // enable pullup resistors on PF4
  GPIO_PORTF_DEN_R |= 0x1E;           // enable digital pins PF4-PF0

  GPIO_PORTF_IS_R &= ~0x10;     // (d) PF4 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x10;    //     PF4 is not both edges
  GPIO_PORTF_IEV_R &= ~0x10;    //     PF4 falling edge event
  GPIO_PORTF_ICR_R = 0x10;      // (e) clear flag4
  GPIO_PORTF_IM_R |= 0x10;      // (f) arm interrupt on PF4
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00A00000; // (g) priority 5
  NVIC_EN0_R = 0x40000000;      // (h) enable interrupt 30 in NVIC
  EnableInterrupts();           // (i) Clears the I bit
}
```

```
//Input: PE0 for SENSOR
void PortE_Init(void){
  SYSCTL_RCGC2_R |= 0x00000010; //Activate Port E clocks
  while ((SYSCTL_RCGC2_R & 0x00000010) != 0x00000010){}
  GPIO_PORTE_AMSEL_R &= ~0x01; // Disable analog function on PE0
  GPIO_PORTE_PCTL_R &= ~0x0000000F; // Enable regular GPIO
  GPIO_PORTE_DIR_R &= ~0x01;   // Inputs on PE0
  GPIO_PORTE_AFSEL_R &= ~0x01; // Regular function on PE0
  GPIO_PORTE_DEN_R |= 0x01;    // Enable digital on PE0

  GPIO_PORTE_IS_R &= ~0x01;     // (d) PE0 is edge-sensitive 1111 1110
  GPIO_PORTE_IBE_R |= 0x01;    //     PE0 is both edges 0000 0001
  GPIO_PORTE_ICR_R |= 0x01;      // (e) clear flag3
  GPIO_PORTE_IM_R |= 0x01;      // (f) arm interrupt on PE0
  NVIC_PRI1_R = (NVIC_PRI1_R&0xFFFFFF1F)|0x00A00000; // (g) priority 5
  NVIC_EN0_R |= 0x00000010;      // (h) enable interrupt 4 in NVIC    (portE: bit 4 -> 0001_0000 -> 0x10)
}

//Turn stepper motor CW/CWW
void Stepper(void){
  if(open){ //door is open
    s = fsm[s].Next[clockwise]; // clock wise circular
    STEPPER = fsm[s].Out; // step motor
  }
  else if(close){ //door is closed
    s = fsm[s].Next[counterclockwise]; // counter clock wise circular
    STEPPER = fsm[s].Out; // step motor
  }
}
```

```
// Output: D3-D0 for STEPPER MOTOR
void Stepper_Init(void){
  SYSCTL_RCGC2_R |= 0x08; // 1) activate port D
  while ((SYSCTL_RCGC2_R&0x00000008)!=0x00000008){} // wait for the clock to be ready
  s = 0;
                                  // 2) no need to unlock PD3-0
  GPIO_PORTD_AMSEL_R &= ~0x0F;      // 3) disable analog functionality on PD3-0
  GPIO_PORTD_PCTL_R &= ~0x0000FFFF; // 4) GPIO configure PD3-0 as GPIO
  GPIO_PORTD_DIR_R |= 0x0F;    // 5) make PD3-0 out
  GPIO_PORTD_AFSEL_R &= ~0x0F;// 6) disable alt funct on PD3-0
  GPIO_PORTD_DR8R_R |= 0x0F;  // enable 8 mA drive
  GPIO_PORTD_DEN_R |= 0x0F;    // 7) enable digital I/O on PD3-0
}

// ***************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 62.5ns (assuming 16 MHz clock)
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
void SysTick_Init(unsigned long period){
  NVIC_ST_CTRL_R = 0;         // disable SysTick during setup
  NVIC_ST_RELOAD_R = period-1;// reload value
  NVIC_ST_CURRENT_R = 0;      // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
                              // enable SysTick with core clock and interrupts
  NVIC_ST_CTRL_R = 0x07;
  EnableInterrupts();
}
```

Conclusion

This project of designing a simplified smart house with some automatic controls helps me learn how to use a stepper motor and obstacle avoidance sensor, and learn how to build an embedded system with timer interrupt and external interrupt. The project is similar to lab4, but we need to

add in a stepper motor. I got a lot of bugs while debugging the code, but then I was able to fix them.