EE4930 Advanced Embedded Systems

Section 011, Winter 2019/20

Draven Schilling

1-16-20

Lab 4 – Finite State Machine

# Objectives

The objective for this lab was to provide a use case for a finite state machine solution. This lab gave us experience using function pointers, enums, and structures in order to successfully implement a finite state machine lookup table using the efficient and robust function pointer implementation. Another objective was to continue learning more about the ADC by using a “scan” approach which allows conversion from multiple inputs in a round robin style.

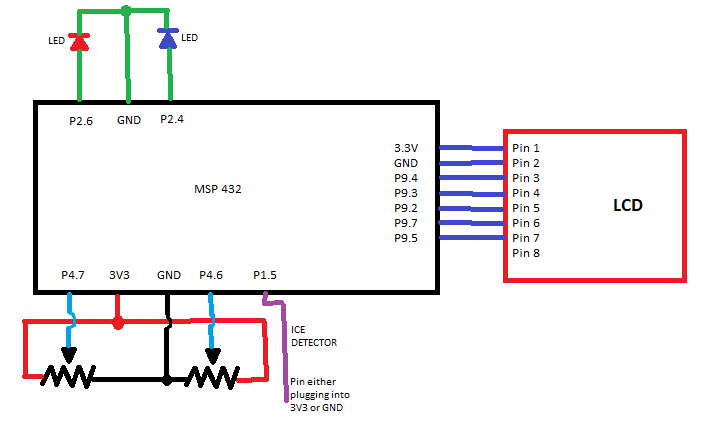
# Description

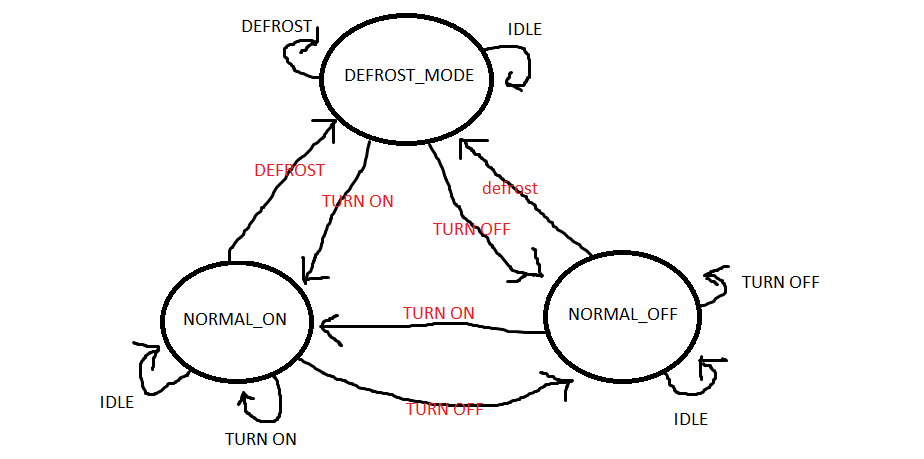
In this lab, I started by configuring 3 GPIO inputs (two for the pushbuttons, and 1 for the ice detector). I next configured the ADC into scan mode and set it up with two separate inputs for each temperature and humidity sensors. I then setup the two GPIO outputs for each the fan and compressor. Once all the pin mappings and interrupt configuration were setup, I then began work on the state machine. Using the outline provided by Dr. Widder, I created a lookup table and methods to run for each state. Finally, I setup the input logic to pass to the state machine and allowed it to run continuously in the main loop. The required inputs were also displayed on the LCD.

# Conclusions

The lab went well overall. The only real problem I encountered was when setting up the ADC, I initially wanted to use the continuous conversion bit so that I could enable the ADC and then forget about it. Unfortunately, I think it was interrupting too quickly and skipping conversions on the second channel; I’m not sure why. But ultimately, I ended up not using continuous conversion mode and instead started a new conversion every loop cycle to slow down the ADC trigger rate. Aside from that, I didn’t encounter many other problems. I was familiar with state machines from embedded 2 and likewise multiple files, so the state machine implementation was fairly smooth. Overall, I feel like this lab was a good learning experience for both ADC scan mode and state machines.

# Attachments





**Source Code:**

/\*

\* fsm.h

\*

\* Created on: Jan 13, 2020

\* Author: schillingdl

\*/

**#ifndef** FSM\_H\_

**#define** FSM\_H\_

**typedef** **enum** {

*NORMAL\_OFF*,

*NORMAL\_ON*,

*DEFROST\_MODE*

}state;

**typedef** **enum** {

*IDLE*,

*TURN\_ON*,

*TURN\_OFF*,

*THERE\_IS\_ICE*

}event;

**typedef** **void** (\*fp)(**void**);

**typedef** **struct**{

state nextstate;

fp action;

} stateElement;

// turns the fan and compressor off

**void** **outputsOff**();

//turns the fan and compressor on

**void** **outputsOn**();

//turns the fan on and compressor off (defrost mode)

**void** **defrost**();

// function to handle the state machine update

// accepts current state and current event, returns next state

state **stateUpdate** (state current, event input);

**#endif** /\* FSM\_H\_ \*/

/\*\*

\* fsm.c

\*/

**#ifndef** FSM\_C\_

**#define** FSM\_C\_

**#include** "fsm.h"

**#include** "msp.h"

// create state table array

stateElement stateTable [3][4] = {

{{*NORMAL\_OFF*, outputsOff},{*NORMAL\_ON*, outputsOn},{*NORMAL\_OFF*, outputsOff},{*DEFROST\_MODE*, defrost}},

{{*NORMAL\_ON*, outputsOn},{*NORMAL\_ON*, outputsOn},{*NORMAL\_OFF*, outputsOff},{*DEFROST\_MODE*, defrost}},

{{*NORMAL\_OFF*, outputsOff},{*NORMAL\_ON*, outputsOn},{*NORMAL\_OFF*, outputsOff},{*DEFROST\_MODE*, defrost}},

};

// function to handle the state machine update

// accepts current state and current event, returns next state

state **stateUpdate** (state current, event input){

stateElement currentstate = stateTable[current][input];

// run the proper action function

(\*currentstate.action)();

//return next state info

**return** currentstate.nextstate;

}

// turns the fan and compressor off

**void** **outputsOff**(){

P2->OUT &= ~((1<<4)|(1<<6));

}

//turns the fan and compressor on

**void** **outputsOn**(){

P2->OUT |= ((1<<4)|(1<<6));

}

//turns the fan on and compressor off (defrost mode)

**void** **defrost**(){

P2->OUT &= ~(1<<4);

P2->OUT |= (1<<6);

}

**#endif**

/\*\*

\* main.c

\*/

**#include** "fsm.h"

**#include** "msp.h"

**#include** <stdint.h>

**#include** <string.h>

**#include** <stdio.h>

**#include** "msoe\_lib\_clk.h"

**#include** "msoe\_lib\_lcd.h"

**#include** "msoe\_lib\_delay.h"

**void** **initGPIOInputs**();

**void** **initGPIOOutputs**();

**void** **initADC**();

**#define** ADC\_RANGE 4096 // for 12 bit conversions

//Global Variables

uint8\_t temperature = 0; //temperature reading

uint8\_t humidity = 0; // humidity reading

uint8\_t humidity\_setpoint = 50; // set point (default 50)

**void** **main**(**void**)

{

event input;

state current = *NORMAL\_OFF*; // start in off mode

outputsOff(); // start with everything off

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

// setup lcd

LCD\_Config();

LCD\_clear();

LCD\_home();

LCD\_contrast(10);

// init ports

initADC();

initGPIOInputs();

initGPIOOutputs();

**while**(1){

ADC14->CTL0 |= 1; // start ADC conversions

LCD\_goto\_xy(0,0);

LCD\_print\_str("Temp: ");

LCD\_print\_udec5(temperature);

LCD\_goto\_xy(0,1);

LCD\_print\_str("Set: ");

LCD\_print\_udec5(humidity\_setpoint);

LCD\_goto\_xy(0,2);

LCD\_print\_str("Humid: ");

LCD\_print\_udec5(humidity);

LCD\_goto\_xy(0,3);

**if**((P1->IN & 1<<5) >> 5 == 1){ // check for high signal from the ice sensor

LCD\_print\_str("DEFROST: Y ");

} **else** {

LCD\_print\_str("DEFROST: N ");

}

// determine input event

input = *IDLE*;

**if**(humidity >= (humidity\_setpoint + 5)){

input = *TURN\_ON*;

} **else** **if**(humidity <= (humidity\_setpoint - 5)){

input = *TURN\_OFF*;

}

**if**((P1->IN & 1<<5) >> 5 == 1){ // check for high signal from the ice sensor

input = *THERE\_IS\_ICE*;

}

current = stateUpdate(current, input); // call state update

}

}

// Setup both MSP onboard pushbuttons & Ice Sensor

// initializes pushbutton P1.1 as GPIO input

// initializes pushbutton P1.4 as GPIO input

// initializes P1.5 as GPIO input (Ice Sensor)

**void** **initGPIOInputs**()

{

// P1.1

P1->SEL0 &= ~0b10; // use GPIO function

P1->SEL1 &= ~0b10;

P1->DIR &= ~0b10; // make input

P1->REN |= 0b10; // allow pull up/down

P1->OUT |= 0b10; // setup as pull up

P1->IES &= ~0b10; // trigger interrupt on rising edge

P1->IE |= 0b10; // enable interrupt

// P1.4

P1->SEL0 &= ~(0b1<<4); // use GPIO function

P1->SEL1 &= ~(0b1<<4);

P1->DIR &= ~(0b1<<4); // make input

P1->REN |= (0b1<<4); // allow pull up/down

P1->OUT |= (0b1<<4); // setup as pull up

P1->IES &= (0b1<<4); // trigger interrupt on rising edge

P1->IE |= (0b1<<4); // enable interrupt

// P1.5

P1->SEL0 &= ~(0b1<<5); // use GPIO function

P1->SEL1 &= ~(0b1<<5);

P1->DIR &= ~(0b1<<5); // make input

P1->REN |= (0b1<<5); // allow pull up/down

P1->OUT |= (0b1<<5); // setup as pull up

P1->IES &= (0b1<<5); // trigger interrupt on rising edge

P1->IE |= (0b1<<5); // enable interrupt

NVIC->ISER[1] |= (1<<3); // enable I/O P1 interrupt in NVIC

}

// Initializes Outputs for both the fan and compressor

// initializes P2.4 as GPIO output (compressor)

// initializes P2.6 as GPIO output (fan)

**void** **initGPIOOutputs**(){

// P2.4

P2->SEL0 &= ~(1<<4); // use GPIO function

P2->SEL1 &= ~(1<<4);

P2->DIR |= (1<<4); // make output

P2->OUT &= ~(1<<4); // setup output low to start

// P2.6

P2->SEL0 &= ~(1<<6); // use GPIO function

P2->SEL1 &= ~(1<<6);

P2->DIR |= (1<<6); // make output

P2->OUT &= ~(1<<6); // setup output low to start

}

//Initialize ADC6 on pin 4.7

//Initialize ADC7 on pin 4.6

//Interrupt upon completed conversion

**void** **initADC**(){

//setup pins 4.6 and 4.7 in analog mode

P4->SEL0 |= 0b11<<6;

P4->SEL1 |= 0b11<<6;

// start sampling on SC bit,

// source a timer,

// use SMCLK,

// repeat sequence of channels conversion mode

// 96 sample/hold time

// turn core on

ADC14->CTL0 &= 0x0;

ADC14->CTL0 |= (1<<26) | (1<<21) | (0b11<<17) | (0b101<<12) | (0b101<<8) | (1<<4);

// 12 bit resolution and use memory location 4 to start

ADC14->CTL1 &= 0xF0000000;

ADC14->CTL1 |= (0b10<<4) | (4<<16);

ADC14->MCTL[4] |= 0x6; // input on A6 for pot1

ADC14->MCTL[5] |= 0x7; // input on A7 for pot2

ADC14->MCTL[5] |= (1<<7); // set EOS for location5

ADC14->IER0 |= (1<<4); // enable interrupt for mem[4]

ADC14->IER0 |= (1<<5); // enable interrupt for mem[5]

ADC14->CTL0 |= 0b10; // enable

NVIC->ISER[0] |= (1<<24); // enable ADC interrupt in NVIC

ADC14->CTL0 |= 1; // start ADC conversions

}

// Interrupt Handler for ADC

// Read ADC and set to temperature/humidity

**void** **ADC14\_IRQHandler**(**void**)

{

uint32\_t pendingInterrupt = ADC14->IFGR0;

**if**((pendingInterrupt & 1<<4) > 0){

//pot1

temperature = 100-(uint8\_t)((((**float**)ADC14->MEM[4])/ADC\_RANGE)\*100);

ADC14->CLRIFGR0 |= 1<<4; // Clear Interrupt

} **else** **if**((pendingInterrupt & 1<<5) > 0){

//pot2

humidity = 100-(uint8\_t)((((**float**)ADC14->MEM[5])/ADC\_RANGE)\*100);

ADC14->CLRIFGR0 |= 1<<5; // Clear Interrupt

}

ADC14->CLRIFGR0 |= 0xFFFFFFFF; // Extra precaution

}

// Interrupt Handler for the GPIO inputs

// increment / decrement the set point

// interrupt for the ice sensor is unused

**void** **PORT1\_IRQHandler**(**void**){

uint16\_t pending\_interrupt = P1->IV;

**if**(pending\_interrupt == 4){

humidity\_setpoint = (humidity\_setpoint < 5) ? 0 : humidity\_setpoint-5;

} **else** **if**(pending\_interrupt == 0xA){

humidity\_setpoint = (humidity\_setpoint > 95) ? 100 : humidity\_setpoint+5;

} **else** **if**(pending\_interrupt == 0xC){

// Unused interrupt

}

}