GreenHouse AUTO MONITOR

Theory of design

Xiaomin Wu 110151559
GREENHOUSE+ long island 11733 NY

Table of contents

Introduction

Overview:

Temperature, CO2 density and humidity are important environment parameters for plant growth. Automated plant growth monitoring device is an embedded system application to monitoring the temperature, CO2 density and humidity for better control plants' growth.

Components:

The system is built on ATmaga128 microcontroller with humidity and temperature sensor, CO2 sensor, real time clock chip and 4X4 keypad.

1. ATmaga128 microcontroller
2. Honeywell HumidIcon™ Digital Humidity/Temperature Sensors
3. DS1306 Serial Alarm Real-Time Clock
4. SEN0219 Analog infrared CO2 Sensor
5. Liquid Crystal Displays (LCDs)
6. 4X4 keypad

Features:

1. Long operating period per battery change.

The software runs on the system is designed to make the entire system energy efficient, so that the monitoring device can be used for long time without changing battery. T

2. User interact allowed.

he device also displays time and can be set by the user. Alarm can also be set for certain control at specific time, for instance, temperature should be increased during the night and decrease at morning.

This theory of design manual will describe in detail of the hardware and software design, the operating parameters. After reading and understanding this manual, reader can re-design the device with sufficient materials.

Front panel and rear panel design:

Front panel layout:

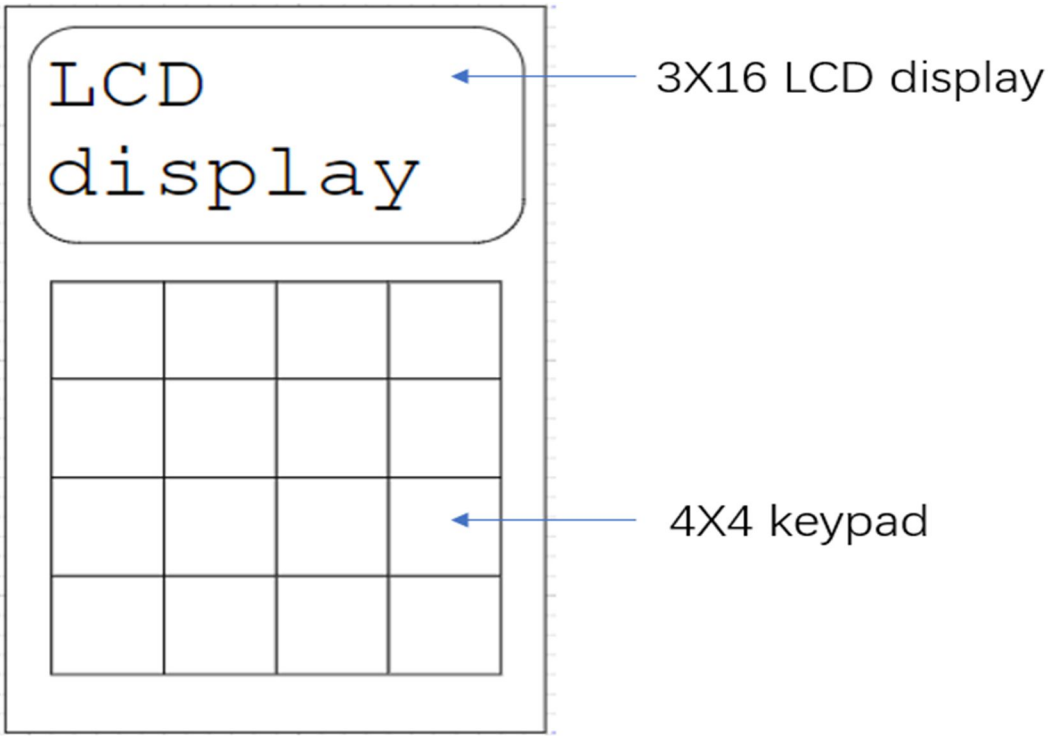


Figure 1 Front panel design

Our device is designed to have a 3X16 LCD display to show the monitoring result including temperature, humidity and CO2 density with a current time display. A 4X4 keypad is put on the front panel to set and control the device, like set the current time and alarm time.

Keypad layout:

1	2	3	Set time
4	5	6	Set alarm
7	8	9	CO2
Not used	0	Not used	Cancel set

Figure 2 4x4 keypad layout

Key function:

•0-9 number key: make device run into time setting mode, and input number digits under time setting mode.

•Set time key: set the input decimal digits into current time and make the device return to temperature humidity measuring mode from time setting mode.

•Set alarm key: set the input decimal digits into alarm time and make the device return to temperature humidity measuring mode from time setting mode.

•CO2 key: make the device run to CO2 measuring and display mode from the temperature and humidity measuring and display mode.

•Cancel set key: abandon the un finished time setting and return to temperature humidity measuring and display mode from time setting mode.

Rear panel layout:

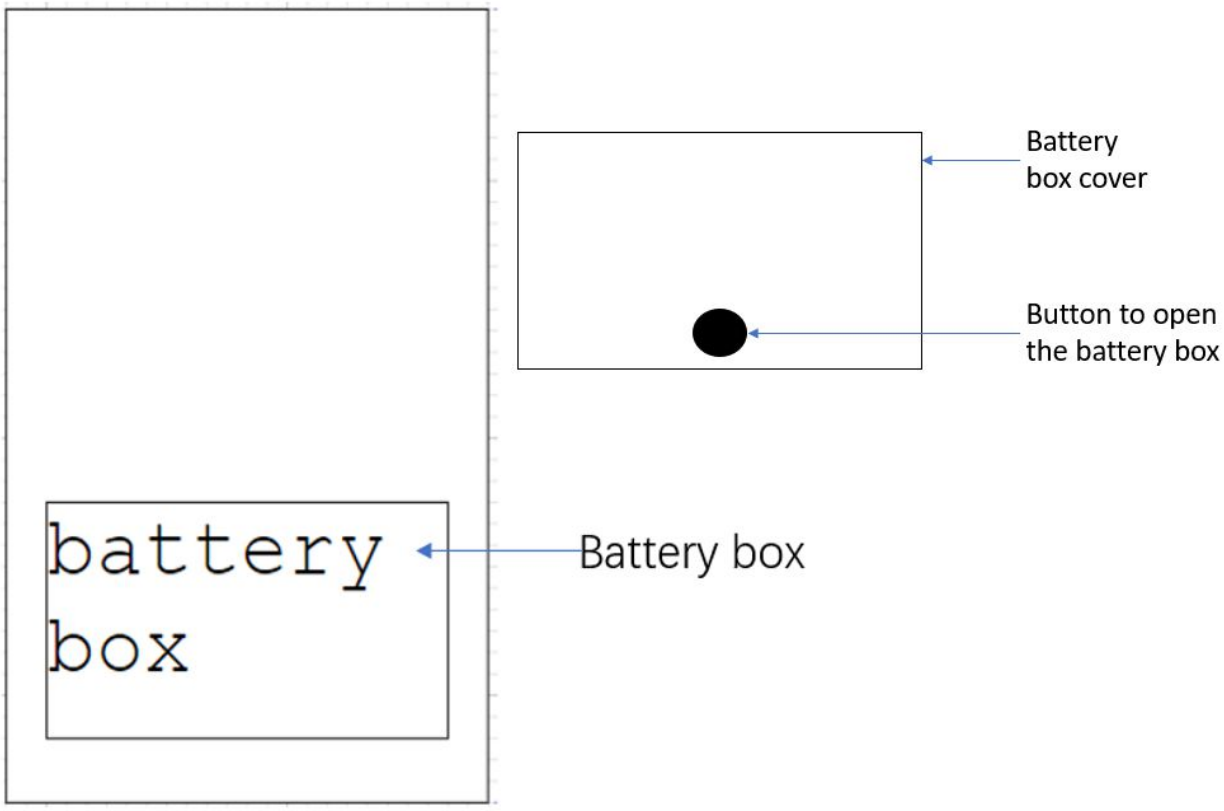


Figure 3 Rear panel design

A space for two AAA batteries is on the rear panel of the device. The device does not need an on/off switch to control. The device will run immediately after the battery was put in. Press the button on the battery box cover to open the battery box.

Hardware design

Hardware block diagram:

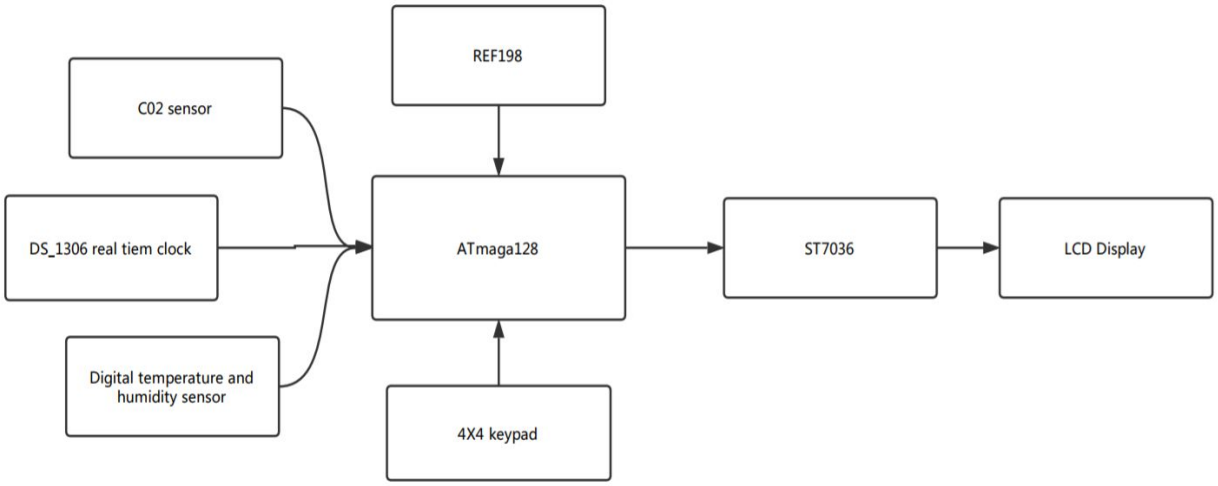


Figure 4 hardware block diagram

•ATmaga128: main logical microcontroller of the device

•CO2 sensor: sensor used to detect the CO2 density

•Digital temperature and humidity sensor: sensor used to detect temperature and humidity

•DS_1306: real time clock chip used to provide current time and alarm function

•REF198: A voltage reference chip to provide reference voltage to the ATmaga128's internal ADC at AREF pin

•4X4 keypad: input device to control the device

•ST7036: Hardware driver chip to control LCD display

ATmaga128 and surrounding circuits：

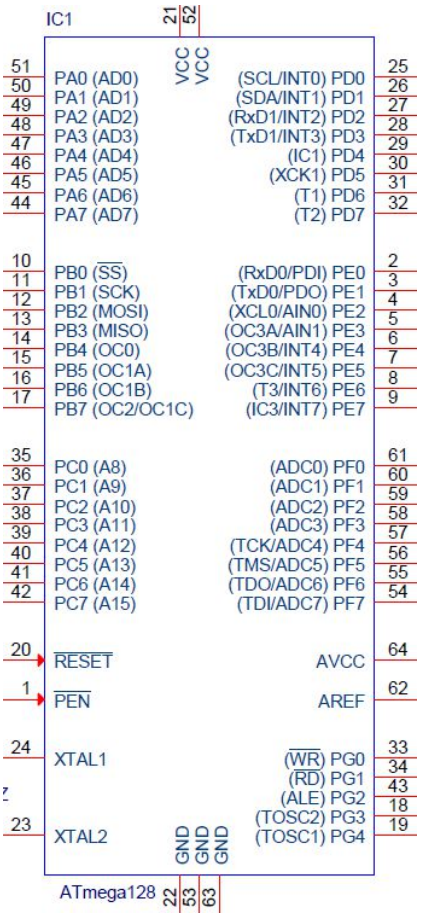


Figure 5 ATmaga128 microchip and pin layout

ATmaga128 microcontroller introduction:

ATmaga128 is a High-performance, Low-power Atmel AVR 8-bit Microcontroller. It is based on Advanced RISC Architecture with 133 Powerful Instructions, 32 × 8 General Purpose Working Registers + Peripheral Control Registers, Fully Static Operation, Up to 16MIPS Throughput at 16MHz, and On-chip 2-cycle Multiplier. It also has High Endurance Non-volatile Memory segments with 128Kbytes of In-System Self-programmable Flash program memory, 4Kbytes EEPROM, 4Kbytes Internal SRAM, Data retention: 20 years at 85°C/100 years at 25°C, Up to 64 Kbytes Optional External Memory Space and SPI Interface for In-System Programming. It supports JATG interface and extensive on-chip debug. It provides plenty of peripherals including: Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes, Two Expanded 16-bit Timer/Counters with

Separate Prescaler, Compare Mode and Capture Mode, Real Time Counter with Separate Oscillator, Two 8-bit PWM Channels, 6 PWM Channels with Programmable Resolution from 2 to 16 Bits, Output Compare Modulator, 8-channel, 10-bit ADC, Byte-oriented Two-wire Serial Interface, Dual Programmable Serial USARTs, Master/Slave SPI Serial Interface, Programmable Watchdog Timer with On-chip Oscillator and On-chip Analog Comparator. ATmaga128 has many useful features including Power-on Reset and Programmable Brown-out Detection, Internal Calibrated RC Oscillator, External and Internal Interrupt Sources, Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby, Software Selectable Clock Frequency and Global Pull-up Disable. ATmaga128 also has 53 Programmable I/O Lines and 64-lead TQFP and 64-pad QFN/MLF. ATmaga128 can operate under 2.7 - 5.5V, with 0 - 16MHz speed grades.

I/O usage:

ATmaga128's I/O resources are selectively used by our device design. PINA was used to select and unselect SPI slaves. PINB was used to do SPI communication. PINC was used for communicating with keypad. PIND was used to get external interrupt requests. PINE was reserved for other extending function. PINF was used for JTAG while modify the device's software and was used for ADC during operating.

Peripheral usage:

ATmaga128's Peripherals are selectively used by our device design. SPI peripheral was used to allow ATmaga128 to communicate with ST7036 for LCD display, DS 1306 for real time clock reading and setting, humidity and temperature sensor for reading humidity and temperature measuring results. 10 bits resolution ADC was used to convert the analog input by the CO2 sensor to digital signal for further processing. Hardware interrupt was used to implement the keypad function, measuring results and LCD display updating and timing alarm function.

Power and ground connection:

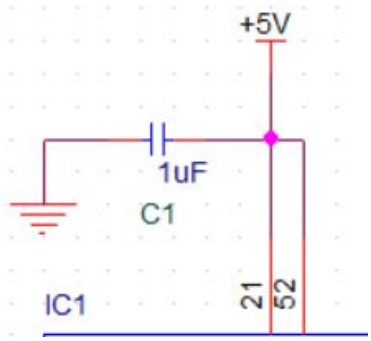


Figure 6 ATmaga128 power connection

5V power supply was connected to the PIN 21 of ATmaga128 chip. 5V power supply was also connected to PIN 52 and to the ground through a 1uf capacitor.

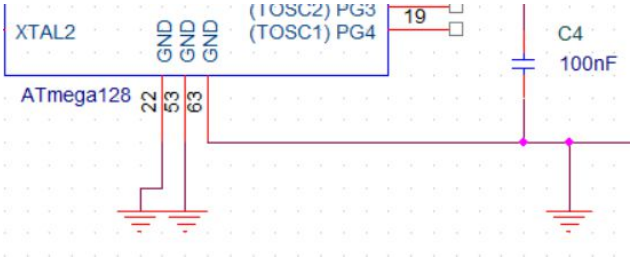


Figure 7 ground connection of ATmaga128

Three GND pins, PIN22, PIN53 and PIN63 are connected directly to ground source.

External crystal oscillator connection:

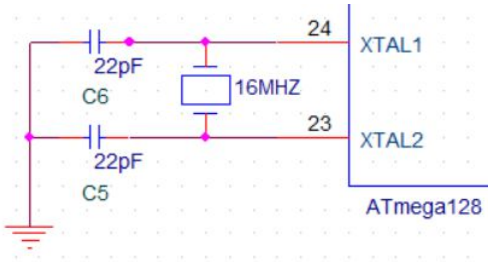


Figure 8 external crystal oscillator connection to ATmaga128

Two ends of the 16MHZ crystal oscillator were connected to the XTAL1 pin (PIN 24) and XTAL2 pin (PIN 23) of ATmaga128. Those two ends of the crystal oscillator were also connected to the ground source through two 22pf capacitors.

RESET bar and PEN bar connection:

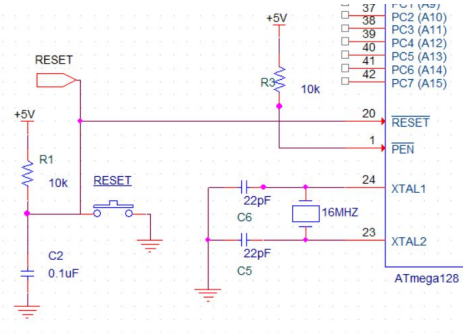


Figure 9 RESET bar and PEN bar connection

A debounced RESET button was connected to the RESET bar pin(PIN 20) of the ATmaga128 chip. PIN 20 was also connected to the JTAG RET output to realize the reset when start emulation. PEN bar pin (PIN 1) was connected to 5V power source through a 10K resistor.

Power LED connection:

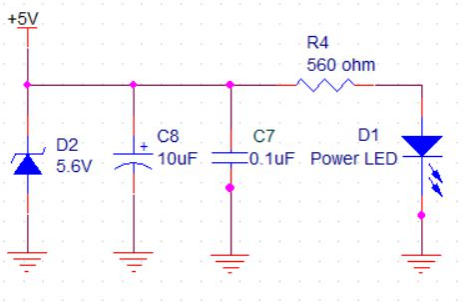


Figure 10 power LED connection

The LED (D1) was serially connected with a 560 ohm resistor and parallelly connected with one 0.1 uf capacitor, one 10 uf capacitor and a 5.6V diode to the 5V power source. All components of this circuit are grounded in the other end. LED will light up once the power is on.

Detail description of used ATmaga128 peripherals:

SPI peripheral:

SPI block diagram (reference from ATmaga128 datasheet):

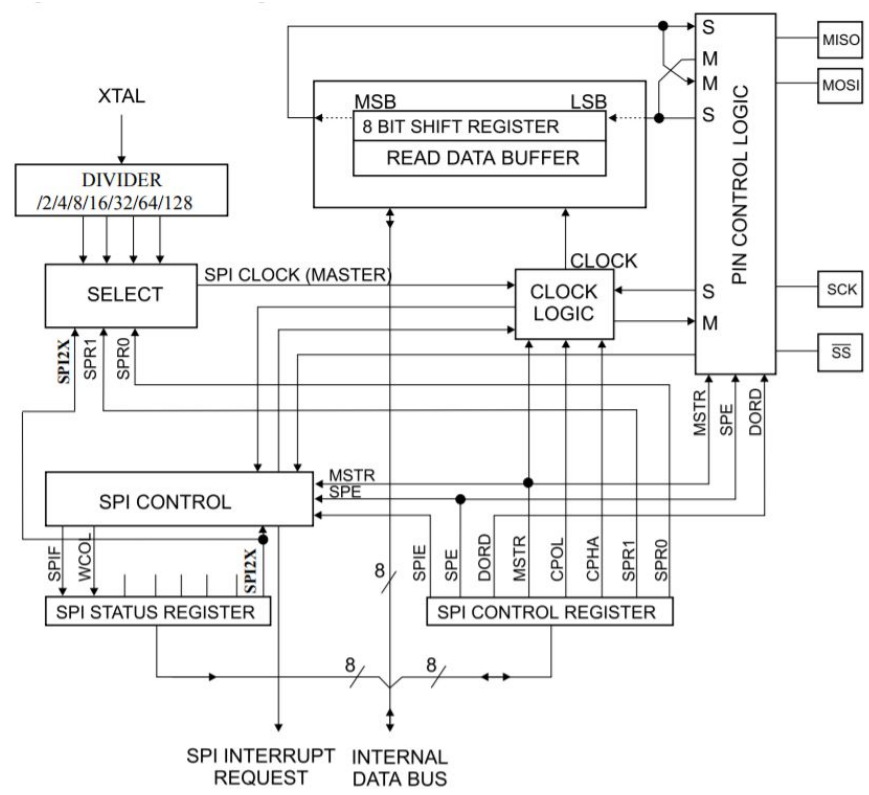


Figure 11 SPI block diagram (reference from ATmaga128 datasheet)

Input crystal oscillator frequency can be programmable divided. It is controlled by the SPI2X, SPR1, SPR0 bits from the SPI status register and SPI control register.

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Figure 12 SPI prescaler operating combinations controlled by SPI2X, SPR1, SPR0 (referenced from ATmaga128 datasheet)

SPI control register also control the clock logic with bits CPOL, CPHA.

SPI Mode	Conditions	Leading Edge	Trailing Edge
0	CPOL=0, CPHA=0	Sample (Rising)	Setup (Falling)
1	CPOL=0, CPHA=1	Setup (Rising)	Sample (Falling)
2	CPOL=1, CPHA=0	Sample (Falling)	Setup (Rising)
3	CPOL=1, CPHA=1	Setup (Falling)	Sample (Rising)

Figure 13 SPI mode selection with CPOL and CPHA (referenced from ATmaga128 datasheet)

The SPI mode is depended on the slave's working timing diagram. If the clock is high level before the slave is selected, the CPOL should be set to 1, otherwise 0. If the data is sampled at the leading edge, then the CPHA should be set to 0, otherwise 1.

MSTR bit must be set if the ATmaga128 was to be configured as master of SPI communication. Clock directly control the read data buffer. If ATmaga128 wants to read data from its SPI slave, ATmaga128 need to start the clock after select the slave device to do the read. Four pins for SPI, MISO, MOSI, SCK, SS bar, are all controlled by the pin control logic block. SPI interrupt request can be generated by the SPI control logic if the SPIE bit in the SPI control register was enabled and the SPIF bit in the SPI status register was set. 8 bits data can be read from the SPI data register from the internal data bus. Data must be read before the second byte reading executing, i.e., second read result will cover the first read result.

Analog digital convertor (ADC) peripheral (referenced from ATmaga128 datasheet):

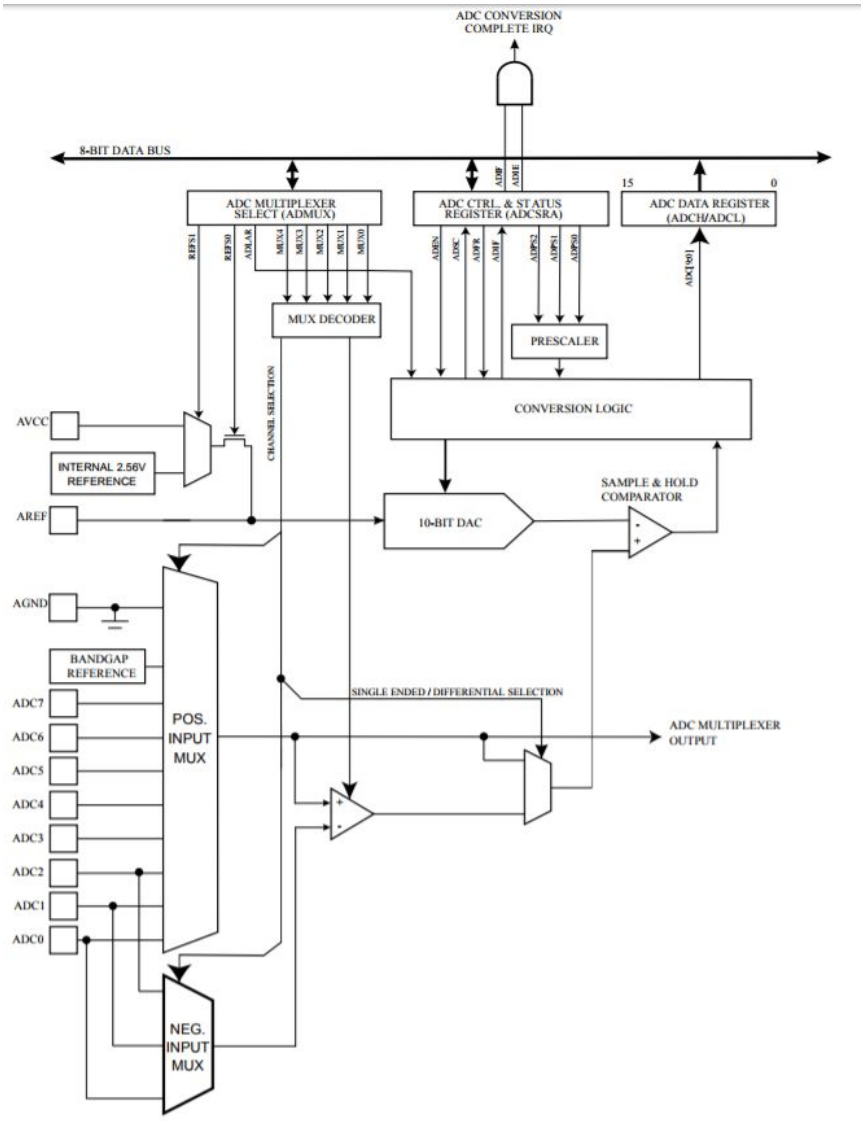


Figure 14 Analog digital convertor (ADC) peripheral (referenced from ATmaga128 datasheet)

ADC module on the ATmaga128 chip can be accessed and controlled by read or write four registers: ADMUX, ADCSRA, ADCH and ADCL. The ADMUX register controls the reference voltage modes with most significant two bits, there are four modes to choose.

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Figure 15 voltage reference modes (referenced from ATmaga128 datasheet)

The ADMUX register also controls which ADC PIN to take analog input in, and determine whether to add amplification to those analog inputs.

MUX[4:0]	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000[1]	Reserved	ADC0	ADC0	10x
01001	Reserved	ADC1	ADC0	10x
01010[1]	N/A	ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101	Reserved	ADC5	ADC2	1x

Figure 16 ADMUX register selecting ADC input pins and amplification (reference from ATmaga128 datasheet)

ADCSRA is the control and status register of ADC module.

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figure 17 ADCSRA bits layout (referenced from ATmaga128 datasheet)

ADEN bit was used to enable the ADC module. ADSC was used to start aN ADC conversion. ADFR was used to enable the ADC free running mode. ADIF is the ADC interrupt status flag. ADIE was used to enable the ADC interrupt. ADPS2-0 bits were used to pre-scale the input clock frequency.

ADPS[2:0]	Division Factor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Figure 18 ADC prescaler operating combination (referenced from ATmaga128 datasheet)

ADC data were stored in ADCL and ADCH registers. When ADLAR bit is unset (0), those two register's bits arrangement is right justified. When ADLAR is set (1), bits in those two registers are left justified.

Bit	7	6	5	4	3	2	1	0
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Figure 19 ADCL register bits arrangement when ADLAR bit is 0

Bit	7	6	5	4	3	2	1	0
							ADC9	ADC8
Access							R	R
Reset							0	0

Figure 20 ADCH register bits arrangement when ADLAR bit is 0

LCD and ST7036 driver:

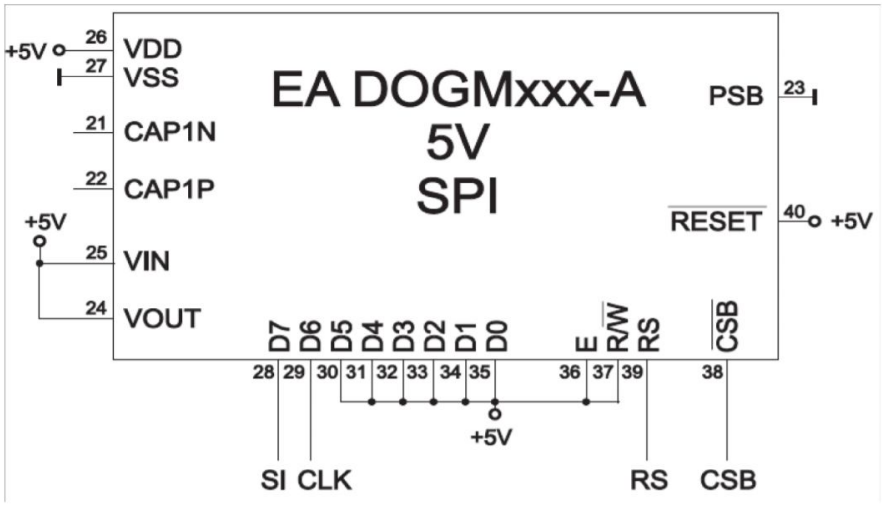


Figure 21 ST7036 pin layout diagram (referenced from ESE 381 Lab manual from Prof. Short)

The LCD display module uses the Sitronix ST7036 Dot Matrix Controller Driver. The DOGM163W-A

module can be operated at a supply voltage of either 3.3 V or 5.0 V.

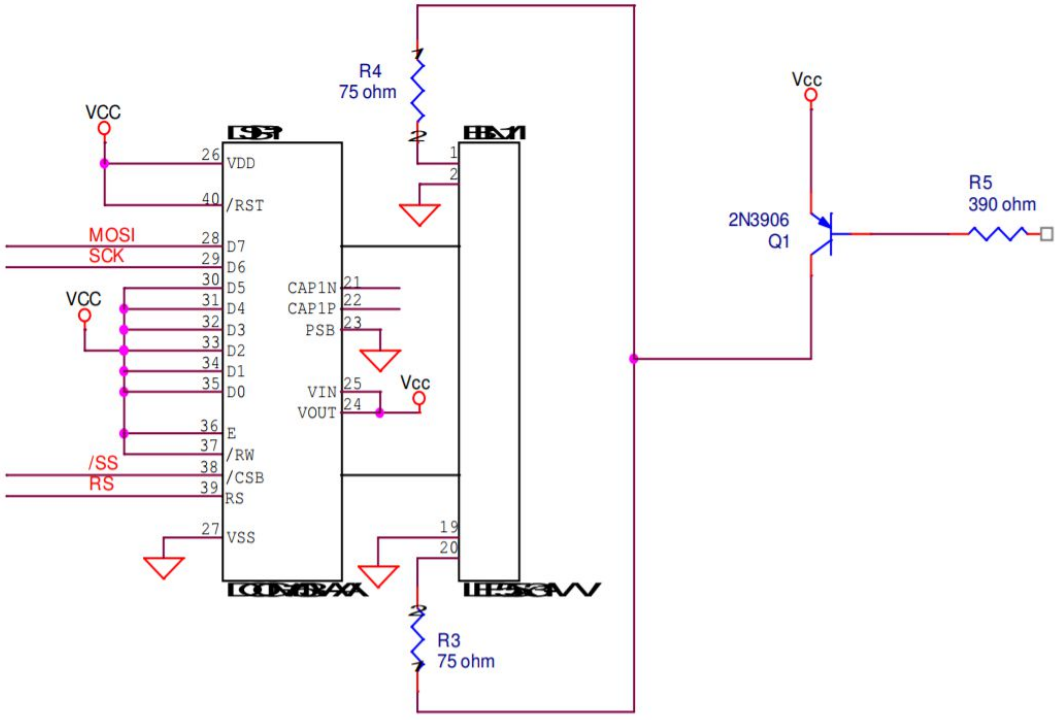


Figure 22 LCD display and ST7037 connection (referenced from ESE 381 Lab manual from Prof. Short)

As shown in the hardware connection figure (fig. 19), D7 is connected to ATmaga128's MOSI PIN, D6 is connected to SCK PIN of ATmaga128, CSB bar is the select PIN connected to SS bar of ATmaga128, RS is connected to the RS pin of ATmaga128. Both LCD display and ST7036 are connected to power and ground source.

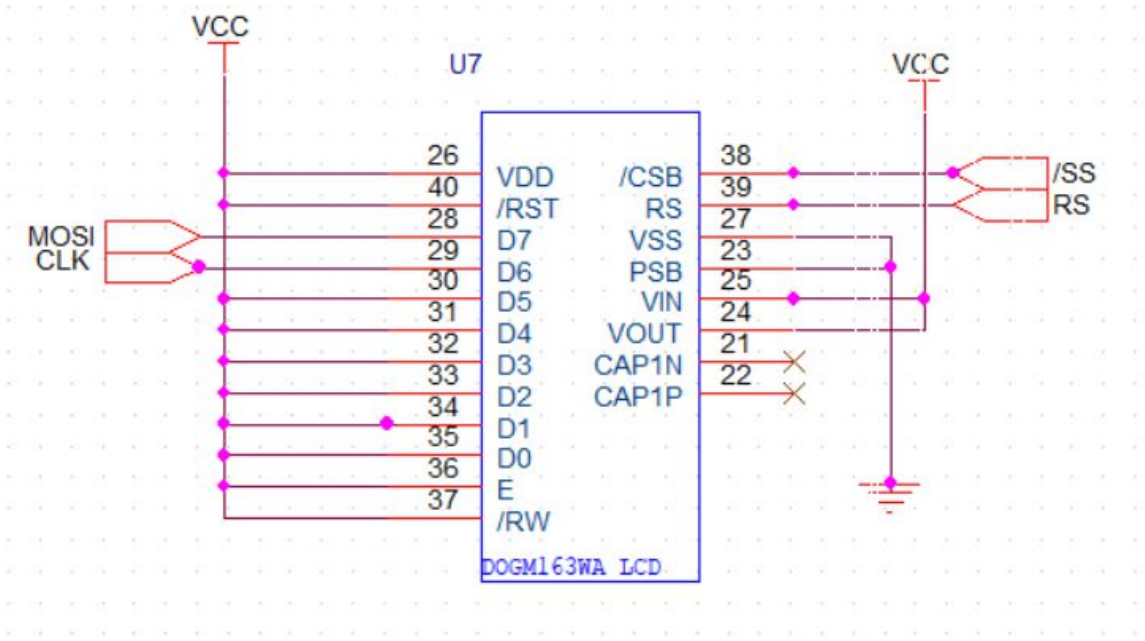


Figure 23 DOGM163WA_LCD schematic

4X4 keypad:

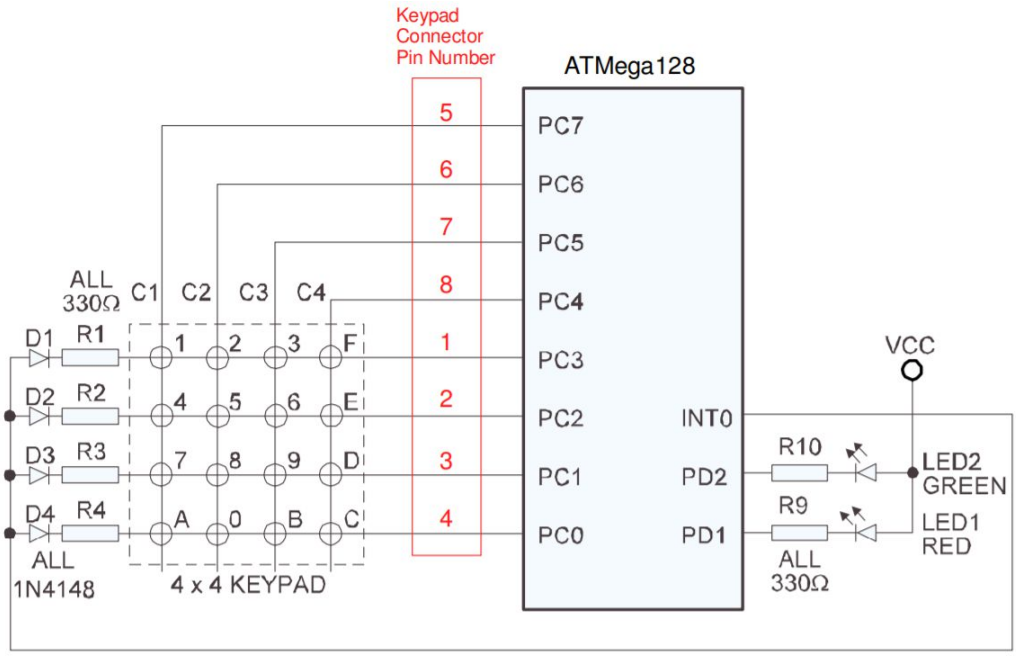


Figure 24 4x4 keypad circuit connection (referenced from ESE381 lab manual from Prof. Short)

Two LEDs were used during product testing and were removed in the finished product. INT0 is enabled with an internal pull-up resister and is set to be sensitive at logic 0. D1 to D4 are four diodes to control the current effective direction of the circuit. When testing which key is pressed, configure row-pins as input and enable internal pull-up, column-pins as output and output logic 0. So, if no key is pressed, the inputs' internal pull-up will balance the INT0's internal pull-up, the INT0 will remain a logic 1. Once any key is pressed, the corresponding output pin output a logic 0 to change the connected input pin to be logic 0, the INT0 will also change to 0 since the balancing logic 1 is changed to logic 0 now. Exchanging the row-pins and column-pins to be the output pins and input pins can let the hardware find out which key is pressed.

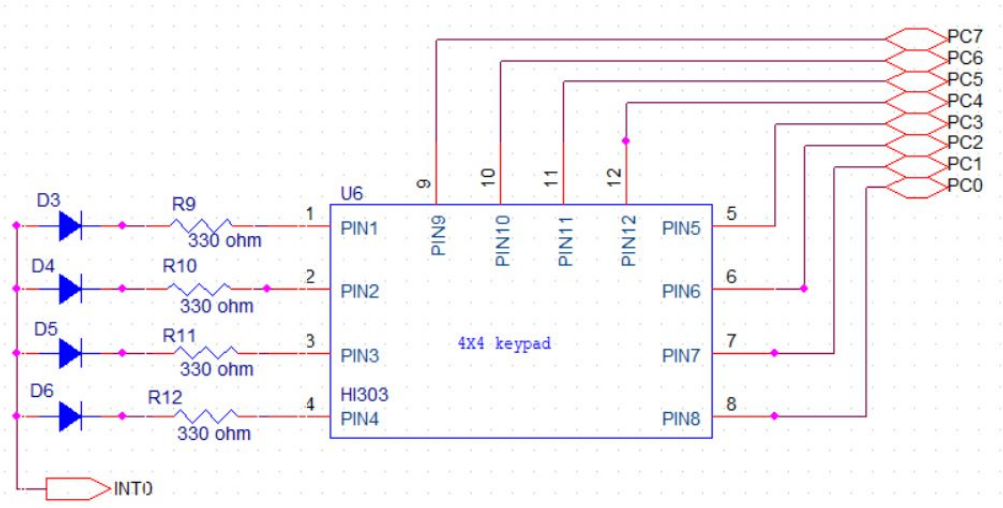


Figure 25 4X4 keypad schematic

Digital Humidity/Temperature Sensors:

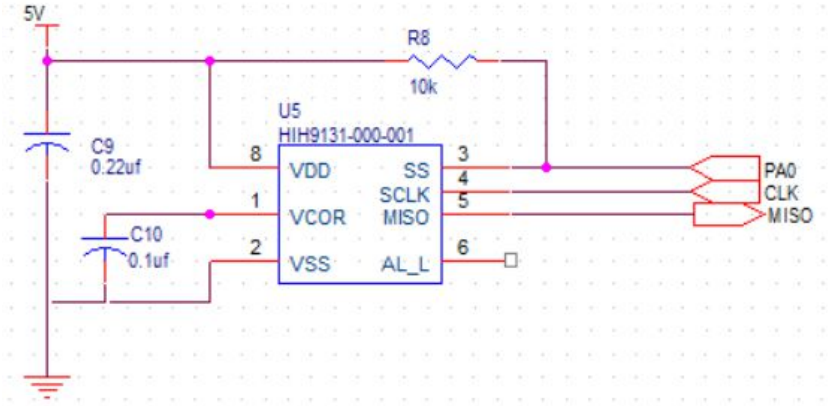


Figure 26 Digital Humidity/Temperature Sensors HIH9131 hardware connection

SS pin was connected to VCC through a 10k resister, it is also connected to PA0 of ATmaga128. SCLK pin is connected to PB1 of ATmaga128. MISO pin is connected to PB3 of ATmaga128. VDD pin is connected to VCC. VCOR pin is connected through one 0.1 uf capacitor to the ground source and through another 0.22 uf capacitor to the VCC source. VSS is connected to the ground source. AL_L pin is left unused.

DS1306 Serial Alarm Real-Time Clock:

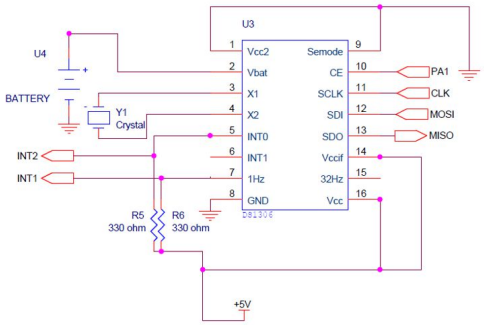


Figure 27 hardware connection of the DS1306 chip

The DS1306 is the third slave device in our system. The other two slave devices are LCD, ST7036 and the HIH9131. DS1306 is selected by the PA1 pin of the ATmaga128. It has independent battery to keep the inside timer running after the whole device out of power. It has its own crystal oscillator.

SEN0219 Analog infrared CO2 Sensor:

Figure 28 SEN0219 CO2 Sensor hardware figure and pins (referenced from SEN0219 datasheet)

The CO2 sensor has three pins with easy connection to power source, ground source and ADC pin on ATmaga128 (ADC0 was used).

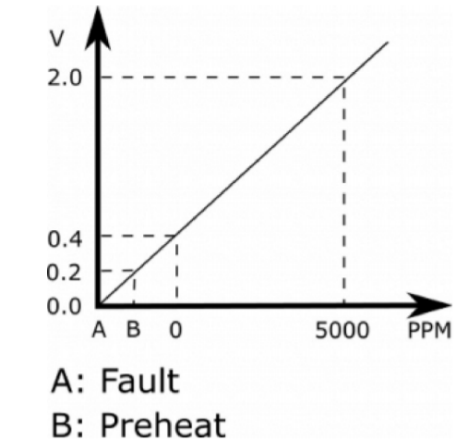


Figure 29 SEN0219 CO2 sensor operating range (referenced from SEN0219 datasheet)

The CO2 sensor has range of operation. Normal operating range is when output voltage between 0.4 V and 2.0 V. Among this range, the CO2 density in PPM is linearly relating to the output voltage. If the output voltage is under 0.4V and above 0.2V, the sensor is under a preheat stage. If the output voltage is under 0.2 V, the device is in a fault measuring range.

Software design

Table driven FSM design:

FSM state diagram:

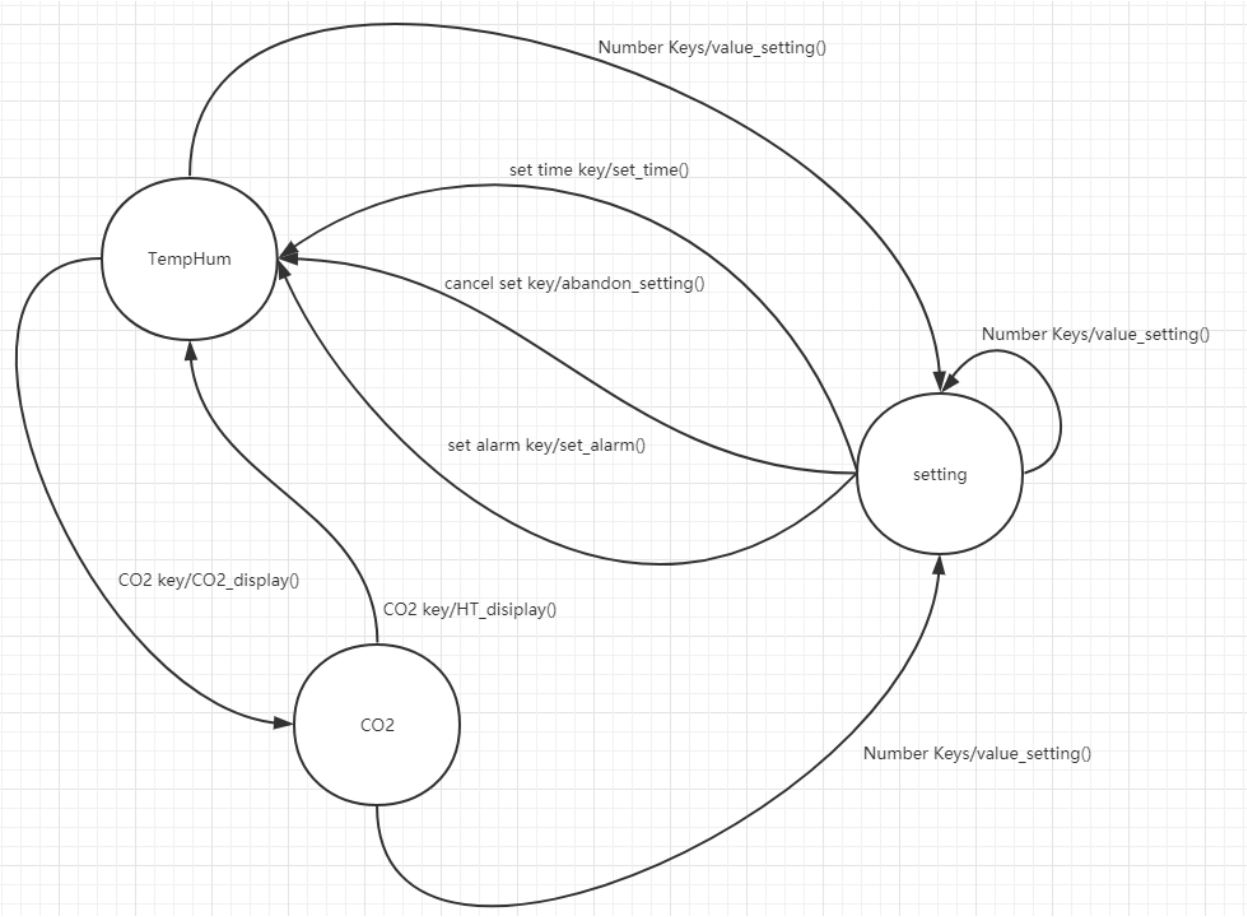


Figure 30 FSM state diagram

The whole software is running based on a table driven finite state machine. The FSM has three states: TempHum, CO2 and setting states. In TempHum state, software mainly measures temperature and humidity from humidicon sensor and convert and display the result. This state can be transferred to setting state by pressing any number keys on the keypad, it can be also transferred to CO2 state by pressing the CO2 key on the keypad. Under setting state, the software mainly controls the device to echo the user input on the LCD. User need to input six values in this state to configure current time or alarm time by pressing set time or set alarm key. The setting state can only be transferred back to tempHum state by pressing one of three keys: set alarm, set time and cancel set. CO2 state can be transferred from TempHum state only, and can be transferred to either setting by

pressing any number keys or back to TempHum state by pressing CO2 key again. Under CO2 state, the software controls the hardware to read CO2 sensor SEN0219 and convert the digital signal into CO2 density.

FSM files:

fsm.h

This file provides important data structures and other typedef information for other fsm file to use.

fsm_table.c

This program file implements transition tables and state table for table driven fsm to use.

fsm_ui.c

This file contain the FSM core function, which will use the input present state and key value to look up tables and call a corresponding task function from the right table, update the next state in the found transition table to the present state.

fsm_function.c

This program file implements functions called by table driven fsm as outputs.

FSM functions:

void value_setting(void):

this function mask out DS1306 interrupt, only keep interrupt for keypad. It records keycode input by the keypad and store them in buffer. Function keeps tracking of the buffer position. Function echoes back the keycode input by user.

void set_time(void):

Function converts keycode saved in the buffer into mainingful time value and stores those time value into another buffer called timeAry. Block writes those time value into DS_1306 register from 0x80 using SPI. It re-enable interrupts masked out by the value_setting() function.

void set_alarm(void):

Function converts keycode saved in the buffer into mainingful time value and stores those time value into another buffer called timeAry. Block writes those time value into DS_1306 register from 0x87 using SPI. It re-enable interrupts masked out by the value_setting() function.

void abandon_setting(void):

Function resets the position tracker for the keycode buffer to 0 and re-enable interrupts masked out by value_setting() function.

void CO2_display(void):

Function sets the tempOrCO2 flag to force the DS_1306 ISR to measure CO2 density and display on LCD.

void HT_display(void):

Function clears the tempOrCO2 flag to force the DS_1306 ISR to measure temperature and humidity and display on LCD.

void error_fn(void):

function be called when eol was hit.

Humidicon driver:

Humidicon.c file

Global variables:

unsigned int humidity; /**< unsigned int variable to hold humidity result */

unsigned int temperature; /**< unsigned int variable to hold temperature result */

Functions:

void SPI_humidicon_config (void):

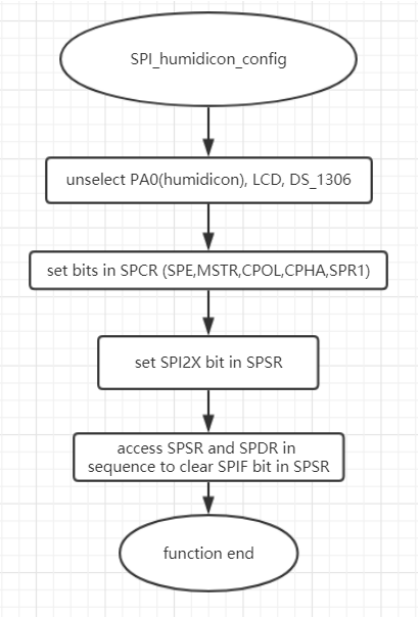


Figure 31 SPI_humidicon_configure function flowchart

This function unselects the HumidIcon and configures it for operation with an ATmega128A operated a 16 MHz. Pin PA0 of the ATmega128A is used to select the HumidIcon.

unsigned char read_humidicon_byte(void):

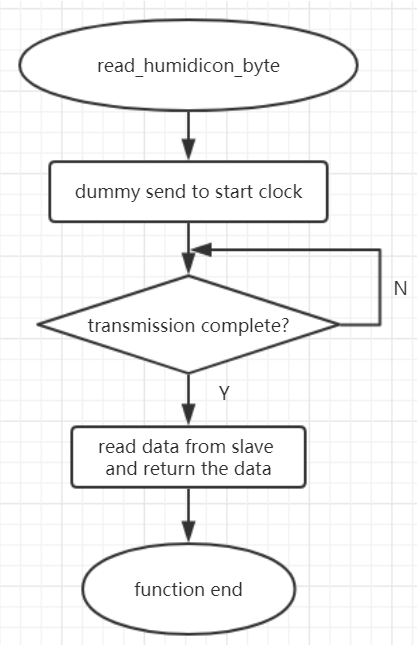


Figure 32 read_humidicon_byte function flowchart

This function reads a data byte from the HumidIcon sensor and returns it as an unsigned char. The function does not return until the SPI transfer is completed. The function determines whether the SPI transfer is complete by polling the appropriate SPI status flag.

void read_humidicon (void):

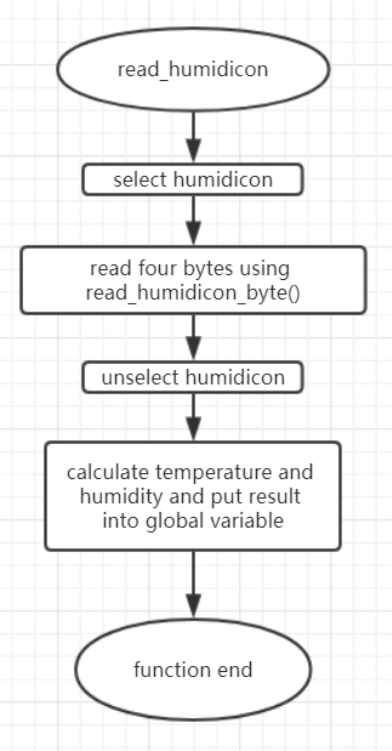


Figure 33 read_humidicon function flowchart

This function selects the Humidicon by asserting PA0. It then calls read_humidicon_byte() four times to read the temperature and humidity information. Is assigns the values read to the global unsigned ints humidicon_byte1, humidion_byte2, humidion_byte3, and humidion_byte4, respectively. The function then deselects the HumidIcon.

The function then extracts the fourteen bits corresponding to the humidity information and stores them right justified in the global unsigned int humidity_raw. Next if extracts the fourteen bits corresponding to the temperature information and stores them in the global unsigned int temperature_raw. The function then returns

temp_humid_humidicon.c file

Functions:

unsigned int compute_scaled_rh(unsigned int rh):

calculation equation: (unsigned int)(((long)rh * 100 * 100)/((1 << 14) - 2));

reconstruct the humidity data scaled by 100, return it as an unsigned int.

unsigned int compute_scaled_temp(unsigned int temp):

calculation equation: (unsigned int)(((((long)temp * 100 *165) / ((1 << 14) - 2))) - (40*100));

reconstruct the temperature data scaled by 100, return it as an unsigned int.

void meas_display_rh_temp(void):

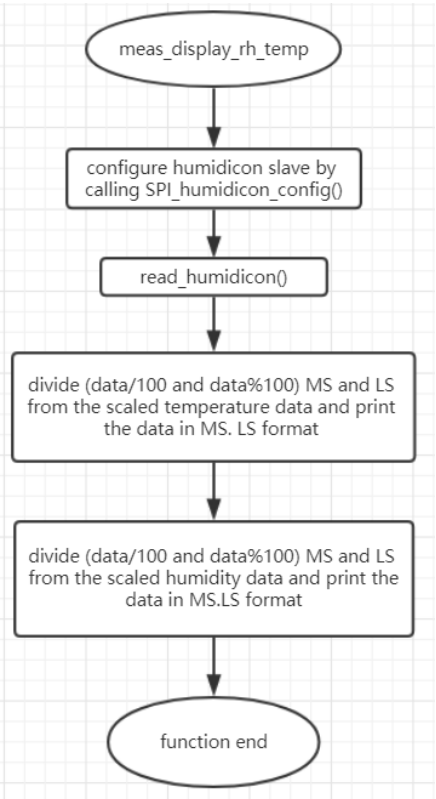


Figure 34 meas_display_rh_temp function flowchart

configure the HumidIcon for a reading get scaled temperature and humidity values then displays the results on a DOG 3 x 16 LCD.

DS1306 driver:

DS1306_RTC_drivers.c file:

Global variables:

volatile unsigned char RTC_time_date_write[4] = {0x80,0x80,0x80,0x80};

/**< array of char to hold contents to write to RTC */

volatile unsigned char RTC_time_date_read[7];

/**< array of char to hold read result of RTC */

Functions:

void SPI_rtc_ds1306_config (void):

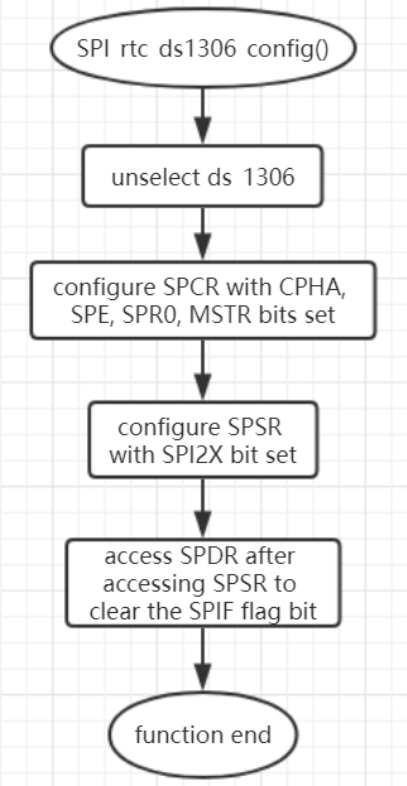


Figure 35 SPI_rtc_ds1306_configure() function flowchart

This function unselects the ds_1306 and configures an ATmega128 operated at 16 MHz to communicate with the ds1306. Pin PA1 of the ATmega128 is used to

select the ds_1306. SCLK is operated under the maximum possible frequency for the ds1306.

void write_RTC (unsigned char reg_RTC, unsigned char data_RTC):

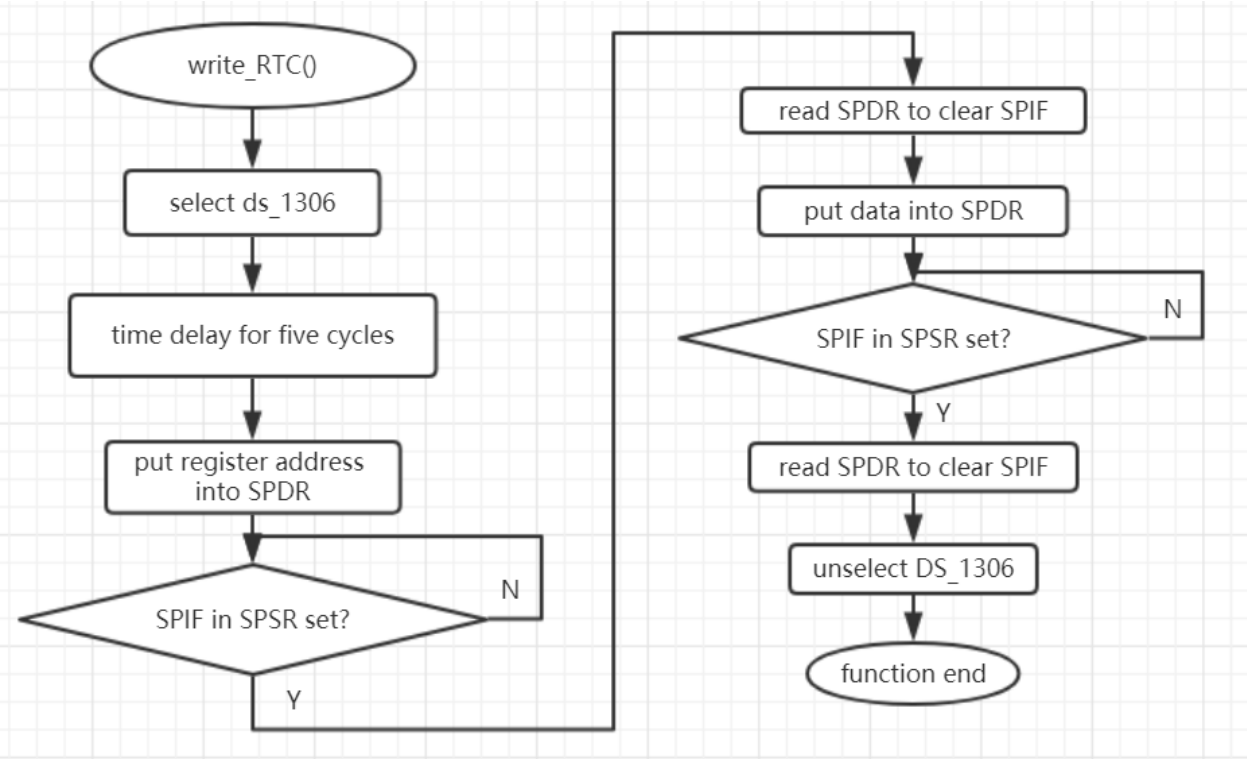


Figure 36write_RTC() function flowchart

This function writes data to a register in the RTC. To accomplish this, it must first write the register's address (reg_RTC) followed by writing the data (data_RTC). In the DS1306 data sheet this operation is called an SPI single-byte write.

unsigned char read_RTC (unsigned char reg_RTC):

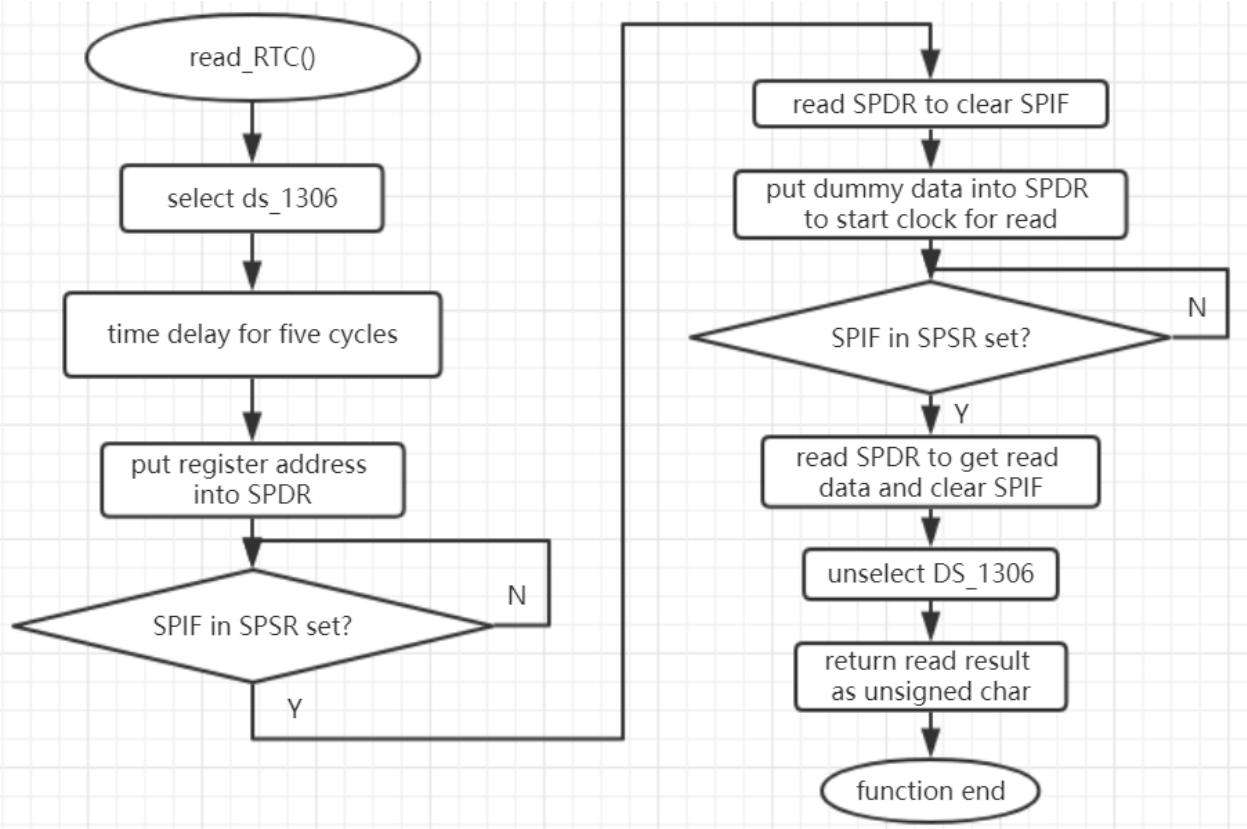


Figure 37 read_RTC() function flowchart

This function reads data from a register in the RTC. To accomplish this, it must first write the register's address (reg_RTC) followed by writing a dummy byte to generate the SCLKs to read the data (data_RTC). In the DS1306 data sheet this operaration is called am SPI single-byte read.

void block_read_RTC (volatile unsigned char *array_ptr, unsigned char strt_addr, unsigned char count):

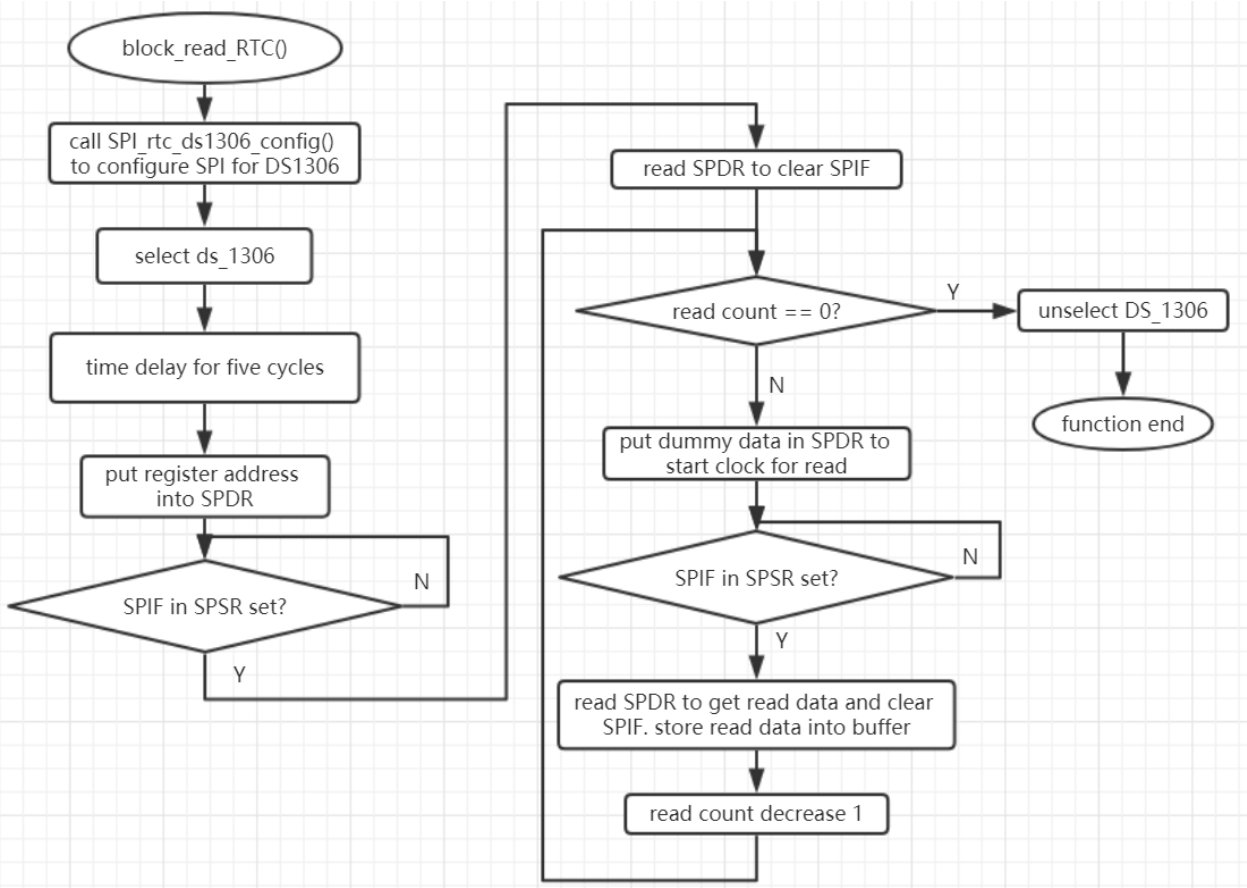


Figure 38 block_read_RTC() function flowchart

This function reads a block of data from the DS1306 and transfers it to an array. strt_addr is the starting address in the DS1306. count is the number of data bytes to be transferred and array_ptr is the address of the destination array.

void block_write_RTC (volatile unsigned char *array_ptr, unsigned char strt_addr, unsigned char count):

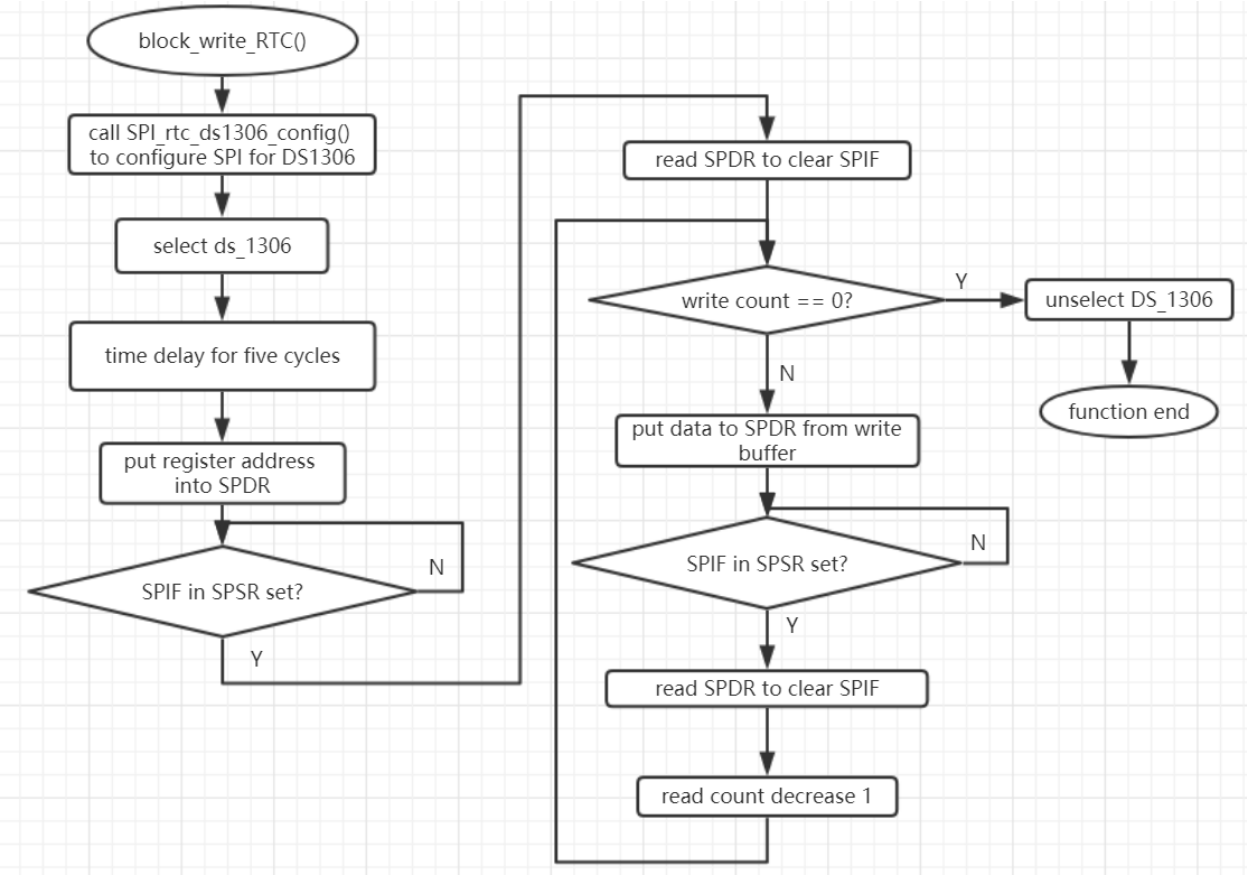


Figure 39 block_write_RTC() function flowchart

This function writes a block of data from an array to the DS1306. strt_addr is the starting address in the DS1306. count is the number of data bytes to be transferred and array_ptr is the address of the source array.

LCD driver:

Lcd_dog_iar_driver.c file:

Global variables:

char dsp_buff_1[16], dsp_buff_2[16], dsp_buff_3[16];

/**< buffers used to store print data for LCD display */

Functions:

void init_spi_lcd(void):

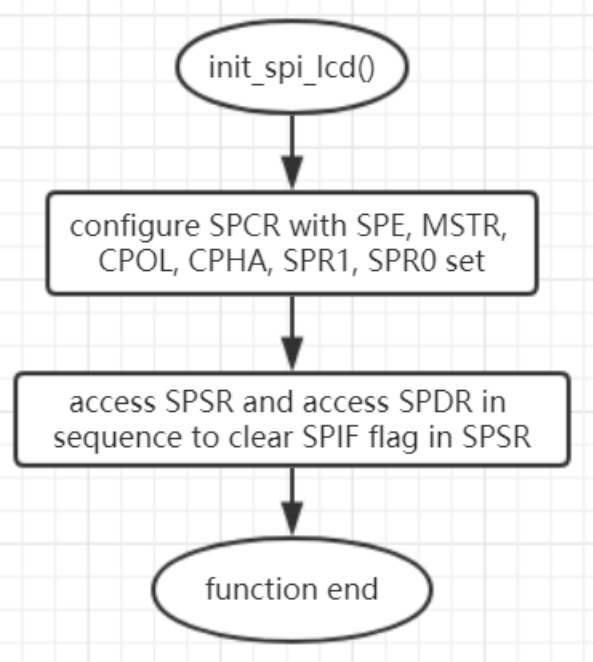


Figure 40 init_spi_lcd() function flowchart

init SPI port for command and data writes to LCD via SPI.

void lcd_spi_transmit_CMD(char cmd):

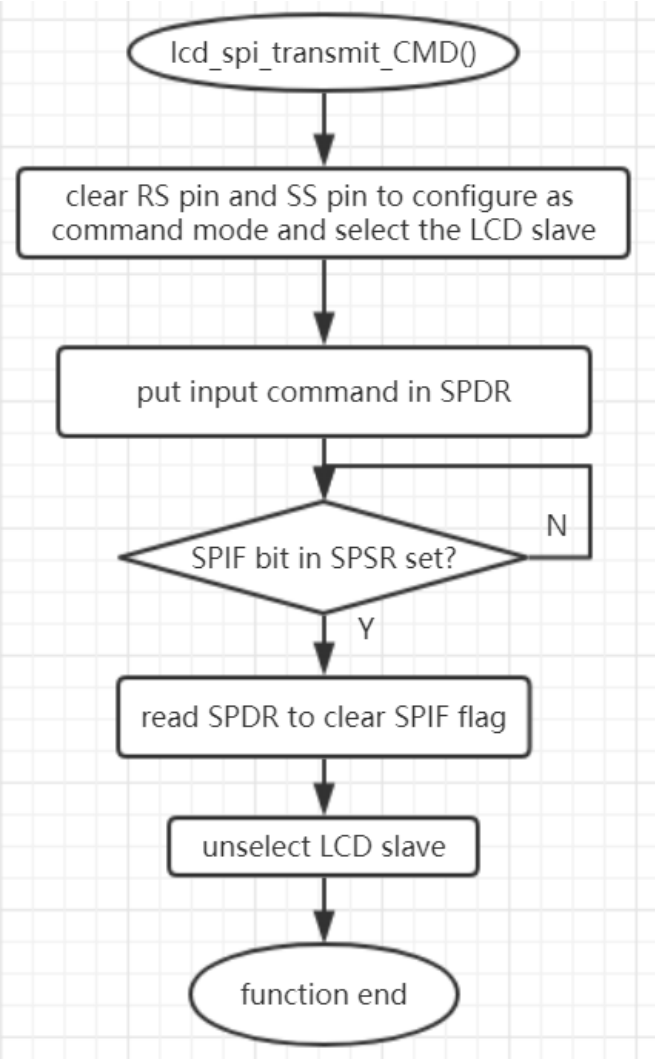


Figure 41 lcd_spi_transmit_CMD() function flowchart

This function outputs a byte passed in r16 via SPI port. Waits for data to be written by spi port before continuing.

void lcd_spi_transmit_DATA(char data):

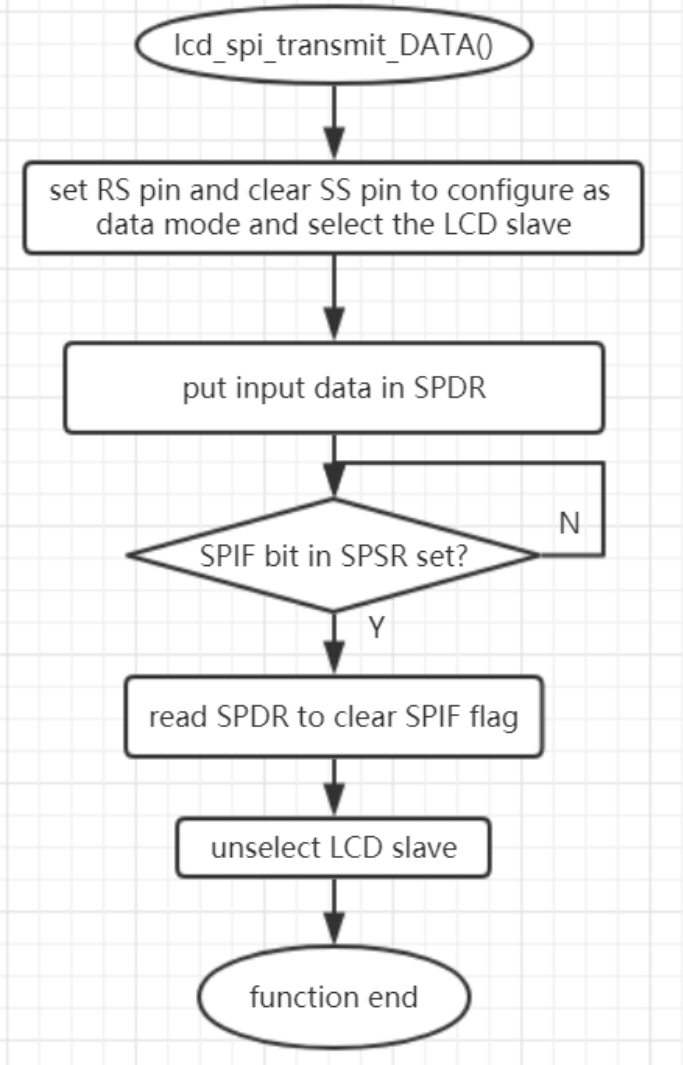


Figure 42 lcd_spi_transmit_DATA() function flowchart

This function outputs a byte passed in r16 via SPI port. Waits for data to be written by SPI port before continuing.

void init_lcd_dog(void):

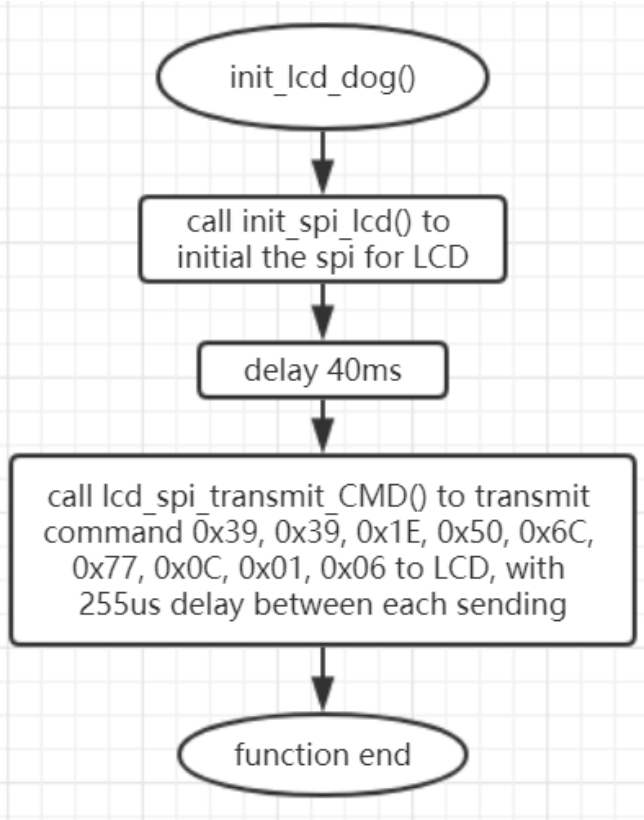


Figure 43 init_lcd_dog() function flowchart

This function inits DOG module LCD display for SPI (serial) operation. NOTE: Can be used as is with MCU clock speeds of 4MHz or less.

void update_lcd_dog(void):

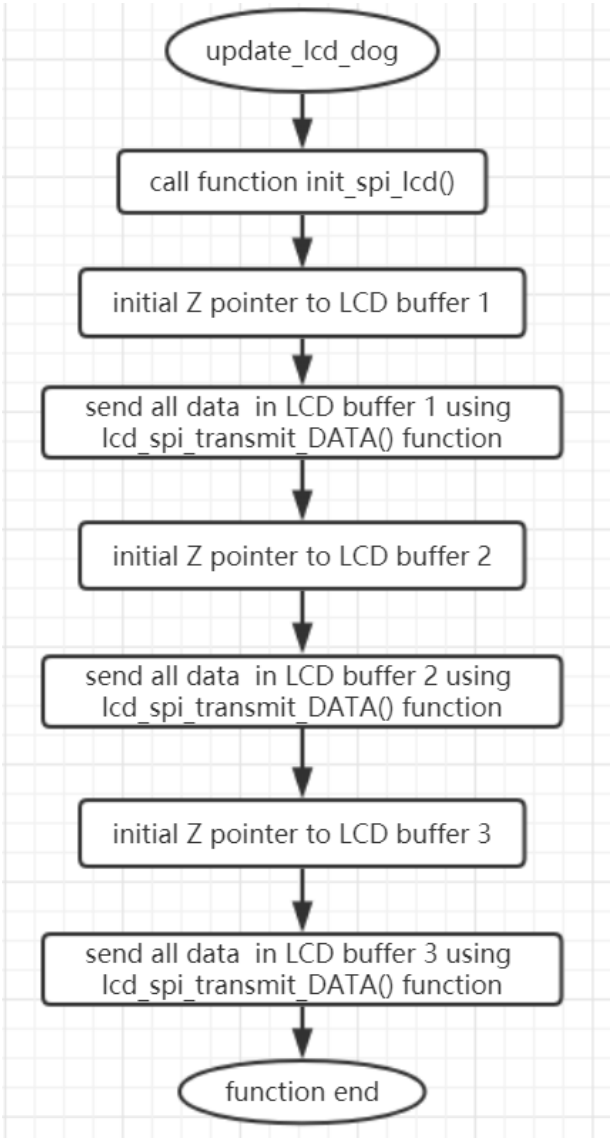


Figure 44 update_lcd_dog() function flowchart

This function updates the LCD display lines 1, 2, and 3, using the contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.

Lcd_ext.c file

Global variable:

static char index; // index into display buffer

Functions:

void clear_dsp(void):

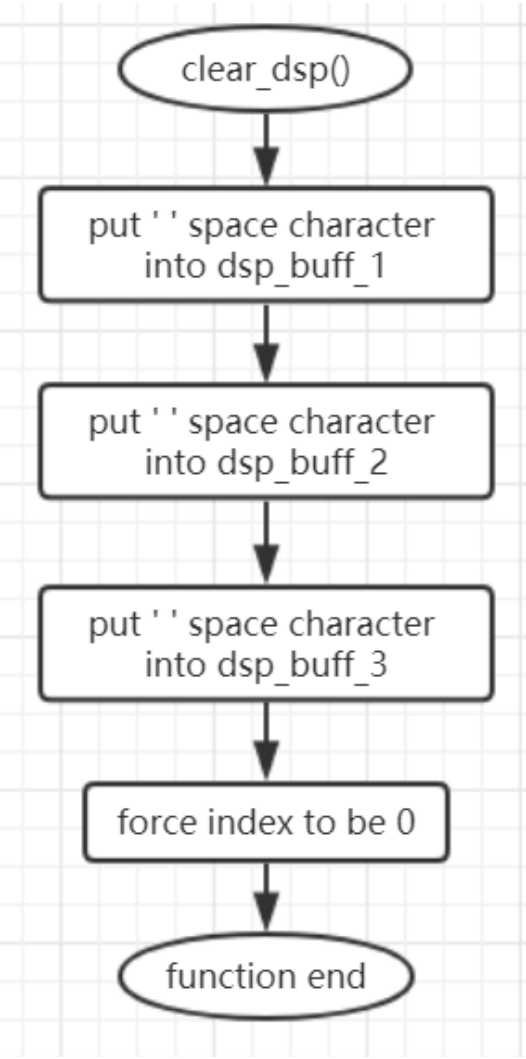


Figure 45 clear_dsp() function flowchart

This function clears the display buffer. Treats each 16 characters array separately.
NOTE: update_dsp must be called after to see results

int putchar(int c):

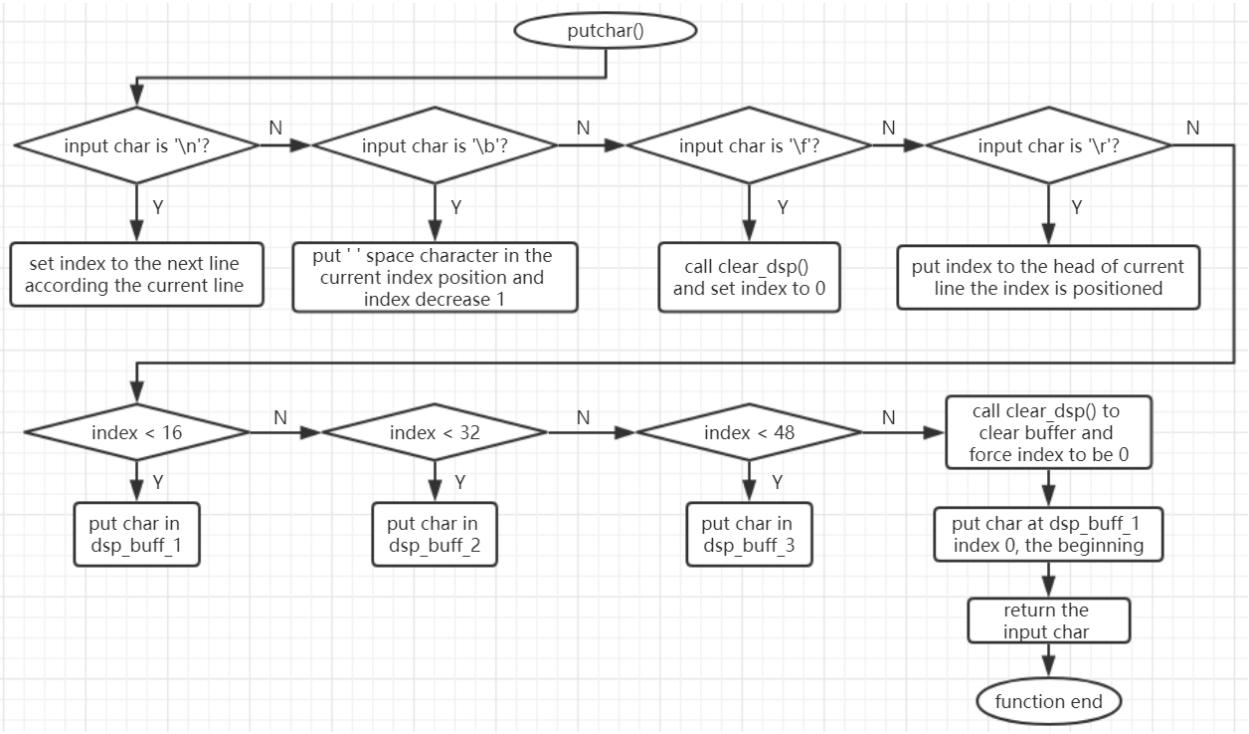


Figure 46 putchar() function flowchart

This function displays a single ascii chararacter c on the lcd at the position specifieb by the global variable index also contain functional inputs: \b, \f, \n, \r. NOTE: update_dsp must be called after to see results

CO2 sensor ADC:

adc.c file:

Functions:

void adc_initial():

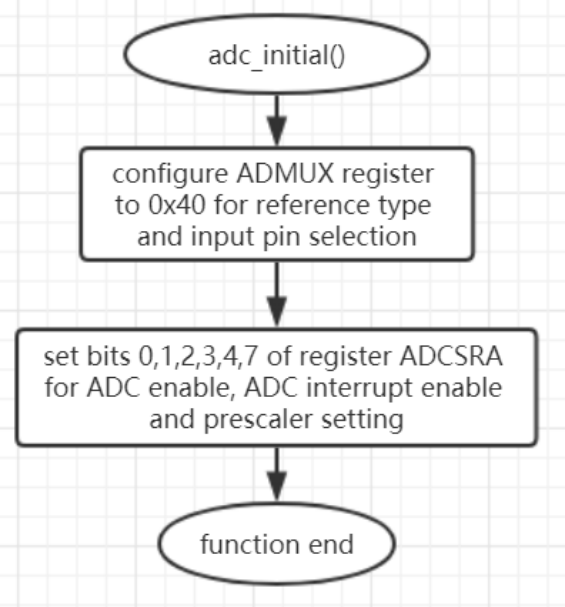


Figure 47 adc_initial() function flowchart

This function is used to initial adc module.

void adc_read():

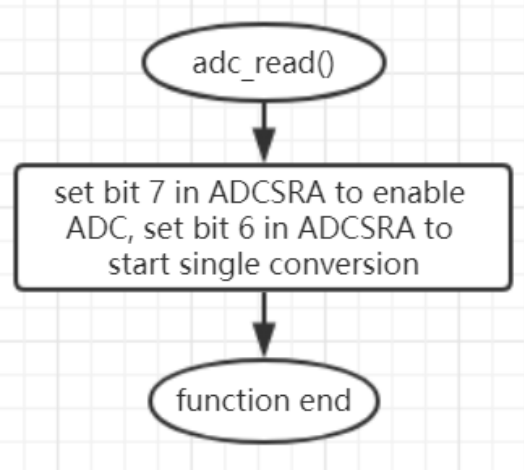


Figure 48 adc_read() function flowchart

This function is used to read from adc module.

adc_isr.c file:

Global variables:

int ADC_result; //global result variable

double VoltResult; //ADC convertion result in volts

int PPM; //ADC convertion result in ppm

Functions:

__interrupt void ISR_ADC(void):

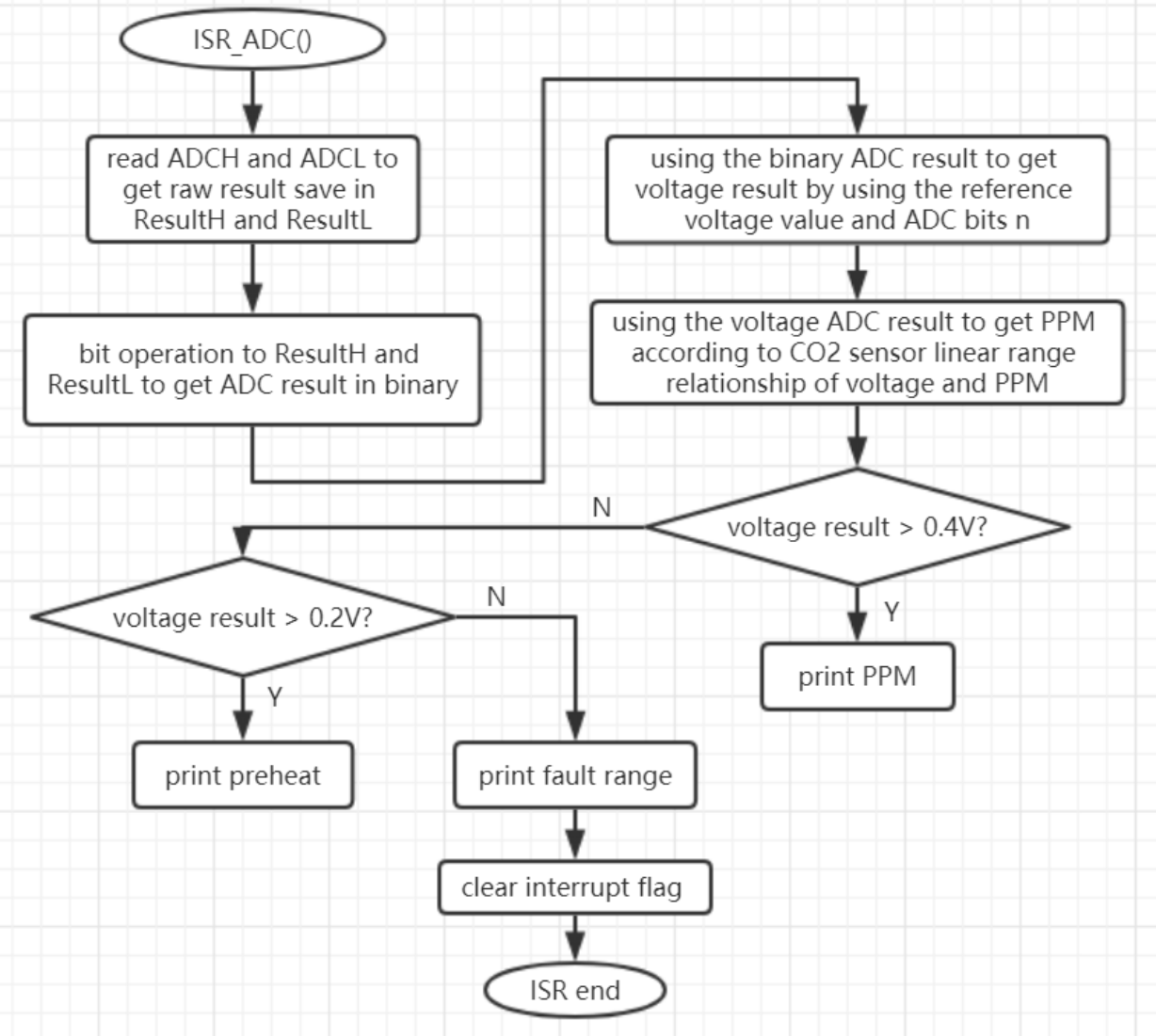


Figure 49 ISR_ADC() interrupt service routine flowchart

This function read raw data from ADCH and ADCL, convert the raw data into ADC result in voltage and in PPM. It also prints the PPM result conditionally.

4X4 keypad:

keyscan_isr.c file:

Global variables:

const char tbl[16] = {1, 2, 3, 'F', 4, 5, 6, 'E', 7, 8, 9, 'D', 'A', 0, 'B', 'C'}; /**< keypad lookup table */

char keycode; /**< char to hold input keycode */

key keypressed; /**< key type variable to hold input key for keypressed */

Functions:

__interrupt void ISR_INT0(void):

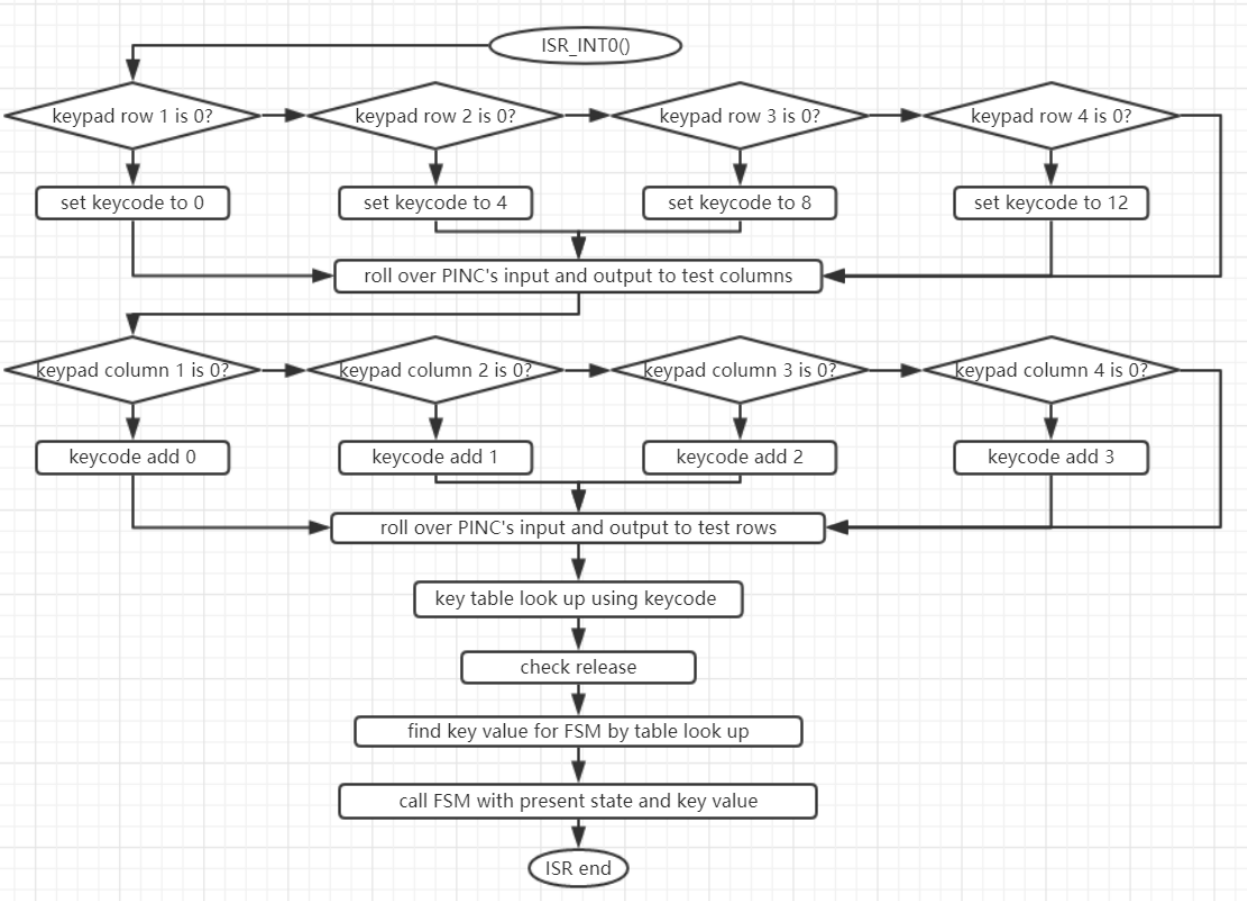


Figure 50 ISR_INT0() interrupt service routine flowchart

When keypad interrupt occurs, key matirx is scanned and is encoded using a table lookup. The keypad is connected to PORTC. The ISR test row and column separately to determine the pressed key's position. Use the key value to look up a FSM's key value. And call FSM function fsm() with the present state and the input key value.

Main file:

fsm_main.c file:

void main(void):

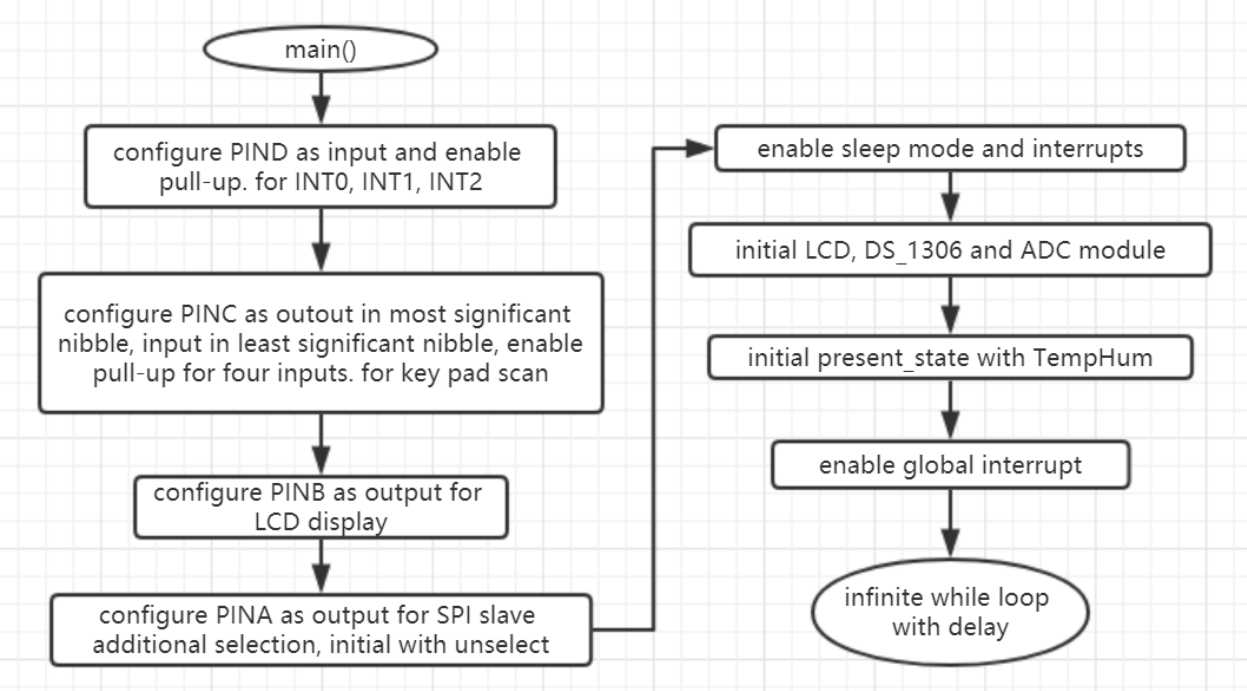


Figure 51 main() function flowchart

The main function configures pins and initial each module used in the application. It also enables interrupts and the present state of the finite state machine used in the application. After finishing those initialization, the main function simply goes into a infinite while loop with inner delays. The application program will be driven purely by interrupts.

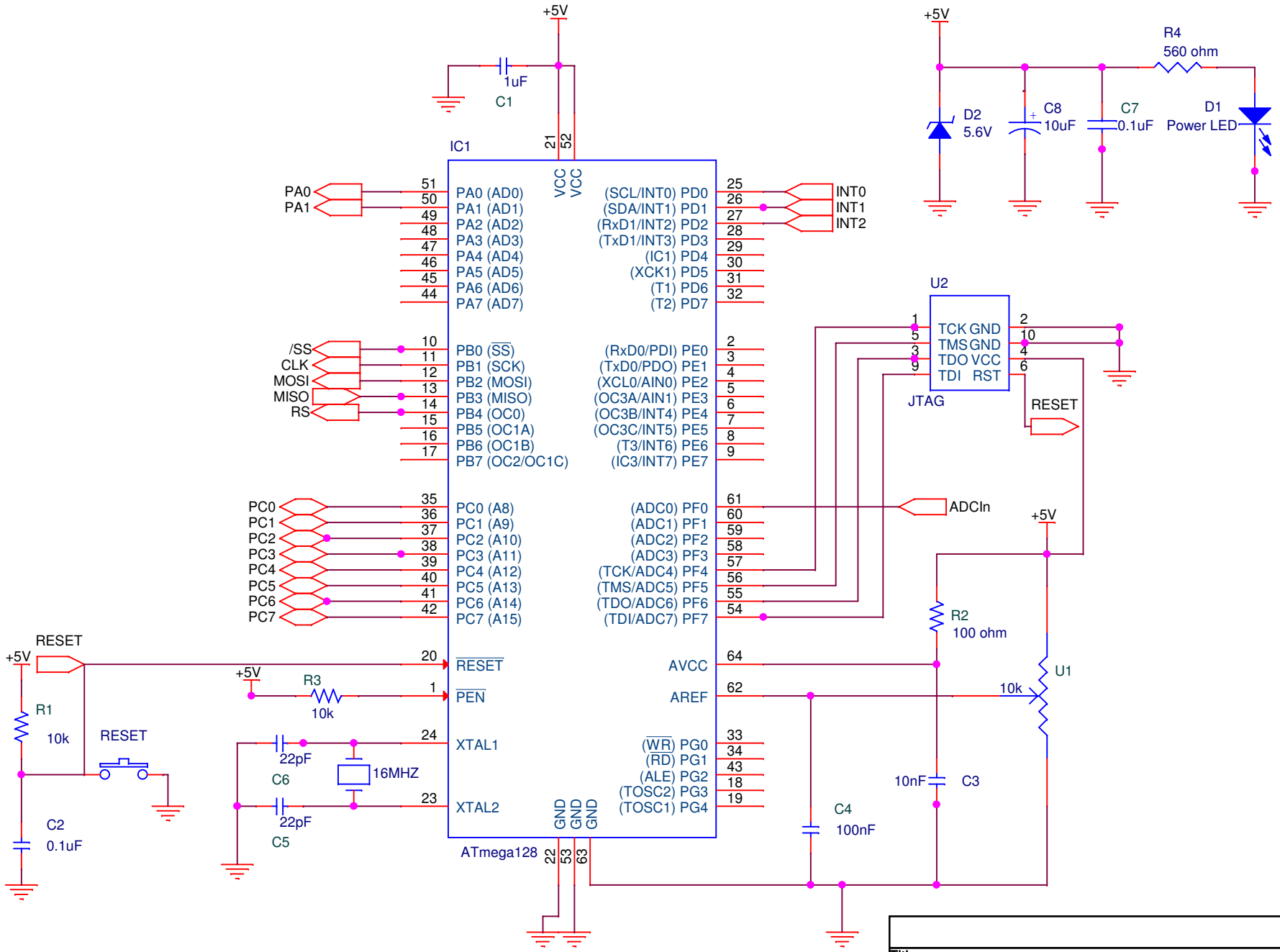
Header files:

All header files contain necessary function declaration for other files which using functions of the header file's corresponding c file to be able to call functions needed.

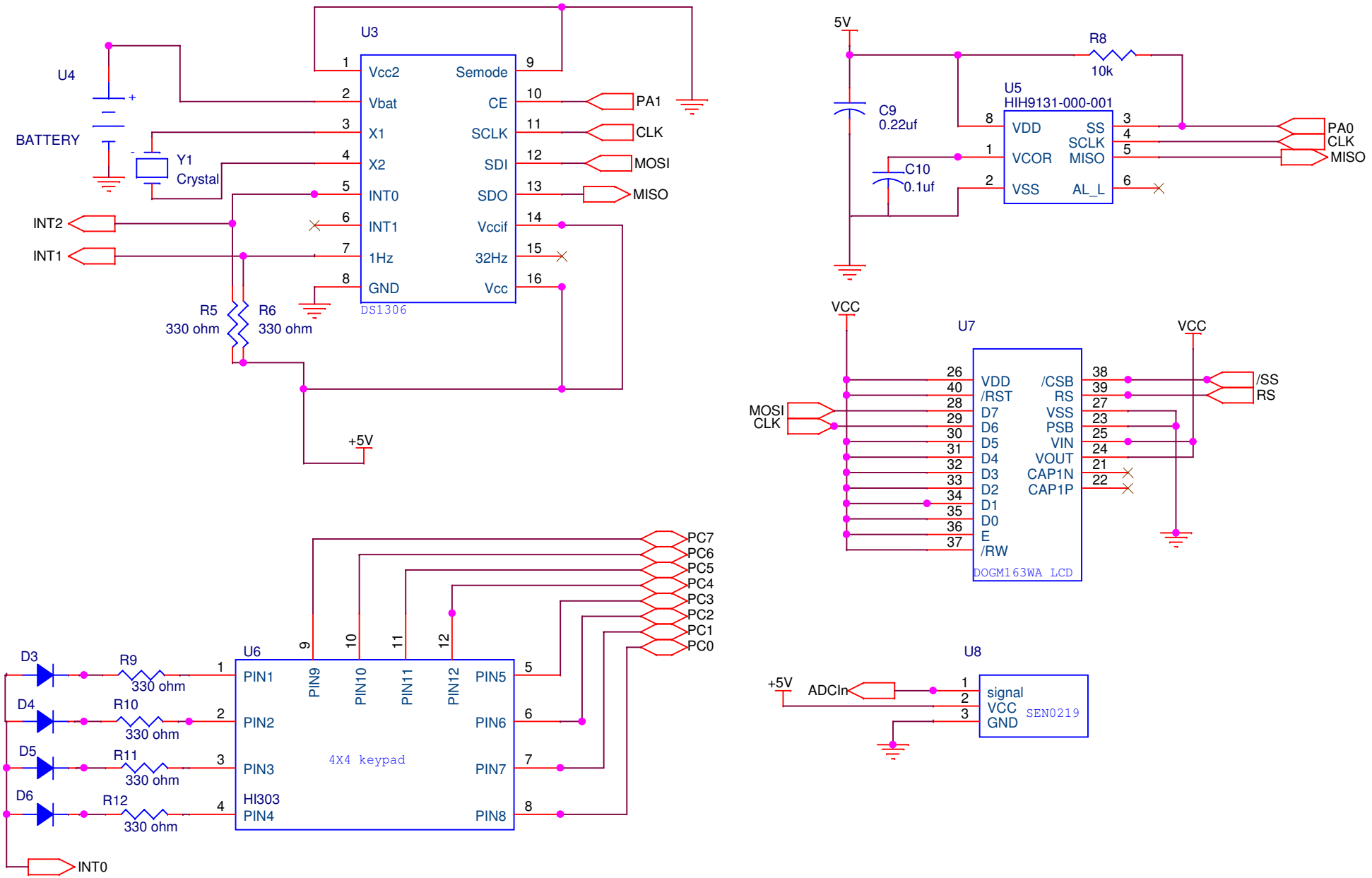
Header files used:

- adc.h
- DS1306 RTC driver.h
- fsm.h
- humidicon.h
- lcd.h

Appendix



Title	ATmega128 wiring	
Size A	Document Number ESE381	Rev 0
Date: Tuesday, May 01, 2018	Sheet 1 of 1	



Title	COMPONENTS	
Size A	Document Number ESE381	Rev 1
Date: Tuesday, May 01, 2018	Sheet 1 of 1	

```
//*****************************************************************************
//
// File Name            : adc.c
// Title                : adc
// Date                 : 03/05/18
// Version              : 1.0
// Target MCU           : ATmega128 @ 16 MHz
// Target Hardware      ;
// Author               : Xiaomin Wu
// DESCRIPTION
// software interfece with ADC on ATmega128 chip
//                       variables
//
// Warnings             : temp not used
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//*****************************************************************************
#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>



//*****************************************************************************
// Function : void adc_initial(void)
// Date and version : version 1.0
// Target MCU : ATmega128A @ 16MHz
// Author :
// DESCRIPTION
// this function is used to initial adc module
//
// Modified
//*****************************************************************************
void adc_initial(){
  ADMUX = 0x40; //AVCC as REF with external cap at AREF pin
                // ADC7 as single analog input
  //ADCSRA = 0x97; //enable ADC . prescaler 128, to 125 Khz fastest, clear INT flag
  SETBIT(ADCSRA,7);//ENABLE
  SETBIT(ADCSRA,4);
  SETBIT(ADCSRA,3);//ENABLE INTERRUPT

  SETBIT(ADCSRA,2);//128 DEVIDE
  SETBIT(ADCSRA,1);
  SETBIT(ADCSRA,0);

}
//*****************************************************************************
// Function : void adc_read(void)
// Date and version : version 1.0
// Target MCU : ATmega128A @ 16MHz
// Author :
// DESCRIPTION
// this function is used to read from adc module
//
// Modified
//*****************************************************************************
void adc_read(){
  //start convertion by writing register ADCSRA
  SETBIT(ADCSRA,7);//ENABLE
  SETBIT(ADCSRA,6);//set bit 6 in ADCSRA to start single convertion


}
```

```
//***********************************************************************
//
// File Name            : adc.h
// Title                : adc
// Date                 : 03/05/18
// Version              : 1.0
// Target MCU           : ATmega128 @ 16 MHz
// Target Hardware      ;
// Author               : Xiaomin Wu
// DESCRIPTION
// software interfece with ADC on ATmega128 chip
//                      variables
//
// Warnings             : temp not used
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//***********************************************************************
extern void adc_read();
extern void adc_initial();
```

```
//*********************************************************************
//
// File Name            : adc_isr.c
// Title                : adc
// Date                 : 03/05/18
// Version              : 1.0
// Target MCU           : ATmega128 @ 16 MHz
// Target Hardware      ;
// Author               : Xiaomin Wu
// DESCRIPTION
// software interfece with ADC on ATmega128 chip
//                      variable. the interrupt service routine of ADC
//
// Warnings             : temp not used
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//*********************************************************************

#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>
#include <stdio.h>

#include "lcd.h"

int ADC_result; //global result variable
double VoltResult; //ADC convertion result in volts
int PPM; //ADC convertion result in ppm

#pragma vector= ADC_vect          // Declare vector location.
__interrupt void ISR_ADC(void)   // Declare interrupt function
{

    int ResultH, ResultL;
    ResultH = ADCH;
    ResultL = ADCL;
    //Reconstruct and store result
    ADC_result = ((ResultH & 0x03) << 8) | ResultL;
    VoltResult = (ADC_result * 4.96)/1024; //reference is AREF (can I read AREF pin?).
    PPM = (5000*VoltResult)/2;//need modify to do conditional calculation

    if(VoltResult > 0.4){
      printf("\nCO2: %d", PPM);
    }else if(VoltResult > 0.2){
      printf("\nCO2: preheat");
    }else{
      printf("\nCO2: fault range");
    }

    update_lcd_dog();

    SETBIT(ADCSRA,4);//clear
    CLEARBIT(ADCSRA,6);
    CLEARBIT(ADCSRA,7);//ENABLE
}
```

```
/*************************************************************************
* @file DS1306_ISR.c
* @author Xiaomin Wu
* @date April 23th
* @brief driver provides Interrupt service routine triggered by DS1306 to display time
*
* and data in 1HZ
*
*
*************************************************************************/

#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>
#include <stdio.h>

#include "DS1306_RTC_drivers.h"
#include "lcd.h"
unsigned char time_read[3];
unsigned char tempOrCO2 = 0; //0 for temp, 1 for CO2 flag. default 0

char sec;/**< least significant byte for second */
char sec2; /**< most significant byte for second */
char min; /**< least significant byte for min */
char min2; /**< most significant byte for min */
char hour; /**< least significant byte for hour */
char hour2; /**< most significant byte for hour */

#pragma vector=INT1_vect          // Declare vector location.
__interrupt void ISR_INT1(void)   // Declare interrupt function
{
  //interrupt triggered by 1HZ
  clear_dsp();        // Call to C function clear_dsp()

  block_read_RTC(time_read, 0x00, 4);//read from 0x00 for 4 bytes

  sec = time_read[0] & 0x0f;
  sec2 = (time_read[0] & 0xf0) >> 4;

  min = time_read[1] & 0x0f;
  min2 = (time_read[1] & 0xf0) >> 4;

  hour = time_read[2] & 0x0f;
  hour2 = (time_read[2] & 0x10) >> 4;

  printf("Time: %d%d%c%d%d%c%d%d\n",hour2,hour, 58, min2,min, 58, sec2,sec);

  if(!tempOrCO2){
    //measure temp and humidity
    meas_display_rh_temp(); //read humidicon and put result into display buffer

#ifndef debug
    update_lcd_dog();   // Call to C function update_lcd_dog()move buffer to LCD
#endif

  }else{
    //measure CO2
    adc_read(); //start ADC read
  }


  char r1;
  r1 = read_RTC(0x07); //clear alarm0 INT FLAG

}
```

```
/*****************************************************************************
* @file DS1306_RTC_drivers.c
* @author Xiaomin Wu
* @date April 23th
* @brief driver provides functions for ATmage128 to communicate with DS1306
*
*
*****************************************************************************/

#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>
#include <stdio.h>
#include "DS1306_RTC_drivers.h"
unsigned char RTC_byte[10]; /**< array of char to hold read result of RTC */

volatile unsigned char RTC_time_date_write[4] = {0x80,0x80,0x80,0x80}; /**< array of char to hold contents to write to RTC */
volatile unsigned char RTC_time_date_read[7]; /**< array of char to hold read result of RTC */

void write_RTC (unsigned char reg_RTC, unsigned char data_RTC);

//*****************************************************************************
// Function : void SPI_rtc_ds1306_config (void)
// Date and version : 031118, version 1.0
// Target MCU : ATmega128 @ 16MHz
// Author : Ken Short
// DESCRIPTION
// This function unselects the ds_1306 and configures an ATmega128 operated at
// 16 MHz to communicate with the ds1306. Pin PA1 of the ATmega128 is used to
// select the ds_1306. SCLK is operated a the maximum possible frequency for
// the ds1306.
//*****************************************************************************
void SPI_rtc_ds1306_config (void){
  char temp;
  CLEARBIT(PORTA,1); //unselect ds_1306
  SPCR = (0 << CPOL) | (1 << CPHA) | (1 << SPE) | (1 << SPR0) | (1 << MSTR);
  SPSR = (1 << SPI2X);

  temp = SPSR;
  temp = SPDR;

}


//*****************************************************************************
// Function :
// void write_RTC (unsigned char reg_RTC, unsigned char data_RTC)
//
// Target MCU : ATmega128 @ 16MHz
// Target Hardware ;
// Author : Ken Short
// DESCRIPTION
// This function writes data to a register in the RTC. To accomplish this, it
// must first write the register's address (reg_RTC) followed by writing the
// data (data_RTC). In the DS1306 data sheet this operation is called an SPI
// single-byte write.
//*****************************************************************************
void write_RTC (unsigned char reg_RTC, unsigned char data_RTC){
  char temp;
  SETBIT(PORTA,1); //select ds_1306
  __delay_cycles(5);
  SPDR = reg_RTC;
  while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
  temp = SPDR; //clear SPIF

  SPDR = data_RTC;
  while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
  temp =  SPDR; //clear SPIF

  CLEARBIT(PORTA,1); //DESELECT
}


//*****************************************************************************
// Function Name :
// unsigned char read_RTC (unsigned char reg_RTC)
// Target MCU : ATmega128 @ 16MHz
// Author : Ken Short
// DESCRIPTION
// This function reads data from a register in the RTC. To accomplish this, it
// must first write the register's address (reg_RTC) followed by writing a dummy
// byte to generate the SCLKs to read the data (data_RTC). In the DS1306 data
// sheet this operaration is called am SPI single-byte read.
//*****************************************************************************

unsigned char read_RTC (unsigned char reg_RTC){
  unsigned char temp;
  SETBIT(PORTA,1);
  __delay_cycles(5);
  SPDR = reg_RTC;
  while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
  temp = SPDR; //clear, may not be correct, not sure if the read should be here
               //after sending address

  SPDR = 0x11; //dummy for clock
```

```
  while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
  temp = SPDR; //data

  CLEARBIT(PORTA,1);

  return temp;
}

//*****************************************************************************
// Function Name : void write_read_RTC_test(void)
// Target MCU : ATmega128 @ 16MHz
// Author : Ken Short
// DESCRIPTION
// This function writes a byte to the NV RAM and then it reads back the location
// just written and places the result in a global array named RTC_byte[]. The
// function repeats this write/read sequencye 10 times. The locations written are
// 0xA0 through 0xA9 and the corresponding locations read are 0x20 through 0x29.
//*****************************************************************************
void write_read_RTC_test(void){
  //SPI_rtc_ds1306_config();

  write_RTC(0x8f,0x00); //configure WP to 0, without influence other bits
  write_RTC(0x8f,0x04); //enable 1HZ outpuut

  int i;
  for(i = 0;i < 10; i ++){
    write_RTC(0xA0 + i, 0x00 + i);
    RTC_byte[i] = read_RTC(0x20 + i);
  }

}
//*****************************************************************************
// Function Name : "block_write_RTC"
// void block_write_RTC (volatile unsigned char *array_ptr,
// unsigned char strt_addr, unsigned char count)
// Target MCU : ATmega128 @ 16MHz
// Author : Ken Short
// DESCRIPTION
// This function writes a block of data from an array to the DS1306. strt_addr
// is the starting address in the DS1306. count is the number of data bytes to
// be transferred and array_ptr is the address of the source array.
//*****************************************************************************
void block_write_RTC (volatile unsigned char *array_ptr, unsigned char strt_addr, unsigned char count){
  char temp;

  SPI_rtc_ds1306_config();

  SETBIT(PORTA,1); //select ds_1306
  __delay_cycles(5);
  SPDR = strt_addr;
  while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
  temp = SPDR; //clear SPIF

  int i;
  for(i = 0;i < count; i++){
    SPDR = *(array_ptr + i);
    while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
    temp =  SPDR; //clear SPIF
  }

  CLEARBIT(PORTA,1); //DESELECT
}


//*****************************************************************************
// Function Name : "block_read_RTC"
// void block_read_RTC (volatile unsigned char *array_ptr,
// unsigned char strt_addr, unsigned char count)
// Target MCU : ATmega128 @ 16MHz
// Author : Ken Short
// DESCRIPTION
// This function reads a block of data from the DS1306 and transfers it to an
// array. strt_addr is the starting address in the DS1306. count is the number
// of data bytes to be transferred and array_ptr is the address of the
// destination array.
//*****************************************************************************
void block_read_RTC (volatile unsigned char *array_ptr, unsigned char strt_addr, unsigned char count){
  unsigned char temp;

  SPI_rtc_ds1306_config();

  SETBIT(PORTA,1);
  __delay_cycles(5);
  SPDR = strt_addr;
  while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
  temp = SPDR; //clear, may not be correct, not sure if the read should be here
               //after sending address

  int i;
  for(i = 0; i<count; i++){
      SPDR = 0x11; //dummy for clock
      while(!(TESTBIT(SPSR,SPIF))); //wait till transfer done
      *(array_ptr+i) = SPDR; //data
  }
```

```
  CLEARBIT(PORTA,1);

}


void block_write_read_RTC_test(void){
  SPI_rtc_ds1306_config();

  write_RTC(0x8f,0x00); //configure WP to 0, without influence other bits
  write_RTC(0x8f,0x04); //enable 1HZ outpuut

  while(1){
    block_read_RTC(RTC_time_date_read, 0x00, 7); //read from 0x00 address, read 7 bytes
    block_write_RTC(RTC_time_date_write, 0x80, 7); //write countent to ds_1306,
                                                   //ccontent in buffer need to be assigned
  }

}


void display_time_configure(){
  SPI_rtc_ds1306_config();

  write_RTC(0x8f,0x00); //configure WP to 0, without influence other bits
  write_RTC(0x8f,0x01); //disable 1HZ outpuut, enable int0

  //congfigure alarm0 alarm every second

  block_write_RTC(RTC_time_date_write, 0x87,4);

}
```

```
/**************************************************************************
* @file DS1306 RTC driver.h
* @author Xiaomin Wu
* @date April 23th
* @brief
* Purpose:
*   The hearder file of DS1306 RTC driver provides necessary function declaration
* for other files using DS1306 RTC driver.
*
**************************************************************************/

void SPI_rtc_ds1306_config (void);
void write_read_RTC_test(void);
void block_write_read_RTC_test(void);
void block_read_RTC (volatile unsigned char *array_ptr, unsigned char strt_addr, unsigned char count);
void block_write_RTC (volatile unsigned char *array_ptr, unsigned char strt_addr, unsigned char count);
void display_time_configure();
unsigned char read_RTC (unsigned char reg_RTC);
```

```c
/*************************************************************************
* @file fsm_function.c
* @author Xiaomin Wu
* @date April 23th
* @brief Table Driven FSM User interface
* Purpose:
*   This program implements functions called by table driven fsm as outputs.
*
*************************************************************************/

#include <iom128.h> //File with register addresses for ATmega128
#include "lcd.h"
#include "humidicon.h"
#include <stdio.h>
#include "DS1306_RTC_drivers.h"

unsigned char Buffer[6] = {0,0,0,0,0,0}; /**< array to hold the input time values */
char positionCount = 0; /**< char to count the position of current time digits */

/**********************************
* value_setting
*
* this function mask out DS1306 interrupt, only keep interrupt for keypad
* it record keycode input by the keypad and store them in buffer
* function keep track of the buffer position
* function echos back the keycode inputed by user
**********************************/
void value_setting(void){
  extern char keycode;
  EIMSK = 0x01;//only enable INT0 for keypad
  clear_dsp();        // Call to C function clear_dsp()
  Buffer[positionCount] = keycode;
  positionCount ++;
  if(positionCount == 6){
    positionCount = 0;
  }

  printf("setting:\n %d%d%c%d%d%c%d%d\n",Buffer[0],Buffer[1], 58, Buffer[2],Buffer[3], 58, Buffer[4],Buffer[5]);
  update_lcd_dog();   // Call to C function update_lcd_dog()move buffer to LCD
}

/************************
*set_time
*function converts keycode saved in the buffer into mainingful time value
*and store those time value into another buffer called timeAry
*block wrtite those time value into DS_1306 register from 0x80 using SPI
*re-enable interrupts masked out by the value_setting function
*
************************/
void set_time(void){
   char timeAry[3]; /**< char array to hold time calculation result for setting */
   timeAry[0] = Buffer[0]*10 + Buffer[1];
   timeAry[1] = Buffer[2]*10 + Buffer[3];
   timeAry[2] = Buffer[4]*10 + Buffer[5];
   block_write_RTC(timeAry, 0x80, 3); //write time to time area to set

   EIMSK = 0x07;//enable INT0,INT1,INT2 generate at low level
}

/*****************************
*set_alarm
*function converts keycode saved in the buffer into mainingful time value
*and store those time value into another buffer called timeAry
*block wrtite those time value into DS_1306 register from 0x87 using SPI
*re-enable interrupts masked out by the value_setting function
*****************************/
void set_alarm(void){
   char timeAry[3]; /**< char array to hold time calculation result for setting */
   timeAry[0] = Buffer[0]*10 + Buffer[1];
   timeAry[1] = Buffer[2]*10 + Buffer[3];
   timeAry[2] = Buffer[4]*10 + Buffer[5];
   block_write_RTC(timeAry, 0x87, 3); //write time to alarm0 area to set

   EIMSK = 0x07;//enable INT0,INT1,INT2 generate at low level
}

/*****************************
* abandon_setting
* function reset the position tracker for the keycode buffer to 0
* and re-enable interrupts masked out by value_setting() function
*****************************/
void abandon_setting(void){
   positionCount = 0; //reset position pointer
   EIMSK = 0x07;//enable INT0,INT1,INT2 generate at low level
}

/*********************************
*CO2_display
* function set the tempOrCO2 flag
* to force the DS_1306 ISR to measure CO2 density and display on LCD
*
*********************************/
void CO2_display(void){
   extern unsigned char tempOrCO2;
   tempOrCO2 = 1; //to display CO2 in the DS1306_ISR
```

```
}
/*********************************
*HT_display
*function clear the tempOrCO2 flag
*to force the DS_1306 ISR to measure temperature and humidity and display on LCD
*********************************/
void HT_display(void){
   extern unsigned char tempOrCO2;
   tempOrCO2 = 0; //to display temp and humidity in the DS1306_ISR
}

/**************************
*error_fn
*function be called when eol was hit
**************************/
void error_fn(void){

}
```

```
/*************************************************************************
* @file fsm.h
* @author Xiaomin Wu
* @date April 23th
* @brief Table Driven FSM User interface header
* Purpose:
*   provide data structure and other typedefine infomation for other fsm file
*   to use
*
*************************************************************************/


#include <iom128.h> //File with register addresses for ATmega128

// states in the FSM
typedef enum{TempHum, CO2, setting}  state ;

// keys on the keypad
// eol is a psuedo key used as a default in the state table
typedef enum {digit, CO2key, time, alarm, abandon, eol} key ;

// functions to implement the task(s) associated with a state transition
// all functions must have the same signature (parameters and return type)

extern void value_setting(void);

extern void set_time(void);

extern void set_alarm(void);

extern void CO2_display(void);

extern void HT_display(void);

extern void abandon_setting(void);

extern void error_fn(void);
// global variables for measured values and settings


// declare type task_fn_ptr as a pointer to a task function
typedef void (* task_fn_ptr) ();

// A structure transition represents one row of a state transition table
// it has a field for the input key value, the next state, and a pointer
// to the task function.
// Declare type transition as a structure with fields for input key value,
// next state value, and pointer to the task function

typedef struct {
  key keyval;
  state next_state;
  task_fn_ptr tf_ptr;
} transition;
```

```
/*********************************************************************
* @file fsm_main.c
* @author Xiaomin Wu
* @date April 23th
* @brief program file contain main function for fsm driven system
*
* \mainpage Description
* finite state machine drived program
* the main file, configure pins
* initial interrupts and slaves: LCD dog and DS_1306
*
*********************************************************************/

#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>
#include <stdio.h>

#include "DS1306_RTC_drivers.h"
#include "lcd.h"
#include "fsm.h"

#define debug

extern state present_state;
extern char keycode;

extern void meas_display_rh_temp(void);
extern void fsm (state ps, key keyval);
extern key lookKey(char KeyBoardcode);

void main(void){

  // Configure PortD for INT0 interrupt.
  DDRD = 0xF8;                    // INT0 input, INT1 input, INT2 input
  PORTD = 0x07;               // INT0, int1, int2 pullup enabled

  // Configure PortC for keypad, initial configuration
  DDRC = 0xF0;         // High nibble outputs, low nibble inputs
  PORTC = 0x0F;
  //low level sensative interruupt



  // Configure PortB for DOG LCD module's SPI interface
  DDRB = 0xFF;              // Set PortB to outputs
  SETBIT (PORTB, 0);        // unassert slave select, PB0 = 1

  DDRA = 0xFF;              // set PortA to outputs, PA0 SELECT humidicon lowlevel SE
  PORTA = 0x01;             // unselect both, PA1 selet SD1306 highlevel SE
  MCUCR = 0x30;      // Sleep enabled for power down mode.

  EIMSK = 0x07;//enable INT0,INT1,INT2 generate at low level



#ifndef debug
  init_lcd_dog();      // Call to asm subroutine init_dsp()
  clear_dsp();         // Call to C function clear_dsp()
  update_lcd_dog();    // Call to C function update_lcd_dog(),include init_lcd_dog
#endif

#ifndef debug
  display_time_configure(); //initial DS_1306
#endif

  adc_initial(); //initial ADC module


  present_state = TempHum; //initial present state

  __enable_interrupt();           //Enable global interrupts.

  while(1){


    __delay_cycles (1000);
  }
}
```

```
/*********************************************************************
*
*
* @file fsm_table.c
* @author Xiaomin Wu
* @date April 23th
* @brief Table Driven FSM User interface
* Purpose:
*   This program implements tables for table driven fsm to use.
*
*********************************************************************/


#include "fsm.h"


// The state transition table consists of an array of arrays of structures.
// Each array of structures corresponds to a particular present state value.
// Each structure in such an array corresponds to a transition from the
// state for a given input value and the task function associated with the
// transition. Accordingly, each structure in an array has fields
// corresponding to an input value, the next state for this input value,
// and a pointer to the function task for this input value.
// The last transition structure in each array has a keyval field value
// of eol. This is a default value meaning any key value that has not
// been explcitly listed in a previous transition structure in the array.

const transition TempHum_transition[] =  // subtable for disp_temp state
{
//  INPUT       NEXT_STATE      TASK
    {digit,       setting,       value_setting},
    {CO2key,       CO2,       CO2_display},
    {eol,         TempHum,          error_fn}
};

const transition CO2_transition[] =  // subtable for disp_temp state
{
//  INPUT       NEXT_STATE      TASK
    {digit,       setting,       value_setting},
    {CO2key,       TempHum,       HT_display},
    {eol,         TempHum,          error_fn}
};

const transition setting_transition[] =    // subtable for disp_hum state
{
//  INPUT       NEXT_STATE      TASK
    {digit,      setting,      value_setting},
    {time,       TempHum,         set_time},
    {alarm,      TempHum,         set_alarm},
    {abandon,    TempHum,         abandon_setting},
    {eol,        TempHum,         error_fn}
};

// The outer array is an array of pointers to an array of transition
// structures for each present state.

const transition * ps_transitions_ptr[3] =
{
 TempHum_transition,
 CO2_transition,
 setting_transition
};
```

```
/*********************************************************************
* @file fsm_ui.c
* @author Xiaomin Wu
* @date April 23th
* @brief Table Driven FSM User interface
* Purpose:
*   This program implements a table driven FSM that process keypress events
*   from a keypad.
*
*********************************************************************/

#include <iom128.h> //File with register addresses for ATmega128
#include "fsm.h"

//extern const transition idle_transition[];
extern const transition * ps_transitions_ptr[4];
// global variable for present state of FSM
state present_state; /**< state type variable to represent the present state of FSM */
// The finite state machine is implemented as a fumction with parameters
// corresponding to the present state and key value that has been input.

void fsm (state ps, key keyval)
{
// Search the array of transition structures corresponding to the
// present state for the transition structure that has has keyvalue
// field value that is equal to current input key value or equal
// to eol.

int i;
for (i = 0; (ps_transitions_ptr[ps][i].keyval != keyval)
     && (ps_transitions_ptr[ps][i].keyval != eol); i++);

// i now has the value of the index of the transition structure
// corresponding to the current intput key value.

// Invoke the task function pointed to by the task function pointer
// of the current transition structure.

ps_transitions_ptr[ps][i].tf_ptr();

// Make present state equal to the next state value of the current
// transition structure.

present_state = ps_transitions_ptr[ps][i].next_state;

}
```

```
/*****************************************************************************
* @file humiodicon.c
* @author Xiaomin Wu
* @date April 23th
* @brief humiodicon
* DESCRIPTION
* contain three functions
* SPI_humidicon_config
*                      configure the SPI to communicate with HumidIcon
* unsigned char read_humidicon_byte
*                      read single byte from HumidIcon. return as unsigned char
* read_humidicon
*                      read humidicon and temperature raw data and put in global
*                      variables
*****************************************************************************/

#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>


unsigned int humidity; /**< unsigned int variable to hold humidity result */
unsigned int temperature; /**< unsigned int variable to hold temperature result */

extern unsigned int compute_scaled_temp(unsigned int temp);
extern unsigned int compute_scaled_rh(unsigned int rh);

//*****************************************************************************
// Function : void SPI_humidicon_config (void)
// Date and version : version 1.0
// Target MCU : ATmega128A @ 16MHz
// Author :
// DESCRIPTION
// This function unselects the HumidIcon and configures it for operation with
// an ATmega128A operated a 16 MHz. Pin PA0 of the ATmega128A is used to select
// the HumidIcon
//
// Modified
//*****************************************************************************
void SPI_humidicon_config (void){
  char temp; /**< char variable used to access SPSR and SPDR, result not used */
  //unselect PA0
  PORTA = PORTA | 0x01;
  //unselect LCD ss
  PORTB = PORTB | 0x01; //  /SS = deselected

  //CPOL = 0; CPHA = 0; DORD = 0;msb first. SPR1 = 1, SPR0 = 0 for 250KHZ
  SPCR = (1<<SPE) | (1<<MSTR) | (1<<CPOL)|(1<<CPHA)| (1<<SPR1);
  //SPI2X = 1 for 500KHZ
  SPSR = (1<<SPI2X);
  //kill any spurious data...
  temp = SPSR;   //clear SPIF bit in SPSR
  temp = SPDR;
}

//*****************************************************************************
// Function : unsigned char read_humidicon_byte(void)
// Date and version : version 1.0
// Target MCU : ATmega128A
// Author : Ken Short
// DESCRIPTION
// This function reads a data byte from the HumidIcon sensor and returns it as
// an unsigned char. The function does not return until the SPI transfer is
// completed. The function determines whether the SPI transfer is complete
// by polling the appropriate SPI status flag.
//
// Modified
//*****************************************************************************
unsigned char read_humidicon_byte(void){

  unsigned char tempRead;/**< unsigned char variable to hold read result */

  //DF command
  SPDR = 0x12; //dummy send to start clock
  while(!((SPSR | 0x7F)&0x80));//Wait for transmission complete
  //tempRead = SPSR; //clear SPIF bit
  tempRead = SPDR; //read data from slave

  return tempRead;
}




//*****************************************************************************
// Function : void read_humidicon (void)
// Date and version : version 1.0
// Target MCU : ATmega128A
// Author :
// DESCRIPTION
// This function selects the Humidicon by asserting PA0. It then calls
// read_humidicon_byte() four times to read the temperature and humidity
// information. Is assigns the values read to the global unsigned ints humidicon_byte1,
// humidion_byte2, humidion_byte3, and humidion_byte4, respectively. The
// function then deselects the HumidIcon.
```

```
//
// The function then extracts the fourteen bits corresponding to the humidity
// information and stores them right justified in the global unsigned int humidity_raw.
// Next if extracts the fourteen bits corresponding to the temperature
// information and stores them in the global unsigned int temperature_raw. The function
// then returns
//
// Modified
//*****************************************************************************
void read_humidicon (void){
  unsigned int humidicon_byte1; /**< unsigned int variable to hold least significant byte for humidicon */
  unsigned int humidicon_byte2; /**< unsigned int variable to hold second byte for humidicon */
  unsigned int humidicon_byte3; /**< unsigned int variable to hold third byte for humidicon */
  unsigned int humidicon_byte4; /**< unsigned int variable to hold most significant byte for humidicon */
  unsigned int humidity_raw; /**< unsigned int variable to hold raw result of humidity */
  unsigned int temperature_raw; /**< unsigned int variable to hold raw result of temperature */
  unsigned char temp; /**< variable used to accesee register, result not used */
  PORTB = PORTB | 0x01; //  /SS = deselected
  PORTA = PORTA & 0xFE; //select slave
  //MR command
  //SPDR = 0x11; //writting a byte toSPDR starts the SPI clock generator, read bytes
  //while(!((SPSR | 0x7F)&0x80));//Wait for transmission complete
  //temp = SPSR;              //clear SPIF bit in SPSR.
  //temp = SPDR;

  humidicon_byte1 = read_humidicon_byte();
  humidicon_byte2 = read_humidicon_byte();
  humidicon_byte3 = read_humidicon_byte();
  humidicon_byte4 = read_humidicon_byte();

  //unselect PA0
  PORTA = PORTA | 0x01;

  //data calculating
  humidity_raw = ((humidicon_byte1 & 0x3f) << 8) | humidicon_byte2; //humidicon
  temperature_raw = (humidicon_byte3 << 6) | ((humidicon_byte4 & 0xfc) >> 2);//tem

  humidity = compute_scaled_rh(humidity_raw);
  temperature = compute_scaled_temp(temperature_raw);

}
```

```
/*****************************************************************************
* @file humiodicon.c
* @author Xiaomin Wu
* @date April 23th
* @brief humiodicon
* DESCRIPTION
* contain two functions declaration
* SPI_humidicon_config
*                      configure the SPI to communicate with HumidIcon
*
* read_humidicon
*                      read humidicon and temperature raw data and put in global
*                     variables
*
*****************************************************************************/

void read_humidicon (void);
void SPI_humidicon_config (void);
```

```
/*************************************************************************
* @file Interrupt2ISR.c
* @author Xiaomin Wu
* @date April 23th
* @brief Keypad scan interrupt service routine
* DESCRIPTION
* response to INT2
*************************************************************************/

#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>
#include <stdio.h>

/*
* Interrupt service routine
*/
#pragma vector=INT2_vect          // Declare vector location.
__interrupt void ISR_INT2(void)   // Declare interrupt function
{
  PORTA = 0x00;
}
```

```
/*****************************************************************************
* @file keyscan_isr.c
* @author Xiaomin Wu
* @date April 23th
* @brief Keypad scan interrupt service routine
* DESCRIPTION
* When keypad interrupt occurs, key matirx is scanned and is encoded using
* a table lookup. The keypad is connected to PORTC. See diagram in laboratory
* description.
*****************************************************************************/

#define debug   //this can be uncommented to remove delays for simulation

#include <iom128.h>         //Atmega128 definitions
#include <intrinsics.h>     //Intrinsic functions.
#include <avr_macros.h>     //Useful macros.
#include <stdio.h>
#include "fsm.h"

/*
* Port pin numbers for columns and rows of the keypad
*/
//PORT Pin Definitions.
#define COL1  7   //pin definitions for PortB
#define COL2  6
#define COL3  5
#define COL4  4
#define ROW1  3
#define ROW2  2
#define ROW3  1
#define ROW4  0

#define INT0  0   //pin definitions for PortD



/*
* Function declarations.
*/
void check_release(void);
key lookKey(char KeyBoardcode);
/*
* Lookup table declaration
*/
const char tbl[16] = {1, 2, 3, 'F', 4, 5, 6, 'E', 7, 8, 9, 'D', 'A', 0, 'B', 'C'}; /**< keypad lookup table */

//*****************************************************************************
//Code

char keycode;       /**< char to hold input keycode */
key keypressed;     /**< key type variable to hold input key for keypressed */
/*
* Interrupt service routine
*/
#pragma vector=INT0_vect          // Declare vector location.
__interrupt void ISR_INT0(void)   // Declare interrupt function
{
  extern state present_state;

  //Note: TESTBIT returns 0 if bit is not set and a non-zero number otherwise.

  if(!TESTBIT(PINC,ROW1))         //Find Row of pressed key.
    keycode = 0;
  else if(!TESTBIT(PINC,ROW2))
    keycode = 4;
  else if(!TESTBIT(PINC,ROW3))
    keycode = 8;
  else if(!TESTBIT(PINC,ROW4))
    keycode = 12;

  DDRC = 0x0F;                    //Reconfigure PORTC for Columns.
  PORTC = 0xF0;

#ifndef debug
  __delay_cycles(256);            //Let PORTC settle.
#endif

  if(!TESTBIT(PINC,COL1))         //Find Column.
    keycode += 0;
  else if(!TESTBIT(PINC,COL2))
    keycode += 1;
  else if(!TESTBIT(PINC,COL3))
    keycode += 2;
  else if(!TESTBIT(PINC,COL4))
    keycode += 3;

  DDRC = 0xF0;                    //Reconfigure PORTC for Rows.
  PORTC = 0x0F;

  keycode = (tbl[keycode]);
  check_release();                //Wait for keypad release.

  keypressed = lookKey(keycode);
  fsm (present_state, keypressed);

}
```

```
/*
*  Check keypad is released and not bouncing.
*/
void check_release(void)
{
#ifndef debug
  while(!TESTBIT(PIND,INT0));     //Check that keypad key is released.

  __delay_cycles(50000);          //Delay (.05secs) / (1 / 1MHz) cycles.

  while(!TESTBIT(PIND,INT0));     //Check that key has stopped bouncing.
#endif
  return;
}

key lookKey(char KeyBoardcode){
  if((int)KeyBoardcode < 10){
    return digit;

  }else if((int)KeyBoardcode == 15){
    return time;
  }else if ((int)KeyBoardcode == 14){
    return alarm;
  }else if((int)KeyBoardcode == 12){
    return abandon;
  }else if((int)KeyBoardcode == 13){
    return CO2key;
  }else{
    return eol;
  }
}
```

```
/************************************************************** v1.1a
;   ATMega128 Version:               PRINT IN LANDSCAPE
;
; revised 01/30/10
;
;   This AVR-asm code module is usable as an include file for assembly
;   language and or mixed asm/C application programs. The code is freely
;   usable by any University of Stonybrook undergraduate students for any
;   and all not-for-profit system designs and or implementations.
;
;   This code is designed to be executed on an AVR ATMega micro-computer.
;   And may be readily adapted for compatibility with IAR/AVR compilers.
;   See the IAR assembler reference guide for more information by
;   clicking 'Help > AVR Assembly Reference Guide" on the above menus.
;
;
;*****************************************************************
;
;   This module contains procedures to initialize and update
;   DOG text based LCD display modules, including the EA DOG163M LCD
;   modules configured with three (3) 16 charactors display lines.
;
;   The display module hardware interface uses a 1-direction, write only
;   SPI interface. (See below for more information.)
;
;   The display module software interface uses three (3) 16-byte
;   data (RAM) based display buffers - One for each line of the display.
;   (See below for more information.)


;
;*****************************************************************
;
;   *** Port B Interface Definitions:
;
;
;  Port B alt names   SCK   MISO  MOSI  /SS    RS    -     -     -
;  LCD Mod Signal     D6     -     D7   /CSB   -     -     -     -
;  LCD Mod Pin #      29     -     28    38    -     -     -     -
;
;      Notes:  RS ==>  0 = command regs, 1 = data regs
;             /SS = active low SPI select signal
;
;*****************/


/*************************************************************************
* @file lcd_dog_iar_driver.c
* @author Xiaomin Wu
* @date April 23th
* @brief driver file for LCD module
* DESCRIPTION
* provide necessary functions to operate lcd_dog hardware
*************************************************************************/

#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>


// All functions used
void init_lcd_dog(void);
void update_lcd_dog(void);

#define SCK     1
#define MISO    3
#define MOSI    2
#define SS_bar  0
#define RS      4
#define BLC     5

#define clk_speed   1   // clock speed in MHz



//***  DATA Segment  ************************************************
//Puts display buffers in near initialize to zero segment.
//See chapter on segments in IAR reference guide

char dsp_buff_1[16], dsp_buff_2[16], dsp_buff_3[16];


//***  CODE Segment Subroutines  ***********************************
//Puts object code in CODE segment.


//************************
//NAME:       init_spi_lcd
//ASSUMES:    IMPORTANT: PortB set as output (during program init)
//RETURNS:    nothing
//MODIFIES:   DDRB, SPCR
//CALLED BY:  init_dsp, update
//DESCRITION: init SPI port for command and data writes to LCD via SPI
//*****************************************************************
void init_spi_lcd(void){
  // Enable SPI, Master, fck/64,
```

```
  char temp;
  SPCR = (1<<SPE) | (1<<MSTR) | (1<<CPOL) | (1<<CPHA) | (1<<SPR1) | (1<<SPR0);
  //kill any spurious data...
  temp = SPSR;   //clear SPIF bit in SPSR
  temp = SPDR;
}


//*******************************
//NAME:       lcd_spi_transmit_CMD
//ASSUMES:    r16 = byte for LCD.
//            SPI port is configured.
//RETURNS:    nothing
//MODIFIES:   R16, PortB, SPCR
//CALLED BY:  init_dsp, update
//DESCRITION: outputs a byte passed in r16 via SPI port. Waits for data
//            to be written by spi port before continuing.
//*********************************************************************
void lcd_spi_transmit_CMD(char cmd){
     char temp;

     PORTB = PORTB & 0xEE; //RS = 0 = command./SS = selected.

     SPDR = cmd;           //write data to SPI port.

     while(!((SPSR | 0x7F)&0x80));//Wait for transmission complete

     //clear SPIF bit in SPSR
     temp = SPDR;          //read of SPSR followed by read of SPDR clears SPIF 013110 kls

     PORTB = PORTB | 0x01; //  /SS = deselected
}


//*******************************
//NAME:       lcd_spi_transmit_DATA
//ASSUMES:    r16 = byte to transmit to LCD.
//            SPI port is configured.
//RETURNS:    nothing
//MODIFIES:   R16, SPCR
//CALLED BY:  init_dsp, update
//DESCRITION: outputs a byte passed in r16 via SPI port. Waits for
//            data to be written by spi port before continuing.
//*************************************************************
void lcd_spi_transmit_DATA(char data){
      char temp;

      PORTB = PORTB | 0x10;    // RS = 1 = data. bit 4
      PORTB = PORTB & 0xFE;    // /SS = selected.
      temp = SPSR;             //clear SPIF bit in SPSR.
      temp = SPDR;

      SPDR = data;             //write data to SPI port.

      while(!((SPSR | 0x7F)&0x80));//Wait for transmission complete

      //clear SPIF bit in SPSR
      temp = SPDR;          //read of SPSR followed by read of SPDR clears SPIF 013110 kls

      PORTB = PORTB | 0x01; //  /SS = deselected
}

//************************
//NAME:       init_lcd_dog
//ASSUMES:    nothing
//RETURNS:    nothing
//MODIFIES:   R16, R17
//CALLED BY:  main application
//DESCRITION: inits DOG module LCD display for SPI (serial) operation.
//NOTE:  Can be used as is with MCU clock speeds of 4MHz or less.
//*********************************************************************


void init_lcd_dog(void){
       init_spi_lcd();   // init SPI port for DOG LCD.
      __delay_cycles(40000*clk_speed); //delay 40ms under 1MHZ

       lcd_spi_transmit_CMD(0x39); //send fuction set #1
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

       lcd_spi_transmit_CMD(0x39); //send fuction set #2
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

       lcd_spi_transmit_CMD(0x1E); //send bias set
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

       lcd_spi_transmit_CMD(0x50); // 0x50 nominal (delicate adjustment).
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

       lcd_spi_transmit_CMD(0x6C);// follower mode on...
      __delay_cycles(40000*clk_speed); //delay 40ms under 1MHZ

       lcd_spi_transmit_CMD(0x77); //7C was too bright
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

       lcd_spi_transmit_CMD(0x0C); //display on, cursor off, blink off
```

```
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

      lcd_spi_transmit_CMD(0x01);//clear display, cursor home
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

      lcd_spi_transmit_CMD(0x06); //clear display, cursor home
      __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us

}



//*************************
//NAME:       update_lcd_dog
//ASSUMES:    display buffers loaded with display data
//RETURNS:    nothing
//MODIFIES:   R16,R20,R30,R31,SREG
//
//DESCRITION: Updates the LCD display lines 1, 2, and 3, using the
//  contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.
//*********************************************************************
// public __version_1 void update_dsp_dog (void)
void update_lcd_dog(void){
       init_spi_lcd(); //init SPI port for LCD.
       int count;
       char* Zptr;

//send line 1 to the LCD module.
       Zptr = dsp_buff_1;
       count = 16;// init 'chars per line' counter.
//snd_ddram_addr
       lcd_spi_transmit_CMD(0x80); //init DDRAM addr-ctr
       __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us
//sed_data
       while(count > 0){

         lcd_spi_transmit_DATA(*Zptr);
         Zptr++;

         __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us
         count--;
       }


//send line 2 to the LCD module.
      Zptr = dsp_buff_2;
      count = 16;// init 'chars per line' counter.

//snd_ddram_addr2:
       lcd_spi_transmit_CMD(0x90); //init DDRAM addr-ctr
       __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us
//snd_buff_2:
      while(count > 0){

         lcd_spi_transmit_DATA(*Zptr);
         Zptr++;

         __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us
         count--;
       }


//send line 3 to the LCD module.
      Zptr = dsp_buff_3;
      count = 16;// init 'chars per line' counter.

//snd_ddram_addr3:
       lcd_spi_transmit_CMD(0xA0); //init DDRAM addr-ctr
       __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us
//snd_buff_3:
      while(count > 0){

         lcd_spi_transmit_DATA(*Zptr);
         Zptr++;

         __delay_cycles(255*clk_speed);    // delay for command to be processed, any delay > 30us
         count--;
       }

}
//******* End Of Module *********************************
```

```
//***********************************************************************
//
// File Name            : lcd_ext.c
// Title                : LCD Utilities
// Date                 : 02/07/10
// Version              : 1.0
// Target MCU           : ATmega128 @  MHz
// Target Hardware      ;
// Author               : Ken Short
// DESCRIPTION
// The file contains two functions that make it easier for a C
// program to use the LCD display. The function clear_dsp() clears the display
// buffer arrays. When followed by the function update_dsp(), the
// display is blanked.
//
// The function putchar() puts a single character, passed to it as an argument,
// into the display buffer at the position corresponding to the value of
// variable index. This putchar function replaces the standard putchar funtion,
// so a printf statement will print to the LCD
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//***********************************************************************



/***********************************************************************
* @file lcd_ext.c
* @author Ken Short
* @date April 23th
* @brief driver file for LCD module
* DESCRIPTION
* The function putchar() puts a single character, passed to it as an argument,
* into the display buffer at the position corresponding to the value of
* variable index. This putchar function replaces the standard putchar funtion,
* so a printf statement will print to the LCD
***********************************************************************/


#include "lcd.h"


static char index;    // index into display buffer

//***************************************************************************
// Function             : void clear_dsp(void)
// Date and version     : 02/07/10, version 1.0
// Target MCU           : ATmega128
// Author               : Ken Short
// DESCRIPTION
// Clears the display buffer. Treats each 16 character array separately.
// NOTE: update_dsp must be called after to see results
//
// Modified
//***************************************************************************

void clear_dsp(void)
{
  // assuming buffers might not be contiguous
  for(char i = 0; i < 16; i++)
    dsp_buff_1[i] = ' ';

  for(char i = 0; i < 16; i++)
    dsp_buff_2[i] = ' ';

  for(char i = 0; i < 16; i++)
    dsp_buff_3[i] = ' ';

  index = 0;
}


//***************************************************************************
// Function             : int putchar(int c)
// Date and version     : 02/17/18, version 1.0
// Target MCU           : ATmega128
// Author               : Xiaomin Wu
// DESCRIPTION
// This function displays a single ascii chararacter c on the lcd at the
// position specifieb by the global variable index
// also contain functional inputs: \b, \f, \n, \r
// NOTE: update_dsp must be called after to see results
//
//
// Modified
//***************************************************************************

int putchar(int c)
```

```
{
  switch((char)c){

  case '\n' :
    if (index < 16)
      index = 16;
    else if (index < 32)
      index = 32;
    else
      index = 0;

    break;

  case '\b' :
    if (index <= 16 && index > 0){
      index--;
      dsp_buff_1[index] = ' ';
    }
    else if (index <= 32){
      index--;
      dsp_buff_2[index - 16] = ' ';
    }
    else if (index <= 48){
      index--;
      dsp_buff_3[index - 32] = ' ';
    }
    break;

  case '\f' :
    clear_dsp();
    index = 0;
    break;

  case '\r' :
    if (index < 16)
      index = 0;
    else if (index < 32)
      index = 16;
    else if (index < 48)
      index = 32;

    break;

  default :
    if (index < 16)
      dsp_buff_1[index++] = (char) c;

    else if (index < 32)
      dsp_buff_2[index++ - 16] = (char) c;

    else if (index < 48)
      dsp_buff_3[index++ - 32] = (char) c;

    else
    {
    clear_dsp();
    index = 0;
    dsp_buff_1[index++] = (char)c;
    }

  }

  return(int) c;
}
```

```
//***********************************************************************
//
// File Name            : lcd.h
// Title                : Header file for LCD module
// Date                 : 02/07/10
// Version              : 1.0
// Target MCU           : ATmega128 @  MHz
// Target Hardware      ;
// Author               : Ken Short
// DESCRIPTION
// This file includes all the declaration the compiler needs to
// reference the functions and variables written in the files lcd_ext.c.
// lcd.asm and lcd_dog_iar_driver.asm
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//***********************************************************************

/***********************************************************************
* @file lcd.h
* @author Xiaomin Wu
* @date April 23th
* @brief Header file for LCD module
* DESCRIPTION
* This file includes all the declaration the compiler needs to
* reference the functions and variables written in the files lcd_ext.c.
* lcd.asm and lcd_dog_iar_driver.asm
***********************************************************************/

/**
 *  To use the lcd functions in lcd_dog_iar_driver.asm lcd_ext.c all that is
 *  needed is to include this file.
 */

/**
 *  This declaration tells the compiler to look for dsp_buff_x in
 *  another module. It is used by lcd_ext.c and main.c to locate the buffers.
 */
extern char dsp_buff_1[16];
extern char dsp_buff_2[16];
extern char dsp_buff_3[16];

/**
 *  Declaratios of low level lcd functions located in lcd_dog_iar_driver.asm
 *  Note that these are external.
 */
 extern void init_lcd_dog(void);
 extern void update_lcd_dog(void);

/**
 *  These functions are located in lcd_ext.c
 */
extern void clear_dsp(void);
extern int putchar(int);
```

```
//*****************************************************************************
//
// File Name            : temp_humid_humidicon.c
// Title                : temp_humid_humidicon
// Date                 : 03/05/18
// Version              : 1.0
// Target MCU           : ATmega128 @ 16 MHz
// Target Hardware      ;
// Author               : Xiaomin Wu
// DESCRIPTION
// contain three functions
//
// unsigned int compute_scaled_rh(unsigned int rh)
//      function take a unsigned int parameter and Computess scaled relative
//      humidity in units of 0.01% RH from the raw 14-bit
//      realtive humidity value from the Humidicon.
//      return as unsigned int
//
// unsigned int compute_scaled_temp(unsigned int temp)
//      function take a unsigned int parameter and Computess scaled relative
//      temperature in units of 0.01°C RH from the raw 14-bit
//      realtive humidity value from the Humidicon.
//      return as float
// void meas_display_rh_temp(void)
//      function configure SPI for humidIcon and read humidIcon
//      print result temperature and humidity to display buffers
//      as scaled integer number
//
// Warnings             : temp not used
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//*****************************************************************************

/*****************************************************************************
* @file temp_humid_humidicon.c
* @author Xiaomin Wu
* @date April 23th
* @brief driver file for LCD module
* DESCRIPTION
* contain three functions
*
* unsigned int compute_scaled_rh(unsigned int rh)
*      function take a unsigned int parameter and Computess scaled relative
*      humidity in units of 0.01% RH from the raw 14-bit
*      realtive humidity value from the Humidicon.
*      return as unsigned int
*
* unsigned int compute_scaled_temp(unsigned int temp)
*      function take a unsigned int parameter and Computess scaled relative
*      temperature in units of 0.01°C RH from the raw 14-bit
*      realtive humidity value from the Humidicon.
*      return as float
* void meas_display_rh_temp(void)
*      function configure SPI for humidIcon and read humidIcon
*      print result temperature and humidity to display buffers
*      as scaled integer number
*****************************************************************************/

#include <stdio.h>
#include <iom128.h>
#include <intrinsics.h>
#include <avr_macros.h>

#include "humidicon.h"
#include "lcd.h"

extern unsigned int humidity;
extern unsigned int temperature;

//*****************************************************************************
// Function : unsigned int compute_scaled_rh(unsigned int rh)
// Date and version : version 1.0
// Target MCU : ATmega128A
// Author : Xiaomin Wu
// DESCRIPTION
// Computess scaled relative humidity in units of 0.01% RH from the raw 14-bit
// realtive humidity value from the Humidicon.
//
//
// Modified
//*****************************************************************************

unsigned int compute_scaled_rh(unsigned int rh){
  return (unsigned int)(((long)rh * 100 * 100)/((1 << 14) - 2));
}

//*****************************************************************************
// Function : unsigned int compute_scaled_temp(unsigned int temp)
// Date and version : version 1.0
// Target MCU : ATmega128A
```

```
// Author : Xiaomin Wu
// DESCRIPTION
// Computess scaled temperature in units of 0.01 degrees C from the raw 14-bit
// temperature value from the Humidicon
//
//
// Modified
//***************************************************************************

unsigned int compute_scaled_temp(unsigned int temp){
  return (unsigned int)(((((long)temp * 100 *165) / ((1 << 14) - 2))) - (40*100));
}

//***************************************************************************
// Function : unsigned int compute_scaled_temp(unsigned int temp)
// Date and version : version 1.0
// Target MCU : ATmega128A
// Author : Xiaomin Wu
// DESCRIPTION
// configure the HumidIcon for a reading
// get scaled temperature and humidity values
// then displays the results on a DOG 3 x 16 LCD
//
// Modified
//***************************************************************************
void meas_display_rh_temp(void){
   int MS; /**< int variable to hold MS nibble */
   int LS; /**< int variable to hold LS nibble */

   SPI_humidicon_config();
   read_humidicon();

   MS = temperature / 100;
   LS = temperature % 100;


   printf("Temp: %d.%d%cC\n", MS, LS,223);

   MS = humidity / 100;
   LS = humidity % 100;

   printf("RH: %d.%d%c", MS, LS,37);
}
```