Coursework 2016/2017

Designing, Developing, and Testing a Museum Showcase Controller

The embedded application we have implemented is: **Controller integrating accessibility management, security, light dimmer and inside environment monitor**.

Author:
Zhili Shao

Supervisor:
Richard Anthony
Jon-Vegard Sørli

Team:
Badis Madani
Zhili Shao

Table of Contents

Contents

Part A

1. Introduction

The subject for the coursework for Modern Embedded System Programming is a museum showcase controller where many issues with museum are to be considered such as security, environment and lighting;

- Higher security is important issue with museums and additional electronic security is much important for the overall security.
- Environment and air circulation control are major issues in museums. Needing cases with a low air exchange rate can exacerbate the need for conservatorial sound materials inside a case. At the very least, cases should be designed with dust seals to reduce housekeeping issues. And they should control the environment to keep the museum case at a certain temperature range and a certain humidity level even when the surrounding museum environment varies slightly from day to day. Usually maintained at 45% RH + 8% RH for humidity and 21°C and 14°C for temperature.
- Lighting the collection is a tricky proposition. In On one hand the collection should be well-lit for visitors to see, but the act of lighting adds a great deal of issues such as ultraviolet and infrared degradation as well as heat that can harm the collection.

Based on the points cited above, a controller integrating accessibility management, security, light dimmer and inside environment monitor functions for the future museum showcase is designed, developed and tested.

2. Design and Development

2.1. Functional requirements of the showcase control system

a. Functional requirements

- Unlock/lock the case using Employee's card.
- Display internal environment (temperature and humidity) on the LCD.
- A colour LED should show the different states of the showcase (normal, abnormal)
- Adjust brightness of lights by remote control.
- Activate/disactivate Alarm for security.

b. Non-Functional requirements

- Must not warm up the case excessively (LED brightness).

2.2. Selecting sensors, inputs, and outputs

1. An **RFID-RC522** is needed to read the employee card for the access management to the showcase and communicate with the microcontroller and requires **4 digital I/O pins**.

2. **LCD1602** is needed to display the status of temperature and humidity inside the showcase and the brightness of the lighting, as well as the door and alarm states, it requires **2 digital output pins**.

3. The **Servo** is need to simulate the lock and it requires a **single digital output pin**.

4. The light **LED** will be used for the lighting in the showcase and it requires a **single digital output pin**.

5. The **three-color LED** (RGB) is needed to show the status of the showcase controller and it requires **3 digital output pins**.

6. The **temperature & humidity sensor (DH11)** is needed to measure the temperature and humidity levels inside the showcase. It requires a **single digital input**.

7. The **security sensor (Button)** is used as the On/Off of the alarm buzzer and it requires a **single digital input pin**. The security sensor (button) will be inversed in the prototype, so it needs to be pushed to activate the alarm, and the remote controller will be used to disactivate the alarm for specific situations such as housekeeping or cleaning the case.

8. The **alarm Buzzer** requires a **single digital output pin**.

9. The **remote control (IR resistor)** requires a **single digital input pin**.

2.3. I/O ports and interfacing

1. **RFID-RC522** will communicate through SPI (Port B bit 0, bit 1, bit 2 and bit 3)

2. **LCD1602** must use I2C protocol (Port D bit 0, and bit 1)

3. The **Servo** must be connected to PWM output for its control, the Port E, bit3 will be used, and Timer3A

4. The light **LED** must be connected to a PWM output so that their brightness can be changed. OC4B, Port H, bit 4.

5. The **three color LED** (RGB) will be connected to Port L bit 0, 1, 2 respectively.

6. The **temperature & humidity sensor (DH11)** is connected to Port B, bit 4.

7. The **security sensor (Button)** is connected to Port D, bit 2.

8. The **alarm Buzzer** is connected to Port A, bit 1.

9. The **remote control (IR resistor)** will be connected to Port D bit 3, Timer1A, ExtInt3.

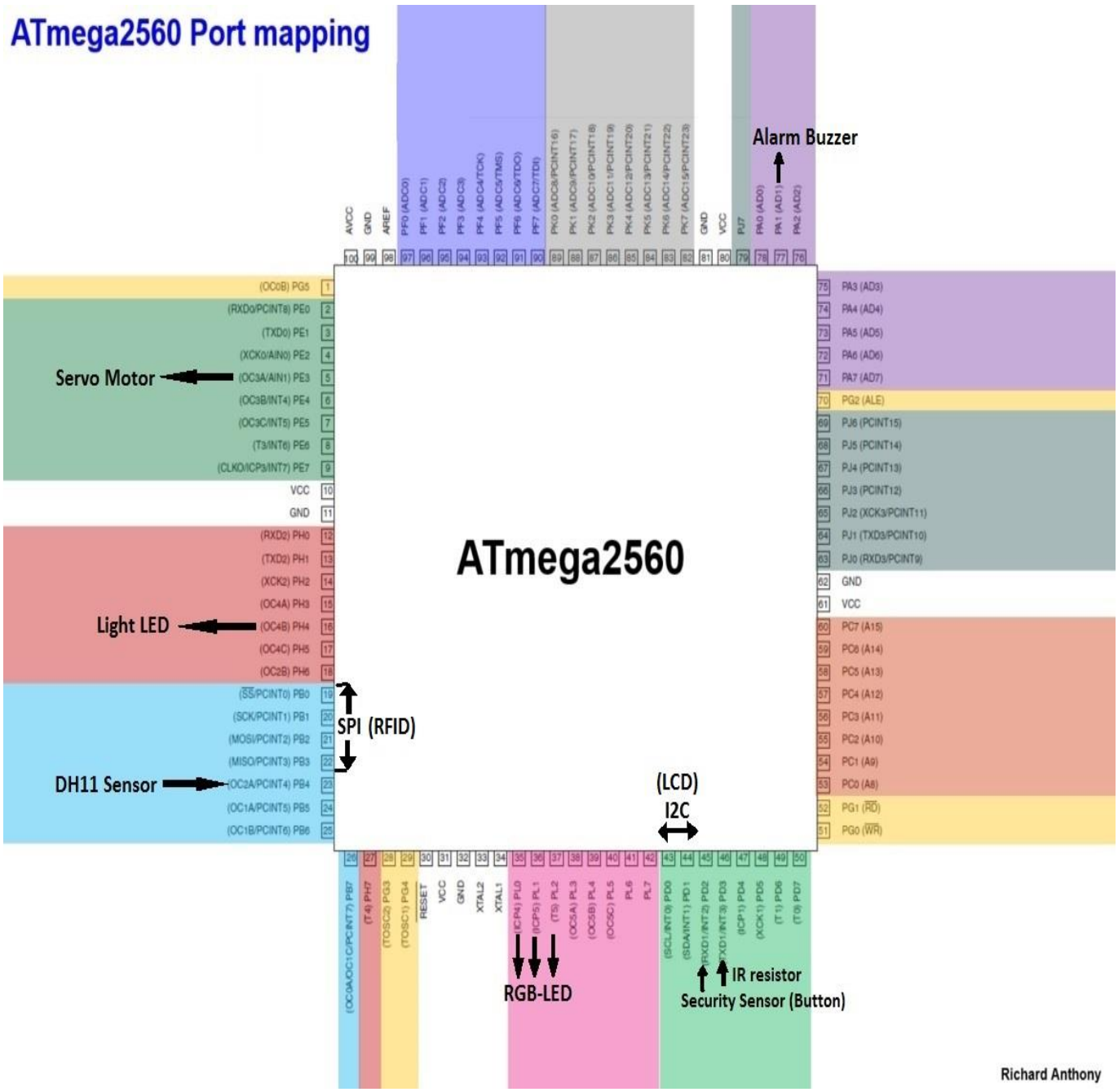

Figure 1. I/O ports and interfacing-ATmega2560 Port mapping

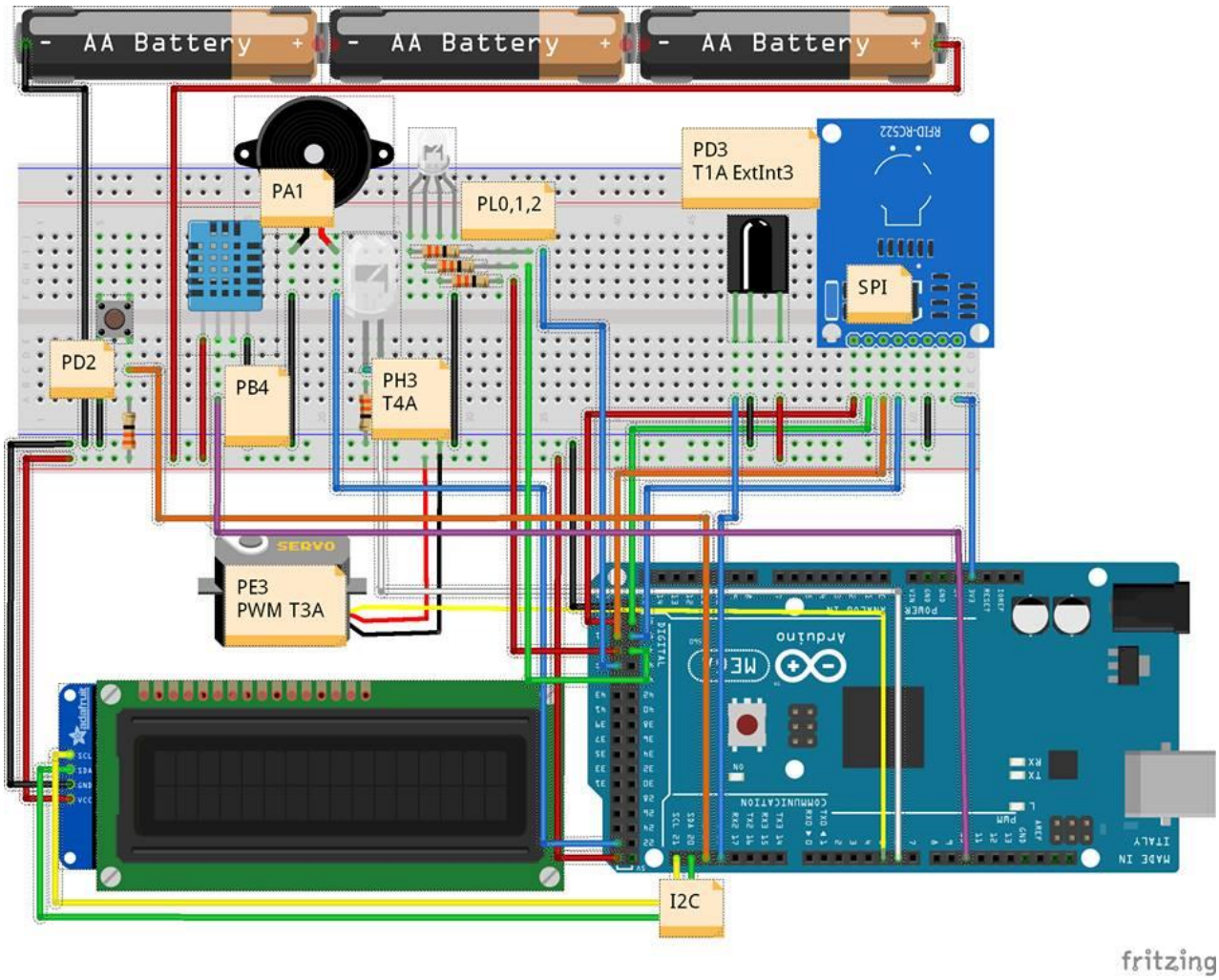

Figure 2. Prototype of the showcase controller

2.4. Use Case Scenarios

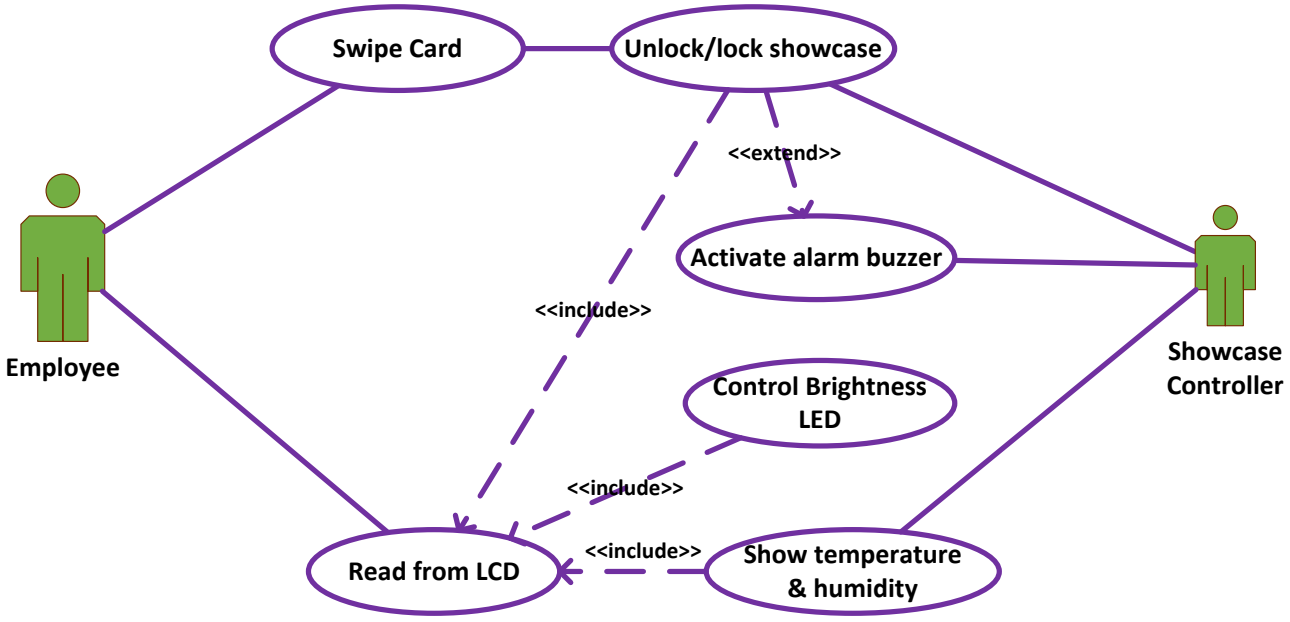

Figure 3. Use Case Diagram

a. The security access system process

To open the showcase， the designated person (museum employee) swipes his card on the RFID reader so that the servo will open the lock. To close the showcase the employee swipes his card again.

A colour LED (RGB) will blink green during normal status, and blue during swiping the card to lock or unlock the case. It will show red when the alarm goes ON. LCD display case door state (D: LOC or OPE)

b. The temperature & humidity monitoring

The LCD displays the state the inside environment of the showcase (Temperature and Humidity) sensed by DHT11 sensor.

c. Alarm process

The button will be kept pushed under the antique object and an interrupt will be set to monitor the button's status. Once the button is released (the object is taken or moved from its place) the buzzer and red led will be active. Also, one button on the remote controller will be used to release the alarm on specific situation. The LCD displays the Alarm state (A: EN "enabled", ON, Dis "disabled").

d. The remote control of the light brightness process

The remote control adjusts the brightness of the light (LED) from 0 to 10 levels of brightness. And enable/disable security sensor (Button) for specific situations. The LCD displays the scale of the LED light brightness (L: from 0 to 10)

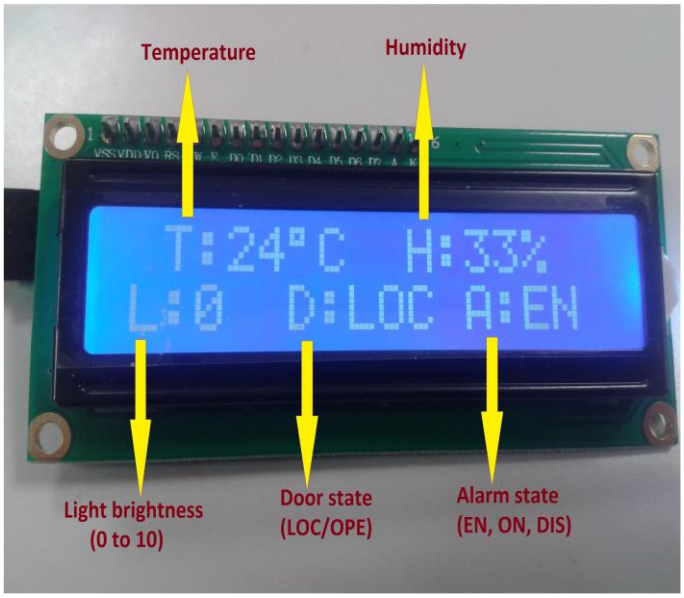

Figure a. LCD display

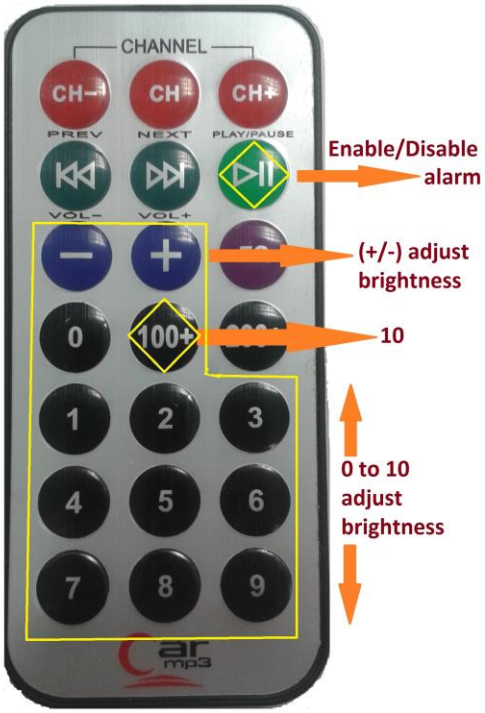

Figure b. Remote control commands

2.5 Flow charts

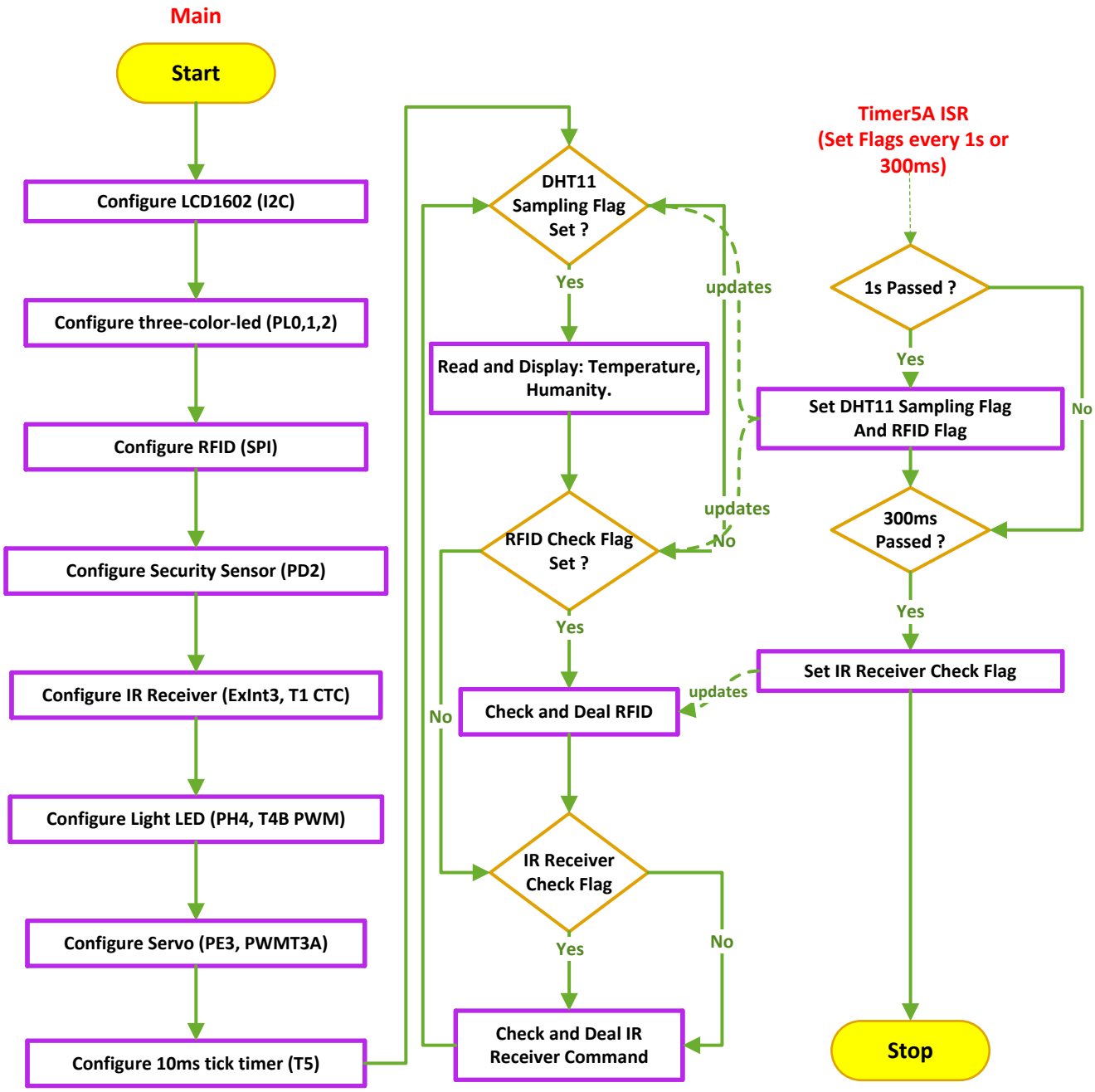

Figure 4.1. Main flow chart

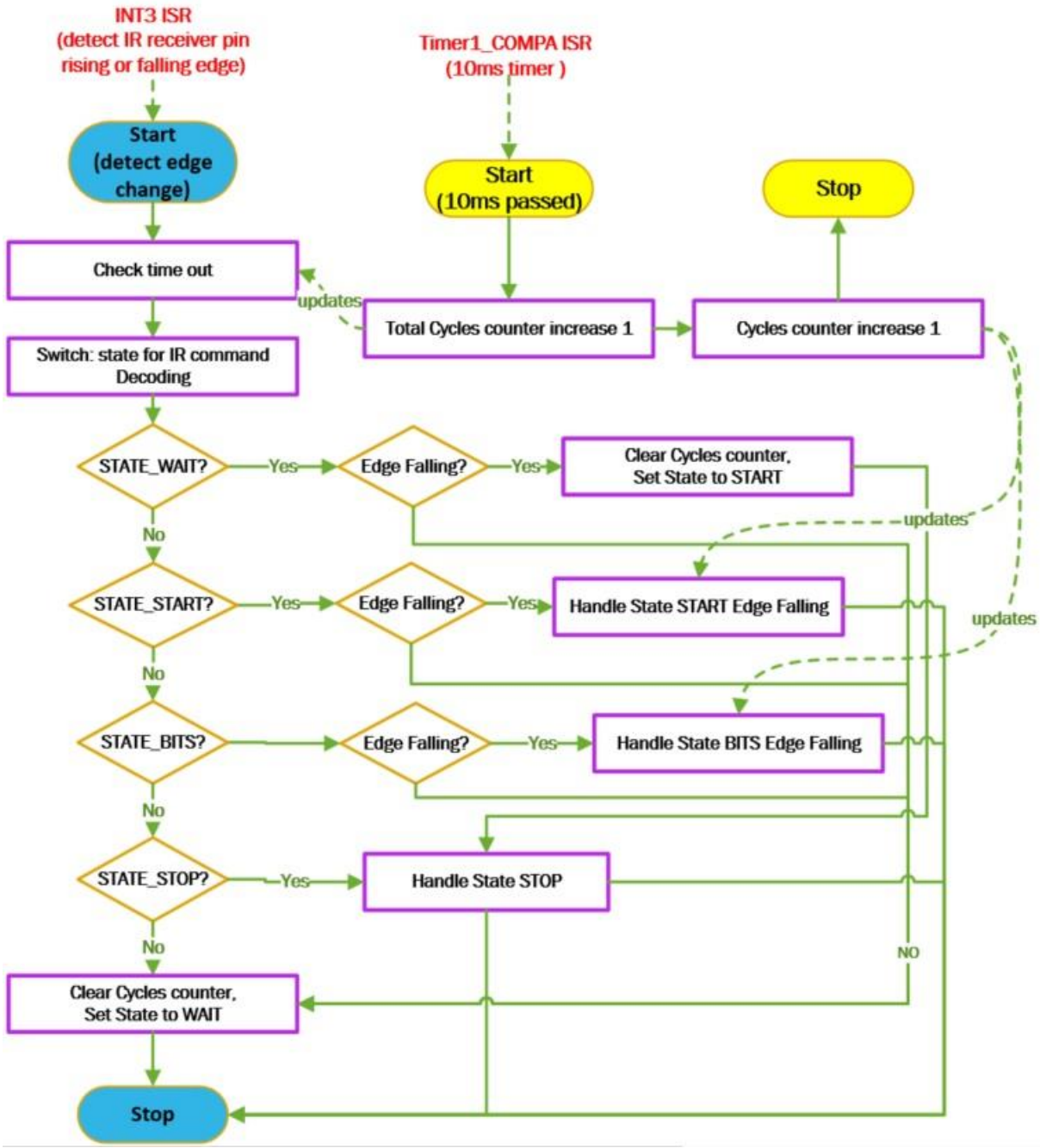

Figure 4.1. Flow chart of the IR remote control command decoding process

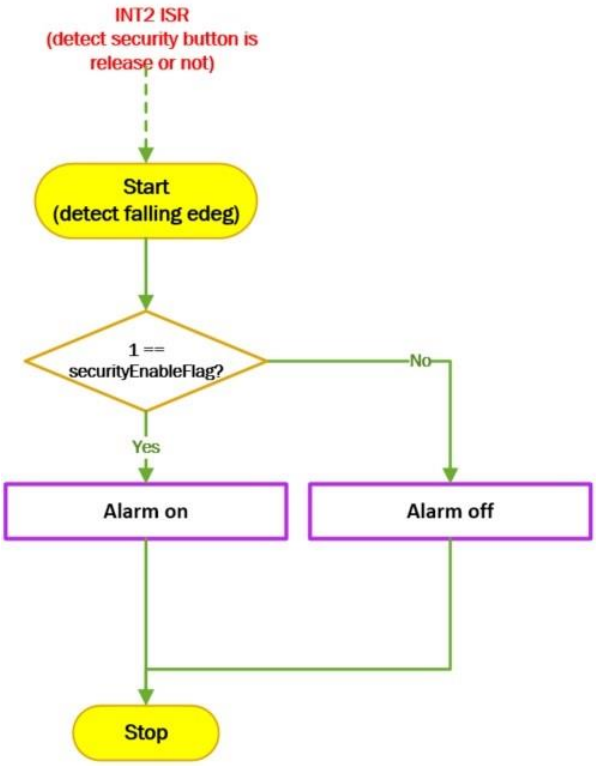

Figure 4.2. Flow chart of the alarm handle process

2.6 Timing diagrams

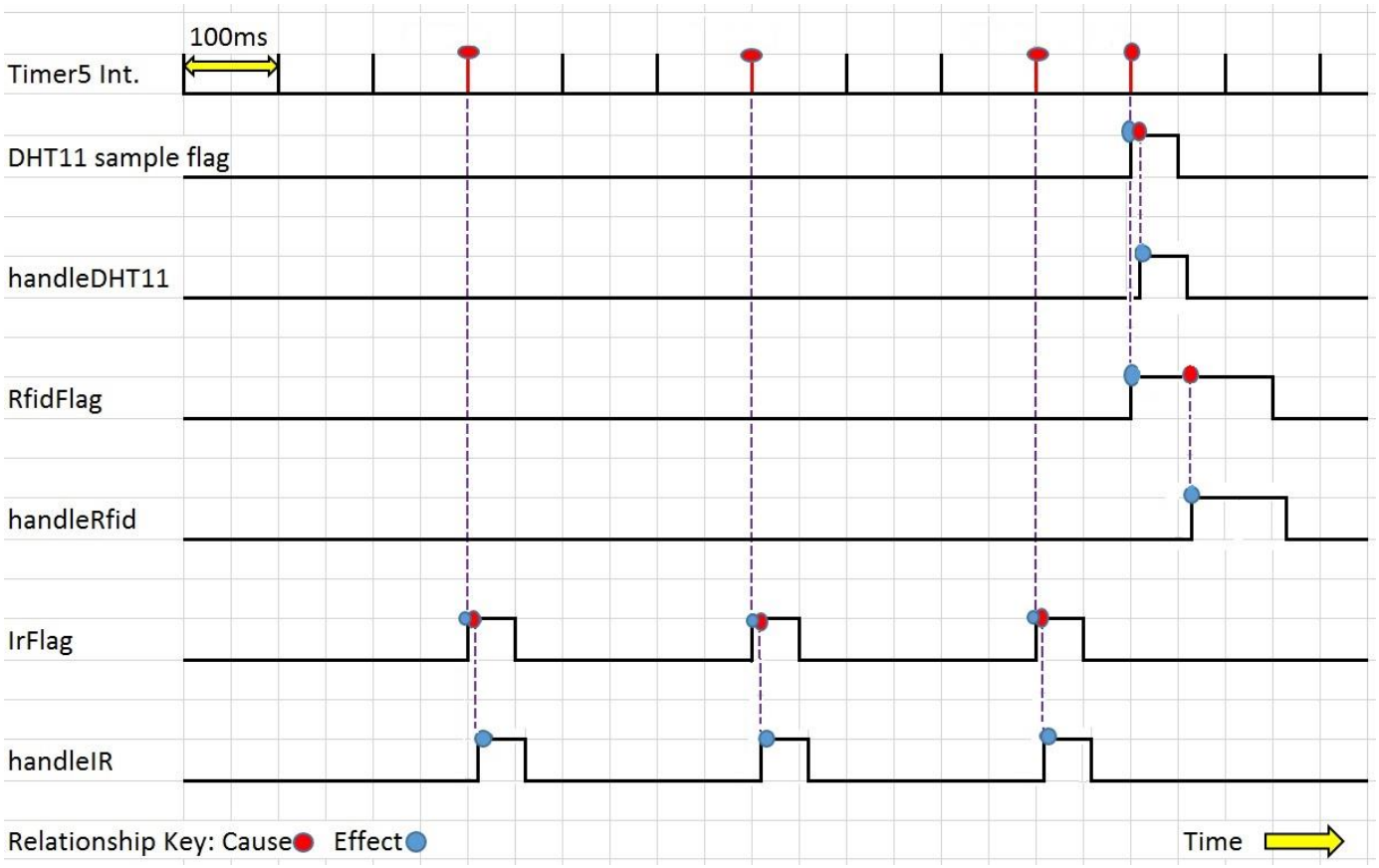

Figure 5. Main timing diagram

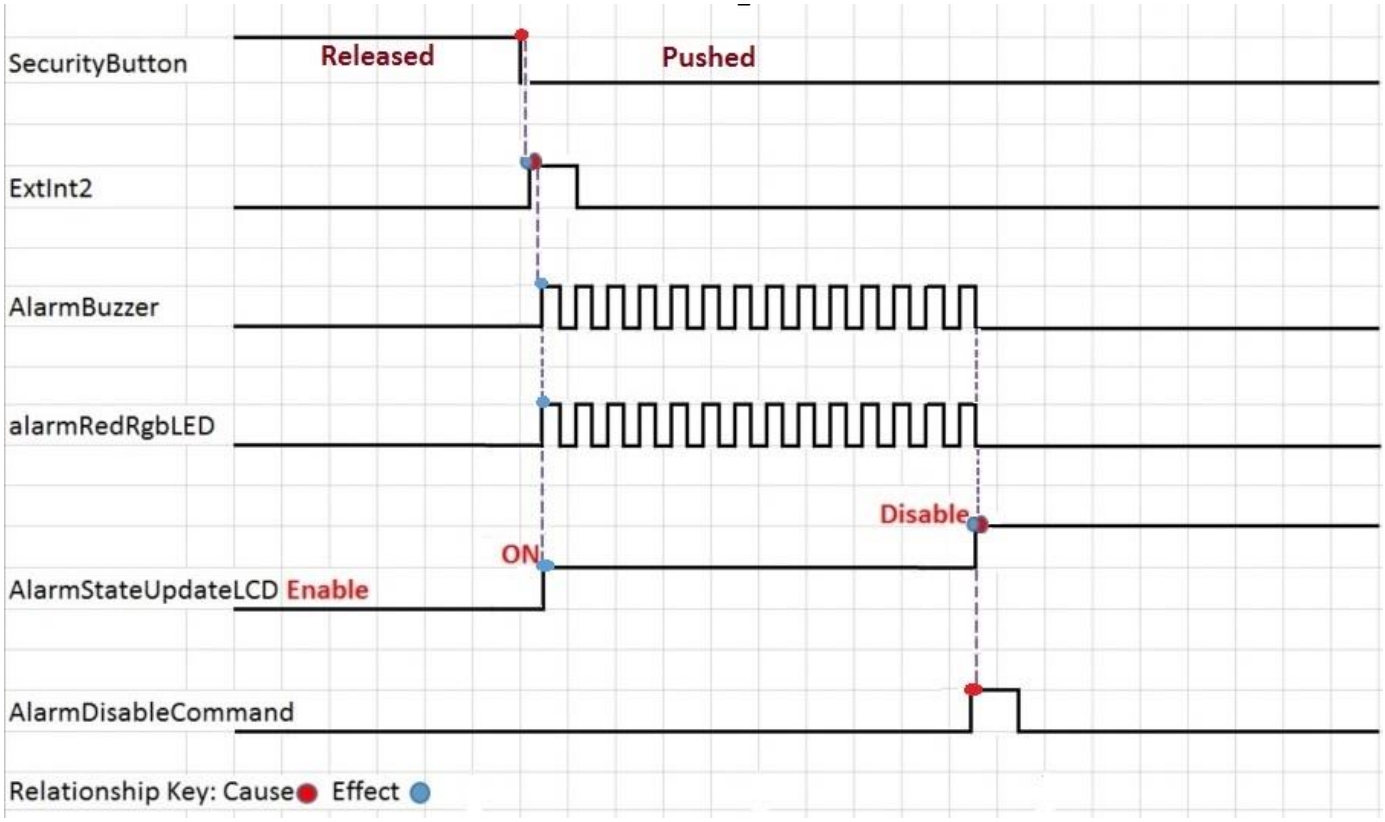

Figure 5.1. Alarm active Process

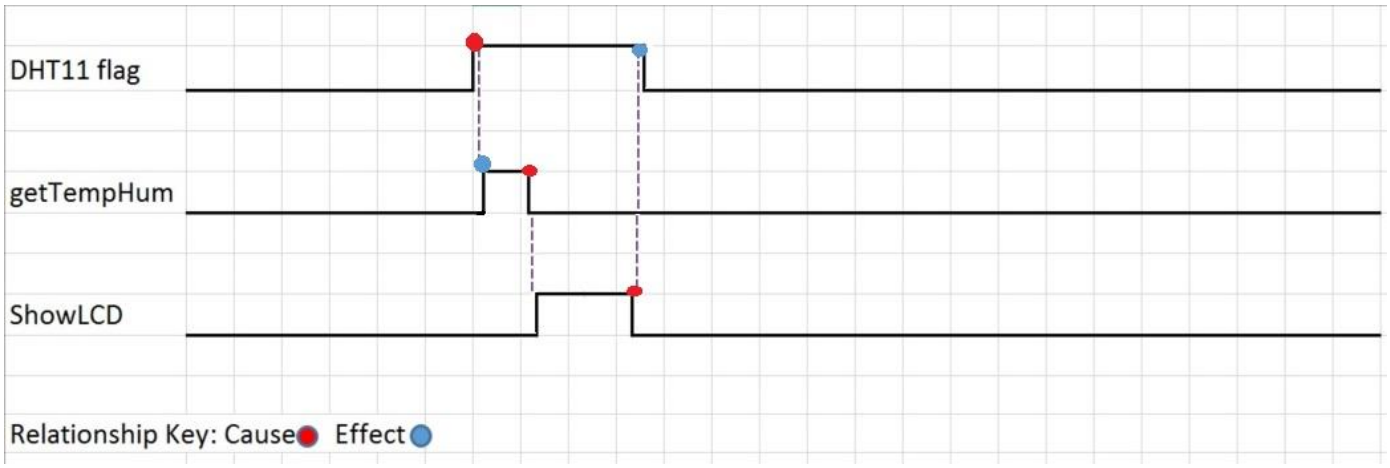

Figure 5.2. DHT11 Temperature & humidity monitoring process

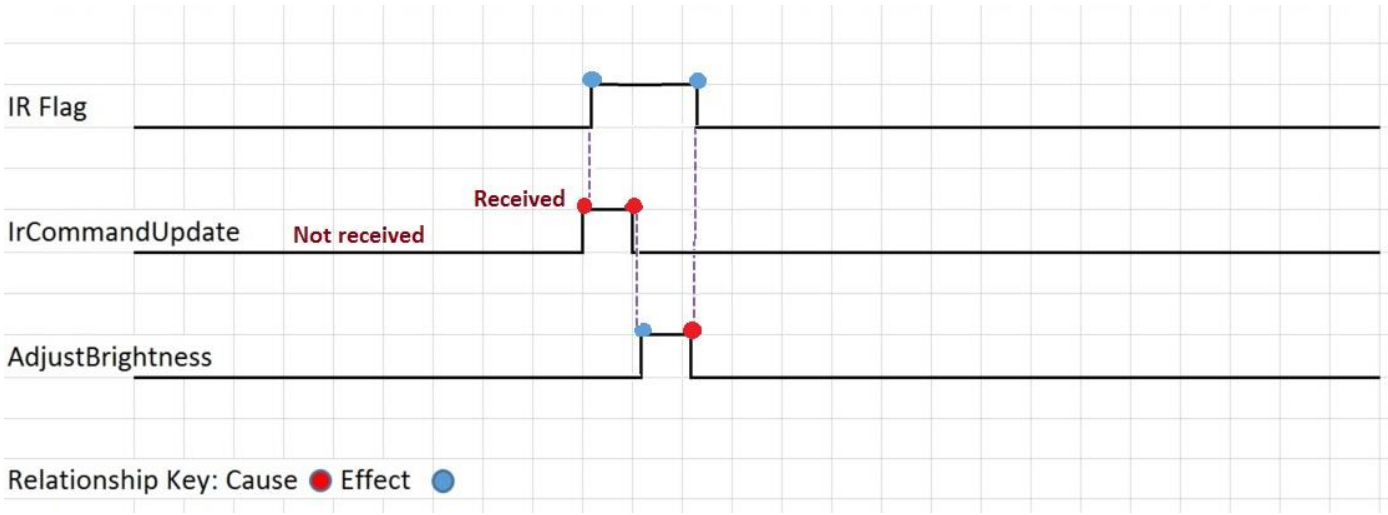

Figure 5.3. Light brightness adjusting process

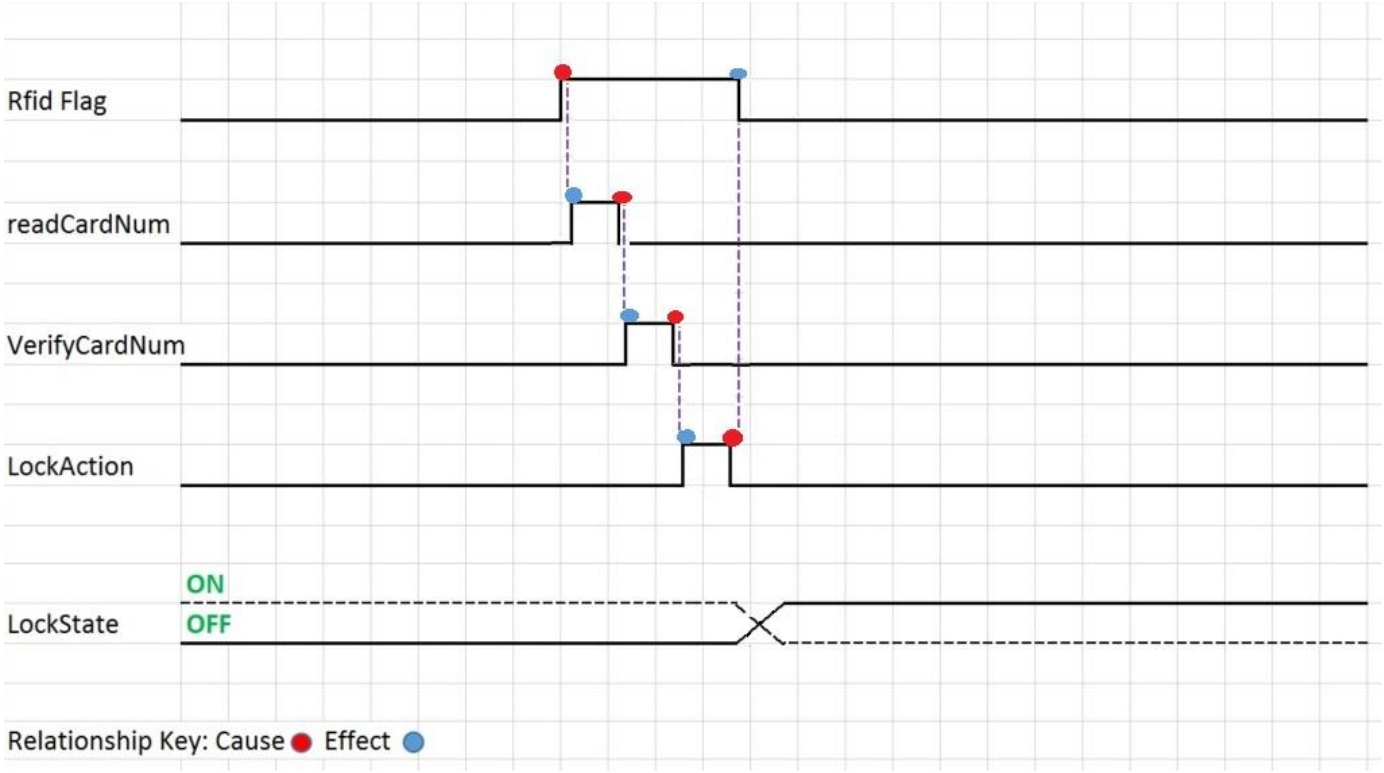

Figure 5.4. RFID Process

2.7 Configuration of internal modules

a. Hardware Interrupts

Two Hardware interrupts are used; **ExInt2** used for security sensor signal detection, **ExInt3** used for IR remote control signal reception.

- **ExInt2** is configured to detect the falling edge. When the security sensor (simulated by button) release, it will create a falling edge, which will trigger ExInt3 to active red colour of led and alarm buzzer.
- **ExInt3** is configured to detect the falling or rising edge to create hardware interrupt during the IR remote control command decoding process.

b. Timers

Four **16bit timers** are used: timer1, 3, 4, 5 and chosen because of their wide counting range.

- **Timer1** used in the IR remote control data receiver protocol (to determine the start and end of each data bit that makes up the command). And used Also as a **timer interrupt**.
- **Timer3** used to generate the Pulse-Width-Modulation signal which controls the position of the servo.
- **Timer4** used to generate the Pulse-Width-Modulation signal which controls the brightness of the LED light.

- **Timer5** used to reset refresh flag for temperature and humanity handle, lock action signal handle and remote control signal handle. And used Also as a **timer interrupt.**

c. SPI

- **SPI** is used to connect RFID reader mfrc522 and main board, helping get key card serial numbers for further evaluation.

d. I2C

- I2C is s configured to send commands to LCD1602, showing temperature, humidity, light level, lock state and alarm state on screen.

e. USART

- The USART is configured to provide a serial connection (9600 baud) to the computer serial terminal for debugging.

3. Testing

The tests were conducted all through the development process. Our testing includes two aspects: hardware test and software hardware. Hardware is the base of our system, and a reliable hardware environment will boost software development. Our system is developed based on Arduino MEGA2560 board and AVR studio IDE, but we also use Arduino IDE to test the circuit and components usability. We use the example on Arduino IDE to verify our connection, then switch to AVR studio and write c library for this component. This method is quite efficient so that we can make hardware work in a short time. Software development test are conducted through the JTAG debugger. Through this debugging tool, we successfully solved many problems. However, we found that JTAG debugger is heavy and easy to be trapped during debug process, we also use USART and serial terminal to test our system. For some simple problems, it can print out what we need to verify on the serial terminal, helping us quickly solve problems. All the tests during the system development period are both important and necessary, helping us successfully developed this museum showcase controller.

We developed driver libraries for individual component at first, then test the functions we need for the further high-level development. Table 1 shows the result of our tests.

Table 1 Component tests

Component	Hardware test on Arduino IDE	Functions tested	Result
RFID-MR522	work	Read card serial number	Success
LCD1602	work	Show letters on specific position	Success
DHT11	work	Read temperature and humanity	Success
IR remote control	work	Get IR remote control command	Success
LED light	work	PWM duty change from 0%-100%	Success
Servo	work	Chang servo position through PWM duty change	Success

During the high-level functions process, we defined four scenarios for our control system. Table 2 shows the test results. During testing, we found all the functions work very well. However, there is one small problem with humidity monitor. On an unpredictable case, the humidity will be showed as a wrong value for a short time, then recover to normal value. We believe it is the problem of DHT11 sensor itself. For Worst-Case Execution Time (WCET), the delay or loss of inner environment monitoring data are acceptable, because it is used just for the employee to refer. But security access and light brightness adjusting should be implemented once the employee give command. In our system, we check the command every one second. Through testing, we found this setting is proper, no delay or loss between sending command and system action. The alarm system is very time-critical, so we use an external interrupt to trigger alarm, this is the simplest and most reliable way for alarm activation.

Table 2 Scenario tests

Senior	Reliability	Usability	Testing process	Result
Security Access	High	High	Servo on lock position at first, swipe key card on RFID reader, state led become blue, servo change position to unlock, swipe and servo go back to lock position.	Success
Inner Environment Monitoring	Medium	High	Breath close to DHT11 for a while, the temperature and humidity on the LCD increase. Then keep away from DHT11, the environment parameters decrease to ambient values.	Success
Alarm Activation	High	High	On alarm enable state: release security button, the red led and buzzer will be active. Press disable button on Remote controller, the red led extinguish and buzzer muted.	Success
Light Brightness Adjusting	High	High	Press IR remote controller buttons from 0 to 10, LED brightness level change accordingly, press Volume +/- button to also adjust brightness.	Success

4. Program structure

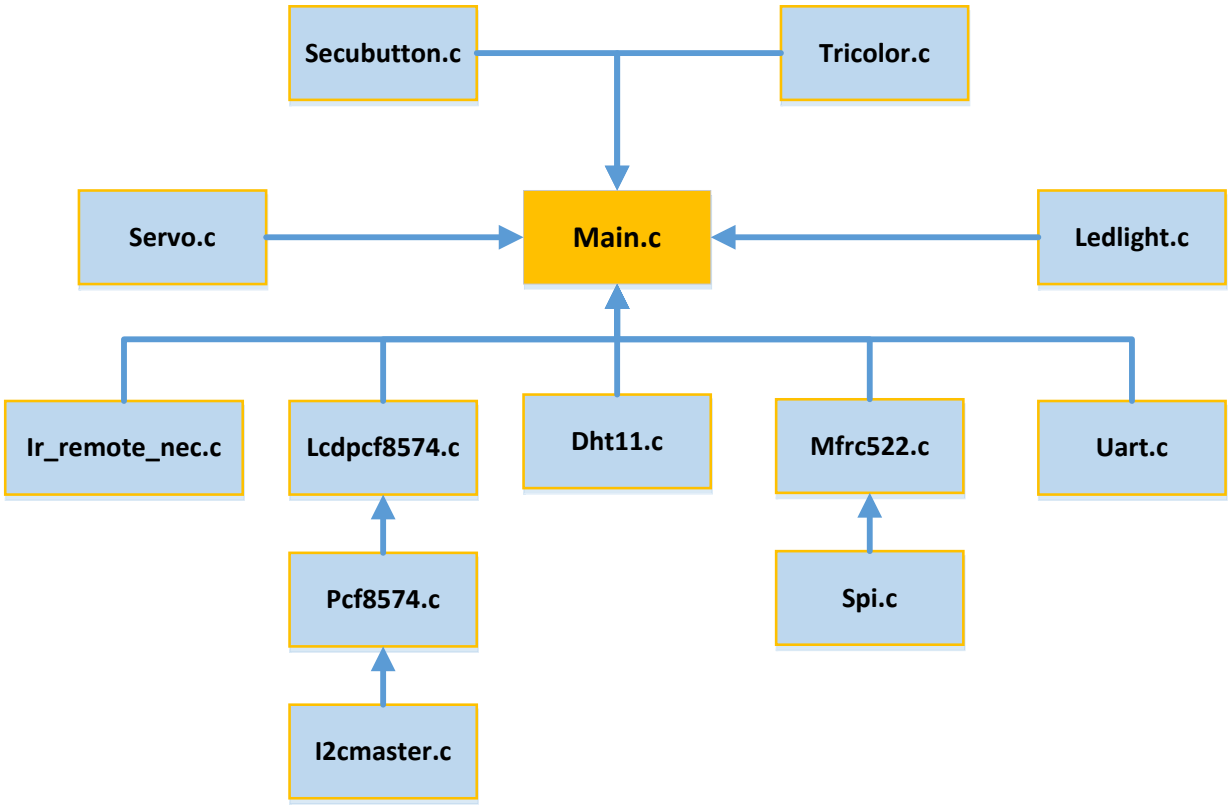

Figure 6. Software structure

The whole system is programed from bottom to up. Basic functions of different component are implement at first. Then high-level functions of showcase are implemented based on these former works. Figure 6 shows all the c files we have written for this project. The source code list is attached at Appendix part of this document.

Part B

5. Critical evaluation and conclusion

Our development process separates to three steps. Firstly, we choose the electronic components according to the functions of showcase control system, develop specific library and test performance of individual components. Secondly, we measure the Arduino MEGA2560 on-board resource and arrange the pins, timers, extend interrupt, I2C, SPI to different components, test their combability and make sure no interference with each other. At last, the whole system is integrated, we use functions from libraries of components like RFID-RC522, DHT11, IR remote control, etc. to build our high-level functions for showcase control, test covers all this process to verify our software design, also a final test is conducted to make sure this showcase controller perfectly implements the requirements.

Through hardworking development and strict tests, we finally implement all the showcase controller specifications excepting the password security check. It will increase complexity and reduce real-time

performance of system, but this system is still very security-critical because we have double protections for access to exhibit: only administrator who have the unique key card can unlock the showcase door; security alarm must be disabled before taking the exhibit. The 10-level LED light dimmer can provide the showcase perfect brightness according to environment. Temperature and humidity monitor will give reference of inner environment, which is very important for exhibit storage. The IR remote controller gives administrator more convenience and makes the system more security than the traditional keyboard. Our system is so practical and safe, but there are still lots of work to do.

For the further improvement, I think our system still have at least two directions to enhance. For now, we just have environment monitor in our system. I think temperature, humidity and oxygen adjusters are very necessary for some very precious exhibit like antique, painting, etc., so we can add the control to these adjusters to this system. The alarm system now is standalone, I hope it can be connected to the Internet later, so that the alarm notification can be set to mobile phone, computer, even police station, and some configuration can be done remotely, but security should be considered seriously for this feature.

This coursework is the most interesting assignment I have done and I learned a lot from it. On the development process, I learned embedded C language and a little assembly language, using them to control hardware directly. We use 4 timers and 2 external interrupts to build this real-time critical system, and I learned how to make them work together without interference. This assignment is a good start for my further exploration to build this kind of embedded systems.

Appendix

Source Code List

The code listing starting by the main.c file followed by the library files.

---------Main.c----------

```
/*
* smartMuseumShowcase_2560_C.c
* Created: 11/9/2016 2:21:16 PM
* Author : Charlie & Badis
* History:
*         11/11/2016: Finish all basic functions.
*         11/13/2016: Fix bug: security's alarm activation affect other functions.
*
* To-do-list: 1. add debug function
*             2. change command style referring to Richard's temple file
*             3. add watchdog
*             4. add IR remote control command to mute alarm
*/
#define  F_CPU 16000000UL

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdlib.h>
#include "library/uart.h"
#include "library/lcd1602//lcdpcf8574.h"
#include "library/tricolorled.h"
#include "library/dht.h"
#include "library/RFID/spi.h"
#include "library/RFID/mfrc522.h"
#include "library/servo.h"
#include "library/secuButton.h"
#include "library/ir_rc/ir_remote_nec.h"
#include "library/ir_rc/ir_nec_commands.h"
#include "library/ledlight.h"

#define LOCK    1 //locker state
#define UNLOCK  0 //locker state

volatile unsigned char sampleFlag=0;//sample flag for DHT11, sample rate 1Hz
volatile unsigned char rfidFlag=0;//sample flag for rfid, sample rate 1Hz
volatile unsigned char irFlag=0;//sample flag for ir remote controller, sample rate 1Hz
int8_t humidity;
int8_t temperature;

unsigned char lockerState=1;
unsigned char RFIDstr[MAX_LEN];
const unsigned char KEYstr[MAX_LEN]={0xf5,0x5b,0x08,0x88,0x2e,0,0,0};

void lcdFormate();//formate lcd display
void tiemr5_10ms_tick_configure();//10ms tick timer, create period(1s and 300ms) of time for sampling,
void HandleTempHum();//get temperature and humidity value from DHT11 every 1 second.
unsigned char HandleRFID(); // control locker servo according to RFID, active every 300 ms
void HandleIR();// change led light according to IR remote controller commands, active every 300 ms
```

```
int main(void)
{
	USART0_SETUP_9600_BAUD_ASSUME_1MHz_CLOCK();//for program debug, show debug information on serial monitor

	lcd_init(LCD_DISP_ON);//initialize lcd,display on, cursor off
	lcd_led(LCD_Backlight_ON); //light lcd backlight
	lcd_home();//lcd cursor go home
	/*
	---0 1 2 3 4 5 6 7 8 9 A B C D E F---
	0--  T £º8 8 ¡ã C     H : 8 8 %    --0
	1--L : 1 0   D : L O C   A : E N  --1
	---0 1 2 3 4 5 6 7 8 9 A B C D E F---
	*/
	lcdFormate();// formate lcd display according to the upside framework

	triColorLed_init();//three color led initialize

	tiemr5_10ms_tick_configure();// tick timer configure, create interrupt very 10ms, used to fresh flag

	spi_init();//initialize SPI port for mfrc522, AVR as master
	mfrc522_init();//initialize RFID

	Servo_Timer3_FastPWM();//initialize timer3 to fastPWM mode, create 18ms period PWM signal for servo
	SetServoPosition(LOCK);//set servo to lock position originally

	securityInit();//security button initialization,release this button will cause Alarm active

	ir_init();//initialize IR remote controller

	LEDLIGHT_Timer4_PWM_ChannelA_Init();//initialize pwm single for led light dimmer

	USART0_TX_String("initialization finish!!!\n");
	while(1)
	{
		HandleTempHum();//every 1s, check DHT11 and display
		HandleRFID();	//every 1s, check RFID input, unlock/lock locker
		HandleIR();	//every 300 ms, check IR remoter command, new command input will be Handleed to change led light

	}
}
//set 10ms tick to measure time
void tiemr5_10ms_tick_configure()
{
	TCCR5A = 0b00000000;		// Normal port operation (OC5A, OC5B, OC5C), Clear Timer on 'Compare Match' (CTC) waveform mode)
	TCCR5B = 0b00001010;		// CTC waveform mode, use prescaler 8

	// For F_CPU Mhz cup clock to achieve a 10 millisecond(100MHz) interval:
	// Need to count F_CPU/100 clock cycles (but already divided by 8)
	// So actually need to count to (F_CPU/100 / 8-1) = 16000000/100/8 -1 = 19999 decimal, = 4E1F Hex
	OCR5AH = 0x4E; // Output Compare Registers (16 bit) OCR5BH and OCR5BL
	OCR5AL = 0x1F;

	TCNT5H = 0b00000000;		// Timer/Counter count/value registers (16 bit) TCNT5H and TCNT5L
	TCNT5L = 0b00000000;
```

```
	TIMSK5 = 0b00000010; // bit 1 OCIE5A		Use 'Output Compare A Match' Interrupt, i.e. generate an interrupt
	// when the timer reaches the set value (in the OCR5A register)
	sei();
}
ISR(TIMER5_COMPA_vect) // TIMER5_CompareA_Handler (Interrupt Handler for Timer 5)
{
   static int s_1_count;//1 second counter
	static unsigned char ms_300_count;//300 millisecond counter

	s_1_count++;
	ms_300_count++;

	if(s_1_count >= 100)
	{
		sampleFlag++;// set sampleFlag, trigger HandleTempHum(sampleFlag) function
		rfidFlag=1;  //set rfidFlag, trigger HandleRFID(rfidFlag)  function
		if(0 == alarmFlag){
			tricolorled_toggle(LED_GREEN);//green state indicator toggle
		}
		s_1_count=0;
		//fprintf(USART,"TimerINT sampleFlag:%d,s_1_count:%d \n\t",sampleFlag,s_1_count);
	}
	if(ms_300_count>=30)
	{
		if(1 == alarmFlag){
		tricolorled_toggle(LED_RED);
		PORTA ^= (1<<DDA1);//active alarm
		}
		irFlag=1;//set irFlag, check IR remoter controller command
		ms_300_count=0;
	}
}
void HandleTempHum()//get temperature and humidity value from DHT11 every 1 second.
  {
	char buf[3];
	//fprintf(USART,"s_1_count_begin:%d,sampleFlag:%d \n\t",s_1_count,sampleFlag);
	if(10 <= sampleFlag)
	{
		//fprintf(USART,"temperature:%d,humidity:%d\n\t",temperature,humidity);
		cli();
		dht_gettemperaturehumidity(&temperature, &humidity);//get temperature and humidity from DHT11, must inside interrupt, put it outside will be interrupt by other interruption
		sei();
		lcd_gotoxy(3, 0);//set cursor to (3,0)
		itoa(temperature, buf, 10);
		lcd_puts(buf);
		lcd_gotoxy(11, 0);
		itoa(humidity, buf, 10);
		lcd_puts(buf);
		sampleFlag = 0;
		//fprintf(USART,"s_1_count_end:%d,sampleFlag:%d \n\t",s_1_count,sampleFlag);
	}
}

unsigned char HandleRFID(){
	unsigned char RFIDbyte;
```

```
	if(1 == rfidFlag)
	{
		RFIDbyte = mfrc522_request(PICC_REQALL,RFIDstr);//read mfrc522
		if(RFIDbyte == CARD_FOUND)
		{
			RFIDbyte = mfrc522_get_card_serial(RFIDstr);
			for(RFIDbyte=0;RFIDbyte<8;RFIDbyte++)
			{
				//fprintf(USART,"serialnumber[%d]: %x\n",RFIDbyte,RFIDstr[RFIDbyte]);
				if(KEYstr[RFIDbyte]!=RFIDstr[RFIDbyte])
				{
					return 0;
				}
			}
			tricolorled_onoff(LED_BLUE, LED_ON);
			if(LOCK==lockerState){
				SetServoPosition(UNLOCK);
				lockerState=UNLOCK;
				lcd_gotoxy(7,1);
				lcd_puts("OPE");
				//fprintf(USART,"show case door unlocked!");
			}
			else
			{
				SetServoPosition(LOCK);
				lockerState=LOCK;
				lcd_gotoxy(7,1);
				lcd_puts("LOC");
				//fprintf(USART,"show case door locked!");
			}
			tricolorled_onoff(LED_BLUE, LED_OFF);
		}
		rfidFlag=0;
		return 1;
	}
	return 0;

}

void HandleIR(){
	uint32_t current_command = 0;
	static unsigned char SwitchesValue;
	char buf[3];
	if(1==irFlag){
		current_command = get_current_command();
		if (current_command != 0){
			//fprintf(USART,"current_command:%x\n",current_command);
			switch (current_command) {
				case COMMAND_VOL_MINUS:
				if( SwitchesValue == 0)
				{
					SwitchesValue = 11;
				}
				SwitchesValue--;
				break;
				case COMMAND_VOL_PLUS:
				SwitchesValue++;
```

```
                    if(SwitchesValue >= 11)
                    {
                            SwitchesValue = 0;
                    };
                    break;
                    case COMMAND_PLAY_PAUSE:
                    securityEnableFlag ^= (securityEnableFlag|0x01);//toggle securityEnableFlag flag
between 0 and 1
                    lcd_gotoxy(13,1);
                    if(0 == securityEnableFlag){
                            alarm_OFF();
                            lcd_puts("DIS");
                    }
                    else{
                            lcd_puts("EN ");
                    }
                    break;
                    case COMMAND_0:SwitchesValue=0;
                    break;
                    case COMMAND_1:SwitchesValue=1;
                    break;
                    case COMMAND_2:SwitchesValue=2;
                    break;
                    case COMMAND_3:SwitchesValue=3;
                    break;
                    case COMMAND_4:SwitchesValue=4;
                    break;
                    case COMMAND_5:SwitchesValue=5;
                    break;
                    case COMMAND_6:SwitchesValue=6;
                    break;
                    case COMMAND_7:SwitchesValue=7;
                    break;
                    case COMMAND_8:SwitchesValue=8;
                    break;
                    case COMMAND_9:SwitchesValue=9;
                    break;
                    case COMMAND_100_PLUS:SwitchesValue=10;
                    break;
                    default:
                    break;
            }
            //change light value on lcd display
            lcd_gotoxy(2,1);
            lcd_puts("  ");
            lcd_gotoxy(2,1);
            itoa(SwitchesValue, buf, 10);
            lcd_puts(buf);
            switch(SwitchesValue)
            {
                    case 0:
                    OCR4AL =  0;                    // led light off
                    break;
                    case 1:
                    OCR4AL = 26;                    // 10% duty cycle
                    break;
                    case 2:
```

```
                                OCR4AL = 51;                    // 20% duty cycle
                                break;
                                case 3:
                                OCR4AL = 77;            // 30% duty cycle
                                break;
                                case 4:
                                OCR4AL = 102;                   // 40% duty cycle
                                break;
                                case 5:
                                OCR4AL = 128;                   // 50% duty cycle
                                break;
                                case 6:
                                OCR4AL = 153;                   // 60% duty cycle
                                break;
                                case 7:
                                OCR4AL = 179;                   // 70% duty cycle
                                break;
                                case 8:
                                OCR4AL = 204;                   // 80% duty cycle (LEDS appear near-full
brightness)
                                break;
                                case 9:
                                OCR4AL = 230;                   // 90% duty cycle (LEDS appear near-full
brightness)
                                break;
                                case 10:
                                OCR4AL = 255;                   // 100% duty cycle (LEDS appear near-full
brightness)
                                break;
                        }
                }
                irFlag=0;
        }
}

void lcdFormate()//formate lcd display
{
        lcd_gotoxy(0, 0);
        lcd_puts(" T:88 C  H:88%");
        lcd_gotoxy(5, 0);
        lcd_putc(0xdf);
        lcd_gotoxy(0,1);
        lcd_puts("L:0  D:LOC A:EN ");
}
```

---------dht.h----------

```
/*
DHT Library 0x03
copyright (c) Davide Gironi, 2012
Released under GPLv3.
Please refer to LICENSE file for licensing information.

References:
  - DHT-11 Library, by Charalampos Andrianakis on 18/12/11
*/
#ifndef DHT_H_
#define DHT_H_
```

```
#include <stdio.h>
#include <avr/io.h>

//setup port
#define DHT_DDR DDRB
#define DHT_PORT PORTB
#define DHT_PIN PINB
#define DHT_INPUTPIN PB4

//sensor type
#define DHT_DHT11 1
#define DHT_DHT22 2
#define DHT_TYPE DHT_DHT11

//enable decimal precision (float)
#if DHT_TYPE == DHT_DHT11
#define DHT_FLOAT 0
#elif DHT_TYPE == DHT_DHT22
#define DHT_FLOAT 1
#endif

//timeout retries
#define DHT_TIMEOUT 200

#ifdef __cplusplus
extern "C" {
#endif

//functions
#if DHT_FLOAT == 1
extern int8_t dht_gettemperature(float *temperature);
extern int8_t dht_gethumidity(float *humidity);
extern int8_t dht_gettemperaturehumidity(float *temperature, float *humidity);
#elif DHT_FLOAT == 0
extern int8_t dht_gettemperature(int8_t *temperature);
extern int8_t dht_gethumidity(int8_t *humidity);
extern int8_t dht_gettemperaturehumidity(int8_t *temperature, int8_t *humidity);
#endif
#ifdef __cplusplus
}
#endif
#endif
```

--------dht.c----------

```
DHT Library 0x03
copyright (c) Davide Gironi, 2012
Released under GPLv3.
Please refer to LICENSE file for licensing information.
*/
#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include "dht.h"
/*
 * get data from sensor
 */
```

```
#if DHT_FLOAT == 1
int8_t dht_getdata(float *temperature, float *humidity) {
#elif DHT_FLOAT == 0
int8_t dht_getdata(int8_t *temperature, int8_t *humidity) {
#endif
	uint8_t bits[5];
	uint8_t i,j = 0;
	memset(bits, 0, sizeof(bits));

	//reset port
	DHT_DDR |= (1<<DHT_INPUTPIN); //output
	DHT_PORT |= (1<<DHT_INPUTPIN); //high
	_delay_ms(100);

	//send request
	DHT_PORT &= ~(1<<DHT_INPUTPIN); //low
	#if DHT_TYPE == DHT_DHT11
	_delay_ms(18);
	#elif DHT_TYPE == DHT_DHT22
	_delay_us(500);
	#endif
	DHT_PORT |= (1<<DHT_INPUTPIN); //high
	DHT_DDR &= ~(1<<DHT_INPUTPIN); //input
	_delay_us(40);

	//check start condition 1
	if((DHT_PIN & (1<<DHT_INPUTPIN))) {
		return -1;
	}
	_delay_us(80);
	//check start condition 2
	if(!(DHT_PIN & (1<<DHT_INPUTPIN))) {
		return -1;
	}
	_delay_us(80);

	//read the data
	uint16_t timeoutcounter = 0;
	for (j=0; j<5; j++) { //read 5 byte
		uint8_t result=0;
		for(i=0; i<8; i++) {//read every bit
			timeoutcounter = 0;
			while(!(DHT_PIN & (1<<DHT_INPUTPIN))) { //wait for an high input (non blocking)
				timeoutcounter++;
				if(timeoutcounter > DHT_TIMEOUT) {
					return -1; //timeout
				}
			}
			_delay_us(30);
			if(DHT_PIN & (1<<DHT_INPUTPIN)) //if input is high after 30 us, get result
				result |= (1<<(7-i));
			timeoutcounter = 0;
			while(DHT_PIN & (1<<DHT_INPUTPIN)) { //wait until input get low (non blocking)
				timeoutcounter++;
				if(timeoutcounter > DHT_TIMEOUT) {
					return -1; //timeout
				}
```

```
                        }
                }
                bits[j] = result;
        }

        //reset port
        DHT_DDR |= (1<<DHT_INPUTPIN); //output
        DHT_PORT |= (1<<DHT_INPUTPIN); //low
        _delay_ms(100);

        //check checksum
        if ((uint8_t)(bits[0] + bits[1] + bits[2] + bits[3]) == bits[4]) {
                //return temperature and humidity
                #if DHT_TYPE == DHT_DHT11
                *temperature = bits[2];
                *humidity = bits[0];
                #elif DHT_TYPE == DHT_DHT22
                uint16_t rawhumidity = bits[0]<<8 | bits[1];
                uint16_t rawtemperature = bits[2]<<8 | bits[3];
                if(rawtemperature & 0x8000) {
                        *temperature = (float)((rawtemperature & 0x7FFF) / 10.0) * -1.0;
                } else {
                        *temperature = (float)(rawtemperature)/10.0;
                }
                *humidity = (float)(rawhumidity)/10.0;
                #endif
                return 0;
        }

        return -1;
}

/*
 * get temperature
 */
#if DHT_FLOAT == 1
int8_t dht_gettemperature(float *temperature) {
        float humidity = 0;
#elif DHT_FLOAT == 0
int8_t dht_gettemperature(int8_t *temperature) {
        int8_t humidity = 0;
#endif
        return dht_getdata(temperature, &humidity);
}

/*
 * get humidity
 */
#if DHT_FLOAT == 1
int8_t dht_gethumidity(float *humidity) {
        float temperature = 0;
#elif DHT_FLOAT == 0
int8_t dht_gethumidity(int8_t *humidity) {
        int8_t temperature = 0;
#endif
        return dht_getdata(&temperature, humidity);
}
```

```
/*
 * get temperature and humidity
 */
#if DHT_FLOAT == 1
int8_t dht_gettemperaturehumidity(float *temperature, float *humidity) {
#elif DHT_FLOAT == 0
        int8_t dht_gettemperaturehumidity(int8_t *temperature, int8_t *humidity) {
#endif
        return dht_getdata(temperature, humidity);
}
```

----------ledlight.h----------

```
/*
 * ledlight.h
 *
 * Created: 11/11/2016 4:08:07 PM
 *  Author: charlie
 */
#ifndef LEDLIGHT_H_
#define LEDLIGHT_H_

void LEDLIGHT_Timer4_PWM_ChannelA_Init();
#endif /* LEDLIGHT_H_ */
```

---------- ledlight.c ----------

```
/*
* ledlight.c
*
* Created: 11/11/2016 4:07:32 PM
*  Author: charlie
*/
#include <avr/io.h>

void LEDLIGHT_Timer4_PWM_ChannelA_Init()
{
        DDRH |=(1<<PH3);
        // TCCR4A ?Timer/Counter 4 Control Register A
        // Bit 7:6 ?COMnA1:0: Compare Output Mode for Channel A
        // Bit 5:4 ?COMnB1:0: Compare Output Mode for Channel B
        // Bit 3:2 ?COMnC1:0: Compare Output Mode for Channel C
        // Bit 1:0 ?WGMn1:0: Waveform Generation Mode (0101 Fast PWM, 8-bit)
        TCCR4A = 0b10000001;            // No output pins in use, set all to normal mode, waveform  = Fast PWM, 8-
bit

        // TCCR4B ?Timer/Counter 4 Control Register B
        // Bit 7 ?ICNCn: Input Capture Noise Canceler
        // Bit 6 ?ICESn: Input Capture Edge Select
        // Bit 4 ?Reserved Bit
        // Bit 4:3 ?WGMn3:2: Waveform Generation Mode (0101 Fast PWM, 8-bit)
        // Bit 2:0 ?CSn2:0: Clock Select (010 = 8 prescaler)
        TCCR4B = 0b00001010; // waveform  = Fast PWM, 8-bit, 8 prescaler

        // TCCR4C ?Timer/Counter 4 Control Register C
        // Bit 7 ?FOCnA: Force Output Compare for Channel A
        // Bit 6 ?FOCnB: Force Output Compare for Channel B
        // Bit 5 ?FOCnC: Force Output Compare for Channel C
        TCCR4C = 0b00000000;

        // TCNT4H and TCNT4L –Timer/Counter 4
```

```
	TCNT4 = 0;

	// OCR4AH and OCR4AL ?Output Compare Register 4 A
	OCR4AH = 0x00;
	OCR4AL = 0x00;
}
```

--------- secuButton.h----------

```
/*
 * secuButton.h
 *
 * Created: 11/10/2016 1:50:56 PM
 *  Author: charlie
 */
#ifndef SECUBUTTON_H_
#define SECUBUTTON_H_

volatile unsigned char securityEnableFlag;//default:1,security enable;0, security disable, mute buzzer and red led
volatile unsigned char alarmFlag;
void securityInit();
void alarm_OFF();

#endif /* SECUBUTTON_H_ */
```

---------- secuButton.c----------

```
/*
* secuButton.c
*
* Created: 11/10/2016 1:50:09 PM
*  Author: charlie
*/
#include <avr/io.h>
#include <avr/interrupt.h>
#include "uart.h"
#include "lcd1602/lcdpcf8574.h"
#include "secuButton.h"
#include "tricolorled.h"

void alarm_ON(){
	alarmFlag=1;
	tricolorled_onoff(LED_GREEN,LED_OFF);
	lcd_gotoxy(13,1);
	lcd_puts("ON");
}

void alarm_OFF(){
	PORTA&= ~(1<<DDA1);//inactive alarm
	alarmFlag=0;
	tricolorled_onoff(LED_RED,LED_OFF);
}

void securityInit(){
	DDRA |= (1<<DDA1);//set portA pin0 to digital output, used as alarm control signal, set 1 will active alarm, set 0 inactive alarm
	PORTA&= ~(1<<DDA1);//turn off buzzer

	DDRD &= ~(1<<DDD2);//set PORTD pin2 to digital input, as security button signal input
	PORTD |= (1<<PD2);//pull up resister enable
	EICRA |= (1<<ISC21);//falling edge on INTn generates an interrupt request
```

```
        EIMSK |= (1<<INT2);//enable external interrupt 2
        EIFR |= (1<<INTF2);//clear external interrupt 2 flag
        sei();
   securityEnableFlag=1; //enable security alarm (buzzer and red state led)
        lcd_gotoxy(13,1);
        lcd_puts("EN ");
}
ISR(INT2_vect)
{
        if(1 == securityEnableFlag){
                alarm_ON();
                //fprintf(USART,"Allarm On!");
        }
        else
        {
                alarm_OFF();
                //fprintf(USART,"Allarm OFF!");
        }
}
```

---------servo.h----------

```
/*
 * servo.h
 *
 * Created: 11/4/2016 1:29:53 PM
 *  Author: charlie
 */
#ifndef SERVO_H_
#define SERVO_H_
#define LockerOFF 0
#define LockerOn 1

void Servo_Timer3_FastPWM();
void SetServoPosition(unsigned char sw);

#endif /* SERVO_H_ */
```

----------servo.c----------

```
/*
* servo.c
*
* Created: 11/4/2016 1:29:42 PM
*  Author: charlie
*/
#include <avr/io.h>
#include <avr/interrupt.h>

#define DDRS DDRE
#define DDSPin   DDE3
#define PORTS PORTE
#define SPin  PE3

void Servo_Timer3_FastPWM()
{
        DDRS |= (1<<DDSPin); //set servo PWM pin as OUTPUT
        PORTS &= ~(1<<SPin); //set servo pin to low

        // TCCR3A ?Timer/Counter 3 Control Register A
```

```
	// Bit 7:6 ?COMnA1:0: Compare Output Mode for Channel A (For FAST PWM 10 = Clear OC3A on Compare match (Non-Inverting))
	// Bit 5:4 ?COMnB1:0: Compare Output Mode for Channel B (For FAST PWM 10 = Clear OC3B on Compare match (Non-Inverting))
	// Bit 3:2 ?COMnC1:0: Compare Output Mode for Channel C (For FAST PWM 10 = Clear OC3C on Compare match (Non-Inverting))
	// Bit 1:0 ?WGMn1:0: Waveform Generation Mode (Waveform bits WGM3(3..0) 1110 Fast PWM ICR3 is TOP)
	TCCR3A = 0b10000010;		// Fast PWM non inverting, ICR3 used as TOP

	// TCCR3B ?Timer/Counter 3 Control Register B
	// Bit 7 ?ICNCn: Input Capture Noise Canceler
	// Bit 6 ?ICESn: Input Capture Edge Select
	// Bit 5 ?Reserved Bit
	// Bit 4:3 ?WGMn3:2: Waveform Generation Mode
	// Bit 2:0 ?CSn2:0: Clock Select
	TCCR3B = 0b00011010;		// Fast PWM, Use Prescaler 8

	// TCCR3C ?Timer/Counter 3 Control Register C
	// Bit 7 ?FOCnA: Force Output Compare for Channel A
	// Bit 6 ?FOCnB: Force Output Compare for Channel B
	// Bit 5 ?FOCnC: Force Output Compare for Channel C
	TCCR3C = 0b00000000;

	// Set Timer/Counter3 Input Capture Register (16 bit) ICR3
	// Can only be written to when using a waveform generation mode that uses ICR3 to define the TOP value
	// For the SERVO, the pulses should occur every 18ms, i.e. 18000uS
	// With a 2MHz clock speed, each clock pulse takes 0.5us, therefore need to count 36000 clock pulses
	// Decimal 36000 = 0x8CA0
	// This count value defines where a single cycle ends.
	// The actual pulse width is much shorter than the whole cycle.
	ICR3H = 0x8C; // 16-bit access (write high byte first, read low byte first)
	ICR3L = 0xA0;

	// Set Timer/Counter count/value registers (16 bit) TCNT1H and TCNT1L
	TCNT3H = 0; // 16-bit access (write high byte first, read low byte first)
	TCNT3L = 0;

	// Initialise Channel A servo to mid-range position
	// Set Timer/Counter Output Compare Registers (16 bit) OCR3AH and OCR3AL
	// Pulse width ranges from 750uS to 2250uS
	// 'Neutral' (Mid range) pulse width 1.5mS = 1500uS pulse width
	OCR3A = 3000;

	// TIMSK3 ?Timer/Counter 3 Interrupt Mask Register
	// Bit 5 ?ICIEn: Timer/Countern, Input Capture Interrupt Enable
	// Bit 3 ?OCIEnC: Timer/Countern, Output Compare C Match Interrupt Enable
	// Bit 2 ?OCIEnB: Timer/Countern, Output Compare B Match Interrupt Enable
	// Bit 1 ?OCIEnA: Timer/Countern, Output Compare A Match Interrupt Enable
	// Bit 0 ?TOIEn: Timer/Countern, Overflow Interrupt Enable
	TIMSK3 = 0b00000000; // No interrupts needed, PWM pulses appear directly on OC3A (Port E Bit3)
	// TIFR3 ?Timer/Counter3 Interrupt Flag Register
	TIFR3 = 0b00101111;		// Clear all interrupt flags

	sei();// Enable interrupts at global level Set Global Interrupt Enable bit
}
// set Servo position to 0-180(+/-10)degree, Pulse width ranges from 500us to 2500us
```

```
void SetServoPosition(unsigned char sw){
	if(0==sw){
		OCR3A=2500;
	}
	else{
		OCR3A=4300;
	}
	/*	OCR3A = position*200/9+1000;//Minimum value*/
}
```

----------tricolorled.h----------

```
#ifndef TRICOLORLED_H_
#define TRICOLORLED_H_

#define LED_RED   0
#define LED_GREEN 1
#define LED_BLUE  2
#define LED_ON  1
#define LED_OFF 0

extern void triColorLed_init();
extern void tricolorled_onoff(unsigned char color, unsigned char ledonoff);
void tricolorled_toggle(unsigned char color);
#endif
```

----------tricolorled.c----------

```
/*
 * CFile1.c
 *
 * Created: 11/9/2016 3:27:16 PM
 *  Author: charlie
 */

#include <avr/io.h>
#include "tricolorled.h"

#define F_CPU 16000000UL
#include <util/delay.h>
//setup port
#define TRICOLORLED_DDR DDRL
#define TRICOLORLED_PORT PORTL
#define TRICOLORLED_PIN PINL

#define TRICOLORLED_RED_PIN PL0
#define TRICOLORLED_GREEN_PIN PL1
#define TRICOLORLED_BLUE_PIN PL2

void triColorLed_init(){

	TRICOLORLED_DDR |= (1<<TRICOLORLED_RED_PIN)|(1<<TRICOLORLED_GREEN_PIN)|(1<<TRICOLORLED_BLUE_PIN)   ;  //set three pins to output
	TRICOLORLED_PORT &= ~((1<<TRICOLORLED_RED_PIN)|(1<<TRICOLORLED_GREEN_PIN)|(1<<TRICOLORLED_BLUE_PIN)); //trun off all the leds

	//check red led
	tricolorled_onoff(LED_RED,LED_ON);
	_delay_ms(1000);
```

```
	tricolorled_onoff(LED_RED,LED_OFF);
	//check green led
	tricolorled_onoff(LED_GREEN,LED_ON);
	_delay_ms(1000);
	tricolorled_onoff(LED_GREEN,LED_OFF);
	//check blue led
	tricolorled_onoff(LED_BLUE,LED_ON);
	_delay_ms(1000);
	tricolorled_onoff(LED_BLUE,LED_OFF);
	};

void tricolorled_onoff(unsigned char color, unsigned char ledonoff){
	switch(color){
		case LED_RED:
			if(ledonoff){
				TRICOLORLED_PORT |= (1<<TRICOLORLED_RED_PIN); //turn on led
			}
			else{
				TRICOLORLED_PORT &= ~(1<<TRICOLORLED_RED_PIN); //turn off
led
			};
			break;
		case LED_GREEN:
			if(ledonoff){
				TRICOLORLED_PORT |= (1<<TRICOLORLED_GREEN_PIN); //turn on
led
			}
			else{
				TRICOLORLED_PORT &= ~(1<<TRICOLORLED_GREEN_PIN); //turn
off led
			};
			break;
		case LED_BLUE:
			if(ledonoff){
				TRICOLORLED_PORT |= (1<<TRICOLORLED_BLUE_PIN); //turn on
led
			}
			else{
				TRICOLORLED_PORT &= ~(1<<TRICOLORLED_BLUE_PIN); //turn
off led
			};
			break;
		default: TRICOLORLED_PORT &=
~((1<<TRICOLORLED_RED_PIN)|(1<<TRICOLORLED_GREEN_PIN)|(1<<TRICOLORLED_BLUE_PIN));
//trun off all the leds
			break;
	}
	};
	void tricolorled_toggle(unsigned char color){
		switch(color){
			case LED_RED:
				TRICOLORLED_PORT ^= (1<<TRICOLORLED_RED_PIN); //toggle led
			  break;
			case LED_GREEN:
				TRICOLORLED_PORT ^= (1<<TRICOLORLED_GREEN_PIN); //toggle led
			  break;
			case LED_BLUE:
```

```
                                TRICOLORLED_PORT ^= (1<<TRICOLORLED_BLUE_PIN); //toggle led
                            break;
                        default:                    TRICOLORLED_PORT                    &=
~((1<<TRICOLORLED_RED_PIN)|(1<<TRICOLORLED_GREEN_PIN)|(1<<TRICOLORLED_BLUE_PIN));
//trun off all the leds
                            break;
                    }
        };
```

----------uart.h----------

```
/*
 * uart.h
 *
 * Created: 10/12/2016 5:59:06 PM
 *  Author: charlie
 */
#ifndef UART_H_
#define UART_H_
#include <stdio.h>
#define USART (&str_uart)

int usart_putchar_printf(char var, FILE *stream);
void USART0_SETUP_9600_BAUD_ASSUME_1MHz_CLOCK();
void USART0_TX_SingleByte(unsigned char cByte);
void USART0_TX_String(char* sData);
void uart_gotoxy(int x, int y);
void uart_clear_screen();

//Declaration of file for Uart
static FILE str_uart = FDEV_SETUP_STREAM(usart_putchar_printf, NULL, _FDEV_SETUP_WRITE);
//fprintf(USART,"Trg: %x\n",Trg);

#endif /* UART_H_ */
```

----------uart.c----------

```
/*
 * uart.c
 *
 * Created: 10/12/2016 5:58:51 PM
 *  Author: charlie
 */
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include <stdio.h>
#include "uart.h"

#define F_CPU 16000000UL
#define CR 0x0D
#define LF 0x0A
#define SPACE 0x20

void USART0_SETUP_9600_BAUD_ASSUME_1MHz_CLOCK()
{
        //UCSR0A ?USART Control and Status Register A
        // bit 7 RXC Receive Complete (flag)
        // bit 6 TXC Transmit Complete (flag)
        // bit 5 UDRE Data Register Empty (flag)
        // bit 4 FE Frame Error (flag) - programmatically clear this when writing to UCSRA
```

```
	// bit 3 DOR Data OverRun (flag)
	// bit 2 PE Parity Error
	// bit 1 UX2 Double the USART TX speed (but also depends on value loaded into the Baud Rate Registers)
	// bit 0 MPCM Multi-Processor Communication Mode
	UCSR0A = 0b00000010; // Set U2X (Double the USART Tx speed, to reduce clocking error)

	// UCSR0B - USART Control and Status Register B
	// bit 7 RXCIE Receive Complete Interrupt Enable
	// bit 6 TXCIE Transmit Complete Interrupt Enable
	// bit 5 UDRIE Data Register Empty Interrupt Enable
	// bit 4 RXEN Receiver Enable
	// bit 3 TXEN Transmitter Enable
	// bit 2 UCSZ2 Character Size (see also UCSZ1:0 in UCSRC)
	// 0 = 5,6,7 or 8-bit data
	// 1 = 9-bit data
	// bit 1 RXB8 RX Data bit 8 (only for 9-bit data)
	// bit 0 TXB8 TX Data bit 8 (only for 9-bit data)
	UCSR0B = 0b10011000;  // RX Complete Int Enable, RX Enable, TX Enable, 8-bit data

	// UCSR0C - USART Control and Status Register C
	// *** This register shares the same I/O location as UBRRH ***
	// Bits 7:6 ?UMSELn1:0 USART Mode Select (00 = Asynchronous)
	// bit 5:4 UPM1:0 Parity Mode
	// 00 Disabled
	// 10 Even parity
	// 11 Odd parity
	// bit 3 USBS Stop Bit Select
	// 0 = 1 stop bit
	// 1 = 2 stop bits
	// bit 2:1 UCSZ1:0 Character Size (see also UCSZ2 in UCSRB)
	// 00 = 5-bit data (UCSZ2 = 0)
	// 01 = 6-bit data (UCSZ2 = 0)
	// 10 = 7-bit data (UCSZ2 = 0)
	// 11 = 8-bit data (UCSZ2 = 0)
	// 11 = 9-bit data (UCSZ2 = 1)
	// bit 0 UCPOL Clock POLarity
	// 0 Rising XCK edge
	// 1 Falling XCK edge
	UCSR0C = 0b00000111;			// Asynchronous, No Parity, 1 stop, 8-bit data, Falling XCK edge

	// UBRR0 - USART Baud Rate Register (16-bit register, comprising UBRR0H and UBRR0L)
	UBRR0H = 0; // 9600 baud, UBRR = 12, and  U2X must be set to '1' in UCSRA
	UBRR0L = F_CPU/8/9600-1;
	//sei();	// Enable interrupts at global level, set Global Interrupt Enable (I) bit
}

 void USART0_TX_SingleByte(unsigned char cByte)
{
	while(!(UCSR0A & (1 << UDRE0)));	// Wait for Tx Buffer to become empty (check UDRE flag)
	UDR0 = cByte;  // Writing to the UDR transmit buffer causes the byte to be transmitted
}

void USART0_TX_String(char* sData)
{
	int iCount;
	int iStrlen = strlen(sData);
	if(0 != iStrlen)
```

```
	{
		for(iCount = 0; iCount < iStrlen; iCount++)
		{
			USART0_TX_SingleByte(sData[iCount]);
		}
		USART0_TX_SingleByte(CR);
		USART0_TX_SingleByte(LF);
	}
}

// void USART0_DisplayBanner()
// {
//	USART0_TX_String("\r\n\n*************************************************************");
//	USART0_TX_String("		Atmel 2560 USART example");
//	USART0_TX_String("*************************************************************");
//	USART0_DisplayPrompt();
// }

// void USART0_DisplayPrompt()
// {
//	USART0_TX_String("Enter command {1,2,3,4} >");
// }

ISR(USART0_RX_vect) // (USART_RX_Complete_Handler) USART Receive-Complete Interrupt Handler
{
//	char cData = UDR0;
//	switch(cData)
//	{
//		case '1':
//		USART0_TX_String("Command '1' received");
//		PORTB = 0b000000001;		// Turn on LED (bit 0)
//		break;
//		case '2':
//		USART0_TX_String("Command '2' received");
//		PORTB = 0b000000010;		// Turn on LED (bit 1)
//		break;
//		case '3':
//		USART0_TX_String("Command '3' received");
//		PORTB = 0b000000100;		// Turn on LED (bit 2)
//		break;
//		case '4':
//		USART0_TX_String("Command '4' received");
//		PORTB = 0b000001000;		// Turn on LED (bit 3)
//		break;
//	}
//	USART0_DisplayPrompt();
}

//Portotype functions
int usart_putchar_printf(char var, FILE *stream)
{
	if (var == '\n') USART0_TX_SingleByte('\r');
	USART0_TX_SingleByte(var);
	return 0;
}

// fprintf(USART,"im usart%d yeeeah",d2);// send format string to Uart "im usart2 yeeeah"
```

```
void uart_gotoxy(int x, int y)
{
	fprintf(USART,"%c[%d;%df",0x1B,y,x);
}

void uart_clear_screen()
{
	printf("\e[1;1H\e[2J");
}
```

----------mrfc522.h----------

```
/*
 * mfrc522.h
 *
 * Copyright 2013 Shimon <shimon@monistit.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */
#ifndef MFRC522_H
#define MFRC522_H

#include <stdint.h>
#include "mfrc522_cmd.h"
#include "mfrc522_reg.h"

#define CARD_FOUND		1
#define CARD_NOT_FOUND	2
#define ERROR			3

#define MAX_LEN				16

//Card types
#define Mifare_UltraLight	0x4400
#define Mifare_One_S50		0x0400
#define Mifare_One_S70		0x0200
#define Mifare_Pro_X		0x0800
#define Mifare_DESFire		0x4403

// Mifare_One card command word
# define PICC_REQIDL		0x26		// find the antenna area does not enter hibernation
# define PICC_REQALL		0x52		// find all the cards antenna area
# define PICC_ANTICOLL		0x93		// anti-collision
# define PICC_SElECTTAG		0x93		// election card
```

```
# define PICC_AUTHENT1A      0x60            // authentication key A
# define PICC_AUTHENT1B      0x61            // authentication key B
# define PICC_READ           0x30            // Read Block
# define PICC_WRITE          0xA0            // write block
# define PICC_DECREMENT      0xC0            // debit
# define PICC_INCREMENT      0xC1            // recharge
# define PICC_RESTORE        0xC2            // transfer block data to the buffer
# define PICC_TRANSFER       0xB0            // save the data in the buffer
# define PICC_HALT           0x50            // Sleep

void mfrc522_init();
void mfrc522_reset();
void mfrc522_write(uint8_t reg, uint8_t data);
uint8_t mfrc522_read(uint8_t reg);
uint8_t  mfrc522_request(uint8_t req_mode, uint8_t * tag_type);
uint8_t  mfrc522_to_card(uint8_t  cmd,  uint8_t  *send_data,  uint8_t  send_data_len,  uint8_t  *back_data,  uint32_t
*back_data_len);
uint8_t mfrc522_get_card_serial(uint8_t * serial_out);
#endif
```

øø
xxc

---

ccc

----------mrfc522_cmd.h----------

```
/*
 * mfrc522_cmd.h
 *
 * Copyright 2013 Shimon <shimon@monistit.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */
#ifndef MFRC522_CMD_H
#define MFRC522_CMD_H

//command set
#define Idle_CMD                    0x00
#define Mem_CMD                         0x01
#define GenerateRandomId_CMD        0x02
#define CalcCRC_CMD                 0x03
#define Transmit_CMD                0x04
#define NoCmdChange_CMD             0x07
```

```
#define Receive_CMD                     0x08
#define Transceive_CMD                  0x0C
#define Reserved_CMD            0x0D
#define MFAuthent_CMD                   0x0E
#define SoftReset_CMD           0x0F

#endif
```

---------mrfc522_reg.h----------

```
/*
 * mfrc522_reg.h
 *
 * Copyright 2013 Shimon <shimon@monistit.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */
#ifndef _MFRC522_REG_H
#define _MFRC522_REG_H

//Page 0 ==> Command and Status
#define Page0_Reserved_1        0x00
#define CommandReg              0x01
#define ComIEnReg               0x02
#define DivIEnReg               0x03
#define ComIrqReg               0x04
#define DivIrqReg               0x05
#define ErrorReg                0x06
#define Status1Reg              0x07
#define Status2Reg              0x08
#define FIFODataReg             0x09
#define FIFOLevelReg            0x0A
#define WaterLevelReg           0x0B
#define ControlReg              0x0C
#define BitFramingReg           0x0D
#define CollReg                 0x0E
#define Page0_Reserved_2        0x0F

//Page 1 ==> Command
#define Page1_Reserved_1        0x10
#define ModeReg                         0x11
#define TxModeReg               0x12
#define RxModeReg               0x13
#define TxControlReg            0x14
```

```
#define TxASKReg                0x15
#define TxSelReg                0x16
#define RxSelReg                0x17
#define RxThresholdReg          0x18
#define  DemodReg               0x19
#define Page1_Reserved_2        0x1A
#define Page1_Reserved_3        0x1B
#define MfTxReg                         0x1C
#define MfRxReg                         0x1D
#define Page1_Reserved_4        0x1E
#define SerialSpeedReg          0x1F

//Page 2 ==> CFG
#define Page2_Reserved_1        0x20
#define CRCResultReg_1          0x21
#define CRCResultReg_2          0x22
#define Page2_Reserved_2        0x23
#define ModWidthReg             0x24
#define Page2_Reserved_3        0x25
#define RFCfgReg                0x26
#define GsNReg                  0x27
#define CWGsPReg                0x28
#define ModGsPReg               0x29
#define TModeReg                0x2A
#define TPrescalerReg           0x2B
#define TReloadReg_1            0x2C
#define TReloadReg_2            0x2D
#define TCounterValReg_1        0x2E
#define TCounterValReg_2        0x2F

//Page 3 ==> TestRegister
#define Page3_Reserved_1        0x30
#define TestSel1Reg             0x31
#define TestSel2Reg             0x32
#define TestPinEnReg            0x33
#define TestPinValueReg         0x34
#define TestBusReg              0x35
#define AutoTestReg             0x36
#define VersionReg              0x37
#define AnalogTestReg           0x38
#define TestDAC1Reg             0x39
#define TestDAC2Reg             0x3A
#define TestADCReg              0x3B
#define Page3_Reserved_2        0x3C
#define Page3_Reserved_3        0x3D
#define Page3_Reserved_4        0x3E
#define Page3_Reserved_5        0x3F

#endif
```

----------mrfc522.c----------

```
/*
 * mfrc522.c
 *
 * Copyright 2013 Shimon <shimon@monistit.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
```

```
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */
#include "mfrc522.h"
#include "spi.h"

#if 0
#include <lcd.h>
#endif
void mfrc522_init()
{
	uint8_t byte;
	mfrc522_reset();//soft reset

	mfrc522_write(TModeReg, 0x8D);// timer starts automatically, TPrescaler[11:8]:0xD
   mfrc522_write(TPrescalerReg, 0x3E);//TPrescalerReg[11:0]: 0xD3E
   mfrc522_write(TReloadReg_1, 30);
   mfrc522_write(TReloadReg_2, 0);	//TReloadVal[15:0]:0x3000,
Td=(TPrescaler*2+1)*(TReloadVal+1)/13.56MHz=6s
	mfrc522_write(TxASKReg, 0x40);		//forces a 100 % ASK modulation independent of the ModGsPReg
register setting
	mfrc522_write(ModeReg, 0x3D);//TxWaitRF:1 transmitter can only be started if an RF field is
generated;polarity of pin MFIN is active HIGH

	byte = mfrc522_read(TxControlReg);
	if(!(byte&0x03))
	{
		mfrc522_write(TxControlReg,byte|0x03);//output signal on pin TX1, TX2
	}
}
void mfrc522_write(uint8_t reg, uint8_t data)
{
	ENABLE_CHIP();
	spi_transmit((reg<<1)&0x7E);//SPI address byte format @ page11 of MFRC522.pdf
	spi_transmit(data);
	DISABLE_CHIP();
}
uint8_t mfrc522_read(uint8_t reg)
{
	uint8_t data;
	ENABLE_CHIP();
	spi_transmit(((reg<<1)&0x7E)|0x80);
	data = spi_transmit(0x00);
	DISABLE_CHIP();
	return data;
```

```
}
void mfrc522_reset()
{
	mfrc522_write(CommandReg,SoftReset_CMD);
}
uint8_t  mfrc522_request(uint8_t req_mode, uint8_t * tag_type)
{
	uint8_t  status;
	uint32_t backBits;//The received data bits

	mfrc522_write(BitFramingReg, 0x07);//TxLastBists = BitFramingReg[2..0]		???

	tag_type[0] = req_mode;
	status = mfrc522_to_card(Transceive_CMD, tag_type, 1, tag_type, &backBits);

	if ((status != CARD_FOUND) || (backBits != 0x10))
	{
		status = ERROR;
	}
	return status;
}
uint8_t  mfrc522_to_card(uint8_t  cmd,  uint8_t  *send_data,  uint8_t  send_data_len,  uint8_t  *back_data,  uint32_t
*back_data_len)
{
	uint8_t status = ERROR;
   uint8_t irqEn = 0x00;
   uint8_t waitIRq = 0x00;
   uint8_t lastBits;
   uint8_t n;
   uint8_t	tmp;
   uint32_t i;

   switch (cmd)
   {
      case MFAuthent_CMD:		//Certification cards close
		{
			irqEn = 0x12;
			waitIRq = 0x10;
			break;
		}
		case Transceive_CMD:  //Transmit FIFO data
		{
			irqEn = 0x77;
			waitIRq = 0x30;
			break;
		}
		default:
			break;
   }

   //mfrc522_write(ComIEnReg, irqEn|0x80);	//Interrupt request
   n=mfrc522_read(ComIrqReg);
   mfrc522_write(ComIrqReg,n&(~0x80));//clear all interrupt bits
   n=mfrc522_read(FIFOLevelReg);
   mfrc522_write(FIFOLevelReg,n|0x80);//flush FIFO data

	mfrc522_write(CommandReg, Idle_CMD);	//NO action; Cancel the current cmd???
```

```
        //Writing data to the FIFO
  for (i=0; i<send_data_len; i++)
  {
                mfrc522_write(FIFODataReg, send_data[i]);
        }

        //Execute the cmd
        mfrc522_write(CommandReg, cmd);
  if (cmd == Transceive_CMD)
  {
                n=mfrc522_read(BitFramingReg);
                mfrc522_write(BitFramingReg,n|0x80);
        }

        //Waiting to receive data to complete
        i = 2000;       //i according to the clock frequency adjustment, the operator M1 card maximum waiting time
25ms???
  do
  {
                //CommIrqReg[7..0]
                //Set1 TxIRq RxIRq IdleIRq HiAlerIRq LoAlertIRq ErrIRq TimerIRq
    n = mfrc522_read(ComIrqReg);
    i--;
  }
  while ((i!=0) && !(n&0x01) && !(n&waitIRq));

        tmp=mfrc522_read(BitFramingReg);
        mfrc522_write(BitFramingReg,tmp&(~0x80));

  if (i != 0)
  {
    if(!(mfrc522_read(ErrorReg) & 0x1B))        //BufferOvfl Collerr CRCErr ProtecolErr
    {
      status = CARD_FOUND;
      if (n & irqEn & 0x01)
      {
                        status = CARD_NOT_FOUND;                        //??
                }

      if (cmd == Transceive_CMD)
      {
                n = mfrc522_read(FIFOLevelReg);
                lastBits = mfrc522_read(ControlReg) & 0x07;
        if (lastBits)
        {
                        *back_data_len = (n-1)*8 + lastBits;
                }
        else
        {
                        *back_data_len = n*8;
                }
        if (n == 0)
        {
                        n = 1;
                }
        if (n > MAX_LEN)
```

```
            {
                                        n = MAX_LEN;
                                    }
                                    //Reading the received data in FIFO
            for (i=0; i<n; i++)
            {
                                        back_data[i] = mfrc522_read(FIFODataReg);
                                    }
         }
      }
      else
      {
                        status = ERROR;
                    }

   }

   //SetBitMask(ControlReg,0x80);          //timer stops
   //mfrc522_write(cmdReg, PCD_IDLE);

   return status;
}


uint8_t mfrc522_get_card_serial(uint8_t * serial_out)
{
          uint8_t status;
   uint8_t i;
          uint8_t serNumCheck=0;
   uint32_t unLen;

          mfrc522_write(BitFramingReg, 0x00);                //TxLastBists = BitFramingReg[2..0]

   serial_out[0] = PICC_ANTICOLL;
   serial_out[1] = 0x20;
   status = mfrc522_to_card(Transceive_CMD, serial_out, 2, serial_out, &unLen);

   if (status == CARD_FOUND)
          {
                    //Check card serial number
                    for (i=0; i<4; i++)
                    {
                              serNumCheck ^= serial_out[i];
                    }
                    if (serNumCheck != serial_out[i])
                    {
                              status = ERROR;
                    }
   }
   return status;
}
```

----------spi.h----------

```
/*
 * spi.h
 *
 * Copyright 2013 Shimon <shimon@monistit.com>
 *
```

```
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */
#ifndef SPI_H
#define SPI_H

#include <stdint.h>
#include "spi_config.h"


void spi_init();
uint8_t spi_transmit(uint8_t data);

#define ENABLE_CHIP() (SPI_PORT &= (~(1<<SPI_SS)))
#define DISABLE_CHIP() (SPI_PORT |= (1<<SPI_SS))
#endif
```

----------spi_config.h----------

```
/*
 * spi_config.h
 *
 * Copyright 2013 Shimon <shimon@monistit.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */
#ifndef SPI_CONFIG_H
#define SPI_CONFIG_H

#include <avr/io.h>
```

```
/*
 * Set to 1, spi api will work in master mode
 * else in slave mode
 */
#define SPI_CONFIG_AS_MASTER      1
/*
 * Config SPI pin diagram
 */
#define SPI_DDR                 DDRB
#define SPI_PORT        PORTB
#define SPI_PIN                 PINB
#define SPI_MOSI        PB2
#define SPI_MISO        PB3
#define SPI_SS          PB0
#define SPI_SCK                 PB1

#endif
```

----------spi.c----------

```
/*
 * spi.c
 *
 * Copyright 2013 Shimon <shimon@monistit.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */
#include "spi.h"

#if SPI_CONFIG_AS_MASTER
void spi_init()
{
        SPI_DDR |= (1<<SPI_MOSI)|(1<<SPI_SCK)|(1<<SPI_SS);
        SPI_DDR &= ~(1<<SPI_MISO);
        SPCR |= (1<<SPE)|(1<<MSTR)|(1<<SPR0);//prescaler 16
}


uint8_t spi_transmit(uint8_t data)
{
        SPDR = data;
        while(!(SPSR & (1<<SPIF)));
```

```
	return SPDR;
}

#else
void spi_init()
{
	SPI_DDR |= (1<<SPI_MISO);
	SPCR |= (1<<SPE);
}

uint8_t spi_transmit(uint8_t data)
{
	while(!(SPSR & (1<<SPIF)));
	return SPDR;
}
#endif
```

----------i2cmaster.h----------

```
#ifndef _I2CMASTER_H
#define _I2CMASTER_H   1
/*************************************************************************
* Title:    C include file for the I2C master interface
*           (i2cmaster.S or twimaster.c)
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: i2cmaster.h,v 1.10 2005/03/06 22:39:57 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device
* Usage:    see Doxygen manual
*************************************************************************/
#ifdef DOXYGEN
/**
 @defgroup pfleury_ic2master I2C Master library
 @code #include <i2cmaster.h> @endcode

 @brief I2C (TWI) Master Software Library

 Basic routines for communicating with I2C slave devices. This single master
 implementation is limited to one bus master on the I2C bus.

 This I2c library is implemented as a compact assembler software implementation of the I2C protocol
 which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for all AVR with built-in TWI hardware (twimaster.c).
 Since the API for these two implementations is exactly the same, an application can be linked either against the
 software I2C implementation or the hardware I2C implementation.

 Use 4.7k pull-up resistor on the SDA and SCL pin.

 Adapt the SCL and SDA port and pin definitions and eventually the delay routine in the module
 i2cmaster.S to your target when using the software I2C implementation !

 Adjust the  CPU clock frequence F_CPU in twimaster.c or in the Makfile when using the TWI hardware implementaion.

 @note
   The module i2cmaster.S is based on the Atmel Application Note AVR300, corrected and adapted
   to GNU assembler and AVR-GCC C call interface.
   Replaced the incorrect quarter period delays found in AVR300 with
   half period delays.
```

```
 @author Peter Fleury pfleury@gmx.ch  http://jump.to/fleury

 @par API Usage Example
  The following code shows typical usage of this library, see example test_i2cmaster.c

 @code

 #include <i2cmaster.h>

 #define Dev24C02  0xA2      // device address of EEPROM 24C02, see datasheet

 int main(void)
 {
    unsigned char ret;

    i2c_init();                          // initialize I2C library

    // write 0x75 to EEPROM address 5 (Byte Write)
    i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
    i2c_write(0x05);                     // write address = 5
    i2c_write(0x75);                     // write value 0x75 to EEPROM
    i2c_stop();                          // set stop conditon = release bus


    // read previously written value back from EEPROM address 5
    i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode

    i2c_write(0x05);                     // write address = 5
    i2c_rep_start(Dev24C02+I2C_READ);       // set device address and read mode

    ret = i2c_readNak();                  // read one byte from EEPROM
    i2c_stop();

    for(;;);
 }
 @endcode

*/
#endif /* DOXYGEN */

/**@{*/

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
#error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC compiler !"
#endif

#include <avr/io.h>
#ifndef F_CPU
#define F_CPU 16000000UL
#endif
/** defines the data direction (reading from I2C device) in i2c_start(),i2c_rep_start() */
#define I2C_READ    1

/** defines the data direction (writing to I2C device) in i2c_start(),i2c_rep_start() */
#define I2C_WRITE   0
```

```
/**
 @brief initialize the I2C master interace. Need to be called only once
 @param  void
 @return none
 */
extern void i2c_init(void);
/**
 @brief Terminates the data transfer and releases the I2C bus
 @param void
 @return none
 */
extern void i2c_stop(void);


/**
 @brief Issues a start condition and sends address and transfer direction

 @param    addr address and transfer direction of I2C device
 @retval   0   device accessible
 @retval   1   failed to access device
 */
extern unsigned char i2c_start(unsigned char addr);
/**
 @brief Issues a repeated start condition and sends address and transfer direction

 @param   addr address and transfer direction of I2C device
 @retval  0 device accessible
 @retval  1 failed to access device
 */
extern unsigned char i2c_rep_start(unsigned char addr);
/**
 @brief Issues a start condition and sends address and transfer direction

 If device is busy, use ack polling to wait until device ready
 @param    addr address and transfer direction of I2C device
 @return   none
 */
extern void i2c_start_wait(unsigned char addr);


/**
 @brief Send one byte to I2C device
 @param    data  byte to be transfered
 @retval   0 write successful
 @retval   1 write failed
 */
extern unsigned char i2c_write(unsigned char data);
/**
 @brief    read one byte from the I2C device, request more data from device
 @return   byte read from I2C device
 */
extern unsigned char i2c_readAck(void);

/**
 @brief    read one byte from the I2C device, read is followed by a stop condition
 @return   byte read from I2C device
```

```
 */
extern unsigned char i2c_readNak(void);

/**
 @brief    read one byte from the I2C device

 Implemented as a macro, which calls either i2c_readAck or i2c_readNak

 @param    ack 1 send ack, request more data from device<br>
           0 send nak, read is followed by a stop condition
 @return   byte read from I2C device
 */
extern unsigned char i2c_read(unsigned char ack);
#define i2c_read(ack)  (ack) ? i2c_readAck() : i2c_readNak();


/**@}*/
#endif
```

---------i2cmaster.c----------

```
/*************************************************************************
* Title:    I2C master library using hardware TWI interface
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: twimaster.c,v 1.3 2005/07/02 11:14:21 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device with hardware TWI
* Usage:    API compatible with I2C Software Library i2cmaster.h
*************************************************************************/
#include <inttypes.h>
#include <compat/twi.h>

#include "i2cmaster.h"

/* define CPU frequency in Mhz here if not defined in Makefile */

/* I2C clock in Hz */
#define SCL_CLOCK  100000L

/*************************************************************************
 Initialization of the I2C bus interface. Need to be called only once
*************************************************************************/
void i2c_init(void)
{
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */

  TWSR = 0;                         /* no prescaler */
  TWBR = ((F_CPU/SCL_CLOCK)-16)/8;  /* must be > 10 for stable operation */

}/* i2c_init */
/*************************************************************************
  Issues a start condition and sends address and transfer direction.
  return 0 = device accessible, 1= failed to access device
*************************************************************************/
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;

        // send START condition
```

```
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wail until transmission completed and ACK/NACK has been received
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

        return 0;

}/* i2c_start */
/*************************************************************************
 Issues a start condition and sends address and transfer direction.
 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device
*************************************************************************/
void i2c_start_wait(unsigned char address)
{
   uint8_t   twst;
   while ( 1 )
   {
            // send START condition
            TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wail until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
        {
           /* device busy, send stop condition to terminate write operation */
              TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
```

```
            // wait until stop condition is executed and bus released
            while(TWCR & (1<<TWSTO));

          continue;
        }
        //if( twst != TW_MT_SLA_ACK) return 1;
        break;
    }

}/* i2c_start_wait */
/*************************************************************************
 Issues a repeated start condition and sends address and transfer direction

 Input:   address and transfer direction of I2C device

 Return:  0 device accessible
        1 failed to access device
*************************************************************************/
unsigned char i2c_rep_start(unsigned char address)
{
    return i2c_start( address );

}/* i2c_rep_start */
/*************************************************************************
 Terminates the data transfer and releases the I2C bus
*************************************************************************/
void i2c_stop(void)
{
    /* send stop condition */
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

        // wait until stop condition is executed and bus released
        while(TWCR & (1<<TWSTO));

}/* i2c_stop */
/*************************************************************************
  Send one byte to I2C device

  Input:    byte to be transfered
  Return:   0 write successful
          1 write failed
*************************************************************************/
unsigned char i2c_write( unsigned char data )
{
    uint8_t   twst;

        // send data to the previously addressed device
        TWDR = data;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits
        twst = TW_STATUS & 0xF8;
        if( twst != TW_MT_DATA_ACK) return 1;
```

```
        return 0;

}/* i2c_write */


/*************************************************************************
 Read one byte from the I2C device, request more data from device
  Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readAck(void)
{
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
        while(!(TWCR & (1<<TWINT)));

   return TWDR;

}/* i2c_readAck */
/*************************************************************************
 Read one byte from the I2C device, read is followed by a stop condition

 Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readNak(void)
{
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT)));

   return TWDR;

}/* i2c_readNak */
```

----------lcdpcf8574.h----------

```
/*
lcdpcf8574 lib 0x01

copyright (c) Davide Gironi, 2013

Released under GPLv3.
Please refer to LICENSE file for licensing information.

References:
  + based on lcd library by Peter Fleury
    http://jump.to/fleury
*/
#ifndef LCD_H
#define LCD_H

#include <inttypes.h>
#include <avr/pgmspace.h>

#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#define LCD_PCF8574_INIT 1 //init pcf8574

#define LCD_PCF8574_DEVICEID 7 //device id, addr = pcf8574 base addr + LCD_PCF8574_DEVICEID
/**
```

```
 *  @name  Definitions for Display Size
 *  Change these definitions to adapt setting to your display
 */
#define LCD_LINES           2     /**< number of visible lines of the display */
#define LCD_DISP_LENGTH     16    /**< visibles characters per line of the display */
#define LCD_LINE_LENGTH  0x40     /**< internal line length of the display    */
#define LCD_START_LINE1  0x00     /**< DDRAM address of first char of line 1 */
#define LCD_START_LINE2  0x40     /**< DDRAM address of first char of line 2 */
#define LCD_START_LINE3  0x14     /**< DDRAM address of first char of line 3 */
#define LCD_START_LINE4  0x54     /**< DDRAM address of first char of line 4 */
#define LCD_WRAP_LINES      1     /**< 0: no wrap, 1: wrap at end of visibile line */



#define LCD_DATA0_PIN    4          /**< pin for 4bit data bit 0  */
#define LCD_DATA1_PIN    5          /**< pin for 4bit data bit 1  */
#define LCD_DATA2_PIN    6          /**< pin for 4bit data bit 2  */
#define LCD_DATA3_PIN    7          /**< pin for 4bit data bit 3  */
#define LCD_RS_PIN       0          /**< pin  for RS line        */
#define LCD_RW_PIN       1          /**< pin  for RW line        */
#define LCD_E_PIN        2          /**< pin  for Enable line    */
#define LCD_LED_PIN      3          /**< pin  for Led            */
/**
 *  @name Definitions for LCD command instructions
 *  The constants define the various LCD controller instructions which can be passed to the
 *  function lcd_command(), see HD44780 data sheet for a complete description.
 */

/* instruction register bit positions, see HD44780U data sheet */
#define LCD_CLR             0     /* DB0: clear display                */
#define LCD_HOME            1     /* DB1: return to home position      */
#define LCD_ENTRY_MODE      2     /* DB2: set entry mode               */
#define LCD_ENTRY_INC       1     /*   DB1: 1=increment, 0=decrement   */
#define LCD_ENTRY_SHIFT     0     /*   DB2: 1=display shift on         */
#define LCD_ON              3     /* DB3: turn lcd/cursor on           */
#define LCD_ON_DISPLAY      2     /*   DB2: turn display on            */
#define LCD_ON_CURSOR       1     /*   DB1: turn cursor on             */
#define LCD_ON_BLINK        0     /*    DB0: blinking cursor ?         */
#define LCD_MOVE            4     /* DB4: move cursor/display          */
#define LCD_MOVE_DISP       3     /*   DB3: move display (0-> cursor) ?  */
#define LCD_MOVE_RIGHT      2     /*   DB2: move right (0-> left) ?    */
#define LCD_FUNCTION        5     /* DB5: function set                 */
#define LCD_FUNCTION_8BIT   4     /*   DB4: set 8BIT mode (0->4BIT mode) */
#define LCD_FUNCTION_2LINES 3     /*   DB3: two lines (0->one line)    */
#define LCD_FUNCTION_10DOTS 2     /*   DB2: 5x10 font (0->5x7 font)    */
#define LCD_CGRAM           6     /* DB6: set CG RAM address           */
#define LCD_DDRAM           7     /* DB7: set DD RAM address           */
#define LCD_BUSY            7     /* DB7: LCD is busy                  */

/* set entry mode: display shift on/off, dec/inc cursor move direction */
#define LCD_ENTRY_DEC           0x04   /* display shift off, dec cursor move dir */
#define LCD_ENTRY_DEC_SHIFT     0x05   /* display shift on,  dec cursor move dir */
#define LCD_ENTRY_INC_          0x06   /* display shift off, inc cursor move dir */
#define LCD_ENTRY_INC_SHIFT     0x07   /* display shift on,  inc cursor move dir */

/* display on/off, cursor on/off, blinking char at cursor position */
#define LCD_DISP_OFF            0x08   /* display off                        */
```

```
#define LCD_DISP_ON             0x0C   /* display on, cursor off                */
#define LCD_DISP_ON_BLINK       0x0D   /* display on, cursor off, blink char    */
#define LCD_DISP_ON_CURSOR      0x0E   /* display on, cursor on                 */
#define LCD_DISP_ON_CURSOR_BLINK 0x0F  /* display on, cursor on, blink char     */

/* move cursor/shift display */
#define LCD_MOVE_CURSOR_LEFT    0x10   /* move cursor left  (decrement)         */
#define LCD_MOVE_CURSOR_RIGHT   0x14   /* move cursor right (increment)         */
#define LCD_MOVE_DISP_LEFT      0x18   /* shift display left                    */
#define LCD_MOVE_DISP_RIGHT     0x1C   /* shift display right                   */

/* function set: set interface data length and number of display lines */
#define LCD_FUNCTION_4BIT_1LINE  0x20   /* 4-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_4BIT_2LINES 0x28   /* 4-bit interface, dual line,   5x7 dots */
#define LCD_FUNCTION_8BIT_1LINE  0x30   /* 8-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_8BIT_2LINES 0x38   /* 8-bit interface, dual line,   5x7 dots */

#define LCD_MODE_DEFAULT     ((1<<LCD_ENTRY_MODE) | (1<<LCD_ENTRY_INC) )

#define LCD_Backlight_ON     0
#define LCD_Backlight_OFF    1

/**
 *  @name Functions
 */

/**
 @brief    Initialize display and select type of cursor
 @param    dispAttr \b LCD_DISP_OFF display off\n
               \b LCD_DISP_ON display on, cursor off\n
               \b LCD_DISP_ON_CURSOR display on, cursor on\n
               \b LCD_DISP_ON_CURSOR_BLINK display on, cursor on flashing
 @return  none
*/
extern void lcd_init(uint8_t dispAttr);
/**
 @brief    Clear display and set cursor to home position
 @param    void
 @return   none
*/
extern void lcd_clrscr(void);

/**
 @brief    Set cursor to home position
 @param    void
 @return   none
*/
extern void lcd_home(void);

/**
 @brief    Set cursor to specified position

 @param    x horizontal position\n (0: left most position)
 @param    y vertical position\n   (0: first line)
 @return   none
*/
extern void lcd_gotoxy(uint8_t x, uint8_t y);
```

```
/**
 @brief    Set illumination pin
 @param    void
 @return   none
*/
extern void lcd_led(uint8_t onoff);
/**
 @brief    Display character at current cursor position
 @param    c character to be displayed
 @return   none
*/
extern void lcd_putc(char c);
/**
 @brief    Display string without auto linefeed
 @param    s string to be displayed
 @return   none
*/
extern void lcd_puts(const char *s);
/**
 @brief    Display string from program memory without auto linefeed
 @param    s string from program memory be be displayed
 @return   none
 @see      lcd_puts_P
*/
extern void lcd_puts_p(const char *progmem_s);
/**
 @brief    Send LCD controller instruction command
 @param    cmd instruction to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_command(uint8_t cmd);

/**
 @brief    Send data byte to LCD controller

 Similar to lcd_putc(), but without interpreting LF
 @param    data byte to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_data(uint8_t data);

/**
 @brief macros for automatically storing string constant in program memory
*/
#define lcd_puts_P(__s)         lcd_puts_p(PSTR(__s))

/*@}*/
#endif //LCD_H
```

----------lcdpcf8574.c----------

```
/*
lcdpcf8574 lib 0x01

copyright (c) Davide Gironi, 2013

Released under GPLv3.
Please refer to LICENSE file for licensing information.
```

```
*/

#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>

#include "pcf8574.h"

#include "lcdpcf8574.h"

#define lcd_e_delay()   __asm__ __volatile__( "rjmp 1f\n 1:" );
#define lcd_e_toggle()  toggle_e()

#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_2LINES
#endif

volatile uint8_t dataport = 0;

/*
** function prototypes
*/
static void toggle_e(void);

/*
** local functions
*/
/************************************************************************
 delay loop for small accurate delays: 16-bit counter, 4 cycles/loop
************************************************************************/
static inline void _delayFourCycles(unsigned int __count)
{
   if ( __count == 0 )
      __asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
   else
      __asm__ __volatile__ (
            "1: sbiw %0,1" "\n\t"
            "brne 1b"                        // 4 cycles/loop
            : "=w" (__count)
            : "0" (__count)
           );
}
/************************************************************************
delay for a minimum of <us> microseconds
the number of loops is calculated at compile-time from MCU clock frequency
************************************************************************/
#define delay(us)  _delayFourCycles( ( ( 1*(F_CPU/4000) )*us)/1000 )


/* toggle Enable Pin to initiate write */
static void toggle_e(void)
{
         pcf8574_setoutputpinhigh(LCD_PCF8574_DEVICEID, LCD_E_PIN);
   lcd_e_delay();
   pcf8574_setoutputpinlow(LCD_PCF8574_DEVICEID, LCD_E_PIN);
```

```
}
/************************************************************************
Low-level function to write byte to LCD controller
Input:    data   byte to write to LCD
          rs     1: write data
                 0: write instruction
Returns:  none
************************************************************************/
static void lcd_write(uint8_t data,uint8_t rs)
{
          if (rs) /* write data        (RS=1, RW=0) */
                    dataport |= _BV(LCD_RS_PIN);
          else /* write instruction (RS=0, RW=0) */
                    dataport &= ~_BV(LCD_RS_PIN);
          dataport &= ~_BV(LCD_RW_PIN);
          pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);

          /* output high nibble first */
   dataport &= ~_BV(LCD_DATA3_PIN);
   dataport &= ~_BV(LCD_DATA2_PIN);
   dataport &= ~_BV(LCD_DATA1_PIN);
   dataport &= ~_BV(LCD_DATA0_PIN);
          if(data & 0x80) dataport |= _BV(LCD_DATA3_PIN);
          if(data & 0x40) dataport |= _BV(LCD_DATA2_PIN);
          if(data & 0x20) dataport |= _BV(LCD_DATA1_PIN);
          if(data & 0x10) dataport |= _BV(LCD_DATA0_PIN);
          pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);
          lcd_e_toggle();

          /* output low nibble */
          dataport &= ~_BV(LCD_DATA3_PIN);
          dataport &= ~_BV(LCD_DATA2_PIN);
          dataport &= ~_BV(LCD_DATA1_PIN);
          dataport &= ~_BV(LCD_DATA0_PIN);
          if(data & 0x08) dataport |= _BV(LCD_DATA3_PIN);
          if(data & 0x04) dataport |= _BV(LCD_DATA2_PIN);
          if(data & 0x02) dataport |= _BV(LCD_DATA1_PIN);
          if(data & 0x01) dataport |= _BV(LCD_DATA0_PIN);
          pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);
          lcd_e_toggle();

          /* all data pins high (inactive) */
          dataport |= _BV(LCD_DATA0_PIN);
          dataport |= _BV(LCD_DATA1_PIN);
          dataport |= _BV(LCD_DATA2_PIN);
          dataport |= _BV(LCD_DATA3_PIN);
          pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);
}
/************************************************************************
Low-level function to read byte from LCD controller
Input:    rs     1: read data
                 0: read busy flag / address counter
Returns:  byte read from LCD controller
************************************************************************/
static uint8_t lcd_read(uint8_t rs)
{
   uint8_t data;
```

```
   if (rs) /* write data        (RS=1, RW=0) */
          dataport |= _BV(LCD_RS_PIN);
   else /* write instruction (RS=0, RW=0) */
          dataport &= ~_BV(LCD_RS_PIN);
   dataport |= _BV(LCD_RW_PIN);
   pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);

   pcf8574_setoutputpinhigh(LCD_PCF8574_DEVICEID, LCD_E_PIN);
          lcd_e_delay();
          data = pcf8574_getoutputpin(LCD_PCF8574_DEVICEID, LCD_DATA0_PIN) << 4;       /* read high nibble
first */
          pcf8574_setoutputpinlow(LCD_PCF8574_DEVICEID, LCD_E_PIN);

          lcd_e_delay();                    /* Enable 500ns low       */

          pcf8574_setoutputpinhigh(LCD_PCF8574_DEVICEID, LCD_E_PIN);
          lcd_e_delay();
          data |= pcf8574_getoutputpin(LCD_PCF8574_DEVICEID, LCD_DATA0_PIN) &0x0F;     /* read low nibble
*/
          pcf8574_setoutputpinlow(LCD_PCF8574_DEVICEID, LCD_E_PIN);

   return data;
}
/************************************************************************
loops while lcd is busy, returns address counter
************************************************************************/
static uint8_t lcd_waitbusy(void)
{
   register uint8_t c;

   /* wait until busy flag is cleared */
   while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}

   /* the address counter is updated 4us after the busy flag is cleared */
   delay(2);

   /* now read the address counter */
   return (lcd_read(0));  // return address counter

}/* lcd_waitbusy */
/************************************************************************
Move cursor to the start of next line or to the first line if the cursor
is already on the last line.
************************************************************************/
static inline void lcd_newline(uint8_t pos)
{
   register uint8_t addressCounter;


#if LCD_LINES==1
   addressCounter = 0;
#endif
#if LCD_LINES==2
   if ( pos < (LCD_START_LINE2) )
      addressCounter = LCD_START_LINE2;
   else
```

```
      addressCounter = LCD_START_LINE1;
#endif
#if LCD_LINES==4
   if ( pos < LCD_START_LINE3 )
      addressCounter = LCD_START_LINE2;
   else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE4) )
      addressCounter = LCD_START_LINE3;
   else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE2) )
      addressCounter = LCD_START_LINE4;
   else
      addressCounter = LCD_START_LINE1;
#endif
   lcd_command((1<<LCD_DDRAM)+addressCounter);

}/* lcd_newline */

/*
** PUBLIC FUNCTIONS
*/
/*************************************************************************
Send LCD controller instruction command
Input:   instruction to send to LCD controller, see HD44780 data sheet
Returns: none
*************************************************************************/
void lcd_command(uint8_t cmd)
{
   lcd_waitbusy();
   lcd_write(cmd,0);
}


/*************************************************************************
Send data byte to LCD controller
Input:   data to send to LCD controller, see HD44780 data sheet
Returns: none
*************************************************************************/
void lcd_data(uint8_t data)
{
   lcd_waitbusy();
   lcd_write(data,1);
}
/*************************************************************************
Set cursor to specified position
Input:    x  horizontal position  (0: left most position)
          y  vertical position    (0: first line)
Returns:  none
*************************************************************************/
void lcd_gotoxy(uint8_t x, uint8_t y)
{
#if LCD_LINES==1
   lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
#endif
#if LCD_LINES==2
   if ( y==0 )
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
   else
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
```

```
#endif
#if LCD_LINES==4
   if ( y==0 )
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
   else if ( y==1)
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
   else if ( y==2)
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
   else /* y==3 */
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);
#endif

}/* lcd_gotoxy */


/*************************************************************************
*************************************************************************/
int lcd_getxy(void)
{
   return lcd_waitbusy();
}
/*************************************************************************
Clear display and set cursor to home position
*************************************************************************/
void lcd_clrscr(void)
{
   lcd_command(1<<LCD_CLR);
}
/*************************************************************************
Set illumination pin
*************************************************************************/
void lcd_led(uint8_t onoff)
{
        if(onoff)
                dataport &= ~_BV(LCD_LED_PIN);
        else
                dataport |= _BV(LCD_LED_PIN);
        pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);
}
/*************************************************************************
Set cursor to home position
*************************************************************************/
void lcd_home(void)
{
   lcd_command(1<<LCD_HOME);
}


/*************************************************************************
Display character at current cursor position
Input:    character to be displayed
Returns:  none
*************************************************************************/
void lcd_putc(char c)
{
   uint8_t pos;
```

```
    pos = lcd_waitbusy();   // read busy-flag and address counter
    if (c=='\n')
    {
        lcd_newline(pos);
    }
    else
    {
#if LCD_WRAP_LINES==1
#if LCD_LINES==1
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#elif LCD_LINES==2
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
        }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ){
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#elif LCD_LINES==4
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
        }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
        }else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
        }else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH ) {
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        }
#endif
        lcd_waitbusy();
#endif
        lcd_write(c, 1);
    }

}/* lcd_putc */
/*************************************************************************
Display string without auto linefeed
Input:    string to be displayed
Returns:  none
*************************************************************************/
void lcd_puts(const char *s)
/* print string on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = *s++) ) {
        lcd_putc(c);
    }

}/* lcd_puts */

/*************************************************************************
Display string from program memory without auto linefeed
Input:     string from program memory be be displayed
Returns:   none
*************************************************************************/
void lcd_puts_p(const char *progmem_s)
```

```
/* print string from program memory on lcd (no auto linefeed) */
{
   register char c;
   while ( (c = pgm_read_byte(progmem_s++)) ) {
      lcd_putc(c);
   }

}/* lcd_puts_p */

/*************************************************************************
Initialize display and select type of cursor
Input:    dispAttr LCD_DISP_OFF            display off
               LCD_DISP_ON             display on, cursor off
               LCD_DISP_ON_CURSOR      display on, cursor on
               LCD_DISP_CURSOR_BLINK   display on, cursor on flashing
Returns:  none
*************************************************************************/
void lcd_init(uint8_t dispAttr)
{
          #if LCD_PCF8574_INIT == 1
          //init pcf8574
          pcf8574_init();
          #endif

          dataport = 0;
          pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);

   delay(16000);        /* wait 16ms or more after power-on       */

   /* initial write to lcd is 8bit */
   dataport |= _BV(LCD_DATA1_PIN);  // _BV(LCD_FUNCTION)>>4;
   dataport |= _BV(LCD_DATA0_PIN);  // _BV(LCD_FUNCTION_8BIT)>>4;
   pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);

   lcd_e_toggle();
   delay(4992);         /* delay, busy flag can't be checked here */

   /* repeat last command */
   lcd_e_toggle();
   delay(64);           /* delay, busy flag can't be checked here */

   /* repeat last command a third time */
   lcd_e_toggle();
   delay(64);           /* delay, busy flag can't be checked here */

   /* now configure for 4bit mode */
   dataport &= ~_BV(LCD_DATA0_PIN);
   pcf8574_setoutput(LCD_PCF8574_DEVICEID, dataport);
   lcd_e_toggle();
   delay(64);           /* some displays need this additional delay */

   /* from now the LCD only accepts 4 bit I/O, we can use lcd_command() */

   lcd_command(LCD_FUNCTION_DEFAULT);     /* function set: display lines  */

   lcd_command(LCD_DISP_OFF);             /* display off                  */
   lcd_clrscr();                          /* display clear                */
```

```
    lcd_command(LCD_MODE_DEFAULT);      /* set entry mode              */
    lcd_command(dispAttr);              /* display/cursor control      */

}/* lcd_init */
```

----------pcf8574.h----------

```
/*
pcf8574 lib 0x02

copyright (c) Davide Gironi, 2012

Released under GPLv3.
Please refer to LICENSE file for licensing information.
*/
#ifndef PCF8574_H_
#define PCF8574_H_

#define PCF8574_ADDRBASE (0x20) //device base address

#define PCF8574_I2CINIT 1 //init i2c

#define PCF8574_MAXDEVICES 8 //max devices, depends on address (3 bit)
#define PCF8574_MAXPINS 8 //max pin per device

//settings
#define PCF8574_I2CFLEURYPATH "../i2chw/i2cmaster.h" //define the path to i2c fleury lib

//pin status
volatile uint8_t pcf8574_pinstatus[PCF8574_MAXDEVICES];
//functions
void pcf8574_init();
extern int8_t pcf8574_getoutput(uint8_t deviceid);
extern int8_t pcf8574_getoutputpin(uint8_t deviceid, uint8_t pin);
extern int8_t pcf8574_setoutput(uint8_t deviceid, uint8_t data);
extern int8_t pcf8574_setoutputpins(uint8_t deviceid, uint8_t pinstart, uint8_t pinlength, int8_t data);
extern int8_t pcf8574_setoutputpin(uint8_t deviceid, uint8_t pin, uint8_t data);
extern int8_t pcf8574_setoutputpinhigh(uint8_t deviceid, uint8_t pin);
extern int8_t pcf8574_setoutputpinlow(uint8_t deviceid, uint8_t pin);
extern int8_t pcf8574_getinput(uint8_t deviceid);
extern int8_t pcf8574_getinputpin(uint8_t deviceid, uint8_t pin);
#endif
```

----------pcf8574.c----------

```
/*
pcf8574 lib 0x02

copyright (c) Davide Gironi, 2012

Released under GPLv3.
Please refer to LICENSE file for licensing information.
*/
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include "pcf8574.h"
#include "i2cmaster.h"

//path to i2c fleury lib
```

```
//#include PCF8574_I2CFLEURYPATH

/*
 * initialize
 */
void pcf8574_init() {
	#if PCF8574_I2CINIT == 1
	//init i2c
	i2c_init();
	_delay_us(10);
	#endif

	//reset the pin status
	uint8_t i = 0;
	for(i=0; i<PCF8574_MAXDEVICES; i++)
		pcf8574_pinstatus[i] = 0;

}

/*
 * get output status
 */
int8_t pcf8574_getoutput(uint8_t deviceid) {
	int8_t data = -1;
	if((deviceid >= 0 && deviceid < PCF8574_MAXDEVICES)) {
		data = pcf8574_pinstatus[deviceid];
	}
	return data;
}

/*
 * get output pin status
 */
int8_t pcf8574_getoutputpin(uint8_t deviceid, uint8_t pin) {
	int8_t data = -1;
	if((deviceid >= 0 && deviceid < PCF8574_MAXDEVICES) && (pin >= 0 && pin < PCF8574_MAXPINS))
{
		data = pcf8574_pinstatus[deviceid];
		data = (data >> pin) & 0b00000001;
	}
	return data;
}

/*
 * set output pins
 */
int8_t pcf8574_setoutput(uint8_t deviceid, uint8_t data) {
	if((deviceid >= 0 && deviceid < PCF8574_MAXDEVICES)) {
		pcf8574_pinstatus[deviceid] = data;
		i2c_start(((PCF8574_ADDRBASE+deviceid)<<1) | I2C_WRITE);
		i2c_write(data);
		i2c_stop();
		return 0;
	}
	return -1;
}
```

```
/*
 * set output pins, replace actual status of a device from pinstart for pinlength with data
 */
int8_t pcf8574_setoutputpins(uint8_t deviceid, uint8_t pinstart, uint8_t pinlength, int8_t data) {
	//example:
	//actual data is	0b01101110
	//want to change		---
	//pinstart		4
	//data			101   (pinlength 3)
	//result		0b01110110
	if((deviceid >= 0 && deviceid < PCF8574_MAXDEVICES) && (pinstart - pinlength + 1 >= 0 && pinstart - pinlength + 1 >= 0 && pinstart < PCF8574_MAXPINS && pinstart > 0 && pinlength > 0)) {
		uint8_t b = 0;
		b = pcf8574_pinstatus[deviceid];
		uint8_t mask = ((1 << pinlength) - 1) << (pinstart - pinlength + 1);
			data <<= (pinstart - pinlength + 1);
			data &= mask;
			b &= ~(mask);
			b |= data;
		pcf8574_pinstatus[deviceid] = b;
		//update device
			i2c_start(((PCF8574_ADDRBASE+deviceid)<<1) | I2C_WRITE);
			i2c_write(b);
			i2c_stop();
			return 0;
	}
	return -1;
}

/*
 * set output pin
 */
int8_t pcf8574_setoutputpin(uint8_t deviceid, uint8_t pin, uint8_t data) {
	if((deviceid >= 0 && deviceid < PCF8574_MAXDEVICES) && (pin >= 0 && pin < PCF8574_MAXPINS))
{
		uint8_t b = 0;
		b = pcf8574_pinstatus[deviceid];
		b = (data != 0) ? (b | (1 << pin)) : (b & ~(1 << pin));
		pcf8574_pinstatus[deviceid] = b;
		//update device
			i2c_start(((PCF8574_ADDRBASE+deviceid)<<1) | I2C_WRITE);
			i2c_write(b);
			i2c_stop();
			return 0;
	}
	return -1;
}

/*
 * set output pin high
 */
int8_t pcf8574_setoutputpinhigh(uint8_t deviceid, uint8_t pin) {
	return pcf8574_setoutputpin(deviceid, pin, 1);
}

/*
 * set output pin low
```

```
 */
int8_t pcf8574_setoutputpinlow(uint8_t deviceid, uint8_t pin) {
	return pcf8574_setoutputpin(deviceid, pin, 0);
}


/*
 * get input data
 */
int8_t pcf8574_getinput(uint8_t deviceid) {
	int8_t data = -1;
	if((deviceid >= 0 && deviceid < PCF8574_MAXDEVICES)) {
		i2c_start(((PCF8574_ADDRBASE+deviceid)<<1) | I2C_READ);
		data = ~i2c_readNak();
		i2c_stop();
	}
	return data;
}

/*
 * get input pin (up or low)
 */
int8_t pcf8574_getinputpin(uint8_t deviceid, uint8_t pin) {
	int8_t data = -1;
	if((deviceid >= 0 && deviceid < PCF8574_MAXDEVICES) && (pin >= 0 && pin < PCF8574_MAXPINS))
{
		data = pcf8574_getinput(deviceid);
		if(data != -1) {
			data = (data >> pin) & 0b00000001;
		}
	}
	return data;
}
```

----------ir_nec_command.h----------

```
#ifndef _IR_NEC_COMMANDS_H_
#define _IR_NEC_COMMANDS_H_

#define COMMAND_CH_MINUS		0x00ffa25d
#define COMMAND_CH			0x00ff629d
#define COMMAND_CH_PLUS			0x00ffe21d
#define COMMAND_PREV			0x00ff22dd
#define COMMAND_NEXT			0x00ff02fd
#define COMMAND_PLAY_PAUSE		0x00ffc23d
#define COMMAND_VOL_MINUS		0x00ffe01f
#define COMMAND_VOL_PLUS		0x00ffa857
#define COMMAND_EQ			0x00ff906f
#define COMMAND_0			0x00ff6897
#define COMMAND_100_PLUS		0x00ff9867
#define COMMAND_200_PLUS		0x00ffb04f
#define COMMAND_1			0x00ff30cf
#define COMMAND_2			0x00ff18e7
#define COMMAND_3			0x00ff7a85
#define COMMAND_4			0x00ff10ef
#define COMMAND_5			0x00ff38c7
#define COMMAND_6			0x00ff5aa5
#define COMMAND_7			0x00ff42bd
#define COMMAND_8			0x00ff4ab5
```

```
#define COMMAND_9                       0x00ff52ad

#endif /* _IR_NEC_COMMANDS_H_ */
```

----------ir_remote_nec.h----------

```
#ifndef __IR_REMOTE_NEC_H_
#define __IR_REMOTE_NEC_H_

#include <avr/io.h>

#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#define IR_PIN PD3

/* Last edge direction */
#define EDGE_FALLING 0
#define EDGE_RISING 1

/* Receiving signal states */
#define STATE_WAIT              0       /* When the receiver is waiting for the signal from remote controller */
#define STATE_START             1       /* When start bit is received */
#define STATE_BITS              2       /* When command bits are received (32 bits: 8 - address, 8 - ~address, 8 - command, 8 - ~command) */
#define STATE_STOP              3       /* When the stop bit is received */


/* Event duration times in microseconds */
#define SAMPLE_RATIO                                    10  /* Sampling rate */
#define TIME_TOLLERANCE                                 250 /* Tolerance */

#define TIME_COMMAND_HALFBIT_HIGH_ZERO                  560 /* High halfbit of the "0" */
#define TIME_COMMAND_HALFBIT_HIGH_ZERO_MIN
        ((TIME_COMMAND_HALFBIT_HIGH_ZERO - TIME_TOLLERANCE) / SAMPLE_RATIO)
#define TIME_COMMAND_HALFBIT_HIGH_ZERO_MAX
        ((TIME_COMMAND_HALFBIT_HIGH_ZERO + TIME_TOLLERANCE) / SAMPLE_RATIO)

#define TIME_COMMAND_HALFBIT_HIGH_ONE                   1690 /* High halfbit of the "1" */
#define TIME_COMMAND_HALFBIT_HIGH_ONE_MIN
        ((TIME_COMMAND_HALFBIT_HIGH_ONE - TIME_TOLLERANCE) / SAMPLE_RATIO)
#define TIME_COMMAND_HALFBIT_HIGH_ONE_MAX
        ((TIME_COMMAND_HALFBIT_HIGH_ONE + TIME_TOLLERANCE) / SAMPLE_RATIO)

#define TIME_START_HALFBIT_HIGH_COMMAND                 4500 /* High halfbit of the start bit, when the frame contains a command (not repeated) */
#define TIME_START_HALFBIT_HIGH_COMMAND_MIN
        ((TIME_START_HALFBIT_HIGH_COMMAND - TIME_TOLLERANCE) / SAMPLE_RATIO)
#define TIME_START_HALFBIT_HIGH_COMMAND_MAX
        ((TIME_START_HALFBIT_HIGH_COMMAND + TIME_TOLLERANCE) / SAMPLE_RATIO)
```

```
#define TIME_START_HALFBIT_HIGH_REPEAT 2250 /* High halfbit of the
start bit, when the frame contains repeat of a command */
#define TIME_START_HALFBIT_HIGH_REPEAT_MIN
        ((TIME_START_HALFBIT_HIGH_REPEAT - TIME_TOLLERANCE) / SAMPLE_RATIO)
#define TIME_START_HALFBIT_HIGH_REPEAT_MAX
        ((TIME_START_HALFBIT_HIGH_REPEAT + TIME_TOLLERANCE) / SAMPLE_RATIO)

#define TIMEOUT (110000 /
SAMPLE_RATIO) /* Timeout */

volatile uint32_t command;
volatile uint32_t current_command;
volatile uint8_t edge;
volatile uint8_t state;
volatile uint32_t cycles_counter;
volatile uint32_t total_cycles_counter;
volatile uint8_t repeat;
volatile uint8_t command_bits_counter;
volatile unsigned char newCommandFlag;

/* Rising edge detection */
void inline check_rising_edge() {
        EICRA |= (1 << ISC31);
        EICRA |= (1 << ISC30);
        edge = EDGE_RISING;
}
/* Falling edge detection */
void inline check_falling_edge() {
        EICRA |= (1 << ISC31);
        EICRA &= ~(1 << ISC30);
        edge = EDGE_FALLING;
}

void inline toggle_edge() {
        if (edge == EDGE_FALLING) {
                check_rising_edge();
                } else if (edge == EDGE_RISING) {
                check_falling_edge();
        }
}

uint32_t inline get_current_command() {
        if(1==newCommandFlag){
                newCommandFlag=0;
                return current_command;
        }
        else{
                return 0;
        }
}
void ir_init();
uint8_t ir_check_command(uint32_t command);
void ir_reset();
void ir_check_timeout();

void ir_handle_state_wait_edge_falling();
void ir_handle_state_wait_edge_rising();
```

```
void ir_handle_state_start_edge_falling();
void ir_handle_state_start_edge_rising();
void ir_handle_state_bits_edge_falling();
void ir_handle_state_bits_edge_rising();
void ir_handle_state_stop();

#endif /* NEC_H_ */
```

----------ir__remote_nec.c----------

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define  F_CPU 16000000UL
#include "ir_remote_nec.h"


void ir_init() {
	TCCR1B |= (1 << WGM12);	//CTC mode
	TCCR1B |= (1 << CS10);			/* prescaler = 1 */
	TIMSK1 |= 1<<OCIE1A;		// bit 1 OCIE1A	Use 'Output Compare A Match' Interrupt, i.e. generate an interrupt
	OCR1A = ((F_CPU/1000000) * SAMPLE_RATIO)-1;; // Output Compare Registers (16 bit) OCR1BH and OCR1BL
   TCNT1 = 0;// initialize counter

   DDRD &= ~(1 << IR_PIN);	//set IR_PIN to input
   EICRA |= (1 << ISC31);			/* Falling edge detection */
   EIMSK |= (1 << INT3);			/* External INT3 enable */
	sei();					// Enable interrupts at global level, set Global Interrupt Enable (I) bit

	ir_reset();
	total_cycles_counter = 0;
}
void ir_reset() {
	state = STATE_WAIT;
	repeat = 0;
	cycles_counter = 0;
}

uint8_t ir_check_command(uint32_t command) {
	uint8_t address_high = (command >> 24);
	uint8_t address_low = (command >> 16);
	return (address_high & address_low) == 0;
}

void ir_check_timeout() {
	if (total_cycles_counter < TIMEOUT)
	return;

	ir_reset();
	total_cycles_counter = 0;
	check_falling_edge();
}
ISR(TIMER1_COMPA_vect) {
	cycles_counter++;
	total_cycles_counter++;
}
ISR(INT3_vect) {
	ir_check_timeout();
```

```
	switch (state) {
		case STATE_WAIT:
		if (edge == EDGE_FALLING)
		ir_handle_state_wait_edge_falling();
		else
		ir_handle_state_wait_edge_rising();
		break;

		case STATE_START:
		if (edge == EDGE_RISING)
		ir_handle_state_start_edge_rising();
		else if (edge == EDGE_FALLING)
		ir_handle_state_start_edge_falling();
		break;

		case STATE_BITS:
		if (edge == EDGE_RISING)
		ir_handle_state_bits_edge_rising();
		else if (edge == EDGE_FALLING)
		ir_handle_state_bits_edge_falling();
		break;

		case STATE_STOP:
		ir_handle_state_stop();
		break;

		default:
		ir_reset();
		break;
	}

	toggle_edge();
}

void ir_handle_state_wait_edge_falling() {
	cycles_counter = 0;
	state = STATE_START;
}
void ir_handle_state_wait_edge_rising() {
	ir_reset();
}
void ir_handle_state_start_edge_falling() {

	if(cycles_counter >= TIME_START_HALFBIT_HIGH_COMMAND_MIN && cycles_counter <=
TIME_START_HALFBIT_HIGH_COMMAND_MAX) {
		state = STATE_BITS;
		command_bits_counter = 0;
		command = 0;
		return;
	}

	if (cycles_counter >= TIME_START_HALFBIT_HIGH_REPEAT_MIN && cycles_counter <=
TIME_START_HALFBIT_HIGH_REPEAT_MAX) {
		state = STATE_STOP;
		repeat = 1;
		command = current_command;
```

```
		return;
	}

	ir_reset();
}

void ir_handle_state_start_edge_rising() {
	cycles_counter = 0;
	state = STATE_START;
}

void ir_handle_state_bits_edge_falling() {
		if (cycles_counter >= TIME_COMMAND_HALFBIT_HIGH_ZERO_MIN && cycles_counter <=
TIME_COMMAND_HALFBIT_HIGH_ZERO_MAX) {
		command = (command << 1);
		command_bits_counter++;

		if (command_bits_counter < 32) {
			state = STATE_BITS;
			} else if (!ir_check_command(command)) {
			ir_reset();
			} else {
			state = STATE_STOP;
			current_command = command;
		}

		return;
	}

	if (cycles_counter >= TIME_COMMAND_HALFBIT_HIGH_ONE_MIN && cycles_counter <=
TIME_COMMAND_HALFBIT_HIGH_ONE_MAX) {
		command = ((command << 1) | 1);
		command_bits_counter++;

		if (command_bits_counter < 32) {
			state = STATE_BITS;
			} else if (!ir_check_command(command)) {
			ir_reset();
			} else {
			state = STATE_STOP;
			current_command = command;
			newCommandFlag=1;
		}

		return;
	}


	command_bits_counter = 0;
	command = 0;
	ir_reset();
}

void ir_handle_state_bits_edge_rising() {
	cycles_counter = 0;
	state = STATE_BITS;
}
```

```
void ir_handle_state_stop() {
	cycles_counter = 0;
	state = STATE_WAIT;
	current_command = command;
	total_cycles_counter = 0;
}
```