CE232 - Advanced Embedded System Design

March 18

2016

Implementation of LED flasher and Alarm System

Michael Bridden

Contents

LED

The task was to use the 8 LED's onboard and alternate between the top and bottom 4 at a desired frequency. This frequency should be variable, it should be altered using the RS232 port to receive a keystroke to determine if the frequency should go up or down.

Timing

For a low speed clock we will use the low reference clock (32.768KHz) also known as a watch crystal if we then use a 2048 clock divider and a counter load value of 3 (+1) we will get a frequency of:

$$f = \frac{\left(\frac{32768}{2048}\right)}{4} = 4Hz$$

This is ideal for the alarm project as it produces a 250ms interrupt (buzzer). From this we can find out how many seconds are required to complete a full count with varying values and thus the frequency for each.

when val = 16 (4 second period)

250mHz 50% duty cycle

when val = 8 (2 second period)

500mHz 50% duty cycle

when val = 4 (1 second period)

1Hz 50% duty cycle

when val = 2 (500ms period)

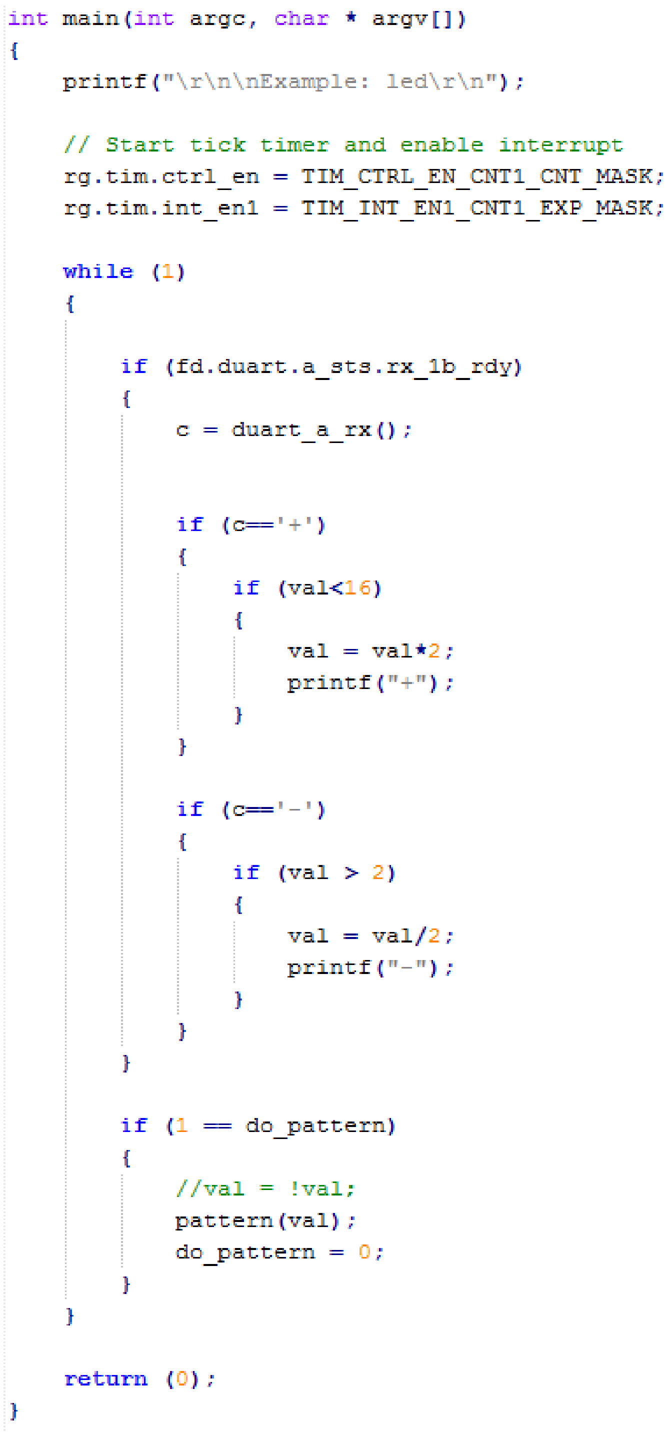
2Hz 50% duty cycle

UART

The eCog1k uses a UART port specifically a rs232 serial port this is setup to transmit/receive at the user defined speed of 9600baud which is sufficiently overkill for this task.

If we check the receive status for the uart using fd.uart.a_sts.rx_1b_rdy when true we are able to use the duart_a_rx() function enabling us to read a key press from a workstation using software such as HyperTerminal from this the system is able to check which character was pressed and perform the relevant action using simple if statements.

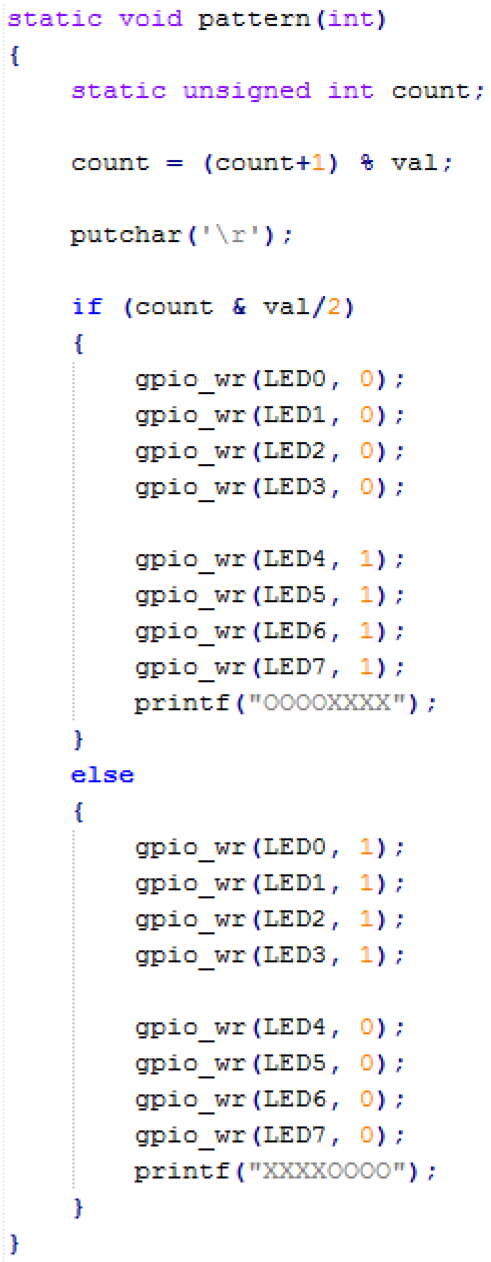
main()



```
int main(int argc, char * argv[])
{
    printf("\r\n\nExample: led\r\n");

    // Start tick timer and enable interrupt
    rg.tim.ctrl_en = TIM_CTRL_EN_CNT1_CNT_MASK;
    rg.tim.int_en1 = TIM_INT_EN1_CNT1_EXP_MASK;

    while (1)
    {

        if (fd.duart.a_sts.rx_1b_rdy)
        {
            c = duart_a_rx();


            if (c=='+')
            {
                if (val<16)
                {
                    val = val*2;
                    printf("+");
                }
            }

            if (c=='-')
            {
                if (val > 2)
                {
                    val = val/2;
                    printf("-");
                }
            }
        }

        if (1 == do_pattern)
        {
            //val = !val;
            pattern(val);
            do_pattern = 0;
        }
    }

    return (0);
}
```

The main function checks for a key press over the uart port if a '+' char is received it multiplies the val variable by 2 and if a '-' char is received it divides the val variable by 2. The do_pattern variable is set to true when the interrupt is called, the pattern function then passes the variable val which can take the 4 possible values 16, 8, 4, 2 or 250mHz, 500mHz, 1Hz, 2Hz respectively.

pattern()



```
static void pattern(int)
{
    static unsigned int count;

    count = (count+1) % val;

    putchar('\r');

    if (count & val/2)
    {
        gpio_wr(LED0, 0);
        gpio_wr(LED1, 0);
        gpio_wr(LED2, 0);
        gpio_wr(LED3, 0);

        gpio_wr(LED4, 1);
        gpio_wr(LED5, 1);
        gpio_wr(LED6, 1);
        gpio_wr(LED7, 1);
        printf("OOOOXXXX");
    }
    else
    {
        gpio_wr(LED0, 1);
        gpio_wr(LED1, 1);
        gpio_wr(LED2, 1);
        gpio_wr(LED3, 1);

        gpio_wr(LED4, 0);
        gpio_wr(LED5, 0);
        gpio_wr(LED6, 0);
        gpio_wr(LED7, 0);
        printf("XXXXOOOO");
    }
}
```

The variable count will increment when the pattern function is called and will "overflow" when it reaches a multiple of the val variable. Initially the first four led's are ON and once the count reaches half that of the val variable the last four led's will turn ON until the count "overflows" when it reaches val and then the cycle repeats with a 50% duty cycle.

Alarm System

The task was to implement the alarm system to the specification/requirements set out in the UML design task.

Requirements

Alarm

Alarm ID	Requirement	State
1	Entry of alarm code followed by '*' transition to exit state	Unset
2	Incorrect code will cause internal buzzer to sound for 500ms	
3	Incorrect code 3 times transition to alarm state	
4	Remains in the exit state until the exit period has expired after which transition to set state (no sensors activated)	Exit
5	Activation of sensors in exit period excluding entry/exit transition to alarm state	
6	Internal buzzer sound 500ms 50% duty cycle in exit period	
7	Entry of alarm code followed by '*' transition to unset state	
8	Incorrect code 4 times transition to alarm state	
9	Activation of sensors excluding entry/exit transition to alarm state	Set
10	Activation of entry/exit sensors transition to entry state	
11	Remains in the entry state until the entry period has expired after which transition to alarm state	Entry
12	Activation of sensors in entry period excluding entry/exit transition to alarm state	
13	Internal buzzer sound 500ms 50% duty cycle in entry period	
14	Entry of alarm code followed by '*' transition to unset state	
15	Entry of alarm code followed by '*' transition to unset state	Alarm
16	Internal buzzer sound	
17	External sounder enabled for 5 minutes	

LCD

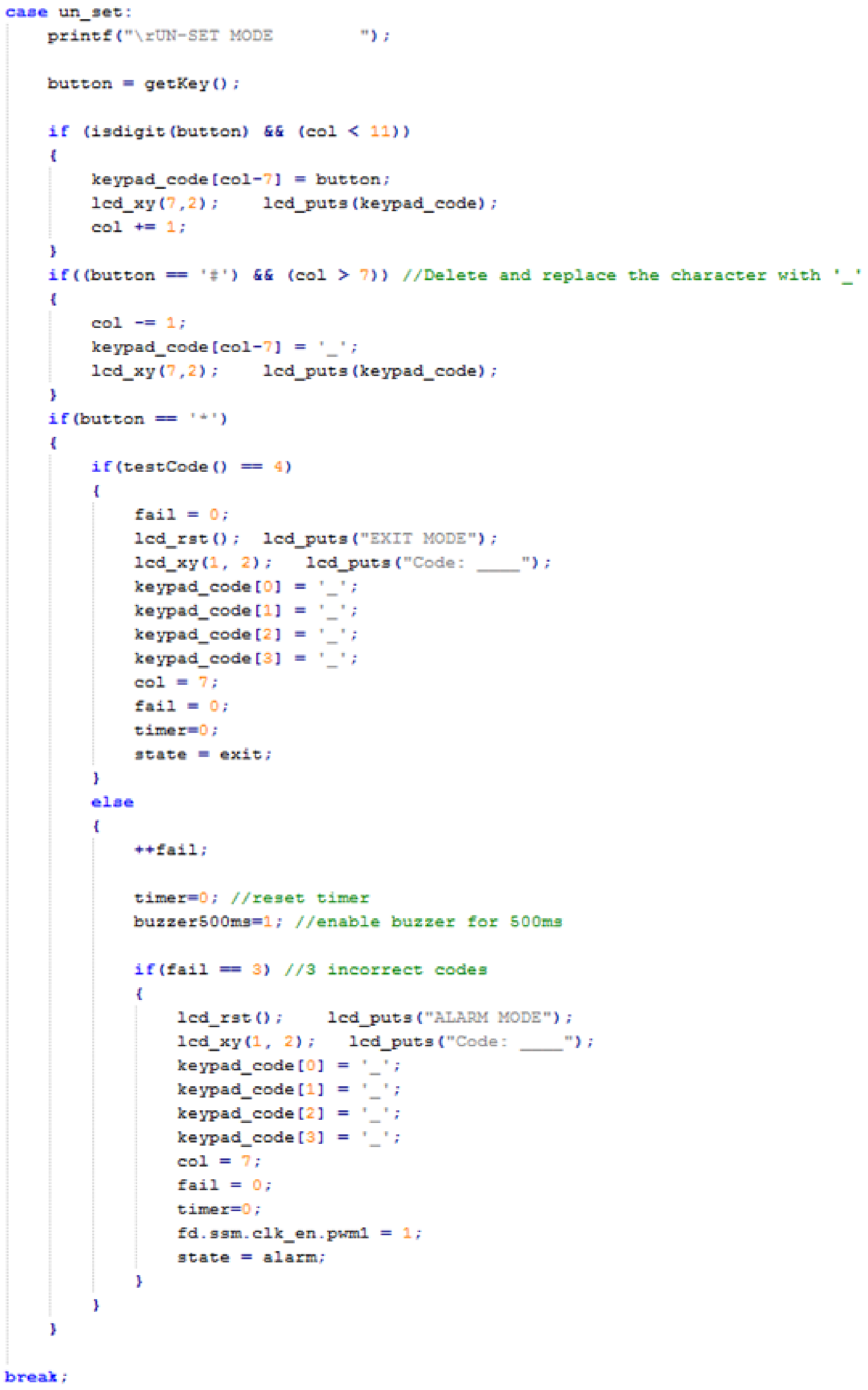
LCD ID	Requirement	State
1	The display should show the current state left aligned on the first line and "Code ____" left aligned on the second line	Unset, Exit, Entry, Alarm
2	The '#' key can be used to delete the previous character press	Unset, Exit, Entry, Alarm
3	When four digits have been entered the display should display "Press * to set"	Unset, Exit, Entry, Alarm

There was some confusion as to what was specifically required to be implemented so above is my personal understanding of the what requirements where to be met for this alarm system.

The 5 states Unset, Exit, Set, Entry and Alarm are described below in sudo code along with the actual code implemented.

States

Unset



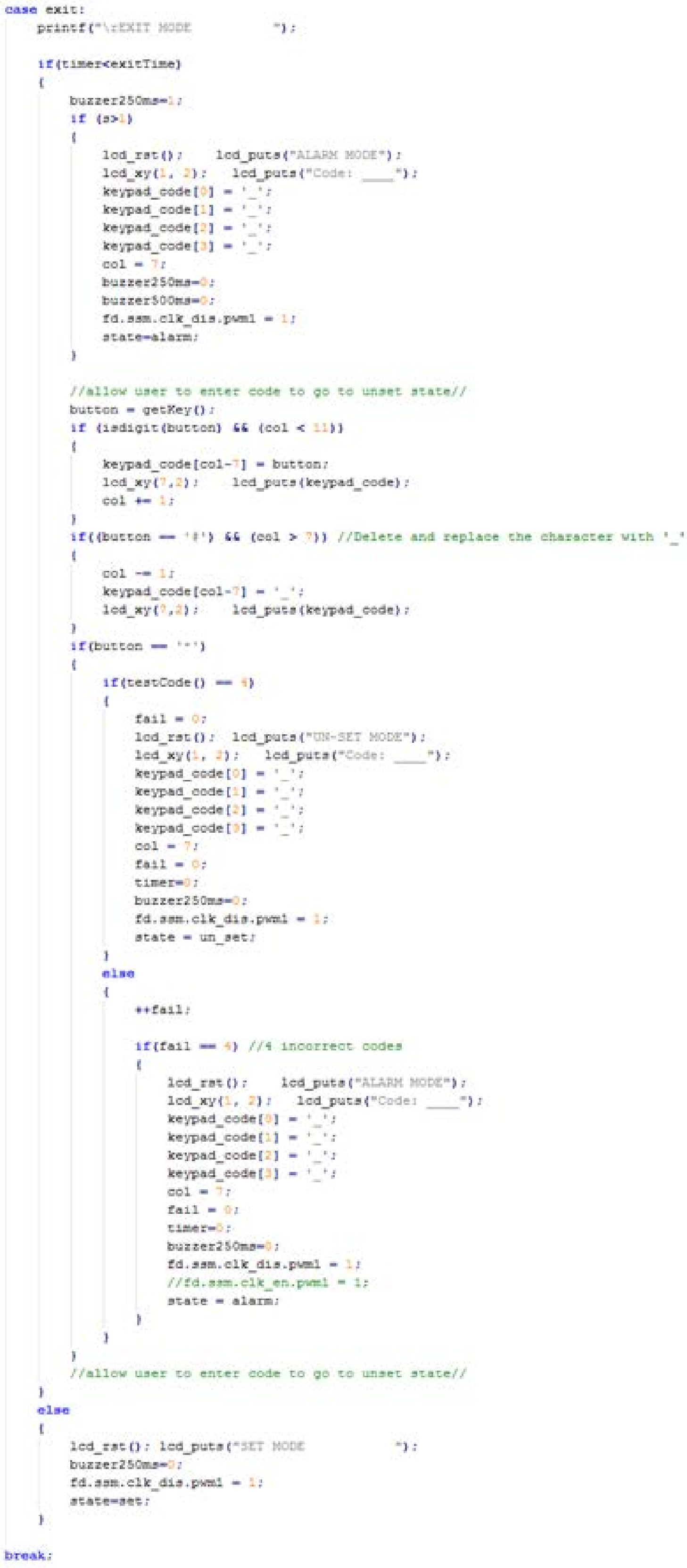
```
case un_set:
    printf("\rUN-SET MODE           ");

    button = getKey();

    if (isdigit(button) && (col < 11))
    {
        keypad_code[col-7] = button;
        lcd_xy(7,2);    lcd_puts(keypad_code);
        col += 1;
    }
    if((button == '#') && (col > 7)) //Delete and replace the character with '_'
    {
        col -= 1;
        keypad_code[col-7] = '_';
        lcd_xy(7,2);    lcd_puts(keypad_code);
    }
    if(button == '*')
    {
        if(testCode() == 4)
        {
            fail = 0;
            lcd_rst();  lcd_puts("EXIT MODE");
            lcd_xy(1, 2);   lcd_puts("Code: ____");
            keypad_code[0] = '_';
            keypad_code[1] = '_';
            keypad_code[2] = '_';
            keypad_code[3] = '_';
            col = 7;
            fail = 0;
            timer=0;
            state = exit;
        }
        else
        {
            ++fail;

            timer=0; //reset timer
            buzzer500ms=1; //enable buzzer for 500ms

            if(fail == 3) //3 incorrect codes
            {
                lcd_rst();    lcd_puts("ALARM MODE");
                lcd_xy(1, 2);   lcd_puts("Code: ____");
                keypad_code[0] = '_';
                keypad_code[1] = '_';
                keypad_code[2] = '_';
                keypad_code[3] = '_';
                col = 7;
                fail = 0;
                timer=0;
                fd.ssm.clk_en.pwm1 = 1;
                state = alarm;
            }
        }
    }

break;
```

The unset state checks for key presses using the getKey function each time a key is pressed; the '*' key calls the testCode function which returns 4 if correct or less than 4 if incorrect; the '#' key deletes the previous digit. The LCD updates after each key press to show the new keypad code.

When the code entered is checked if it returns 4 (all digits match) then the alarm updates the LCD for the next state; clears all relevant variables and changes state. Else when an incorrect code is entered the alarm buzzer will enable for 500ms and the fail counter is incremented when this reaches 3 the system will change to the alarm state.

Exit

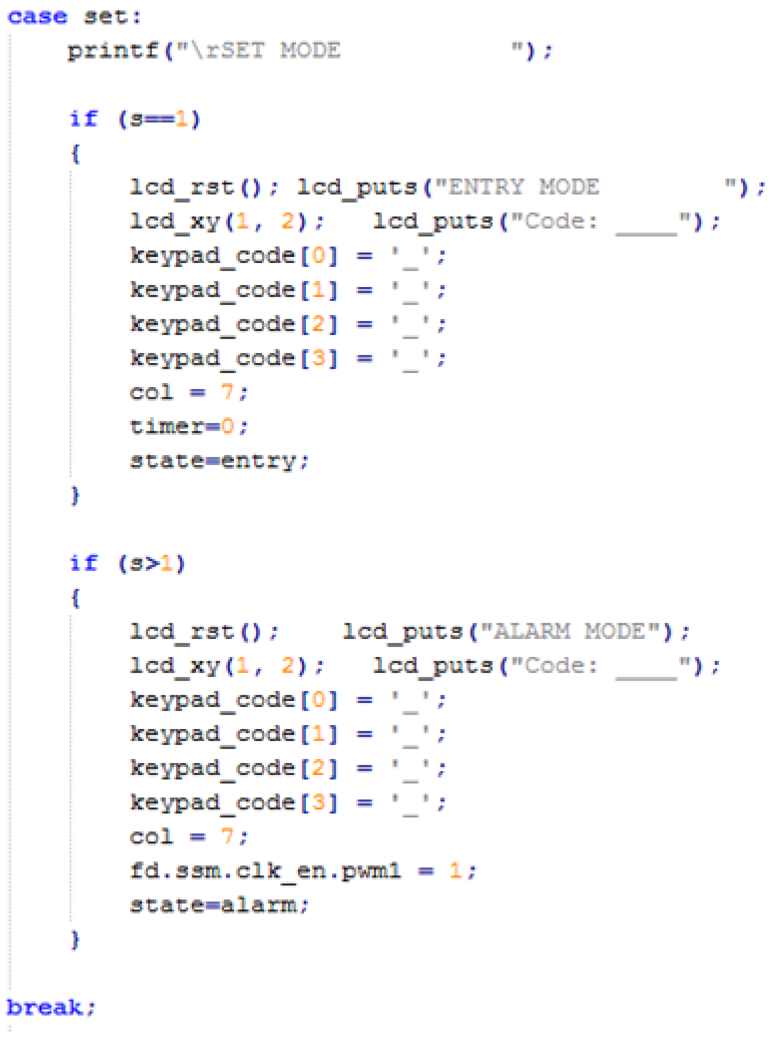
The exit state has a countdown period in which it checks if any sensors are triggered, if so it updates



```
case exit:
    printf("\rEXIT MODE          ");

    if(timer<exitTime)
    {
        buzzer250ms=1;
        if (s>1)
        {
            lcd_rst();    lcd_puts("ALARM MODE");
            lcd_xy(1, 2);   lcd_puts("Code: ____");
            keypad_code[0] = '_';
            keypad_code[1] = '_';
            keypad_code[2] = '_';
            keypad_code[3] = '_';
            col = 7;
            buzzer250ms=0;
            buzzer500ms=0;
            fd.ssm.clk_dis.pwm1 = 1;
            state=alarm;
        }

        //allow user to enter code to go to unset state//
        button = getKey();
        if (isdigit(button) && (col < 11))
        {
            keypad_code[col-7] = button;
            lcd_xy(7,2);    lcd_puts(keypad_code);
            col += 1;
        }
        if((button == '#') && (col > 7)) //Delete and replace the character with '_'
        {
            col -= 1;
            keypad_code[col-7] = '_';
            lcd_xy(7,2);    lcd_puts(keypad_code);
        }
        if(button == '*')
        {
            if(testCode() == 4)
            {
                fail = 0;
                lcd_rst(); lcd_puts("UN-SET MODE");
                lcd_xy(1, 2);   lcd_puts("Code: ____");
                keypad_code[0] = '_';
                keypad_code[1] = '_';
                keypad_code[2] = '_';
                keypad_code[3] = '_';
                col = 7;
                fail = 0;
                timer=0;
                buzzer250ms=0;
                fd.ssm.clk_dis.pwm1 = 1;
                state = un_set;
            }
            else
            {
                ++fail;

                if(fail == 4) //4 incorrect codes
                {
                    lcd_rst();    lcd_puts("ALARM MODE");
                    lcd_xy(1, 2);   lcd_puts("Code: ____");
                    keypad_code[0] = '_';
                    keypad_code[1] = '_';
                    keypad_code[2] = '_';
                    keypad_code[3] = '_';
                    col = 7;
                    fail = 0;
                    timer=0;
                    buzzer250ms=0;
                    fd.ssm.clk_dis.pwm1 = 1;
                    //fd.ssm.clk_en.pwm1 = 1;
                    state = alarm;
                }
            }
        }
        //allow user to enter code to go to unset state//
    }
    else
    {
        lcd_rst(); lcd_puts("SET MODE          ");
        buzzer250ms=0;
        fd.ssm.clk_dis.pwm1 = 1;
        state=set;
    }

break;
```

the LCD for the next state resets relevant variables and sets the state to alarm. Otherwise the user is allowed to enter a code to return to the un_set state within the period. The buzzer in on for 500ms with a 50% duty cycle during this period. After the countdown passes with no trigger it will set the state to set.

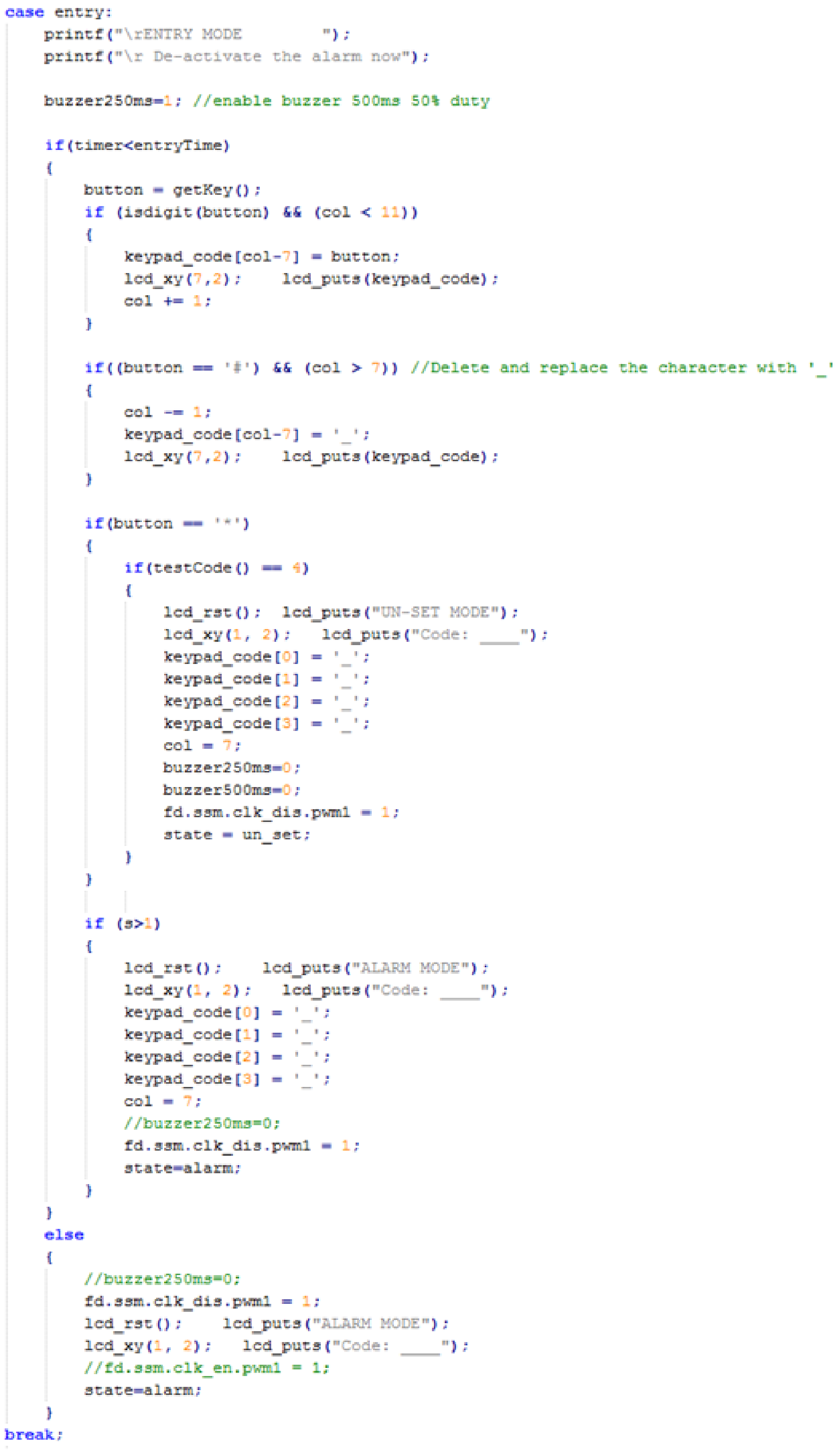
Set



```
case set:
    printf("\rSET MODE           ");

    if (s==1)
    {
        lcd_rst(); lcd_puts("ENTRY MODE        ");
        lcd_xy(1, 2);   lcd_puts("Code: ____");
        keypad_code[0] = '_';
        keypad_code[1] = '_';
        keypad_code[2] = '_';
        keypad_code[3] = '_';
        col = 7;
        timer=0;
        state=entry;
    }

    if (s>1)
    {
        lcd_rst();    lcd_puts("ALARM MODE");
        lcd_xy(1, 2);   lcd_puts("Code: ____");
        keypad_code[0] = '_';
        keypad_code[1] = '_';
        keypad_code[2] = '_';
        keypad_code[3] = '_';
        col = 7;
        fd.ssm.clk_en.pwm1 = 1;
        state=alarm;
    }

break;
```

When in the set state if a zone sensor is triggered it will give the variable s a value greater than 1, triggering the entry/exit sensor will set s equal to 1. For each case we update the LCD and display the new state data and reset all variables that need clearing.

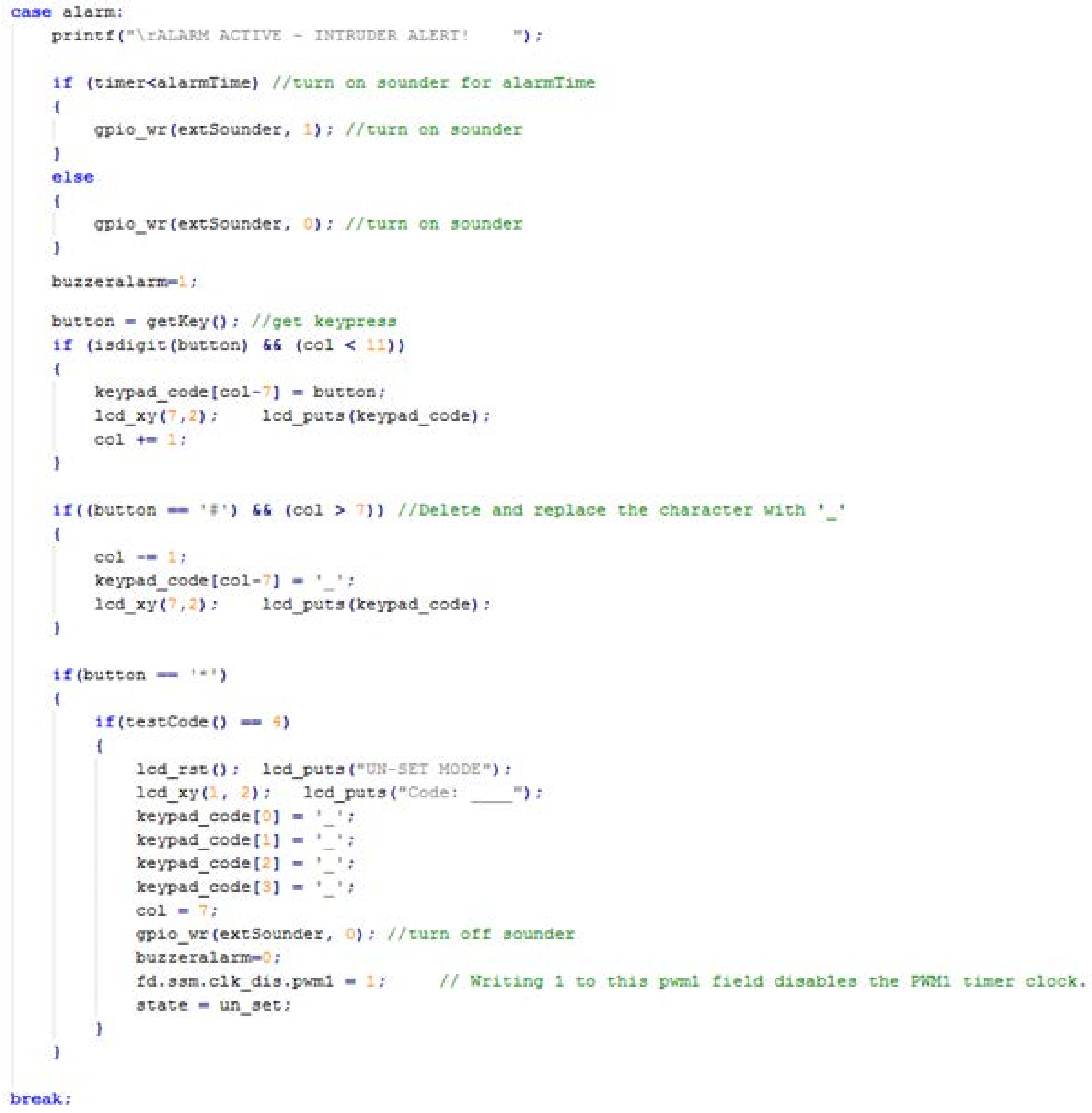
Entry



```
case entry:
    printf("\rENTRY MODE         ");
    printf("\r De-activate the alarm now");

    buzzer250ms=1; //enable buzzer 500ms 50% duty

    if(timer<entryTime)
    {
        button = getKey();
        if (isdigit(button) && (col < 11))
        {
            keypad_code[col-7] = button;
            lcd_xy(7,2);    lcd_puts(keypad_code);
            col += 1;
        }

        if((button == '#') && (col > 7)) //Delete and replace the character with '_'
        {
            col -= 1;
            keypad_code[col-7] = '_';
            lcd_xy(7,2);    lcd_puts(keypad_code);
        }

        if(button == '*')
        {
            if(testCode() == 4)
            {
                lcd_rst();  lcd_puts("UN-SET MODE");
                lcd_xy(1, 2);   lcd_puts("Code: ____");
                keypad_code[0] = '_';
                keypad_code[1] = '_';
                keypad_code[2] = '_';
                keypad_code[3] = '_';
                col = 7;
                buzzer250ms=0;
                buzzer500ms=0;
                fd.ssm.clk_dis.pwm1 = 1;
                state = un_set;
            }
        }

        if (s>1)
        {
            lcd_rst();    lcd_puts("ALARM MODE");
            lcd_xy(1, 2);   lcd_puts("Code: ____");
            keypad_code[0] = '_';
            keypad_code[1] = '_';
            keypad_code[2] = '_';
            keypad_code[3] = '_';
            col = 7;
            //buzzer250ms=0;
            fd.ssm.clk_dis.pwm1 = 1;
            state=alarm;
        }
    }
    else
    {
        //buzzer250ms=0;
        fd.ssm.clk_dis.pwm1 = 1;
        lcd_rst();    lcd_puts("ALARM MODE");
        lcd_xy(1, 2);   lcd_puts("Code: ____");
        //fd.ssm.clk_en.pwm1 = 1;
        state=alarm;
    }
break;
```

Upon entry the system will check the sensors however if the countdown is reached and the correct code isn't entered within that time then state will equal alarm. The user can enter codes until the countdown is reached and if correct it will move to the un_set state otherwise it will set the state to alarm.

Alarm



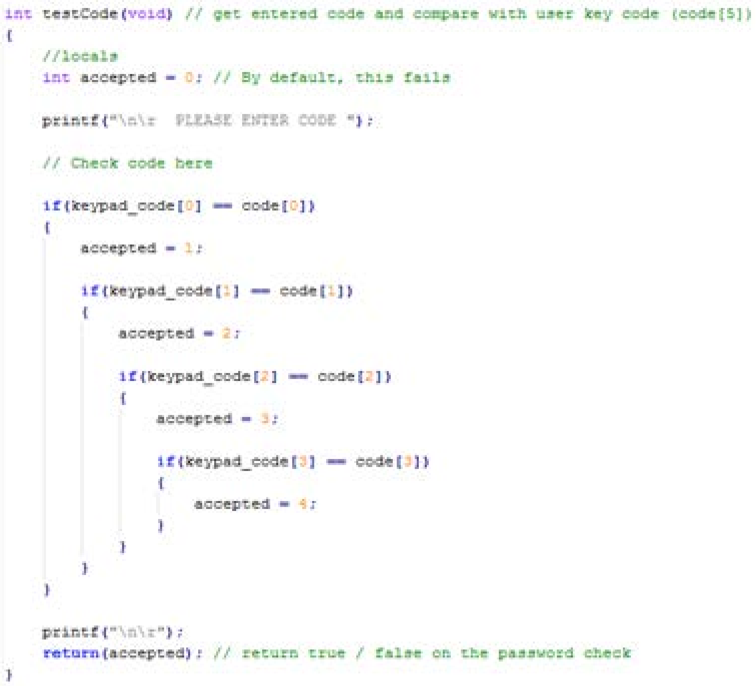
```
case alarm:
    printf("\rALARM ACTIVE - INTRUDER ALERT!     ");

    if (timer<alarmTime) //turn on sounder for alarmTime
    {
        gpio_wr(extSounder, 1); //turn on sounder
    }
    else
    {
        gpio_wr(extSounder, 0); //turn on sounder
    }
    buzzeralarm=1;

    button = getKey(); //get keypress
    if (isdigit(button) && (col < 11))
    {
        keypad_code[col-7] = button;
        lcd_xy(7,2);    lcd_puts(keypad_code);
        col += 1;
    }

    if((button == '#') && (col > 7)) //Delete and replace the character with '_'
    {
        col -= 1;
        keypad_code[col-7] = '_';
        lcd_xy(7,2);    lcd_puts(keypad_code);
    }

    if(button == '*')
    {
        if(testCode() == 4)
        {
            lcd_rst();  lcd_puts("UN-SET MODE");
            lcd_xy(1, 2);   lcd_puts("Code: ____");
            keypad_code[0] = '_';
            keypad_code[1] = '_';
            keypad_code[2] = '_';
            keypad_code[3] = '_';
            col = 7;
            gpio_wr(extSounder, 0); //turn off sounder
            buzzeralarm=0;
            fd.ssm.clk_dis.pwm1 = 1;     // Writing 1 to this pwm1 field disables the PWM1 timer clock.
            state = un_set;
        }
    }
break;
```

The alarm state has a countdown using the timer variable so that the system can turn off the sounder after 5 minutes. We set the relevant variable to turn the buzzer by enabling the pwm signal. The state constantly checks for the users key presses and is able to delete characters; check if the code is correct upon pressing '*' and updates the LCD. If the user has entered the correct code we need to reset the keypad_code; reset LCD column index; turn off the sounder/buzzer and change state to un_set.

Functions

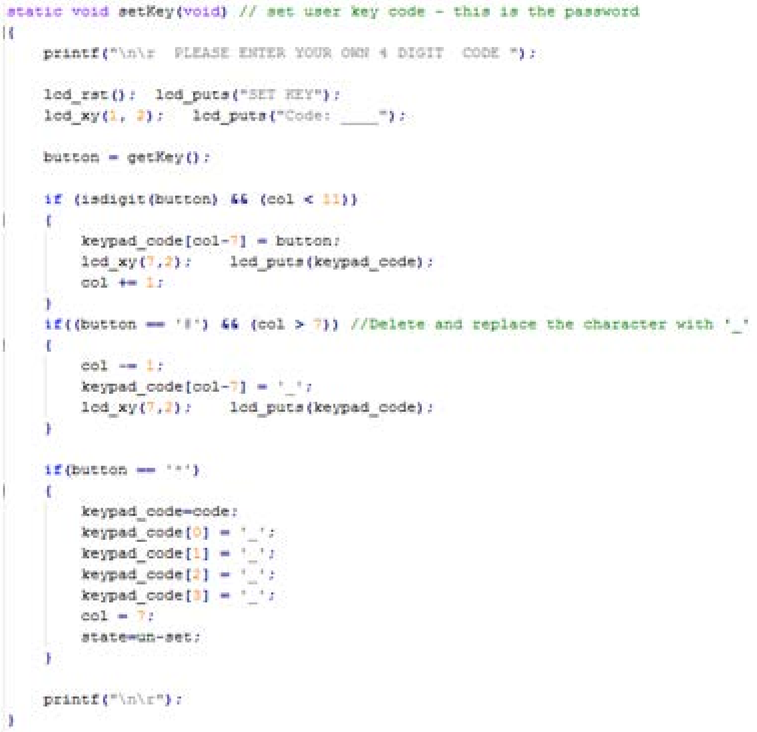
testCode()

The test code function simply checks if each character in the keypad_code[index] array matches that of the code[index] if so it will return 4 otherwise it will return less than 4.



```
int testCode(void) // get entered code and compare with user key code (code[5])
{
    //locals
    int accepted = 0; // By default, this fails

    printf("\n\r  PLEASE ENTER CODE ");

    // Check code here

    if(keypad_code[0] == code[0])
    {
        accepted = 1;

        if(keypad_code[1] == code[1])
        {
            accepted = 2;

            if(keypad_code[2] == code[2])
            {
                accepted = 3;

                if(keypad_code[3] == code[3])
                {
                    accepted = 4;
                }
            }
        }
    }

    printf("\n\r");
    return(accepted); // return true / false on the password check
}
```

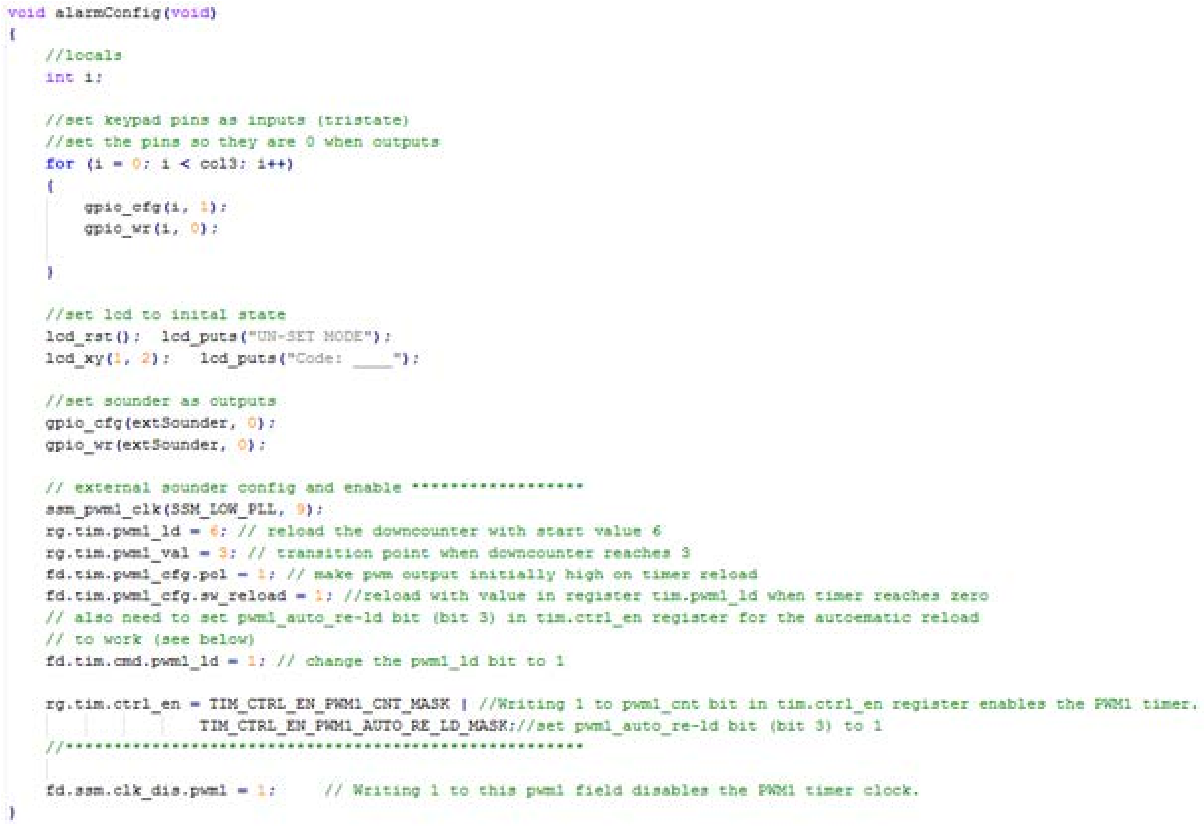
setKey()



```
static void setKey(void) // set user key code - this is the password
{
    printf("\n\r  PLEASE ENTER YOUR OWN 4 DIGIT  CODE ");

    lcd_rst();  lcd_puts("SET KEY");
    lcd_xy(1, 2);   lcd_puts("Code: ____");

    button = getKey();

    if (isdigit(button) && (col < 11))
    {
        keypad_code[col-7] = button;
        lcd_xy(7,2);    lcd_puts(keypad_code);
        col += 1;
    }
    if((button == '#') && (col > 7)) //Delete and replace the character with '_'
    {
        col -= 1;
        keypad_code[col-7] = '_';
        lcd_xy(7,2);    lcd_puts(keypad_code);
    }

    if(button == '*')
    {
        keypad_code=code;
        keypad_code[0] = '_';
        keypad_code[1] = '_';
        keypad_code[2] = '_';
        keypad_code[3] = '_';
        col = 7;
        state=un-set;
    }

    printf("\n\r");
}
```

setkey calls the getKey function using this we store the value of each button press in the keypad_code array once the user presses the '*' key we set the code to keypad_code ; clear the keypad_code array; reset the column index and change the state to un-set. This function also allows for deleting key presses using the '#' button.

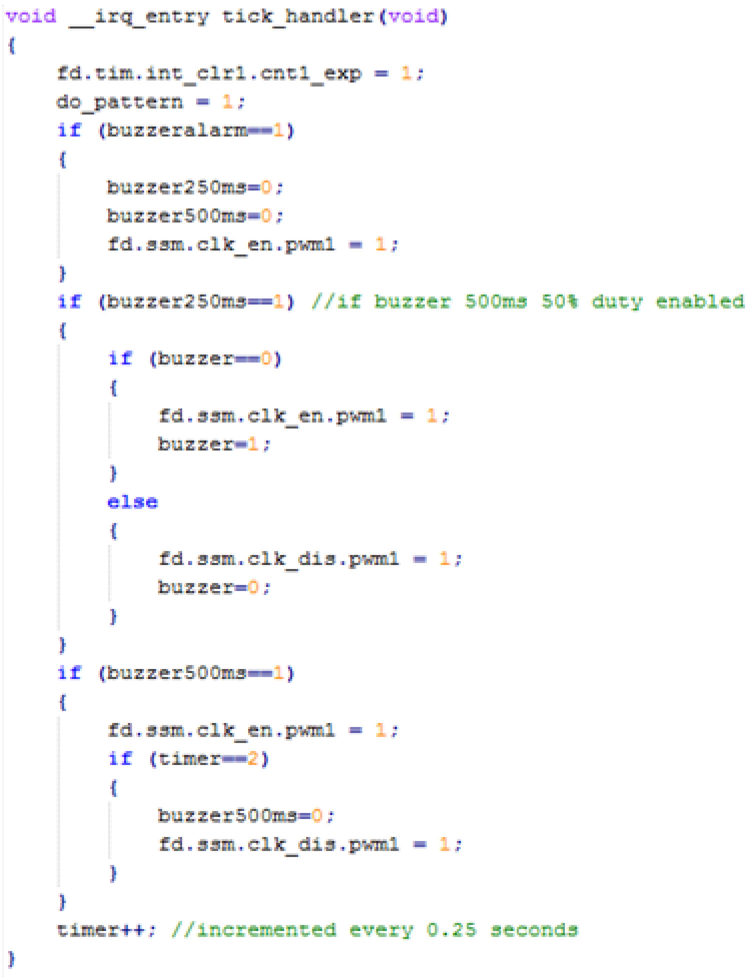
alarmConfig()



```
void alarmConfig(void)
{
    //locals
    int i;

    //set keypad pins as inputs (tristate)
    //set the pins so they are 0 when outputs
    for (i = 0; i < col3; i++)
    {
        gpio_cfg(i, 1);
        gpio_wr(i, 0);

    }

    //set lcd to inital state
    lcd_rst();  lcd_puts("UN-SET MODE");
    lcd_xy(1, 2);   lcd_puts("Code: ____");

    //set sounder as outputs
    gpio_cfg(extSounder, 0);
    gpio_wr(extSounder, 0);

    // external sounder config and enable ******************
    ssm_pwm1_clk(SSM_LOW_PLL, 9);
    rg.tim.pwm1_ld = 6; // reload the downcounter with start value 6
    rg.tim.pwm1_val = 3; // transition point when downcounter reaches 3
    fd.tim.pwm1_cfg.pol = 1; // make pwm output initially high on timer reload
    fd.tim.pwm1_cfg.sw_reload = 1; //reload with value in register tim.pwm1_ld when timer reaches zero
    // also need to set pwm1_auto_re-ld bit (bit 3) in tim.ctrl_en register for the autoematic reload
    // to work (see below)
    fd.tim.cmd.pwm1_ld = 1; // change the pwm1_ld bit to 1

    rg.tim.ctrl_en = TIM_CTRL_EN_PWM1_CNT_MASK | //Writing 1 to pwm1_cnt bit in tim.ctrl_en register enables the PWM1 timer.
                     TIM_CTRL_EN_PWM1_AUTO_RE_LD_MASK;//set pwm1_auto_re-ld bit (bit 3) to 1
    //*****************************************************

    fd.ssm.clk_dis.pwm1 = 1;     // Writing 1 to this pwm1 field disables the PWM1 timer clock.
}
```

This initialises all the inputs and outputs. The buzzer is configured using a pwm interrupt which can be enabled or disabled using fd.ssm.clk_dis.pwm1=1 or fd.ssm.clk_en.pwm1=1.

Interrupt



```
void __irq_entry tick_handler(void)
{
    fd.tim.int_clr1.cnt1_exp = 1;
    do_pattern = 1;
    if (buzzeralarm==1)
    {
        buzzer250ms=0;
        buzzer500ms=0;
        fd.ssm.clk_en.pwm1 = 1;
    }
    if (buzzer250ms==1) //if buzzer 500ms 50% duty enabled
    {
        if (buzzer==0)
        {
            fd.ssm.clk_en.pwm1 = 1;
            buzzer=1;
        }
        else
        {
            fd.ssm.clk_dis.pwm1 = 1;
            buzzer=0;
        }
    }
    if (buzzer500ms==1)
    {
        fd.ssm.clk_en.pwm1 = 1;
        if (timer==2)
        {
            buzzer500ms=0;
            fd.ssm.clk_dis.pwm1 = 1;
        }
    }
    timer++; //incremented every 0.25 seconds
}
```

The interrupt code checks which state multiple variables are in and from that puts the buzzer in to a specific mode such as turning on and off at every interrupt (500ms 50% duty); on for 2 interrupts (500ms on) or a solid buzzer until turned off. The timer variable is used for all the countdown section for the entry, exit and alarm states as it increments at a known frequency giving an accurate timing.

Summary

The implemented system met all but one of the requirements this was specifically LCD ID 3 in which the keypad_code isn't checked if its currently 4 digits and if so update the LCD to say "Press * to set" however this could be easily implemented using something similar to shown in the improvements section.

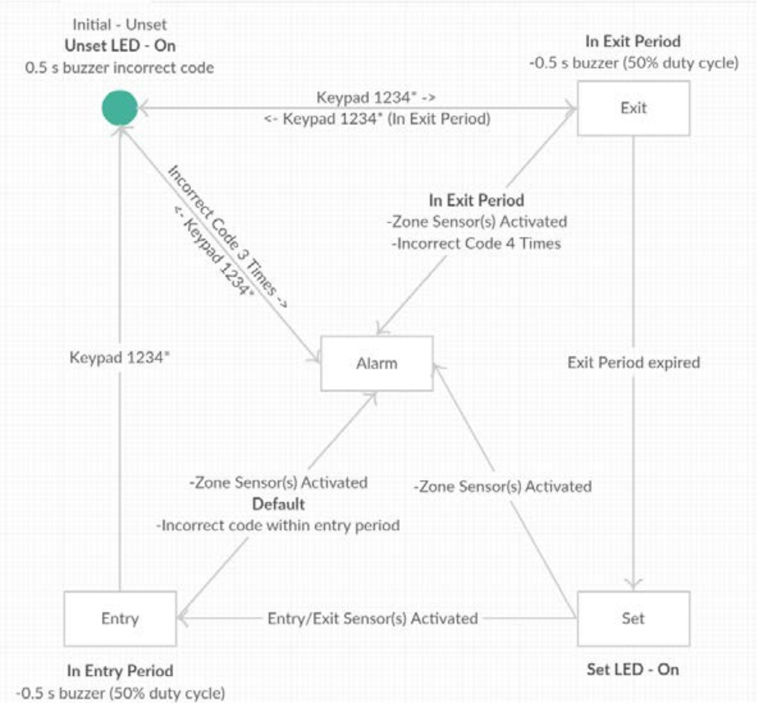
Requirements

Alarm ID	Requirement	Met
1	Entry of alarm code followed by '*' transition to exit state	Yes
2	Incorrect code will cause internal buzzer to sound for 500ms	Yes
3	Incorrect code 3 times transition to alarm state	Yes
4	Remains in the exit state until the exit period has expired after which transition to set state (no sensors activated)	Yes

5	Activation of sensors in exit period excluding entry/exit transition to alarm state	Yes
6	Internal buzzer sound 500ms 50% duty cycle in exit period	Yes
7	Entry of alarm code followed by '*' transition to unset state	Yes
8	Incorrect code 4 times transition to alarm state	Yes
9	Activation of sensors excluding entry/exit transition to alarm state	Yes
10	Activation of entry/exit sensors transition to entry state	Yes
11	Remains in the entry state until the entry period has expired after which transition to alarm state	Yes
12	Activation of sensors in entry period excluding entry/exit transition to alarm state	Yes
13	Internal buzzer sound 500ms 50% duty cycle in entry period	Yes
14	Entry of alarm code followed by '*' transition to unset state	Yes
15	Entry of alarm code followed by '*' transition to unset state	Yes
16	Internal buzzer sound	Yes
17	External sounder enabled for 5 minutes	Yes

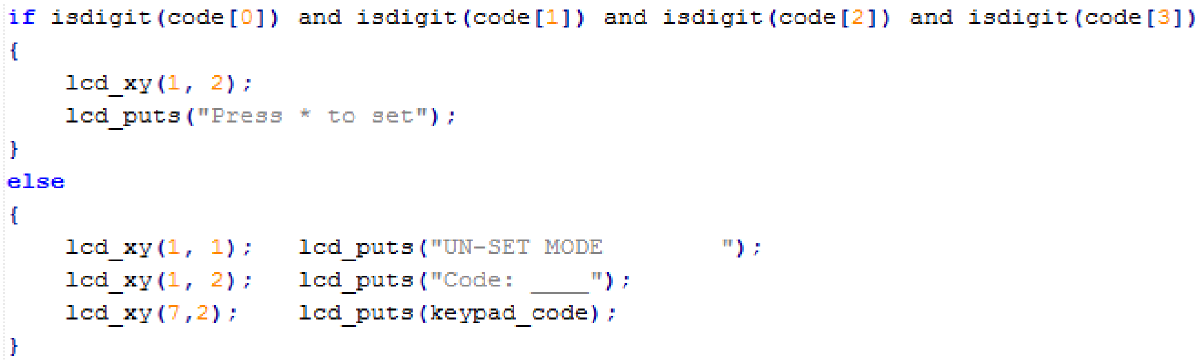
LCD ID	Requirement	State
1	The display should show the current state left aligned on the first line and "Code _____" left aligned on the second line	Yes
2	The '#' key can be used to delete the previous character press	Yes
3	When four digits have been entered the display should display "Press * to set"	No

State Machine


Initial - Unset
Unset LED - On
0.5 s buzzer incorrect code
In Exit Period
-0.5 s buzzer (50% duty cycle)
Keypad 1234* ->
<- Keypad 1234* (In Exit Period)
Exit
Incorrect Code 3 Times ->
<- Keypad 1234*
In Exit Period
-Zone Sensor(s) Activated
-Incorrect Code 4 Times
Keypad 1234*
Alarm
Exit Period expired
-Zone Sensor(s) Activated
Default
-Incorrect code within entry period
-Zone Sensor(s) Activated
Entry
Entry/Exit Sensor(s) Activated
Set
In Entry Period
-0.5 s buzzer (50% duty cycle)
Set LED - On

The final implementation included all the states previously discussed in the design stage. Each state transitions as expected whether from external stimuli or a counter triggered interrupt.

Improvements



```
if isdigit(code[0]) and isdigit(code[1]) and isdigit(code[2]) and isdigit(code[3])
{
    lcd_xy(1, 2);
    lcd_puts("Press * to set");
}
else
{
    lcd_xy(1, 1);    lcd_puts("UN-SET MODE        ");
    lcd_xy(1, 2);    lcd_puts("Code: ____");
    lcd_xy(7,2);     lcd_puts(keypad_code);
}
```

To meet ID 3

This simply checks if each character used is a digit if so it can update the LCD to inform the user to press '*'.

The getkey function and debounce could have also been improved by using the generic counter variable timer which is incremented every interrupt at a frequency of 4Hz. The clock divider would have to be lower to allow for a faster interrupt to trigger at 5ms as it currently interrupts at 250ms which can be changed in the configuration settings.

A hardware implementation of button denouncing would vastly reduce the keypad code and would allow for more accurate keystrokes however it would come at the cost of increased BOM.