CE323 ADVANCED EMBEDDED SYSTEMS DESIGN – ASSIGNMENT 2

1804336

SHINSKIY MAXIM University of Essex

Contents

Requirement Form

Name	Home security system.
Purpose	Notify owner of intruders, secure are by monitoring entrance into zones.
Inputs	4x4 Keypad; 1 door sensor; 1 zone sensor.
Outputs	3 Red LEDs; 16x2 LCD screen.
Functions	Read user input from keypad, display current state/error code, monitor zones' entrance when alarm set.
Performance	Can take several sensors if necessary. Updates sensor status every 50 ms.
Manufacture costs	£55
Physical size/weight	< 1kg in total; 18x10x5 (cm) control block, 10x10x5 (cm) motion zone sensor, 8x8x6 (cm) magnetic door sensor.
Power	1 W

Documentation

All functionality and features are implemented as requested by the assignment brief. Hence, the behaviour of the program is as expected. But the focus of this will be on the feature implementation and some specific parts of code that need explanation.

By default, the program starts from Unset state and waits for the user to enter a correct passcode: 1234. The passcode is store as a global constant char array. To check the users guess (input 4-digit code) the method passcode(char*) calls method checkPasscode(char[ ]) and passes the users input, the checkPasscode returns boolean representing the correctness of the pin (true – correct, otherwise false). If the code is correct the passcode method returns 1 as a boolean true and 0 as false, after 3 fails attempts. However, the passcode also returns 2 that indicates that the exit/entry period has finished. To print the state that the keypad is used in, passcode also calls method showState(char*) that takes name of the state as an argument.

Finite state machine logic is implemented using swich statement in a loop method. Each case in a switch statement is a state and current state is stored as a global "current" variable and is check in switch as an argument. Inside the case the current variable is passed to a respective state's method. unState (unset), exState (exit), setState (set), alState (alarm), enState (entry), restate (reset) methods all have un, ex, se, al, en and re constant variables respectively to represent and store current state. Every state accepts as an argument previous method's variable and return next method's variable that has been produced by state's logic.

The system has some time constrains like exit period, entry period and LED ON time. Exit period is the time that is allowed for a user to leave the house before the system goes into SET state and is equal to 20 seconds. Entry period is a time starting from activating the door sensor and lasts for 30 seconds, during this period the user has to enter a correct code or else ALARM state will be activated. LED ON time is set to 30 seconds and is a period of time from entering the ALARM state, during this period a dedicated LED will stay on and will only go off after the time period elapses or the system transitions into the next state.

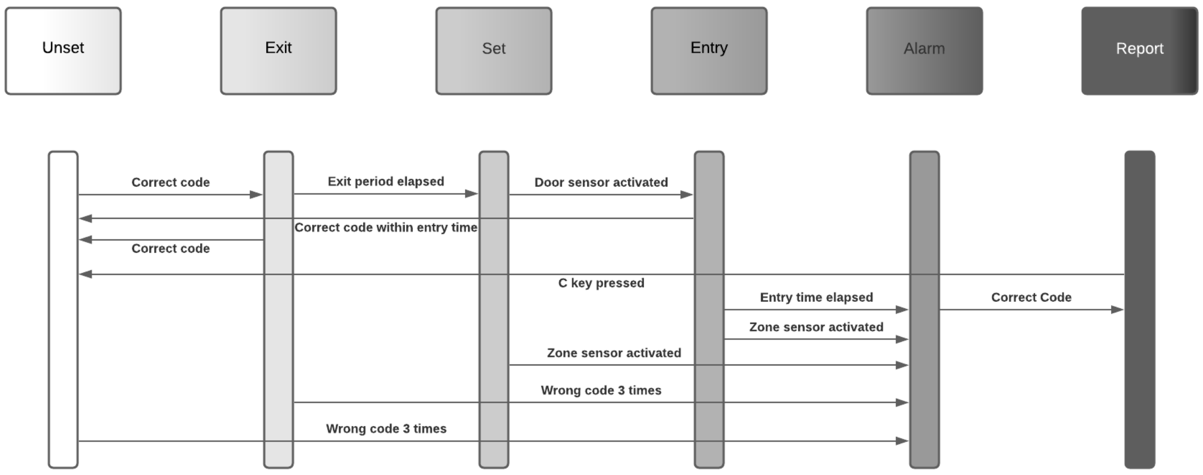
To count the time, the system uses Internal Service Routine that counts seconds. It is initialized by register TCNTx (TCCRxA and TCCRxB), OCRxA register and TIMSKx register to request interrupt, where x is number (1 in this case). TCCRxA defines the mode to work in (CTC in this case) and TCCRxB defines the prescaler value. Control Logic receives a pulse from clock cycle that passes through the prescaler, and increments TCNTx register by 1. After the value that has been incremented it is compared with OCRx register and when they match, the interrupt occurs. In this system the prescaler is 1024 and the interrupt occurs every 1 second. [1]

[1]: https://create.arduino.cc/projecthub/Marcazzan_M/internal-timers-of-arduino-58f6c9

Diagrams

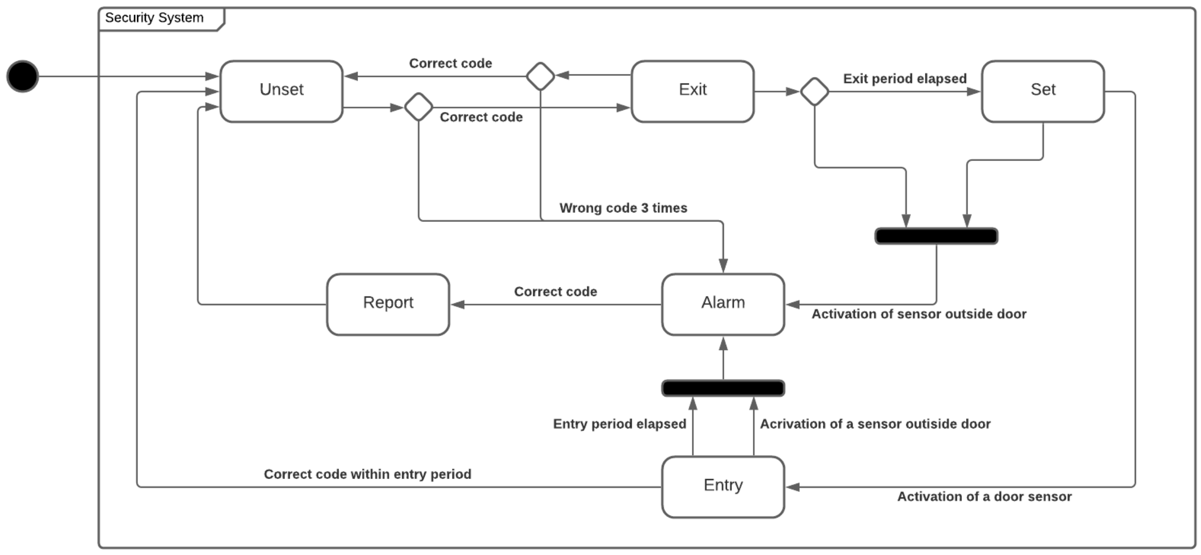
Sequence Diagram

The diagram the flow of the system. It starts from unset state. Every rectangle on the top represents the state and each arrow shows the direction of a transition of a state.



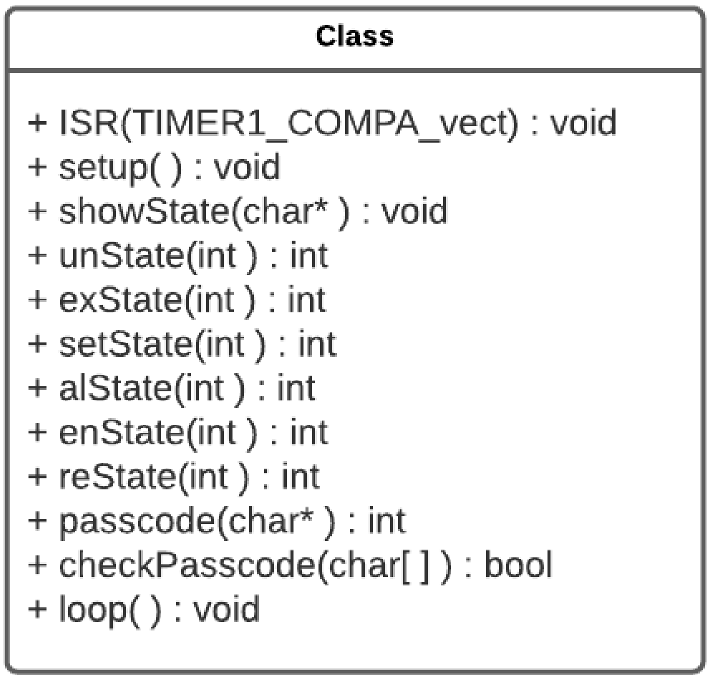
State Diagram

To keep the diagram clear and sensible only states are represented, other transitional methods are commented above the arrows. Black bar basically acts as an OR logic.



Class Diagram

Class diagram in C language basically shows a list of methods that a program has, because C doesn't have classes.



Class

+ ISR(TIMER1_COMPA_vect) : void
+ setup( ) : void
+ showState(char*) : void
+ unState(int) : int
+ exState(int) : int
+ setState(int) : int
+ alState(int) : int
+ enState(int) : int
+ reState(int) : int
+ passcode(char*) : int
+ checkPasscode(char[ ]) : bool
+ loop( ) : void

Appendix

The code has been written to run on Arduino and was evaluated in TinkerCAD.

Code

```
#include <Keypad.h>
#include <LiquidCrystal.h>

//Password
const char code[4] = {'1', '2', '3', '4'};


//Initial cursor position
int row;
int col;

//Keypad=============
const byte ROWS = 4;
const byte COLS = 4;

char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {11, 10, 9, 8};
byte colPins[COLS] = {7, 6, 5, 4};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

// LCD===============
LiquidCrystal lcd(A0, A1, A2, A3, A4, A5);

// LEDs==============
const int ledA = 13;

// Switches with LEDs
const int doorZone = 3;
const int sensorZone = 2;

//States consts======
int current = 0;
const int un = 1;
const int ex = 2;
const int se = 3;
const int al = 4;
const int en = 5;
const int re = 6;

//Interrupt
int timer = 0;
int ledVal = 0;
ISR(TIMER1_COMPA_vect){
    ledVal = ~ledVal;    //flip led (do blink)
    timer++;             //count time

    /*
    Alarm LED logic
    ---------------
    Blink if the currrent state is either exit or entry
    LED is ON for 30 seconds from transition into alarm state
    LED is OFF after 30 seconds in alarm state or
    if the current state is none of what have been mentioned above
    */
    if(current == ex || current == en) {
        digitalWrite(ledA, ledVal);
    } else if(current == al) {
        if(timer < 30){
            digitalWrite(ledA, HIGH);
        } else {
            digitalWrite(ledA, LOW);
        }
    } else {
        digitalWrite(ledA, LOW);
    }
    Serial.println(timer);
}

// Prepare===========
void setup() {
    //intialize components
    Serial.begin(9600);
    lcd.begin(16, 2);
    pinMode(ledA, OUTPUT);
    pinMode(doorZone, INPUT);
    pinMode(sensorZone, INPUT);

    current = un; //Initial state
}

// States===========
// states take previous state as an argument
// and return next state

//UNSET
int unState(int  from){
    if(passcode("UNSET")){
        //if passcode is correct enter exit state
        return ex;
    } else {
        //if passcode is wrong enter alarm state
        return al;
    }
}

//EXIT
int exState(int from){

    int returnState = passcode("EXIT");

    switch(returnState){
        //Wrong password/sensor
        case 0:
            return al;
        //Correct password
        case 1:
            return un;
        //Time is up -> set
        case 2:
            return se;
    }

}

//SET
int setState(int from){
    lcd.clear();
    lcd.setCursor (2, 0);
    lcd.print("State : SET");

    //Check zones
    while(1){
        if(digitalRead(doorZone)){
            return en;
        }
        if(digitalRead(sensorZone)){
            return al;
        }
        delay(50);
    }
}

//ALARM
int alState(int from){
    while(1){
        //reset timer
        timer = 0;

        //Go to report state after correct passcode
        if(passcode("ALARM")){
            return re;
        }
        delay(50);
    }
}

//ENTRY
int enState(int from){
    int returnState = passcode("ENTRY");

    switch(returnState){
        //Wrong password/sensor/time is up
        case 0:
        case 2:
            return al;
        //Correct password
        case 1:
            return un;
    }
}

//DONE
int reState(int from){
    lcd.clear();
    //print error code
    lcd.setCursor(2, 0);
    lcd.print("Error: 001");
    //print message
    lcd.setCursor(0, 1);
    lcd.print("Press C to clear");
    while(1){
        char key = keypad.getKey();
```

```
            if(key !=NO_KEY){
                //Check for C character
                if(key == 'C'){
                    return un;
                }
                Serial.println(key);
            }
        }
    }

    // Print state on LCD
    void showState(char* state){
        lcd.clear();                //clear screen
        row = 1;                    //set row and col to start position
        col = 0;

        //print state line
        lcd.setCursor(2, 0);
        lcd.print(state);
        //print code line
        lcd.setCursor(col, row);
        lcd.print("Code:____");
    }
    // Password==========
    int passcode(char *state){
        //Store users guess
        char guess[4];

        timer = 0;
        //Initialize interrupt
        TCCR1A = 0b00000000;
        TCCR1B = 0b00001101;
        OCR1A = 15625*1;    //1 second timer
        TIMSK1 = 0b00000010;

        sei();


        //Do lcd routine before beginning
        //Pass state string to print it
        char printState[16] = "State: ";
        strcat(printState, state);
        showState(printState);
        int passFail = 0;
        //go in a loop to count the fail times
        while(passFail < 3){
            //retrieve key
            char key = keypad.getKey();

            //Do special routine is these states
            if(current == en || current == ex){
                //check for exit period time
                if(current == ex && timer > 20){
                    return 2;
                }
                //check for entry period time
                if(current == en && timer > 30){
                    return 2;
                }

                //check zone switch
                if(digitalRead(sensorZone)) {
                    return 0;
                }
            }

            if(key != NO_KEY){
                //press C to delete last char
                if(key == 'C'){
                    //delete the message
                    //if last char deleted
                    if(col > 3){
                        lcd.setCursor(0, 0);
                        lcd.print("                   ");
                    }
                    if(col > 0){
                        //move cursor back
                        col--;
                    }
                    //Set delete char in the array
                    //and set cursor
                    guess[col] = NULL;
                    lcd.setCursor(col+5, row);
                    lcd.print('_');

                } else {
                    //until pasword is full
                    if( col < 4){
                        lcd.setCursor(col+5, row);
                        //put key into an array
                        //and do the routine for LCD
                        guess[col] = key;
                        lcd.print('*');
                        col++;
                    }
                    //print message if password fills
                    if(col > 3){
                        lcd.setCursor(0, 0);
                        lcd.print("Press B to set  ");
                    }
                    //when password is filled up
                    //and 'B' pressed
                    if(col == 4 && key == 'B'){
                        //check password
                        //if correct
                        if(checkPasscode(guess)){
                            //do routine
                            Serial.println("Correct!");
                            lcd.setCursor(0,0);
                            lcd.print("                ");  //delete previous
                            lcd.setCursor(0,0);

                            delay(100);

                            //return true when code is correct
                            return 1;
                        //if wrong
                        } else {
                            //count the number of failed times
                            passFail++;
                            Serial.print("Wrong! n: ");
                            Serial.println(passFail);
                        }
                        //reset the screen and password
                        showState(printState);
                    }
```

```
                }
                //print every key that was pressed
                Serial.println(key);
            }
            delay(50);
        }
        //after while loop
        //return false if user fails to
        //enter passcode 3 times IN TOTAL
        Serial.println("Alarm!");
        return 0;
    }

    //Compares global variable GUESS with global CODE
    bool checkPasscode(char guess[]){
        //check password
        int check = 0;
        for(int x = 0; x < 4; x++){
            if(guess[x] == code[x]){
                check++;
            }
        }
        //if correct
        if(check == 4){
            //do routine
            return true;
        //if wrong
        } else {
            return false;
        }
    }

    //Main======================
    void loop() {
        /*
        Main loop with the switch statement.
        As for FSM, this code has several states that are listed here as cases.
        The current state is tracked with 'current' variable.
        As a convention, 'current' variable as only updated in this switch case
        and never inside other functions.
        */

        switch(current){
            case un:
                Serial.println("Entered UNSET state");
                current = unState(current);
                break;

            case ex:
                Serial.println("Entered EXIT state");
                current = exState(current);
                break;

            case se:
                Serial.println("Entered SET state");
                current = setState(current);
                break;

            case al:
                Serial.println("Entered ALARM state");
                current = alState(current);
                break;
        case en:
            Serial.println("Entered ENTER state");
            current = enState(current);
            break;

        case re:
            Serial.println("Entered RESET state");
            current = reState(current);
            break;
    }
    delay(50);
}
```