DAT110 - PROJECT 4

GROUP 6

By Sondre Gjesdal (571736), Raida Talukdar (183566) & Arja Sivapiragasam (571334)

Introduction

The aim of the project in was to create a hardware/software co-design of an access control device. We solved this by creating a simulation of an Arduino circuit board system using TinkerCad.

The simulation was then connected to the cloud using REST service API, creating a functioning IoT device.

Access Control Design Model

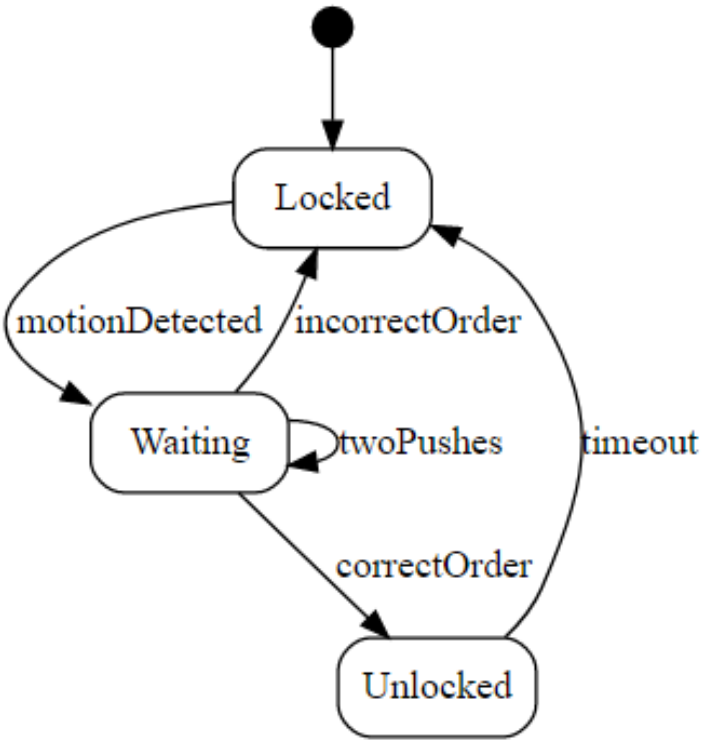

Figure 1. Finite State Machine using UMPLE.

We created a finite state machine using the online tool Umple Online. It shows the states transitioning between LOCKED, WAITING and UNLOCKED.

When the system is in a locked state it can be put into a waiting state by motion detection using a sensor. In the waiting state two pushes of the buttons in the correct order will cause the system to become unlock. If the order of the buttons being pushed are incorrect, the system will automatically go back to the locked state. Finally, if the system is unlocked it will wait for a given time period until it goes back to the locked state.

Access Control Hardware/Software Implementation

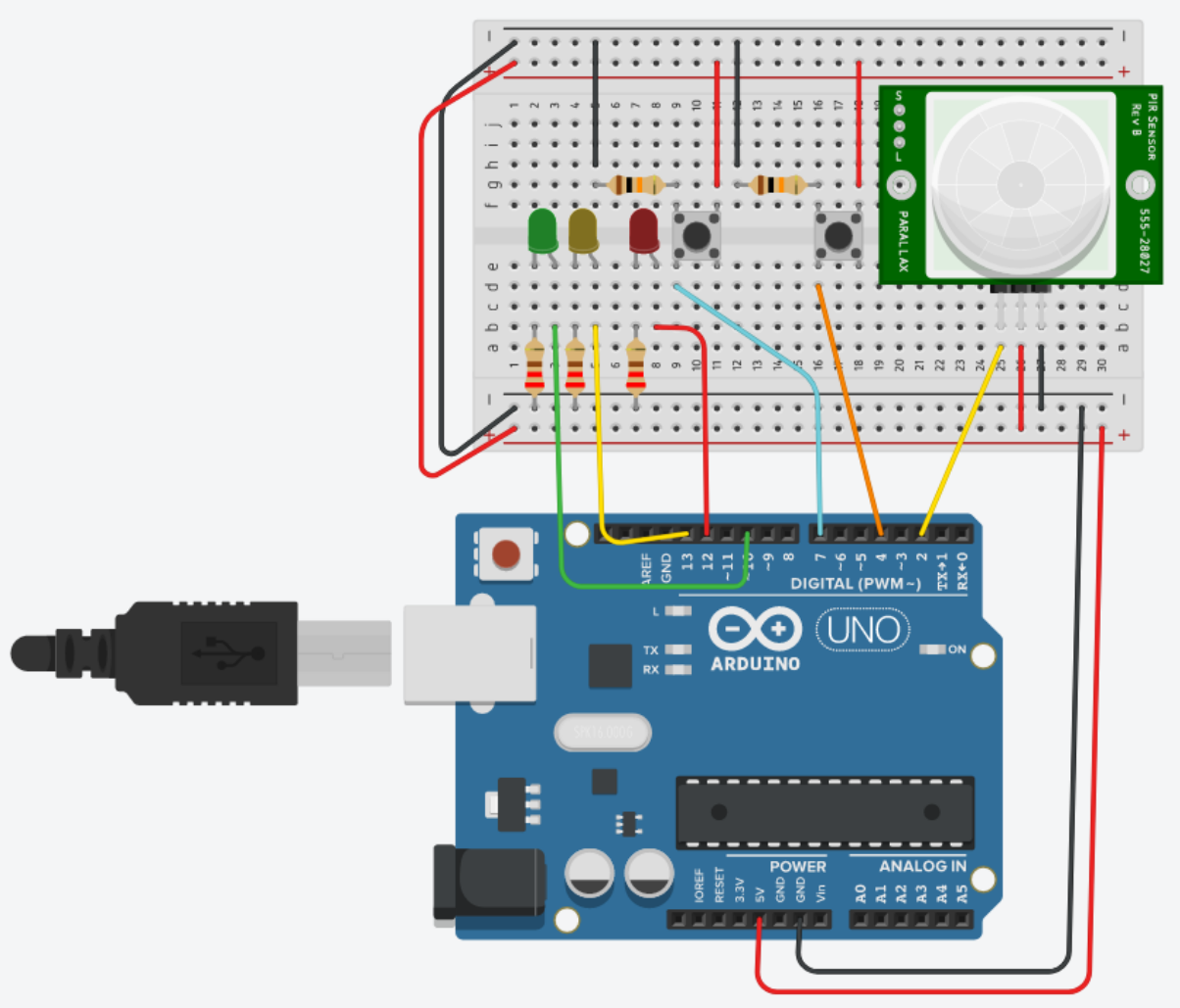

Figure 2. Arduino Board Simulation.

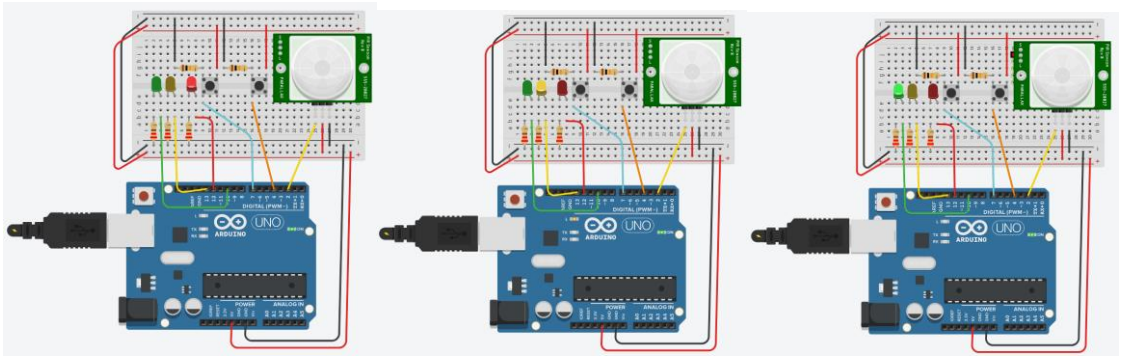

Figure 3. Arduino Board in locked state, waiting state and unlocked state.

We began connecting the hardware components by placing them on the the arduino board. We started by connecting the red, yellow and green LED-lights, each with a 220 Ω resistor. We wrote a simple code to make sure all the lights were connected correctly. Then we connected a passive infrared-sensor (PIR) for motion detection and proceeded with writing code for the light to change from red to yellow (waiting) when motion was detected. The red light would get switched off whenever the system entered the waiting state.

The next stage was to connect the two buttons. This was the tricky part because we had to write a code that would make sure the order of the buttons being pushed followed a certain pattern. In this particular system the code was written so that the button the right first, followed by a push of the left button, in order to enter the unlocked state.

We solved this problem by initiating a counter to keep count of how many times the button was pushed. Every time the button is pushed within the loop the counter is incremented. The system stays the waiting state until the counter been incremented twice.

Coding the Arduino circuit board simulation

Here is the final source code including short explanations:

```
1  //initiating variables
2  int sensor = 0;
3
4  int btn = 0;
5
6  int btn2 = 0;
7
8  int go = 0;
9
10 int locked = 0;
11
12 int waiting = 1;
13
14 int unlocked = 3;
15
16 int state = locked;
17
18 int antallTrykket = 0;
19
```

```
// setting up the various input and output connections
 // and relating them to a number
void setup()
{
  pinMode(2, INPUT);
  pinMode(4, INPUT);
  pinMode(7, INPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(10, OUTPUT);
}


// this section of code will loop until the system is switched off
void loop()
{
  sensor = digitalRead(2); // reads sensor input
  btn = digitalRead(4);  // reads button iinput
  boolean pushed = false;
```

```
    // if sensor has been activated the red light is switched off
 if (sensor == HIGH) {
   digitalWrite(12, LOW);
   if (state == locked) { // if the state was locked the state changes to waiting
       state = waiting;
   }

       // if the state is waiting while in this state this section of code loops
        // until the button
       // has been pushed twice
   while (state == waiting && antallTrykket < 2) {
     digitalWrite(13, HIGH); // the yellow light is switched on
     btn = digitalRead(4);   // reads button nr 1
     btn2 = digitalRead(7); // reads button nr 2

     if (btn == HIGH) {
       pushed = true;  // first button has been pushed
     }

     // if btn1 has been pushed and btn2 is pushed after
     if (pushed && btn2 == HIGH) {
       state = unlocked;  //the state is set to unlocked
     }
```

```
      // the yellow light will blink if either of the buttons are pushed while in
        // the waiting state
      if ((btn == HIGH || btn2 == HIGH) && state == waiting){
        antallTrykket++; //when a button is pushed the counter increments by 1
        digitalWrite(13, LOW);
        delay(200);  // waits for 2 seconds (blinking effect)
        digitalWrite(13, HIGH);
      } // if-yellow
    } // while not unlocked


   // once the state is set to unlocked, the yellow light is switched off and the
     // green light
   // turns on, the system then "waits" for 6 seconds until the green light turns
     // off again
   // and it goes back to the locked state
  if (state == unlocked) {
      digitalWrite(13, LOW);
      digitalWrite(10, HIGH);
      delay(6000);  // wait for 6000 millisecond(s)
      digitalWrite(10, LOW);
    } // if-green
      } // if 1
```

```
   //system goes back to the locked state and the red light turns back on
   digitalWrite(13, LOW);
   digitalWrite(12, HIGH);
   antallTrykket = 0;
   state = locked;
}
```

REST API cloud service

The routes in the service are directed to two sites, /accessdevice/log and /accessdevice/code.The first line is using spark/java frameworks post method for storing messages sent to the accessdevice. The method takes the message in the body and stores it in the log as shown later since it's done in the AccessLog.java class.
Second line retrieves the entire log of access entries and shows it as JSON representation for structure.

Third line selects a specified log by entering the specified ID of the access entry and shows it in the JSON format as seen that the method converts the log entry into JSON using gson. Fourth line clears the log using clear method from Spark and returns message that the log is cleared.

Fifth line retrieves the entry code for the IoT application using Spark/java frameworks get method. Sixth line updates the entry code to the new accesscode.

```
post( path: "/accessdevice/log", (req, res) -> accesslog.add(req.body()));
get( path: "/accessdevice/log", (req, res) -> accesslog.toJson());
get( path: "/accessdevice/log:id", (req, res) -> gson.toJson(accesslog.get(Integer.parseInt(req.params(":id")))));

delete( path: "/accessdevice/log", (req, res) -> {
    accesslog.clear();
    return gson.toJson( src: "Log is now clear");

});
// TODO: implement the routes required for the access control service

get( path: "/accessdevice/code", (req, res) -> gson.toJson(accesscode));
put( path: "/accessdevice/code", (req, res) -> {
    accesscode.setAccesscode(req.queryParams("accesscode"));
    return gson.toJson( src: "Success, added new code!");
});
```

The storage is a concurrent hashmap using AtomicInteger as ID key. It also uses gson to convert to and from String and JSON.

```
public AccessLog () {
    this.log = new ConcurrentHashMap<Integer,AccessEntry>();
    cid = new AtomicInteger( initialValue: 0);
    gson = new Gson();
}
```

The add method takes the message and converts it to JSON using gson library.
The method also uses the incrementAndGet method from AtomicInteger for simple way of incrementing the ID value.

```
public int add(String message) {

    int id = cid.incrementAndGet();
    log.put(id, gson.fromJson(message, AccessEntry.class));
    return id;
}
```

Get method is a simple get method by retrieving the specified id as shown.

```
public AccessEntry get(int id) {
    return log.get(id);

}
```

Clear method uses the ConcurrentHashMap clear method and resets the AtomicInteger to 0.

```
public void clear() {
    log.clear();
    cid.set(0);

}
```

toJson method is an easier way of converting the log to JSON, using gson, so it's possible to show it through the deviceaccess/log page.

```
public String toJson () {

    String json = gson.toJson(log);

    return json;
}
```

Device Communication

RestClient.java is the main communicator between the "Arduino" and the cloud service. The class contains a HTTP POST and a HTTP GET method.

First is the doPostAccessEntry(String message) which sends the current message from the Arduino to the cloud service. This results in a unlocked and a locked entry to the log.

```
public void doPostAccessEntry(String message) {

    // TODO: implement a HTTP POST on the service to post the message
    AccessMessage accessMessage = new AccessMessage(message);

    try (Socket s = new Socket(Configuration.host, Configuration.port)) {
        Gson gson = new Gson();
        String jsonbody = gson.toJson(accessMessage);

        String httpputrequest =
                "POST " + logpath + " HTTP/1.1\r\n" +
                        "Host: " + Configuration.host + "\r\n" +
                        "Content-Type: application/json\r\n" +
                        "Content-length: " + jsonbody.length() + "\r\n" +
                        "Connection: close\r\n" +
                        "\r\n" +
                        jsonbody +
                        "\r\n";

        OutputStream output = s.getOutputStream();

        PrintWriter pw = new PrintWriter(output,  autoFlush: false);
        pw.print(httpputrequest);
        pw.flush();

        InputStream in = s.getInputStream();
        Scanner scan = new Scanner(in);
        StringBuilder jsonResponse = new StringBuilder();
        boolean header = true;

        while (scan.hasNext()) {
            String nextLine = scan.nextLine();

            if (header) {
                // System.out.println(nextLine);
            } else {
                jsonResponse.append(nextLine);
            }

            if (nextLine.isEmpty()) {
                header = false;
            }
        }
        System.out.println(jsonResponse.toString());
        scan.close();

    } catch (IOException ex) {...}
```

Second is the doGetAccessCode() which is used by the arduino in network mode to check for the current access code using a HTTP GET request.

```
public AccessCode doGetAccessCode() {

    AccessCode code = null;

    try (Socket s = new Socket(Configuration.host, Configuration.port)) {
        // construct the GET request
        String httpgetrequest = "GET " + codepath + " HTTP/1.1\r\n"
                + "Accept: application/json\r\n"
                + "Host: localhost\r\n" + "Connection: close\r\n" + "\r\n";

        // sent the HTTP request
        OutputStream output = s.getOutputStream();

        PrintWriter pw = new PrintWriter(output, autoFlush: false);
        pw.print(httpgetrequest);
        pw.flush();

        // read the HTTP response
        InputStream in = s.getInputStream();

        Scanner scan = new Scanner(in);
        StringBuilder jsonresponse = new StringBuilder();
        boolean header = true;

        while (scan.hasNext()) {
            String nextline = scan.nextLine();
            if (header) {
                // System.out.println(nextline);
            } else {
                jsonresponse.append(nextline);
            }
            // simplified approach to identifying start of body: the empty line
            if (nextline.isEmpty()) {
                header = false;
            }
        }
        Gson gson = new Gson();
        code = gson.fromJson(jsonresponse.toString(), AccessCode.class);

        System.out.println(jsonresponse.toString());
        scan.close();

    } catch (IOException ex) {...}
```

System Testing

By running the Cloudservice and IoT device at the same time we can test the system with a bit of help from the Postman tool.

While the two programs are running we used the Postman tool to issue a PUT to /accessdevice/code to change the accesscode to something other than the standard.
If we now open Chrome and direct the browser to /code, we should see that the code has changed to the value we issued.

If we then try the new code on the Arduino in network mode, we should be able to unlock it. Now we should see in /log that there are two log entries, one for unlocked state and the other for locked state. If we redirect the browser to /log/1 we should see the first of the log entries for itself.

Lastly, we can use the Postman tool to issue a DELETE command to /log to check that our delete method is working. this should return a message that says: “Log is now clear”.
All of these tests should be working and that means that the program works as it should by the requirements

Conclusions

The project is fully finished and working as intentionally after the requirements. This project has spiked our interest for working with Arduinos and was an interesting new experience for all of the members. We’re looking forward to an opportunity to test this with a physical Arduino another time.

Links to the source codes

Github repository for the IoT device and for the cloud service:
https://github.com/S571736/DAT110_Project4.git

TinkerCad simulation with the Arduino code:
https://www.tinkercad.com/things/5NDVw6EtW62