# Access Control System

Steven Diviney 08462267

December 21, 2012

# 1 Introduction

This project consists of three main parts. The LCD screen, the I2C buttons and the controlling logic. These three components are separated conceptually and in the projects physical structure. The LCD screen is controlled in almost exactly the same fashion as the previous assignment. Four buttons are used to simulate the doors and communication is done with I2C. The controlling logic is implemented as a state machine.

FreeROTS is used to coordinate communication between the three components. Each of the three components executes as a separate FreeROTS task.

The console is used extensively to display output detailing the state of the system. The LEDs are on when doors are locked. The code for the LCD display is 0001.

# 2 State Machine

The state machine resembles a state table with action methods. It contains a number of states, events and actions. The states and events are used to index a two-dimensional array of function pointers. These functions transition the state and perform any additional processing needed to transition, i.e. Turning LEDs on or off.

The LCD and button components have a reference to a FreeROTS queue managed by the state machine. When the correct password is entered using the LCD or a button is pushed an event is placed on the queue. The state machine then resumes execution and processes the event.

This approach produced relatively clean and easy to understand code. This was partially due to the small number of states and events, although more states would also be manageable. The state machine logic is kept to a minimum and follows intuitively from a state table drawn up during design. Debugging is also greatly simplified. Each state and transition is checked to validate the logic of the application. Any remaining issues are with the hardware.

## 2.1 States

The states and events are encoded as enumerated integers. There are 5 states;

- s1: The start state in which all the doors are locked.
- s2: Outer door closed.
- s3: Outer door open.
- s4: Inner door open.
- s5: Inner door closed.

## 2.2 Events

The seven possible events are also encoded as enumerated integers;

- AB: Valid code entered or button B pressed.
- C: Button C pressed.
- D1O: Outer door opened.
- D1C: Outer door closed.
- D2O: Inner door opened.
- D2C: Inner door closed.
- T: Timer expired.

## 2.3 Implementation

The state machine code resembles the table below. The table is encoded as an array. The current state of the machine is used to index the column while the event being processed is used to index the row. Each entry in the table is a function pointer. The function performs the necessary processing and advances the state. The action methods are very simple. Apart from advancing the state they toggle the LEDs and start the 5 second timer when needed.

The table below has scope for reduction. This was not done because of time constraints, and resulted in many of the action methods being essentially redundant. However, methods for each transition could be useful for debugging.

The invalid and don't care states must also be implemented as functions, otherwise the machine would try and execute NULL entires. The invalid and don't care action methods simply return.

The bulk of the implementation time was spent on implementing message passing using FreeROTS queues. The logic of the application was relatively easy to encode.

| | AB | C | D1O | D1C | D2O | D2C | T |
|---|---|---|---|---|---|---|---|
| s1 | s2 | s4 | - | - | - | - | - |
| s2 | - | - | s3 | - | - | - | s1 |
| s3 | - | - | - | s1 | - | - | x |
| s4 | - | - | - | - | s5 | - | s1 |
| s5 | - | - | - | - | - | s1 | x |

## 2.4 Timer

The timer was implemented in software using FreeROTS. After 5 seconds a callback function is executed. This callback explicitly advances the state (theres no reason for this other than I ran out of time to test the callback function placing an event on the queue. The code is there however and should work.) As can be seen from the table the timer is ignored if a door is open as there is no point locking the door if it is open.

# 3 I/O

The I/O contains 3 parts. The LCD screen and the buttons on the development boards are used as input. The LEDs above the buttons are used as output. The console is also used to display any changes in the inputs or outputs.

## 3.1 LCD

The LCD module is almost identical to the work submitted for assignment 2. The only additions are the inclusion of a password hard-coded in the file as a string. When the user presses "OK" their input is checked against this string. If the two match an event is placed on the queue and processed by the state machine.

The user input is rendered on the screen after each key-press. This may not be a great idea from a security standpoint but makes testing the system much easier. When the input is cleared the top portion of the screen is drawn over. This saves having to redraw the entire screen.

## 3.2 Buttons

The button layout is as follows;

- Button 1 is used to simulate "B" being pressed. Falling edge
- Button 2 is used to simulate the sensor on the outer door. Falling edge is door open, rising edge is door closed.
- Button 3 is used to simulate "C" being pressed. Falling edge.
- Button 4 is used to simulate the sensor on the inner door. Falling edge is door open, rising edge is door closed.

The buttons are read over I2C. A pole is done every 20ms. If the state of the buttons changes from the last poll the event is determined and sent to the state machine.

## 3.3 LEDs

The LEDs over buttons 1 and 3 are used to display system output. LED1 is lit if the outer door is locked and LED3 is lit if the inner door is locked. The LED code is relatively simple. It consists of one function within the button module. The process is very similar to determining the button values only values are written out instead of being read.

The sensor class originally made the LED states mirror the button states. This has been removed.

# 4 Testing

The nature of the implementation made debugging relatively easy. Using the state table each input event was tested against each state. There were no defects found. State tables are generally very clearly defined. There are no edge cases or real possibilities for interference. As such a series of formal tests, as used in the last assignment, were not needed.

The LCD had been tested during the last assignment so it was assumed (relatively) bug free.

The buttons were verified simply by pressing and releasing them.

Any possible race conditions are guarded against in two ways. Each event is placed on a queue which guarantees events are processed in the order they occur. A door being locked, open and closed are considered separate events. A door must be closed before it can be locked. This is encoded in the state table.