Tawanda Muzanenhamo (MZNTAw004)

Wanna McMoses (WNNMCM001)

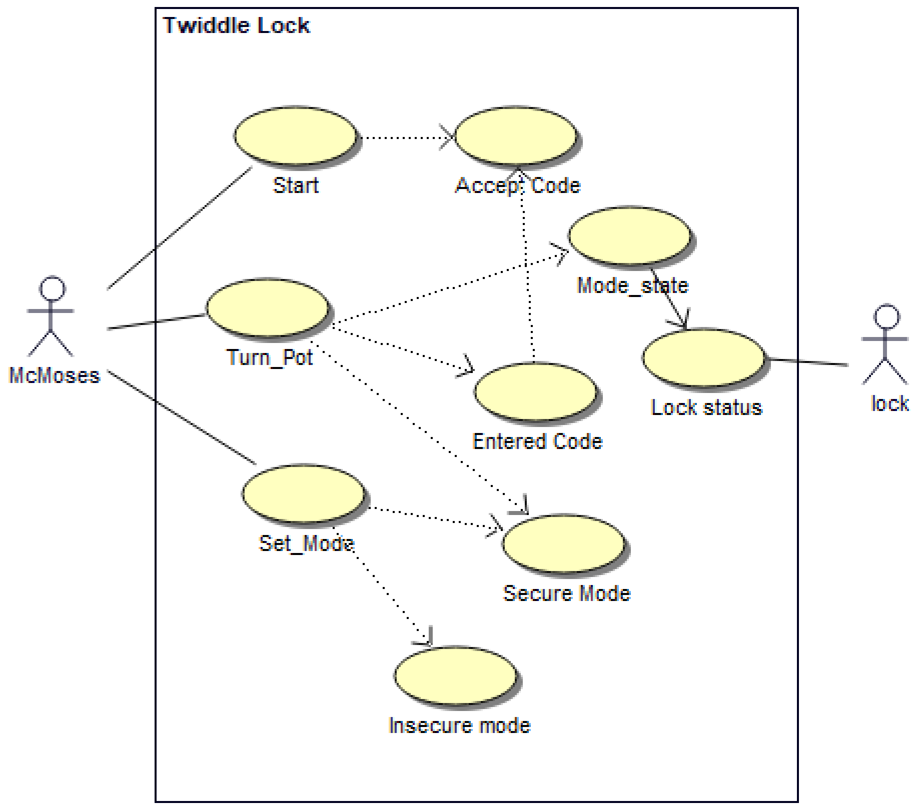
Project A Twiddle lock

Twiddle lock is a representative embedded product which implements the electronic form of the Dial Combination Safe which implemented using a python program which was executed on Raspberry Pi.

This report describes the design choices made by Tawanda Muzanenhamo and Wanna McMoses during the implementation on the Twiddle lock. The design of the circuit and reasons for decisions is at the beginning and develops into the Use case showing the requirements for the system and how they were implemented, after the requirements the specifications and implementation of the python program are illustrated through a flow chart and state chart. The validation and performance of system rounds up the report together with the conclusion on the success of the system

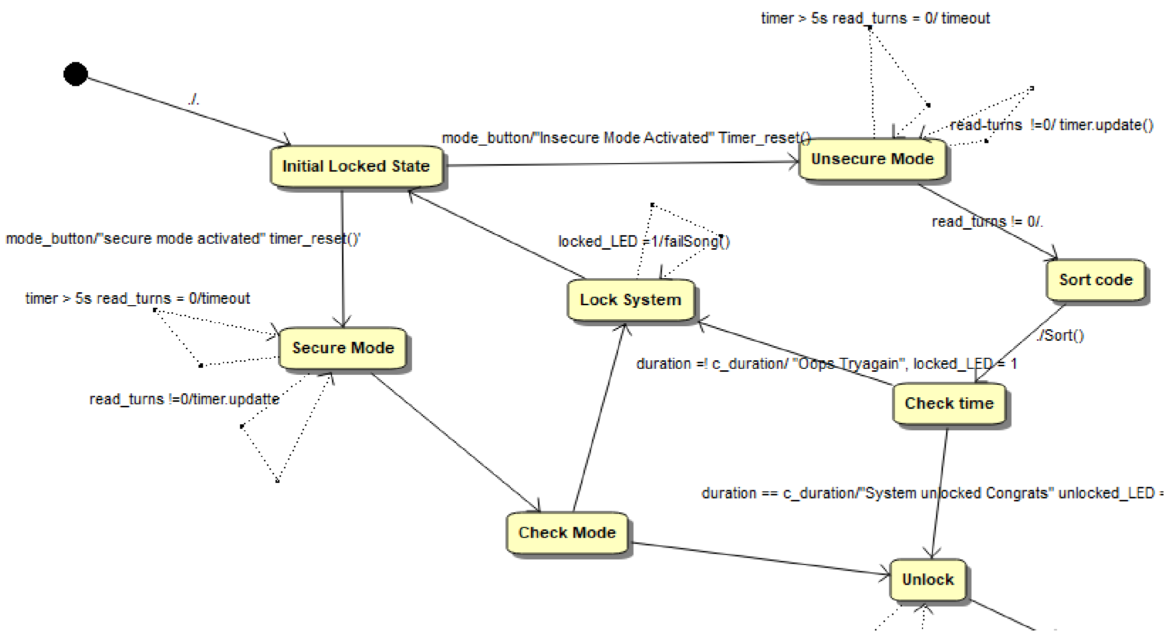
The circuit design included two LEDs a red one to show locked mode and a green one to show unlocked mode, this was do so that the user could easily tell the mode in which the system is in. An MCP3008 was used as an ADC converter because raspberry pi does not execute ADC conversion. Two switch buttons were used with one being used to indicate start and the other one being used to toggle between secure and unsecure mode. The GPIO BCM pinout of the raspberry pi was used because it made it easier to connect the channels of the MCP3008 i.e. MOSI, MISO etc. A potentiometer was used to enter the combo code required to unlock the system, the Right direction of the combo code was selected in the event that the start voltage output from the potentiometer was greater than the voltage after turning the potentiometer and the Left direction was selected when the voltage after turning the potentiometer was greater than the voltage before turning the potentiometer. The code entered by user was allowed to have a tolerance of up to 50ms so as to arrow room for error small errors during the entering of the code

Requirements

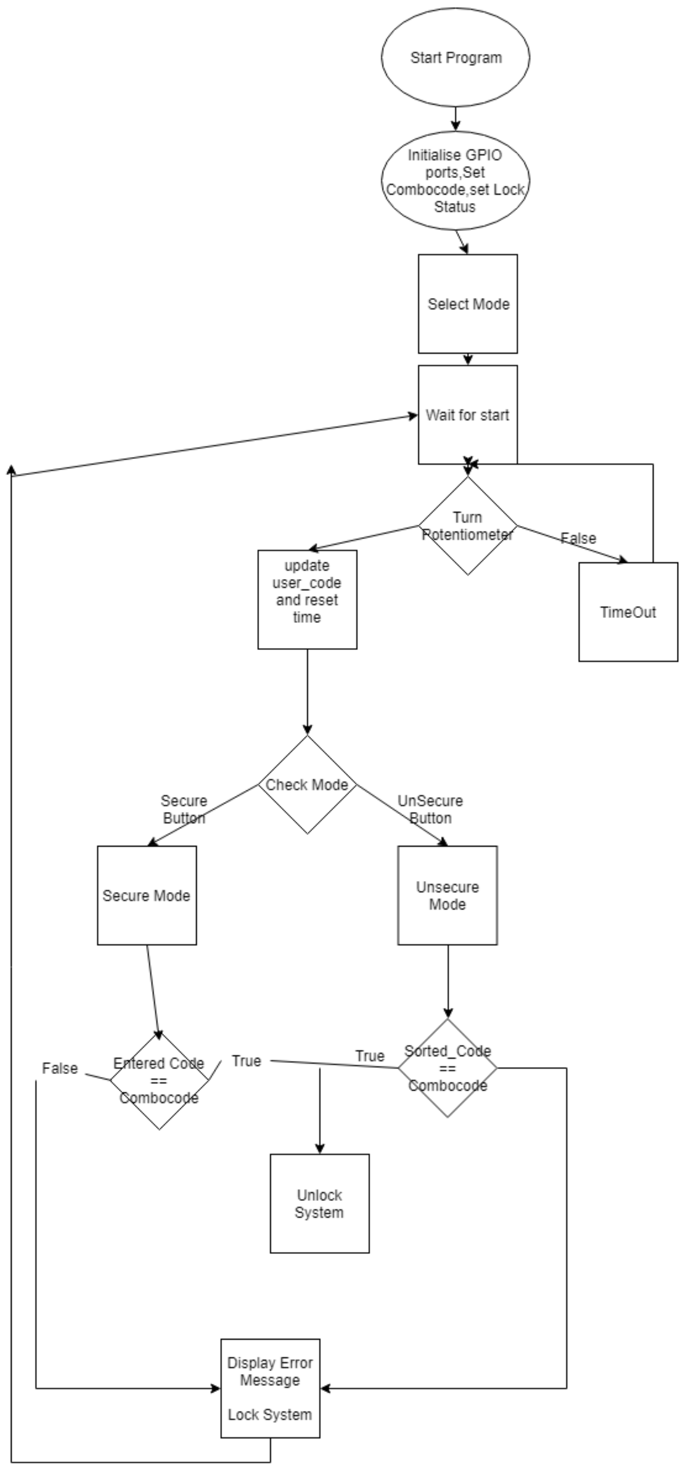


During the implementation of the Twiddle lock, the service line was replaced by the start/stop button which worked in the same way but made it easier for the user to understand what it's doing. The dial was implemented by turning a potentiometer in different directions because we could not get hold of the dial knob. A use case which checks the mode state selected so it determines how to interpret the entered code was added, this was implemented by having a sort algorithm within the Unsecure mode so that it would arrange the code entered regardless of the order.

Specifications and Design


timer > 5s read_turns = 0/ timeout
./.
mode_button/"Insecure Mode Activated" Timer_reset()
read-turns !=0/ timer.update()
Initial Locked State
Unsecure Mode
read_turns != 0/.
mode_button/"secure mode activated" timer_reset()'
locked_LED =1/failSong()
Sort code
timer > 5s read_turns = 0/timeout
Lock System
Secure Mode
./Sort()
duration =! c_duration/ "Oops Tryagain", locked_LED = 1
read_turns !=0/timer.updatte
Check time
duration == c_duration/"System unlocked Congrats" unlocked_LED =
Check Mode
Unlock

Flow chart of how the Twiddle.py program was implemented


Start Program
Initialise GPIO ports,Set Combocode,set Lock Status
Select Mode
Wait for start
Turn Potentiometer
False
update user_code and reset time
TimeOut
Check Mode
Secure Button
UnSecure Button
Secure Mode
Unsecure Mode
False
Entered Code == Combocode
True
True
Sorted_Code == Combocode
Unlock System
Display Error Message
Lock System

Text

Implementation

The function below was used to read the directions from the potentiometer, as illustrated in the introduction, the Right direction was recorded when the start “at Initial position” voltage was greater than the end voltage “at end position” and the Left direction was recorded when the end voltage was smaller than the start voltage

```
def read_turns(start_voltage, end_voltage):
    if start_voltage > end_voltage:
        return "R" # right direction relative to start voltage
    else:
        return "L"
```

The time spent entering the code was recorded by finding the differences in times when the first turn was made to when the direction was changed i.e. at the end of a direction as illustrated in the code below

```
            duration.append((end-begin)*100)
            user_code.append(read_turns(start_voltage, end_voltage))
            value = False
            start_voltage = mcp.read_adc(read_pot)
            end_voltage = mcp.read_adc(read_pot)
```

the value was multiplied by a 100 so as to have a value easier for human understanding

The secure mode was implemented by checking if the code entered by the user matched that supplied in combocode as illustrated in the code below

```
if secure == False:
        return True
    for i in range(0, len(user_code)):
        if user_code[i] != combocode[i]:
            return False
    return True
```

The true is only returned when the codes matches otherwise false is returned the system stays in locked state.

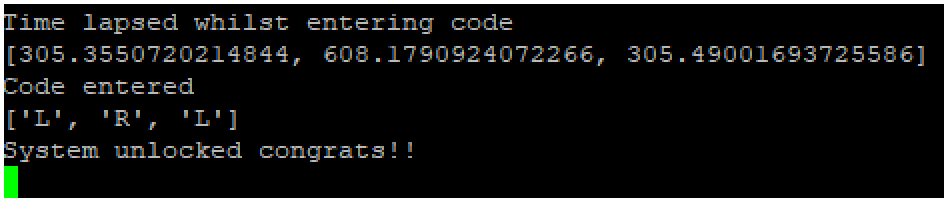
The duration was compared by taking the time lapsed during the entering of the code and comparing it to the supplied duration at the beginning of the code, the compare times function was Implemented in the code below

```
if secure == False:
        duration.sort()
        c_duration.sort()
    for i in range(0, len(duration)):
        if abs(duration[i] - c_duration[i]) > tolerance:
            return False
    return True
```

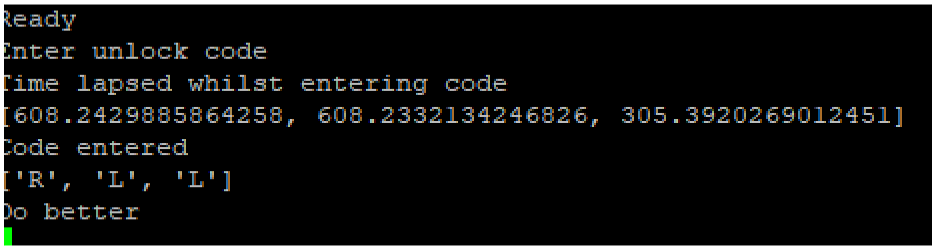
The tolerance allowed room for deviation in the duration of entered code and the supplied duration

Validation and Performance

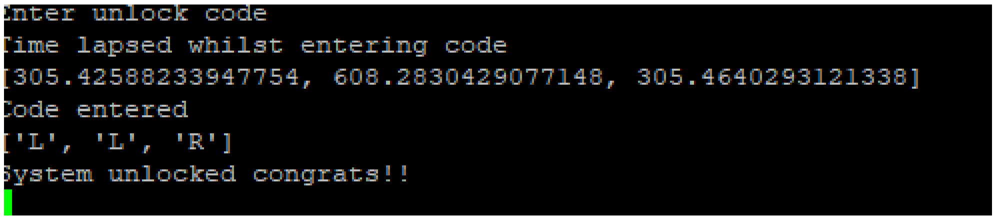
The Twiddle lock accepted user input from the potentiometer and then compared it to the combocode. The system only transition from locked in secure mode which is the initial state to unlocked in the event that the user entered code is equal to the combocode and the duration is within the duration specified taking into account the tolerance


```
Time lapsed whilst entering code
[305.3550720214844, 608.1790924072266, 305.49001693725586]
Code entered
['L', 'R', 'L']
System unlocked congrats!!
```

If a wrong code is entered the system would remain locked and indicates through the red LED that its locked


```
Ready
Enter unlock code
Time lapsed whilst entering code
[608.2429885864258, 608.2332134246826, 305.3920269012451]
Code entered
['R', 'L', 'L']
Do better
```

In unsecure mode the code did not require user input code to be in order it would use the implemented sort to match the sort and transition the state given that the code is close to being correct as shown in the shot below


```
Enter unlock code
Time lapsed whilst entering code
[305.42588233947754, 608.2830429077148, 305.4640293121338]
Code entered
['L', 'L', 'R']
System unlocked congrats!!
```

The actual combocode was L R L but the entered code was L L R the sort made the code look the same

Conclusion

The twiddle lock project was implemented successfully using limited resources without any problems. The potentiometer replaced the knob and performed the role of transferring the input to the code with great ease. The Time duration which was used in secure mode to unlock the system helped make the system even more secure since not only was the user supposed to know the code but they were supposed to know the right duration thereby increasing the security of the programme

However, the unsecure mode ensured that the system would unlock even if the code entered was in the wrong order, this was a drawback on the security of the program since it would allow for users to guess codes thereby letting in intruders

Despite the drawback of unsecure mode, the twiddle lock was a successful