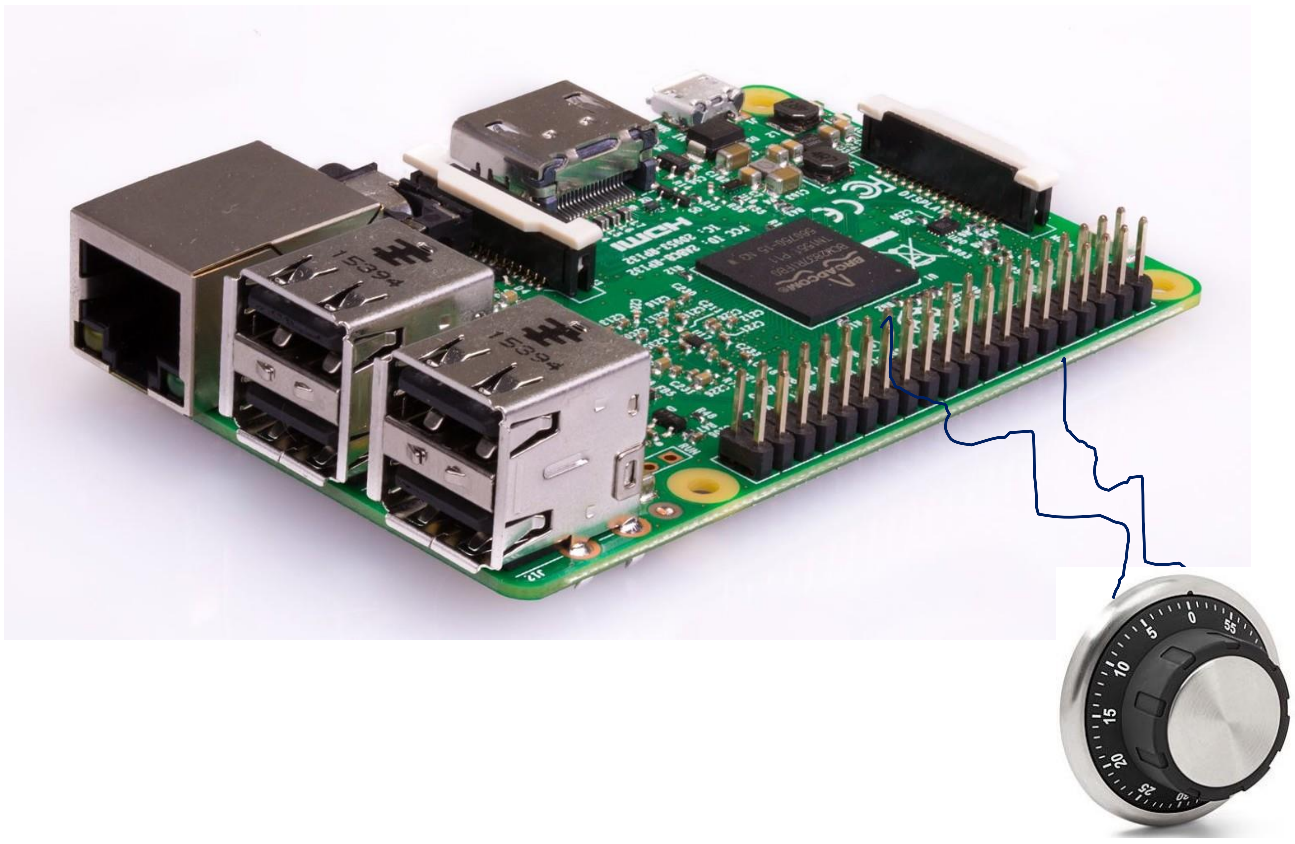
EEE3096/5S



Mission:

‘Twiddle Lock’

Practical 6 / Project A

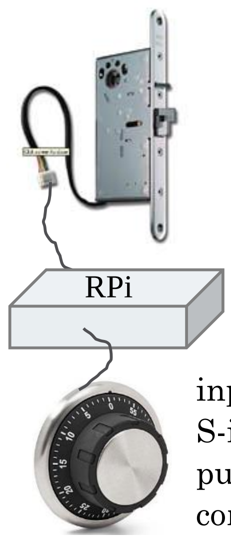
&

ELO5.2 Assessment

Practical 6 / Project A

The objective of this practical (or rather mini-project) is the development of a representative embedded system product, it will be a very small system due to the duration of the time available for this prac. A short report needs to be prepared and submitted for this practical.

The Mission: code-named Twiddle Lock



You are tasked to develop the Twiddle Lock. In brief this is an electronic form of the classic 'dial combination safe' (or DCS as they call it in the trade) combinational lock mechanism. But in this project you will use a Raspberry-based application that will listen to a button that will indicate you are about to put in a 'dialed code', and this dialed code will be sensed by an ADC connected to a POT that is attached to the dial, or in our case just a knob that represents the 'twiddle knob'. The application needs to read in sampled values from the ADC connected to the twiddle knob, we can call this the C-input (for code input). There will be a second input line, let's call it the S-input (for service request line) which the user will press in order to put the system into a mode that it will start sensing the input code combination. The application has another two GPIO lines that are outputs, the L-line and the U-line. When the L-line is held HIGH by the processor for 2s, it represents a lock signal. Whereas, when the U-line is held HIGH for 2s, it represents an unlock signal. These two signals are generally kept low most of the time. Note that holding either the L-line or the U-line for high much beyond 2s may cause the automatic locking mechanism to fail. The block diagram below illustrates the system design.

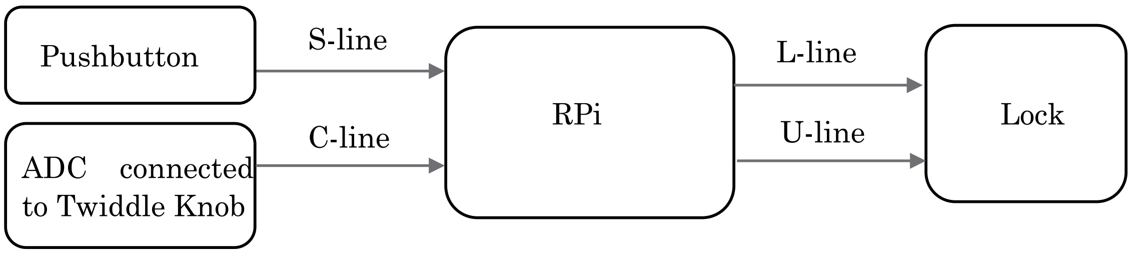


Fig. 1: Block diagram of Twiddle Lock system.

The S and C inputs work as follows: the user presses the S button to indicate that a code will be entered within the next two seconds. The C input, as you may expect, is used like the dial on a DCS; i.e. the user turns the C knob in one direction, clockwise or anticlockwise, to a certain point – this corresponds to the first symbol of the combination code. Here we are assuming that the direction and duration of turning the knob corresponds to entering a symbol, where the combination code is a collection of these symbols. The user either pauses for a little while (say less that 1s), or immediately turns the knob in the opposite direction to start entering another symbol. This is then repeated any number of times until the user stops turning the knob for more than 2s. After this the system then checks the code that has been entered to see if it is accepted, i.e. matches the combination code that it is programmed to accept. If it is a match then the lock is made to open, and can make a suitable sound (like a short buzz, or something more fancy – which might need a speaker or earphone hooked up – that makes the sound of e.g. a lock opening, hands clapping or a slot machine jackpot sound, whatever you like). Otherwise, on an invalid combination the lock stays locked (and make a negative sound like a long buzz or a 'boo hoo' sound).

The system needs to support two modes: secure more and unsecure mode.

In secure mode the system operates as indicated above – and to work the user may need to move the knob to the right position before pressing the S button; so this is obviously going to be quite a secure combination lock. However, in unsecure mode it is all about the durations of the turns, not the sequence or direction. Consider that the combination sequence is set to L2R3L1 (where L and R indicates the direction and the number indicated the duration of the turn in multiples of 100ms, i.e. this sequence would be turning left for 200 ms, right for 300 ms and left again for 100ms). You could assume the system is tolerant to about 50ms (you should have this as a setting, you may need more that 50ms tolerance depending on the knob used, but this may need to be decided experimentally). If in secure mode the user would need to enter the combination sequence exactly to lock/unlock the device. Of course, if you are using a POT there may be a limit to how far you can turn the knob. So if you don't know the starting point it may be difficult to repeat the sequence.

In unsecure mode, using the above example combination, the user only needs to indicate durations of 200ms, 300ms, 100ms in any direction, either just with a delay of not turning to indicate a break between symbols words or just switching direction. Furthermore the delays do not need to be in the correct sequence when in unsecure mode, although the delays much be of the necessary duration needed (according to the tolerance setting). So in unsecure mode if the combination is set to L2R3L1 you could unlock using delays in the sequence 100ms. 200ms, 300ms, or more concisely you could enter in L1L2L3 or R1L3R2.

(If you like math then you are most welcome, and indeed encouraged, to compare the strength of the secure vs. unsecure modes assuming an alphabet of say 20 symbols and maximum combination code lengths of 5 symbols – here I'm assuming an alphabet of 20 symbols alphabet would imply the symbols are L1 - L10 and R1 - R10. If you want to be really fancy (not recommended) then you could work around the knob having a limited amount of turns, e.g. R10R10 would be impossible to enter, R10R4 being the maximum possible if the POT is at its leftmost extent). Part of implementing unsecure mode will involve sorting the delays, which will be stored in an array – this is explained further in the software section below.

You can use the same combination code for both lock and for unlock.

The above aspects are the minimum requirements for this mini-project, if you are feeling particularly enthusiastic you are welcome to take it a bit further so that you can add a mode to that a new combination code can be entered and then saved.

ELO5.2 Checks and Demonstration/Documentation Requirements

ELO5.2 requires the student to develop a representative embedded system. This needs hardware/software interfacing so that a digital-to-analogue convertor is sampled by software and then signal processing operations are applied. See the Requirements and Specifications section to see desired operation of the system.

The following sections elaborate more on the development to be carried out.

Parts:

It is pretty clear what parts you need to use for this practical. It happens that the RPi is rather stingy in terms of buttons and LEDs. Coincidentally, this provide an opportunity for you to wire up a basic pushbutton to the S-Line and LEDs for showing the status of the U and L output lines. You don't need to worry about a lock but you could use something like a Set Reset Flip-Flop (SR-FF) connected to another LED to remember what state the lock is in. In terms of the C-input… well this is more special: you could try to get hold of a nice big knob or something that looks like a dial for a safe but which you can hook up to a potentiometer.

Software:

You need to develop the following code for this project:

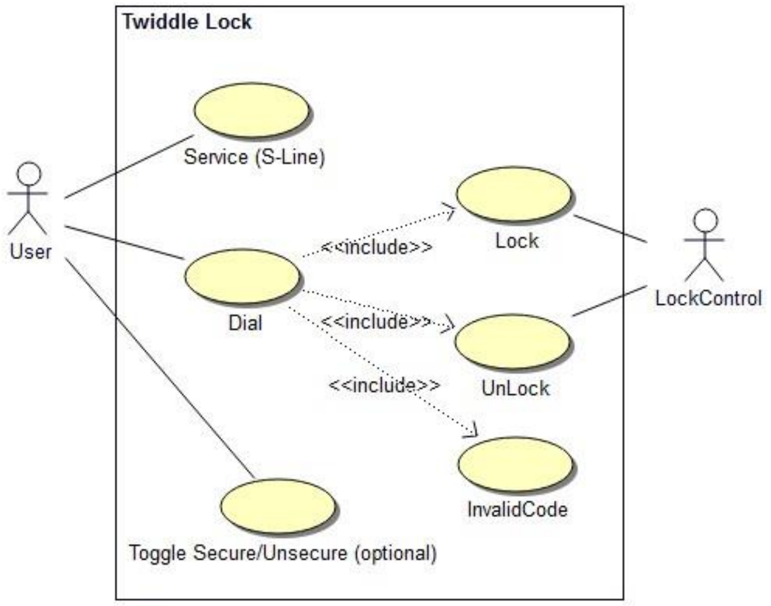
1. A main function that will perform the startup of the application and configuration. This will also include the main while(1) loop to poll the pushbuttons used, control the output lines to the L and U lines and manage timing. By default assume the lock is locked (if you are using a SR-FF circuit you could initially send a high pulse out of L-line just to be sure).
2. During the main function you need to keep a log (of up to 16 entries) of durations in milliseconds of the symbol that was entered (e.g. this log can be called int log[16]) and you need a second array storing the direction in which the knob was turned (e.g. call it int dir[16]). As an example if you turned the knob left for 220ms and then right for 550ms then log will be {220,550} and dir

will be {0,1} assuming 0=left, 1=right. If the S-button is pressed this will do a clear operation (e.g. reset the index for wiring to *log* and *dir*).

3. Develop a sort routine for the unsecure mode, it can use selection sort which is easy to implement. Pass the *log* array to sort the times in order to implement unsecure mode. The routine can have interface *sort (int x, int y, int n)* where the elements in *x* are placed into sorted order in *y*. You need to provide an ARM assembly implementation of sort even if you are using only Python, this can be provided in a separate assembly file – if you are using Python you can write an equivalent Python sort function which you use in your program, but you still need to write an ARM assembly version even if you don't call it from Python (why? Because you need to demonstrate some assembly programming).
4. Hard code a lock/unlock sequence, call it *combocode*, store as a const array in your code (this could be a class if you want to be a bit OO otherwise a struct to store direction and delay details). Implement code so that when the user inputs a combination code sequence, which will get stored in *log*, the system will match this will the stored *combocode* variable – note you may need to write a function that will convert the delays in *log* and the direction values in *dir* to an appropriate form to match with *combocode*. The appropriate lock or unlock must be triggered when a valid passcode is entered (remember to raise the L or U line for 2s only and make a suitable sound).
5. Implement code for unsecure mode so that the hard-coded passcode is sorted and then when the C-input has stopped (i.e. user is done entering in the code) the log variable will be sorted and then compare to *combocode* to decide if the lock should be opened.

Requirements

The operation of the locking system explained previously need to be implemented. The use case diagram on right gives a brief summary of the main operations, essentially the user interacts with the S-Line (for requesting service) and then dials in a combination that will cause a lock or unlock to occur if successful otherwise indicate an incorrect code was given.



Report and Demo:

You need to prepare a minimum 3-page report for this assignment. This does not include the cover page, if you put one in. If you don't have a cover page indicating the assignment and your team's names then at least put this information at the top of the first page to facilitate the marking process.

The table below explains parts of the report that need to be provided and related aspects that need to be demoed to a tutor. Mark allocations are shown at the end of the description for each part.

#	Part	Report	Demo
1	Introduction	Provide a short introduction (~ ½ page) to your project explaining your main design choices and how the report has been structured. [15 marks]	Introduce yourselves and your project. [8 marks]
2	Requirements	The requirements section should provide a refined UML Use Case diagram of the system, according to your implementation, and any accompanying text that is needed to clarify the requirements. Highlight any departures or additions that you may have made compared to the original project description given in this document. [15 marks]	Have an (at least draft) UML Use Case to show the tutor. Show this briefly, indicating any departures/additions. [6 marks]
3	Specification and Design	This section should provide a UML State Chart describing the main operation. Add a UML class or deployment diagram (or other suitable diagram) to indicate the structuring of your implementation (e.g. code modules / classes you may be using). You don't need to provide fine detail of the system, the diagram(s) can be e.g. at the level of functions. [20 marks]	Briefly show your design, you need not show more than one rough diagram (e.g. draft state chart) to the tutor. [10 marks]
4	Implementation	This section should give some snippets of important code and explanations for this (or referring to particular functions in code files). The point here is elaborating any parts of the State Chart that are not so straightforward to turn into code. [20 marks]	You should have a code file open already (e.g. where the main function is) before you start the demo. Briefly confirm to the tutor that this is part of the program that will be demoed. [6 marks]

5	Validation and Performance	Provide at least a paragraph or two explaining the performance of the system. A snapshot could be included and you could show test cases where you have tested that the system works reliably (e.g. using a time to compare if manual timing of turning one way and then the other looks correct to what the system reads this as, you might need additional debug messages for this). [20 marks]	This is a main aspect of the demonstration. See Validation Check Sheet in Appendix A. [40 marks]
6	Conclusion	Give a summary of the extent that the system was found to be successful. Discuss if you think that a system working in this way might be considered a potentially useful product. [10 marks]	End you demo with a short conclusion, reflect on the activity. [10 marks]
7	References	Provide a few references if relevant.	Questions [20 marks]
	TOTAL	100	100

Notice on Mark Penalties for the report:

Please make sure your assignment report provides the following information otherwise the indicated mark deductions may be applied:

- Make sure that you report clearly indicates the assignment name (i.e. Prac06 and/or Mini-Project A) ... otherwise you get a -5 marks penalty.
- Make sure that you put your name on the first page / cover page – as well as your prac partner if you are doing it as a team ... otherwise -5 marks penalty.
- If you are found not to have referenced the report appropriately you may be marked down (up to -15 depending on the severity of the issue. Please avoid plagiarism as this would be dealt with more seriously).

END OF ASSIGNMENT DESCRIPTION

EEE3096S Embedded Systems II
Mini-Project A
Appendix A: Validation Check Sheet

This elaborates part 6 of the Report and Demo marking schedule.

The tutors will use this sheet to validate your system and note its performance.

Procedure for validation checking: This forms part of ELO5.2. The performance that your system achieves may influence the mark you receive but it does not influence the ELO pass/fail – if your successfully demonstrated the desired aspect you will get a PASS; but if the aspect is lacking or not adequate a FAIL is given and you will be given another attempt (up to another two attempts) for which you can implement improvements and schedule a future demonstration to demonstrate the aspect.

EEE3096S Validation Check Sheet - 2018

STUDENTS: ____

Please provide at least first initial and last name including student number.

DATE: ____________

Attempt #1 Pass/Fail	Attempt #2 Pass/Fail	Attempt #3 Pass/Fail	Description	Max Marks	Max Marks
			Show startup of system (the embedded program executes)	2	
			Change to secure mode (might mean rebuilding the code)	2	
			Show how combination code has been stored/encoded (might need to show code)	3	
			Show lock operation is unsuccessful if wrong code is entered (i.e. dialed in)	5	
			Show lock operation is successful if right code is entered (i.e. dialed in)	5	
			Show unlock/lock unsuccessful if timing of dialed in combination obviously out	5	
N/A	N/A	N/A	Change to unsecure mode (might mean rebuilding the code) *Note:* not essential for ELO requirement	2	
			Show *sort()* assembly routine and briefly explain its implementation. Attempt to demonstrate that it operates. *Note:* is not essential to demonstrate this function is fully operational to pass the ELO but you need to explain it suitably.	8	
N/A	N/A	N/A	Show unsecure mode works (will get you marks but it is not necessary for passing the ELO)	8	
			TOTAL	40	

COMMENTS:

MARKER (please provide at least your initials) : __________