TEAM 10
Final Project
<Password protected door lock with alarm system>

110000167 郭光輝
110000265 張凱盛

A. Introduction

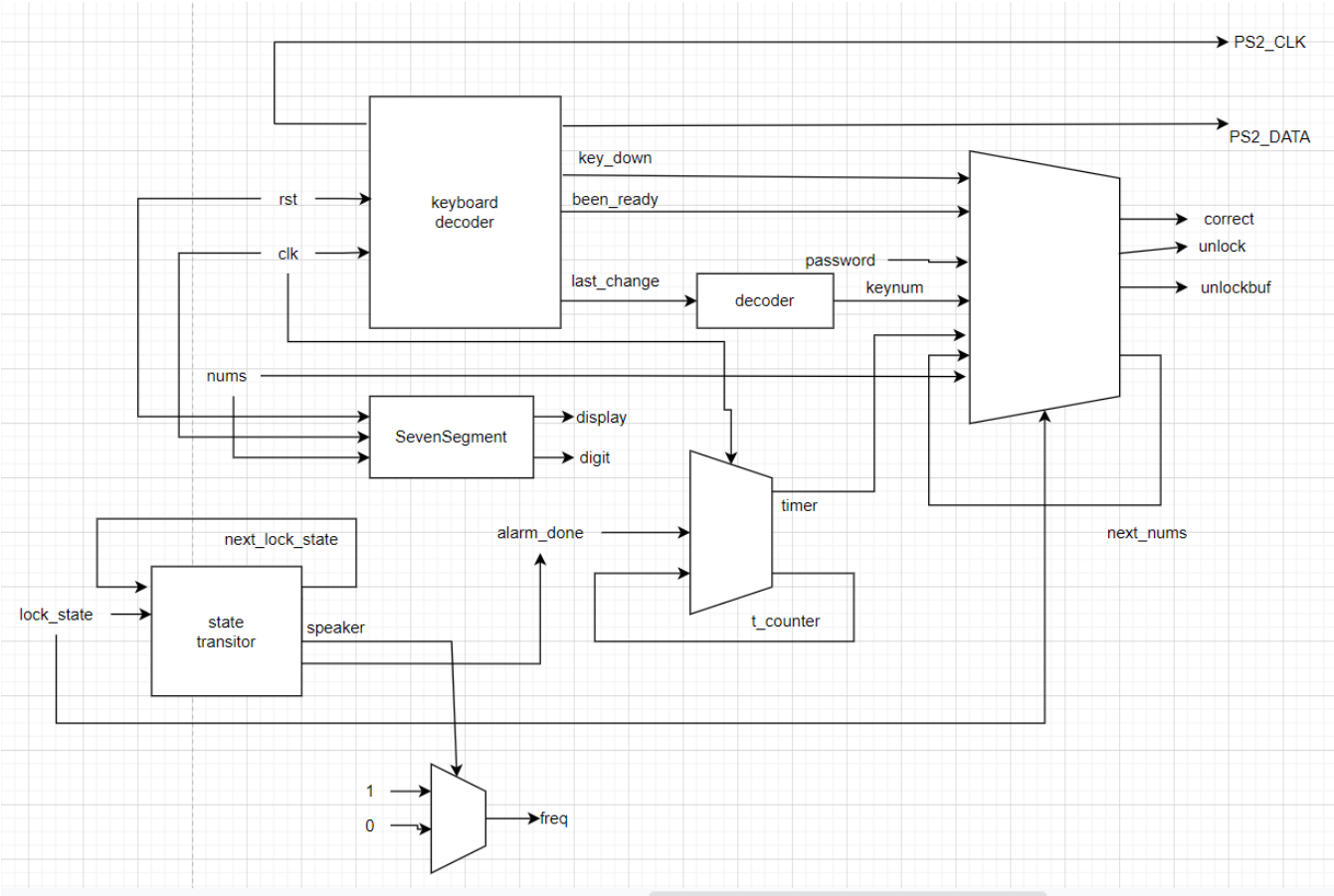
Nowadays, security is an important aspect of life. We might have never experienced it yet, but there will be 1 day when we will lose a personal belonging. We now have many inventions to protect our belongings. A popular way is to put them in a container, secure them with a lock, and make the lock password-protected. In this report, we will explain how we made a password-protected lock system.

B. Motivation

In this era, we should always expect the unexpected. One day we will have our personal belongings taken from us and we might not realize it. To prepare for such a situation and protect ourselves, we decided to implement a password-protected lock system. This will save us time from having to look for locks in stores, and since we have the source code, we can modify it to make the protection as strict as we want.

C. System Specification

- **Block diagram**



From the block diagram shown, when every positive edge clock is triggered, the status of lock_state and num will be updated. The lock_state will decide the states of the lock, while the num is the current digits shown for the keypad.

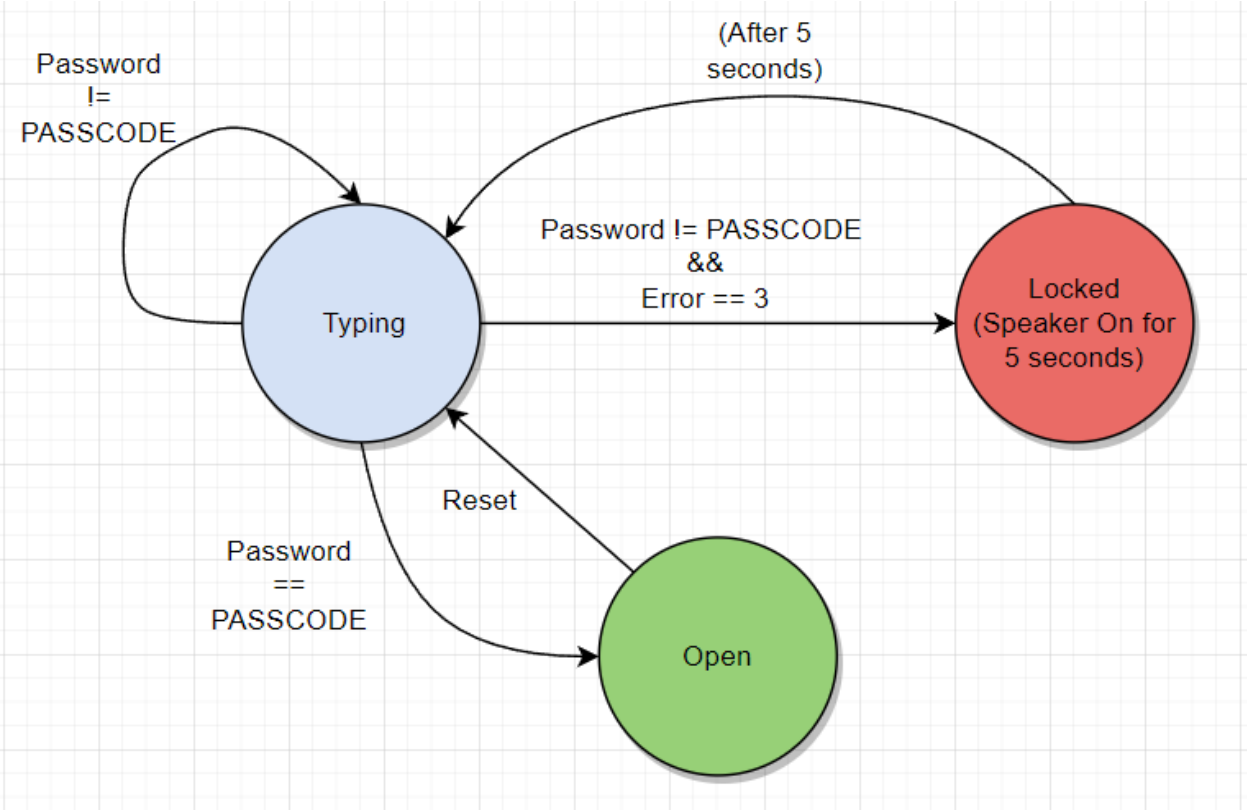
For the keypad part, we have a keyboard decoder which was similar to the basic lab that we had done before. Whenever there is a key being pressed, the key pressed will decide what will happen to the keypad digit. Keys like “Spacebar”, “Enter” and also the keypad for the keyboard are used to implement the password function. The “Spacebar” will reset all the digits shown on the seven-segment and the “Enter” will check the current digits to see if it is the same as the predefined PASSCODE, and last for the keypad, it is just for the numbers input. The numbers inputted will be updated every time into the “SevenSegment” module, so that the numbers pressed by the user will be shown on the seven segment displays.

For the current state of the lock, every time the lock_state is updated, the register variable “speaker” will be updated as well, when it is on 0, the alarm should be turned off and when it is

on 1 the alarm should be turned on. Once the alarm is turned on, it will be turned on until the register variable “timer” is changed to 1, once triggered it will then go to the next state of the lock_state.

The “timer” is being controlled by a MUX which will keep implementing by 1 until it reaches 5 seconds. But there is a variable called “alarm_done” which will always be 1 to make sure that the counter will stay 0 until the state of the lock is changed to “locked” state, then the “alarm_done” is changed to 0, then the counter will start working. By then, the state transition of this system will be completed.

- **State transition diagrams**



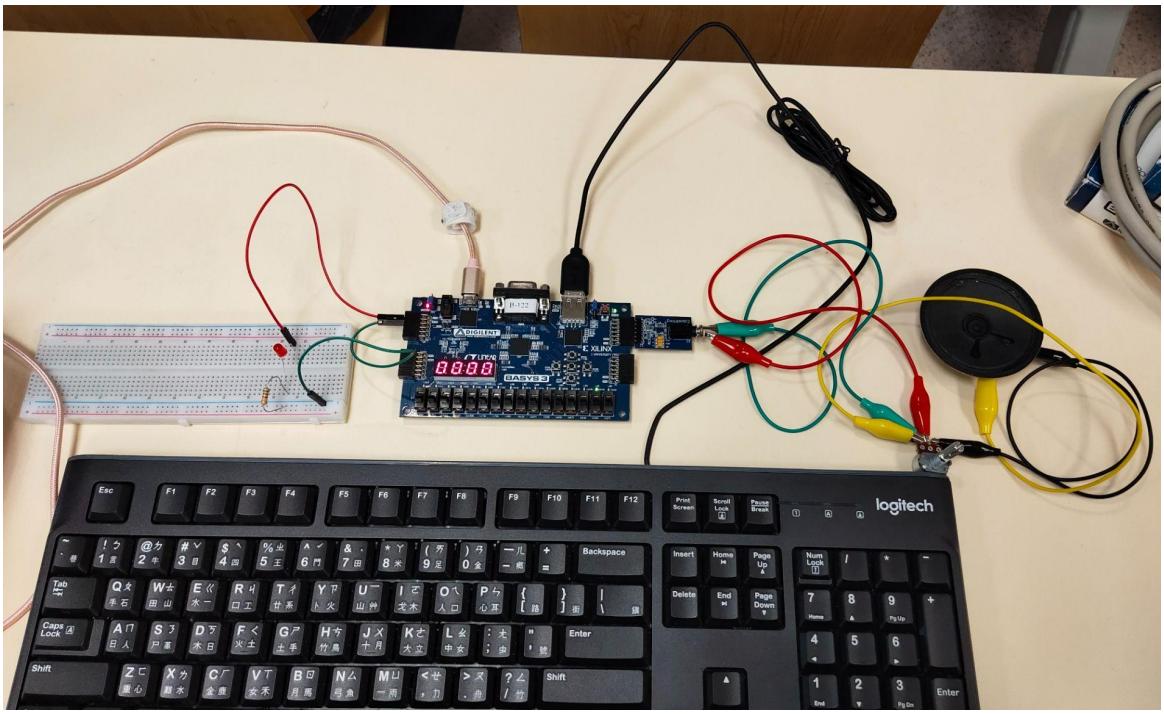
This is the state transition diagram for this project. We divided the whole operating structure into 3 parts, which are the typing state, the open state, and the lock state. When in the typing state, the user can insert any combination of 4-digit password, after clicking ‘ENTER’, it will check whether the password is the same as the predefined PASSCODE.

If the password is correct, it will then go to the open state, and the lock will be opened. Or else it will stay in the typing state if the error counts haven’t reached the trying limit, once the

error limit is reached, it will then go to the locked state and the alarm will be triggered for 5 seconds. During the alarm triggered period, nothing can be done, even with the reset key.

In the open state, whenever the reset key is pressed, it will go back to the typing state and the lock will be locked up again. For the locked state, after the alarm goes on for 5 seconds, then everything will be reset back to the initial typing state.

- **Result**



The final result is actually a little different compared to the proposal. The part of the solenoid we didn't manage to implement. We changed it to a LED to see whether there is a digital signal coming out from the FPGA board and it actually does have a signal coming out. We assumed that the solenoid is not working because it probably had some error with the circuit connecting on the breadboard. We were using IC4N35 and a MOSFET to power the output signal up because the solenoid needs 12V to be driven up but the board is giving 3.3V out only. Till the end, we couldn't figure out what's the problem with the circuit, so we decided to use LED to represent that we actually have a signal coming out from the board.

D. Code Explanation

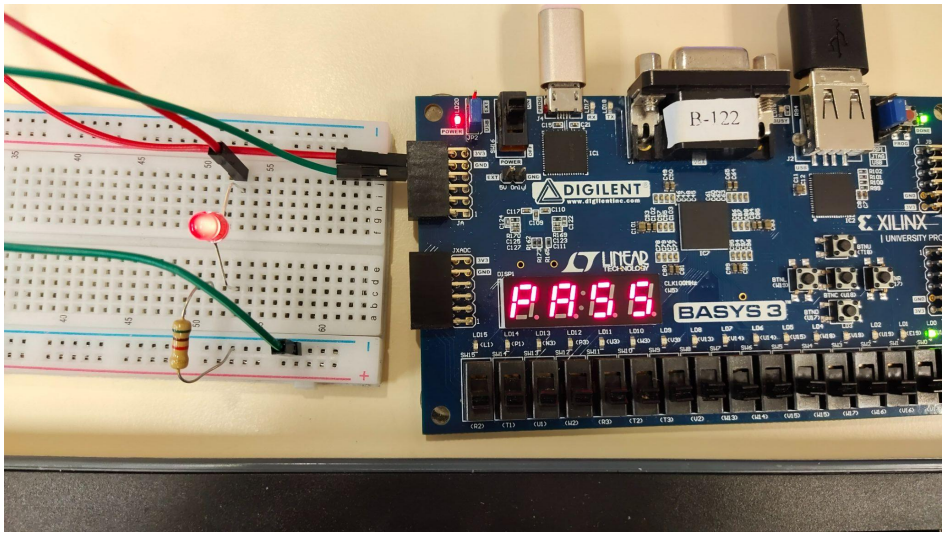
We reuse some blocks of code from the keyboard and audio basic labs. The keyboard will display the recently input digit to the right-most seven segment display and whenever a new digit is entered, the display bits are shifted. We use a variable "password" to check whether or not the 4 bits are the same as the defined passcode. The space bar acts as a reset button, so when pressed, all the seven segment displays will be reset to 0, and the state resets back to the default state.

For the states, we create separate states to indicate how many errors in a row the user has entered. The default state is "typing1" which means this is the 1st chance (no errors entered in a row). This changes to "error1" which indicates that there is 1 error. From here it changes to "typing2" after the space button is pressed, if the 2nd error occurs, it goes to "error2' and then to "typing3" when the space button is pressed. Finally, if the user enters the wrong code for the 3rd time, it enters the "locked" state that will trigger the alarm and disables all the keyboard functions until the alarm stops. When the alarm stops, it switches back to "typing1". If the user enters correctly from any typing state, it goes to the "open" state which sends the signal "unlockbuf" to tell that the input is correct. The signal is transferred to the red LED. The user can press the space button to close the lock, which in real life can also function as a lock function.

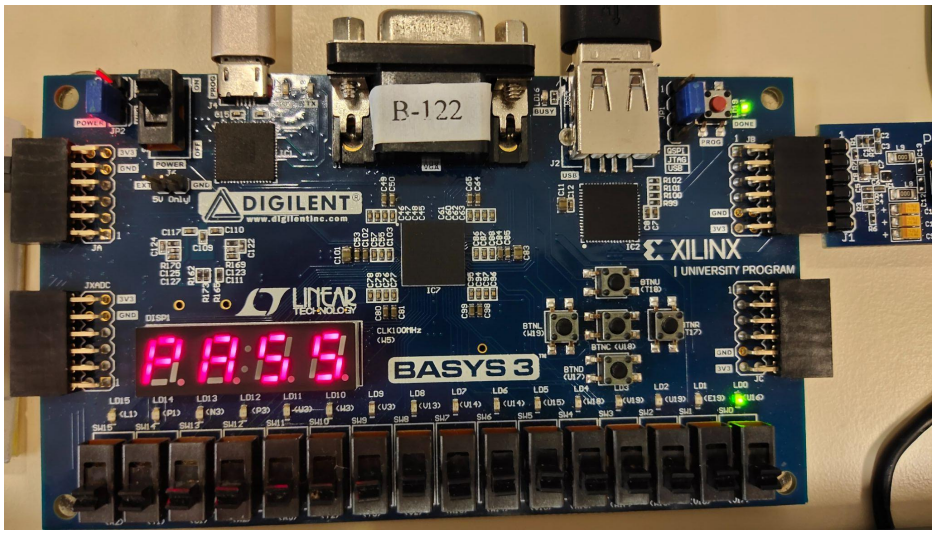
The alarm by default has a frequency of 0. It will change its frequency to 232 whenever the program enters the "locked" state. We create the variable "t_counter" to count to 5 seconds, during which period the alarm frequency should be 232. The variable "timer" will detect whether or not the alarm has rung for 5 seconds. Once it has, it changes values to make the program transition itself from an error state to the next typing state and also to reset the seven segment display to all 0-s, then resets its value when the 5-second period ends. We also created a variable "alarm_done" that will set itself to 1 whenever the program enters any typing state. It is used to reset "t_counter".

E. Experimental Results

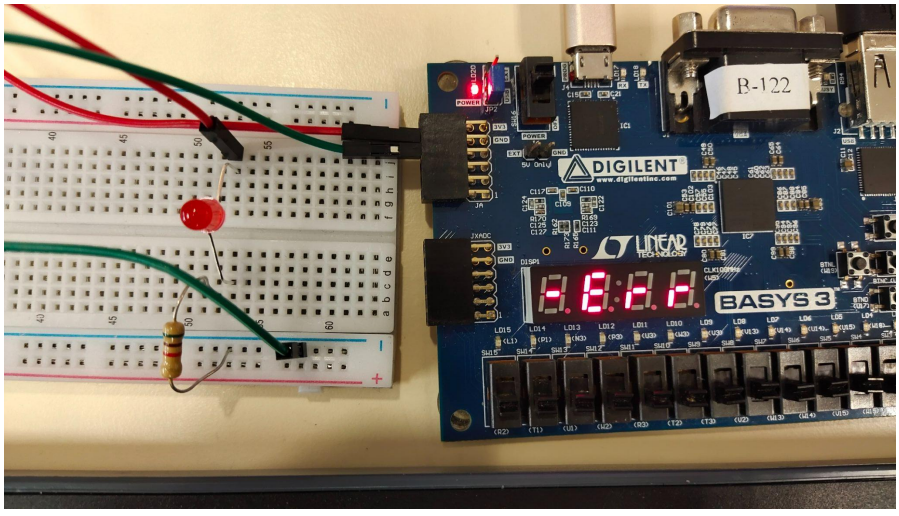
When the user types the correct passcode, the 7 segments will display "PASS" and the red LED light will be turned on, otherwise, it will display "-Err" which means error. If a user gets 3 errors in a row, the alarm will trigger for a few seconds. While the alarm is triggered, the user cannot input any codes and cannot use the reset button. After which, the program resets by itself.



The result when the entered passcode is correct



The LED on the bottom right indicates a signal is transferred when the passcode is correct



The result when the entered passcode is incorrect

F. Conclusion

The program can successfully retrieve the user's input from the keyboard and detect whether or not it matches the defined passcode. The program can also output a signal that detects whether the input passcode is correct and how many times the user enters the wrong code. The alarm rings exactly when the user enters the wrong code for the 3rd time and during which, the user cannot perform any action with the keyboard. We learned to utilize the state transition that we learned before, and tried to implement our own keyboard and alarm usage with the system. Unfortunately, we couldn't make the solenoid work, but in the end, it was a good experience to finish this project.