UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

CIPHER CODE SYSTEM FOR SECURING LOCKERS

DIGITAL SYSTEM DESIGN

Mureșan Claudiu **Stan Paul-Ioan**

Group: 30415

Advisor: Șuta Andrei

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Contents

1. Introduction

1.1 Project specifications

This project aims to design and implement a numeric locking system that enables users to secure a locker using a 3-character numeric code. The system is inspired by the types of locks commonly found in locker rooms at gyms, shopping malls, schools, swimming pools, and other public facilities, where simplicity, ease of use, and basic security are key requirements.

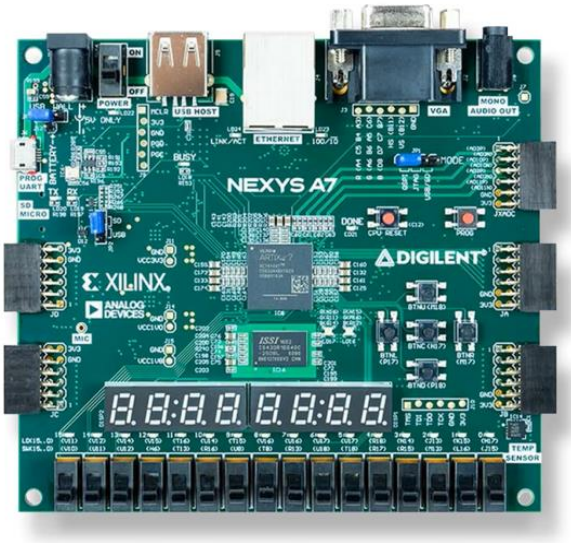
Functional Requirements

1. An indicator LED named **AVAILABLE** will signal the locker's status:
    - **OFF**: Locker is free
    - **ON**: Locker is occupied
2. The user presses the **ADD_DIGIT** button to begin entering the code. An **INPUT_DIGIT** LED will turn on to indicate the input state.
3. The user will enter three characters sequentially using the **UP** and **DOWN** buttons.
4. Allowed characters are in the range: **0–9** and **A–F**.
5. The currently selected character is displayed on a 7-segment display (SSD).
6. To proceed to the next character, the user presses the **ADD_DIGIT** button.
7. The previously entered character remains visible on the display.
8. The next character will be shown in the next position on the display.
9. After entering the third character, pressing the **ADD_DIGIT** button will:
    - Turn **OFF** the SSD display
    - Lock the locker by turning **ON** the **AVAILABLE** LED
10. The **INPUT_DIGIT** LED will turn **OFF**.
11. A **RESET** button/switch is available during code entry. Pressing it will reset the system to the initial state:
    - **AVAILABLE** LED turns **OFF**
    - The SSD display is cleared
    - **INPUT_DIGIT** LED turns **OFF**

12. To unlock the locker, the user presses the **ADD_DIGIT** button again to begin entering the unlock code.
13. Steps 2–8 are repeated for code entry.
14. After entering the third character, pressing the **ADD_DIGIT** button will trigger a verification step:
    - If the entered code matches the previously stored code:
        - **AVAILABLE** LED turns **OFF**
        - **INPUT_DIGIT** LED turns **OFF**
        - The SSD display is cleared
    - If the code does not match:
        - **INPUT_DIGIT** LED remains **ON**
        - **AVAILABLE** LED turns **OFF**
        - The SSD display is cleared

1.2 Software / Hardware

This project was developed using Vivado 2024.2, a powerful FPGA design suite by AMD-Xilinx. The logic was implemented in VHDL, a hardware description language suited for designing reliable, modular digital systems.



Digilent Nexys™ A7 board

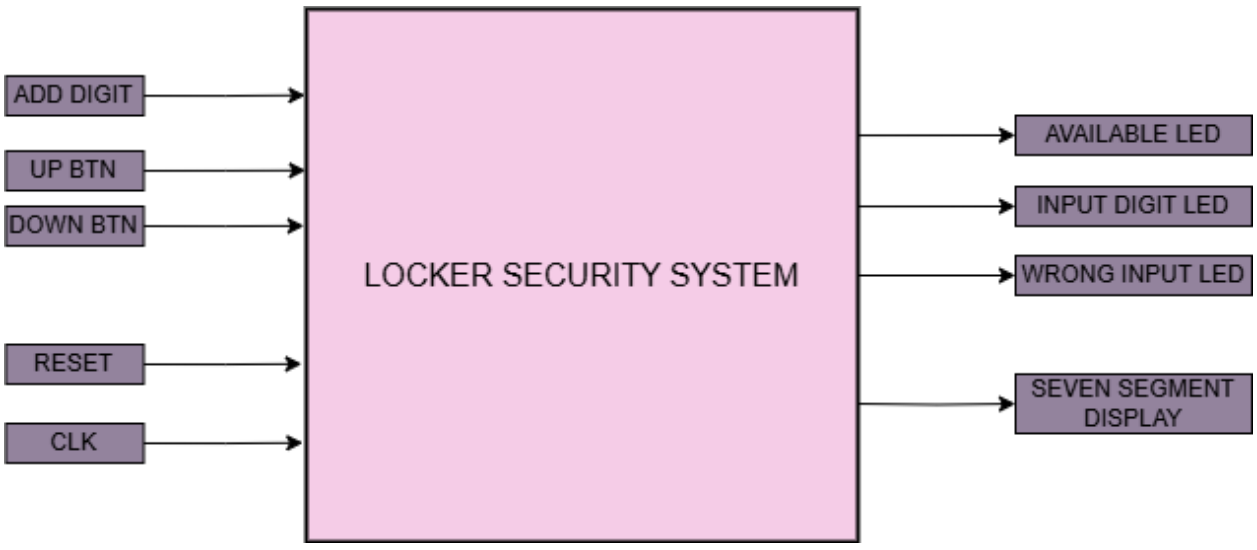
The target hardware was the Artix-7 XC7A100T FPGA. Vivado was used for simulation, synthesis, implementation, and bitstream generation. The design is easily adaptable to other FPGA boards by simply modifying the XDC constraints file to match different pin assignments and I/O configurations.

2. Schematics

2.1 Black Box

The initial step in our design process involved identifying the various input and output signals that our system would need to interface with. This was a critical phase, as understanding the signal flow is essential for defining the architecture and ensuring the system meets its intended functionality.

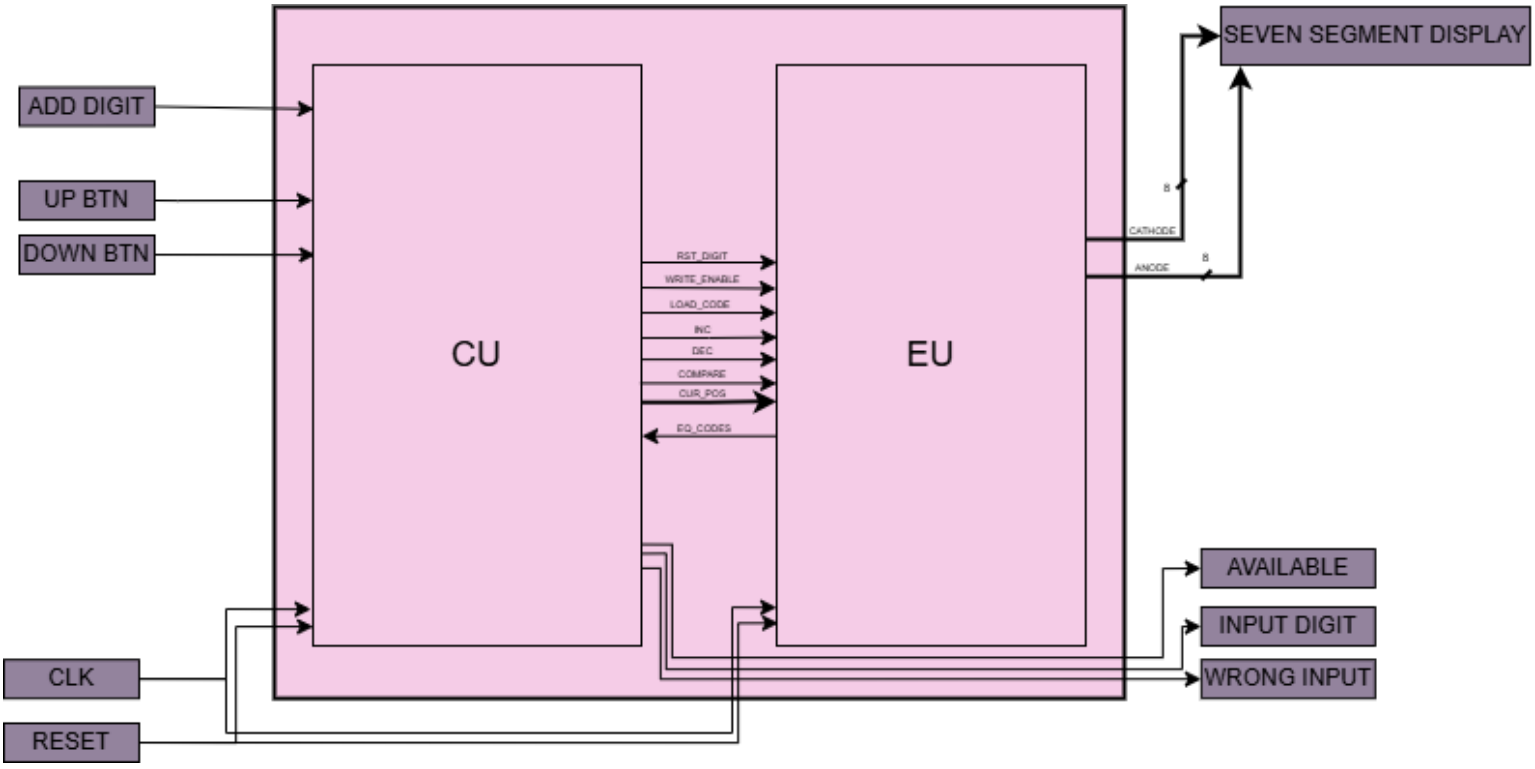
Following a thorough analysis of the functional requirements, we determined that the system requires a total of five input signals. Of these, there are four external input signals originating from the FPGA development board, specifically push-buttons and switches, while the fifth input signal is internal, coming from the development board itself, specifically, the FPGA clock signal. This clock input will serve as the system's timing reference, coordinating the operation of logic within the FPGA. For the output signals, four key categories have been identified. Three of these are discrete control signals used to indicate critical system states: one signals that the locker is available, another indicates that the system is awaiting code input, and the third flags an incorrect code entry. The fourth output is a composite group of signals responsible for driving the cathodes and anodes of the Seven-Segment Display (SSD), used for visual digit representation.



2.2 Control and Execution Unit

Following the preliminary black-box design and the identification of the system's primary inputs and outputs, the architecture is further refined by decomposing the system into two principal modules: the **Control Unit** (CU) and the **Execution Unit** (EU). The Control Unit is responsible for orchestrating the overall logic and operational sequencing of the locker system. The Execution Unit is tasked with handling the low-level operations and resource management required to effectively execute the system's functionalities.

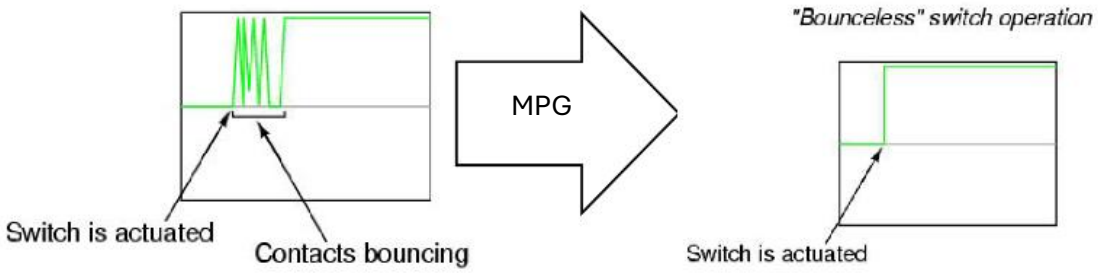
During this phase, we also identified the interconnecting signals between the Control Unit (CU) and the Execution Unit (EU). These signals serve as control and status interfaces, enabling synchronisation and coordination between the CU's logic control flow and the operational components within the EU. More information about these signals will be given in the next section.



2.3 Components

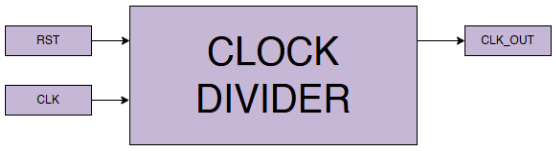
In this section, we will elaborate on the sub-components of the Execution Unit and other components used in our system, mainly: MPG, CLOCK_DIVIDER, SSD_MUX, CODE_REGISTER, and HEX_TO_SSD.

MPG – Due to mechanical issues and imperfections, the button of the FPGA board tends to bounce, giving multiple signals. This has a negative effect on our system because during specific operations, it can detect more than one input, thus giving an unwanted set of inputs and leading to an undesired result. For this reason, we use an MPG (Multi-pulse Generator) which can filter our multiple signals, giving one constant signal:



BUTTON – The signal that must be filtered
CLK – FPGA board clock (f = 100MHz)
CLEAN SIGNAL – The filtered signal

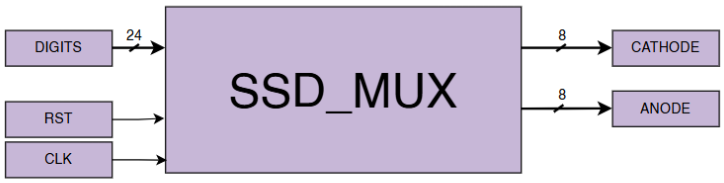
CLOCK_DIVIDER – Used to reduce the frequency of the FPGA board clock from 100 MHz to 140 Hz (an appropriate amount for the usage of 3 of the 8 SSD displays).



RST – Used to reset the clock divider to an initial state
CLK - FPGA board clock (f = 100MHz)
CLK_OUT – Divided clock signal (f = 140Hz)

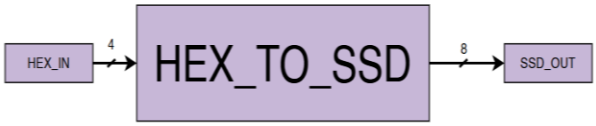
SSD_MUX – Used to display the introduced code on the Seven Segment Display (SSD). Since all 8 displays share a common cathode, showing different digits on each display requires quickly switching the active anode and updating the segment values. This

switch happens fast enough that the human eye perceives all digits as being shown at the same time.



RST – Used to reset the display and the starting position
CLK – Divided clock signal (f = 140Hz)
DIGITS – The 24-bit array stores three digits, each encoded in an 8-bit cathode format for SSD display.
CATHODE – Cathodes alternate across each set of 3 digits for multiplexing.
ANODE – Anodes cycle through each digit to enable multiplexing.

HEX_TO_SSD – Used to convert a 4-bit hexadecimal input into the corresponding 8-bit encoding required to display the character on a seven-segment display.
(Conversion table can be found in Annexe 1)



HEX_IN – Selected character encoded in 4-bit format
SSD_OUT – Selected character encoded in 8-bit format

CODE_REGISTER – A component designed for two main functions: first, to store both the locking code and the input (opening) code; second, to compare the two codes and determine if they match.

RST – Resets the display and sets the cursor to the starting position.
CLK – FPGA board clock signal (frequency = 100 MHz).
WRITE_ENABLE – Enables writing of CUR_CHAR at the position specified by CUR_POS.
LOAD_CODE – Triggers the saving of the entered code into the component.

CUR_POS – Indicates the current position where CUR_CHAR will be stored.
CUR_CHAR – The character selected for input.
SAVED_CODE – The stored code currently saved in the component.
CURRENT_CODE – The code currently being entered.
EQ_CODES – Signals that SAVED_CODE = CURRENT_CODE

2.4 Internal Signals

RST_DIGIT - Mapped from the Control Unit (CU) to the Execution Unit (EU), is used to reset the code register counter responsible for selecting the digit to be displayed. It is set to 1 whenever the **ADD_DIGIT** button is pressed and the system transitions from selecting one digit to another, passing through an intermediary reset state (such as **RST_X** or **RST_U_X**). In all other cases, the signal remains at 0.

WRITE_ENABLE - Mapped from the Control Unit (CU) to the Execution Unit (EU), is used to enable the writing of each digit into a temporary memory. It is set to 1 starting from the **ADD_1** state and remains active until the system reaches the **LOAD** state, where it is set to 0. It is then set to 1 again when the system enters the **UNLOCK_1** state and remains active until the **CHECK** state, after which it is deactivated.

LOAD_CODE - Mapped from the Control Unit (CU) to the Execution Unit (EU), is used to load the initial code entered from temporary memory into a permanent memory that retains its value until the locker is opened again, or the system is reset. It is set to 1 during the **LOAD** state and set to 0 immediately afterwards.

INC | DEC - Mapped from the Control Unit (CU) to the Execution Unit (EU), these signals have the purpose of transmitting further the input from the UP_BTN and DOWN_BTN, respectively. In the EU, these signals control the incrementing or decrementing of the counter associated with navigating through the digits the user intends to select.

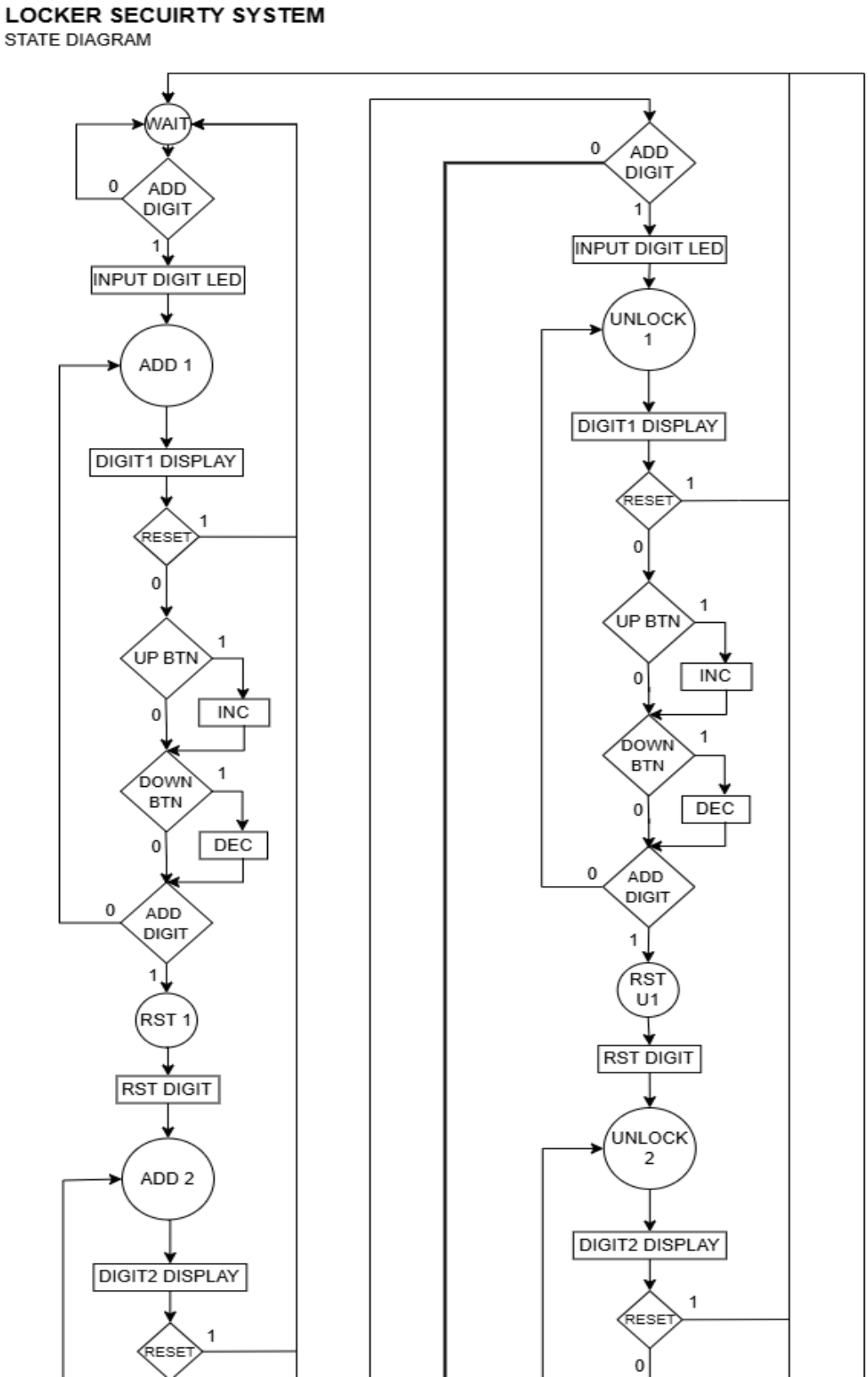
COMPARE - Mapped from the Control Unit (CU) to the Execution Unit (EU), is used to instruct the EU to perform a comparison between the initially stored code and the code entered during the unlocking process. It is set to 1 only during the **CHECK** state and remains 0 in all other states.

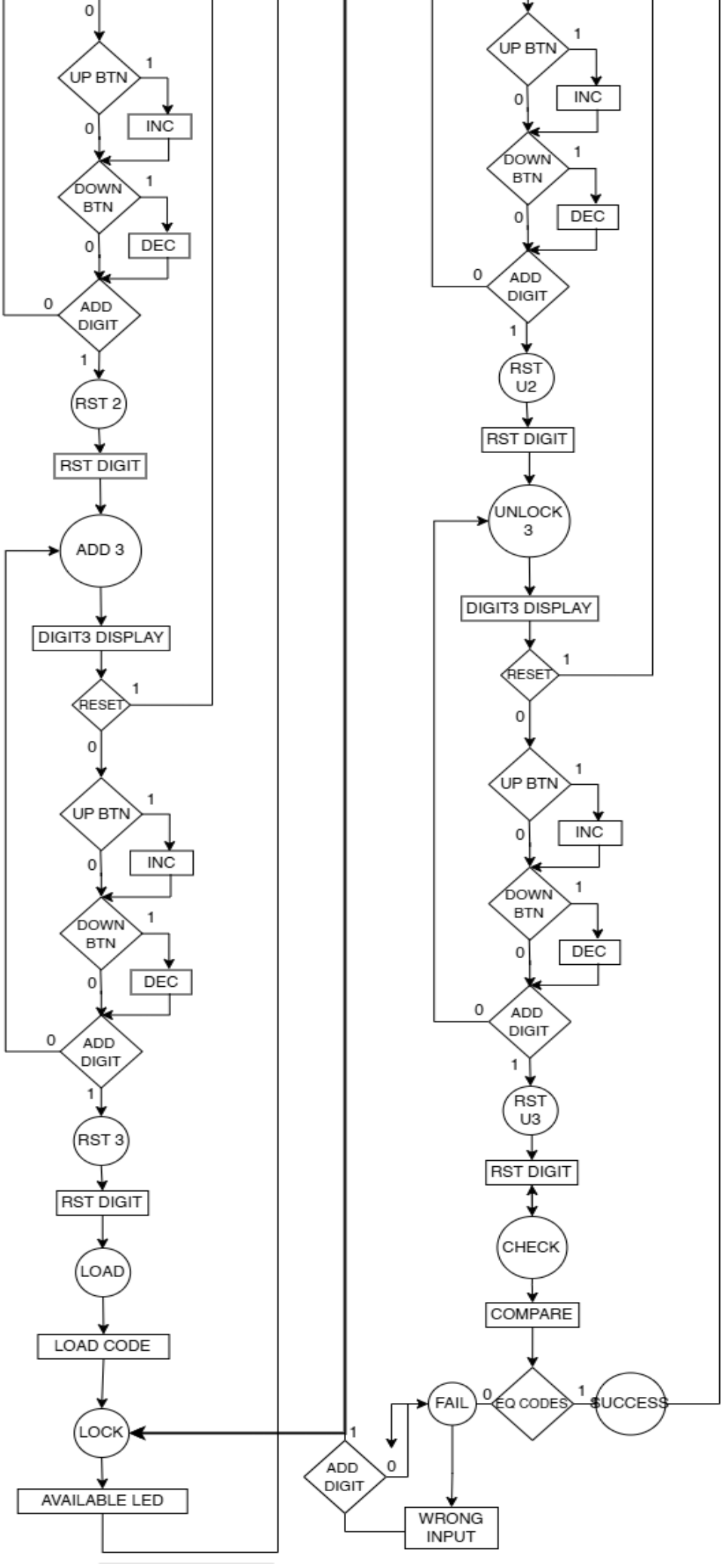
CUR_POS - Mapped from the Control Unit (CU) to the Execution Unit (EU), is a signal on 2 bits meant to indicate which temporary memory should save the digit introduced. It takes value 00 for states **ADD_1 | UNLOCK_1**, 01 for states **ADD_1 | UNLOCK_1**, 10 for states **ADD_1 | UNLOCK_1.**

EQ_CODES - Mapped from the Execution Unit (EU) to the Control Unit (CU), it has the purpose to signal the CU that the 2 codes are equal/not equal and move to the appropriate next state. If the value of the signal is '1', then the next state is **SUCCESS,** signalling the successful opening of the locker, otherwise, the next state is **FAIL** signalling that the code introduced by the user is not correct.

2.5 State Diagram

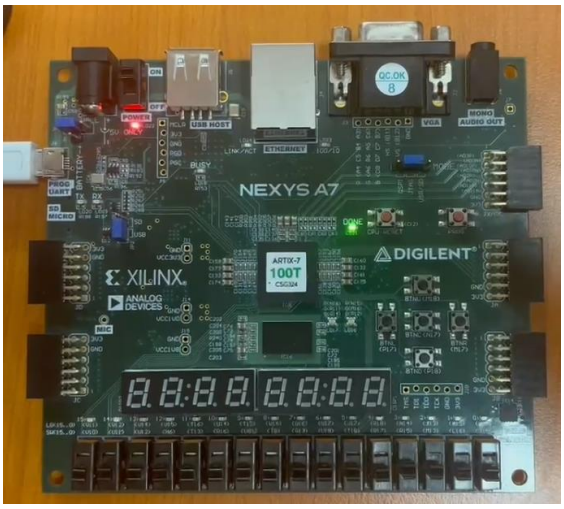
Below is a state diagram that illustrates the operational flow of our system. It outlines the various states the system can occupy, the transitions between them, and the conditions or inputs that trigger those transitions, providing a clear overview of the control logic and behavior throughout the code entry and validation process.



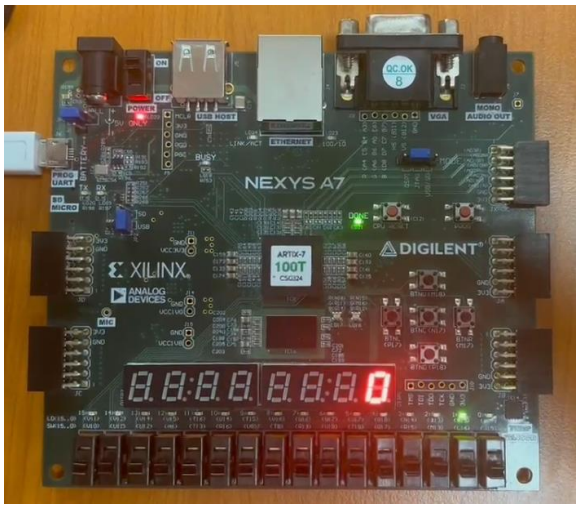

UP BTN
INC
DOWN BTN
DEC
ADD DIGIT
RST 2
RST DIGIT
ADD 3
DIGIT3 DISPLAY
RESET
UP BTN
INC
DOWN BTN
DEC
ADD DIGIT
RST 3
RST DIGIT
LOAD
LOAD CODE
LOCK
AVAILABLE LED
UP BTN
INC
DOWN BTN
DEC
ADD DIGIT
RST U2
RST DIGIT
UNLOCK 3
DIGIT3 DISPLAY
RESET
UP BTN
INC
DOWN BTN
DEC
ADD DIGIT
RST U3
RST DIGIT
CHECK
COMPARE
EQ CODES
FAIL
SUCCESS
ADD DIGIT
WRONG INPUT

3. User manual

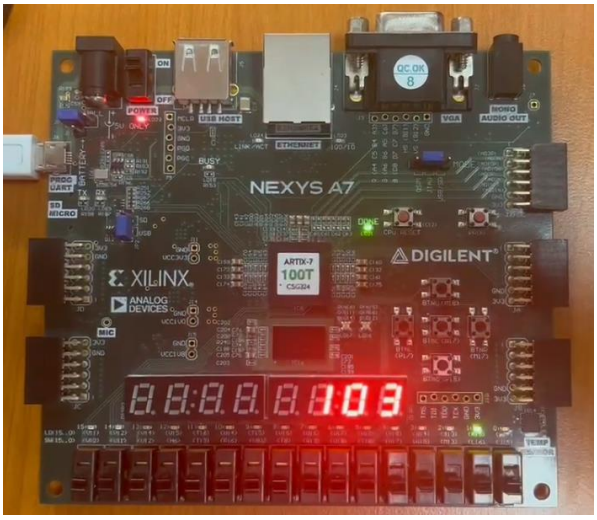
1. The user is expected to find the system in an **idle state**, waiting for input. This state is characterized by an **empty display** and **all LEDs turned off**.
2. Press the ADD_DIGIT button to **activate** the system. The display will show the leftmost digit. At the same time, an LED that indicates to the user that a digit should be introduced is **activated.** In this state, the user can choose the digit using the UP and DOWN buttons, the pressing of these buttons having **immediate action on the display**. Once the user is satisfied with the digit they chose, they must **press** the ADD_DIGIT button, and the system will **move to the next digit**. This process is **repeated until all three digits have been introduced**.
3. After selecting the third and final digit, **press** the ADD_DIGIT button. The system will then enter the locked state. At this point, the locker is **locked** and an LED that indicates this will become **active**. The LED stays active **until the locker is later unlocked**. At the same time, the LED, which was activated when the user began the introduction of the code, will be **turned off**, and the **display will be cleared**.
4. To **unlock** the locker, **press** the ADD_DIGIT button again. The introduction of the code is **almost identical** to the one described before, but this time, **besides** the LED that indicates the introduction of digits, the LED that indicates that the system is locked is also **active**. Same as before, if the user is satisfied with the 3rd digit, **pressing** the ADD_DIGIT button will **confirm the code,** which is **compared** to the one previously introduced. If both are **identical**, the system is **unlocked** (the locked LED is **turned off**), and it **comes back** to the idle state. Otherwise, the locked LED continues to be **active**, and, besides that, an LED which signals that the code was wrong becomes active too. To bring the system **back** to the locked state, the user should **press** the ADD_DIGIT button once again.
5. At any point during these steps, should the user activate the RST reset switch, the system will automatically be brought back to the idle state.



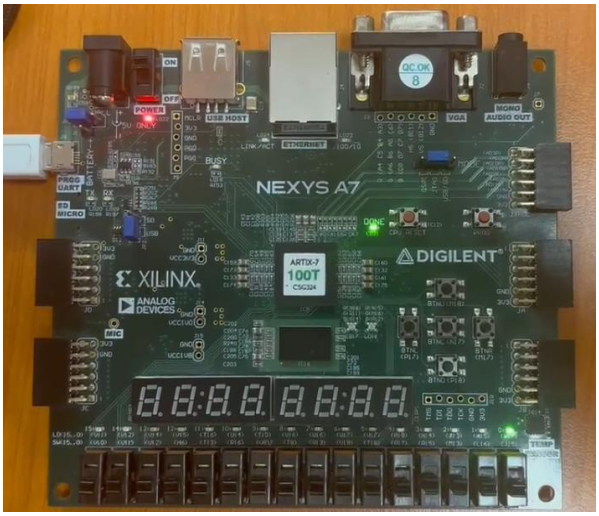
IDLE STATE



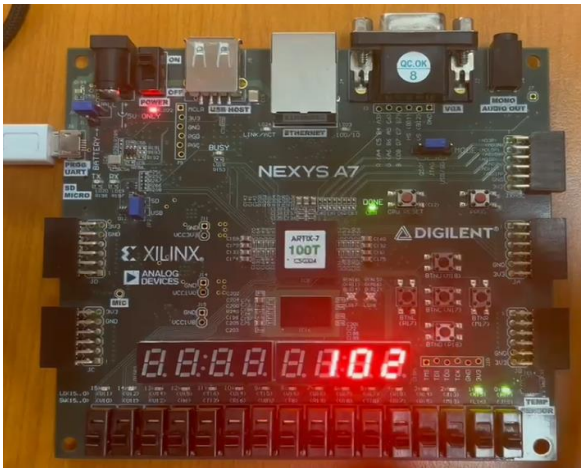
ADD_DIGIT was pressed first time
Rightmost digit is displayed
INPUT_DIGIT LED lights up



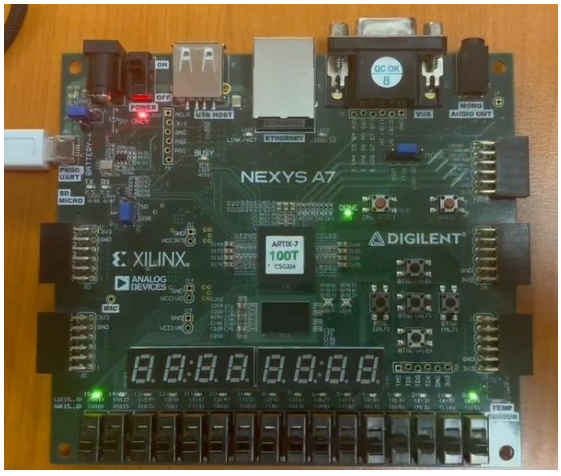
ADD_3 STATE
ADD_DIGIT was pressed 3 times



ADD_DIGIT was pressed fourth time
SSD display is disabled
INPUT_DIGIT LED turns off
AVAILABLE LED turns on



Incorrect code is introduced



WRONG_INPUT LED turns on

4. Method justification

At first glance, the system might appear challenging, but looking at the state diagram, one could understand why we picked such a linear approach. This way, the program and the algorithms are easy to follow due to their modularity, which, together with the state-machine-based design, provides a clean, simple, and efficient way to control a 3-digit locker system. Using this design, every component can be tested individually, making it easier to find errors on a local level and to understand what every component does. The result is an efficient locker system that responds well to the user's input and has an easy-to-understand interface.

All the components of the system are linked in a structural, top-down way, making it very useful if they are to be integrated into other systems later.

5. Further developments

Future improvements to the 3-character cipher system could enhance both functionality and user experience. One key development would be to display status messages on the Seven Segment Display (SSD), such as "LOC" (Locked), "OPN" (Open), "WRG" (Wrong), or "COR" (Correct), depending on the locker's state and the correctness of the entered code. Additionally, refining and organizing the code for improved readability, modularity, and efficiency would make the system more maintainable. Other possible enhancements include extending the cipher length to increase security, adding real-time user feedback through LEDs or sound signals, and implementing a keypad interface for easier and more intuitive input. These developments would move the project closer to a fully practical and user-friendly embedded security system.

6. Annexes

6.1 Annex 1

CHAR	4-BIT BINARY ENCODING	SSD CATHODE ENCODING	HEXADECIMAL ENCODING
0	b0000	b1100 0000	xC0
1	b0001	b1111 1001	xF9
2	b0010	b1010 0100	xA4
3	b0011	b1011 0000	xB0
4	b0100	b1001 1001	x99
5	b0101	b1001 0010	x92
6	b0110	b1000 0010	x82
7	b0111	b1111 1000	xF8
8	b1000	b1000 0000	x80
9	b1001	b1001 0000	x90
A	b1010	b1000 1000	x88
B	b1011	b1000 0011	x83
C	b1100	b1100 0110	xC6
D	b1101	b1010 0001	xA1
E	b1110	b1000 0110	x86
F	b1111	b1000 1110	x8E

6.2 Annex 2