FH Joanneum

Graz

Model Based Design

Numeric Sequence Lock

Training Unit 04

Authors
David B. Heer
Jakob Soukup
Graz, January 8, 2019

Lecturer
Alfred Steinhuber

Contents

Introduction

The task was to implement an Finit State Machine in Simulink. Afterwards this Finit State Machine is to be tested for its requirements. Finally code should be generated and loaded onto the Arduino board.

1 Requirements

1.1 Order of Priorities

1. **Unlocking**
   This is the core of the task and so the most important requirement.
2. **Locking**
   It is necessary to return to the locked state.
3. **Sampling**
   With a sampletime of 10ms you do not have to debounce the inputs. In this case is sampling important.
4. **Wrong Sequence**
   This requirement makes it difficult to crack the code using the brute force method.
5. **Input Handling**
   This is a nice feature and can be useful if you have mistyped.
6. **State**
   It would be a nice feature to see the current state. But it is not essential.
7. **Keypad**
   It is absolutely irrelevant whether the numbers are compared as integers between 0-9 or as ACII integers between 48 - 57.

The Voltage Monitoring is an own requirement. Because it has no connection with the remaining requirements.

1.2 Funcional and non functional requirements

Functional **[TODO - Sort]**

- Unlockin
- Wrong Sequence
- Input Handling

- State
- Sampling
- Locking
- Keypad
- Voltage Monitoring

Non functional

- sdgs

1.3 Missing requirements

hange Sequence It would be useful if you could change the sequence while the program is running and not just when the program is freshly loaded onto the arduino board.

label description

1.4 Stateflow implementation

Based on the previously defined requirements the state machine of the numeric sequence lock shall be implemented in Simulink using stateflow. This task begins with defining the needed cases and then working on the transitions between said cases.

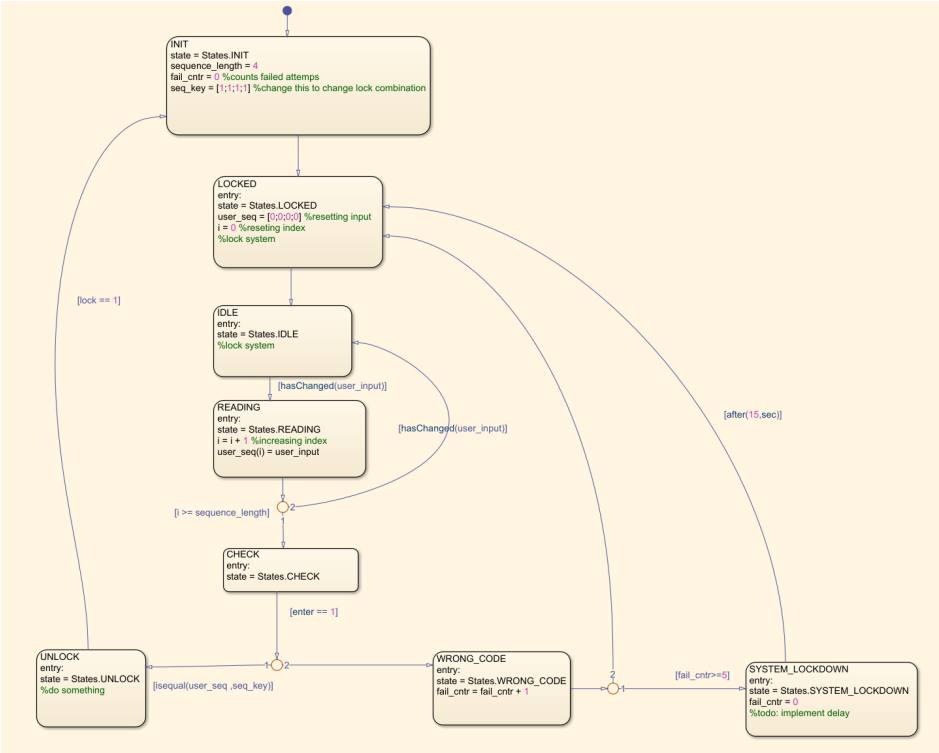

Figure 1: *stateflow implementation of the state machine*

1.4.1 Statemachine detailed

In order to further understand the working of the models, each case will be briefly explained and reviewed.

INIT
As the name implies, this state sets the initial condition for the lock. The fail counter is reset to zero and other initial settings such as the sequence length and the sequence key are set in here. The Transition to the next state is made immediately after setting all parameters, without any conditions.

LOCKED
The system gets locked and the buffer for the user input is reset. The index i is used to keep track of how many digits were already entered. Again, the transition to the next state is done immediately after finishing all tasks without any conditions.

IDLE
This state is used to wait for user inputs, other than that it serves no purpose. After a user input was detected, the transition to the READING state is made.

READING
Here the detected input is stored in the previously declared buffer and the index is increased. If the index exceeds the sequence length, the state machine transitions to the CHECK state, if not, and the user input changes again (button was released) it moves back to the IDLE state and waits for the next digit.

CHECK
In this state the machine waits for the user to activate the enter key to confirm the input. After receiving it the sequence is compared to the key and if it matches the machine moves to the UNLOCK state. If the sequence is wrong, it moves on to the WRONG_CODE state.

UNLOCK
This is the state in which, if there was an actual locking hardware attached, the system unlocks itself. After completing the unlocking phase, it moves back to the INIT state to reset the system.

WRONG_CODE
Here the fail counter is increased and if it exceeds the allowed fail attempts, it moves to the LOCKDOWN state. If the limit Is not yet reached the machine jumps back into the LOCKED state in order to allow another try.

LOCKDOWN
The system waits for 15 seconds before returning into the LOCKED state. In addition, the fail counter is reset.

1.4.2 Input translation

Since the Arduino later will receive ascii values for the available some sort if translation to the system must be implemented in order to feed the statemachine the correct inputs. This was done by creating a simple subsystem in Simulink.

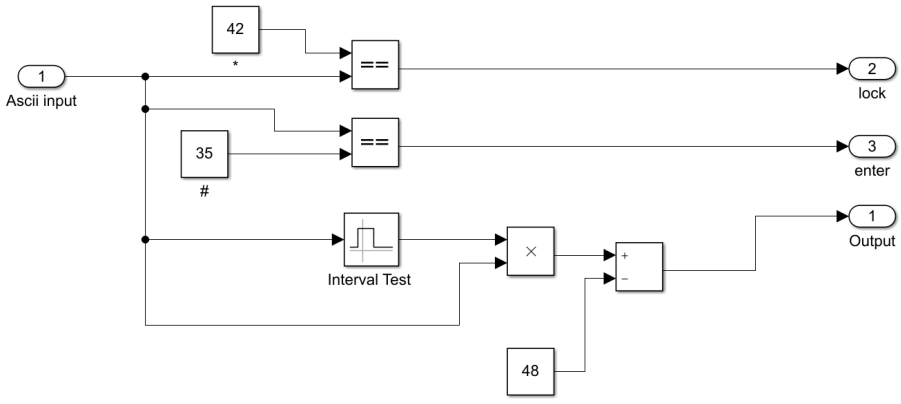

Figure 2: *check input subsystem in Simulink*

The different inputs are filtered by their ascii value and fed into their outputs. The star an the hashtag are rather obvious. The interval filters for the ascii value of all 10 digits and the resets them to values from zero to nine.

1.4.3 Testing

In order to test the system without its hardware a constant was attached to the input of the check input subsystem. The constant got linked to multiple push buttons which made testing the functionality possible.

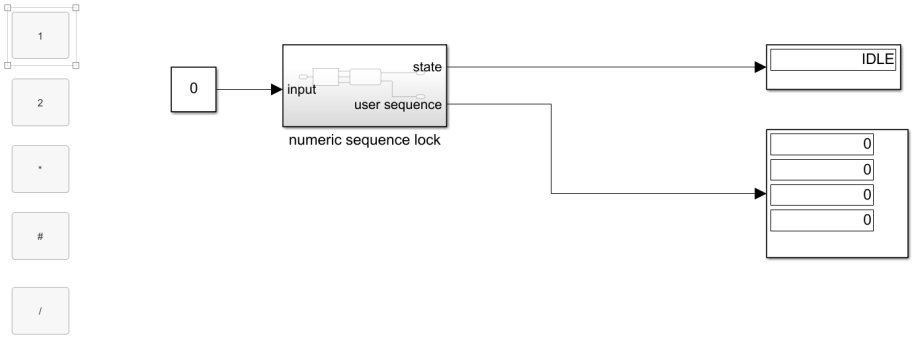

Figure 3: *Test structure of the sequence lock*

All possible cases where tested and the system held up to the expectations. If the correct sequence was entered and the hashtag was pressed the system unlocked itself. The input of the '/' character was ignored. After then pressing the start button the system was locked again. If the wrong sequence was entered the system returned into the IDLE state. If the wrong code was entered too many times the LOCKDOWN state was entered. However, the time the system spent in this state was way too short due to Simulink not simulating in real time.

1.5 Arduino implementation

After having a working model of the sequence lock the next step was to transfer the software onto the actual hardware, this time being an Arduino UNO with a numerical pad attached. Simulink shall automatically generate the required C code. For that purpose, the numerical keypad got attached to the Arduino in the following manner:

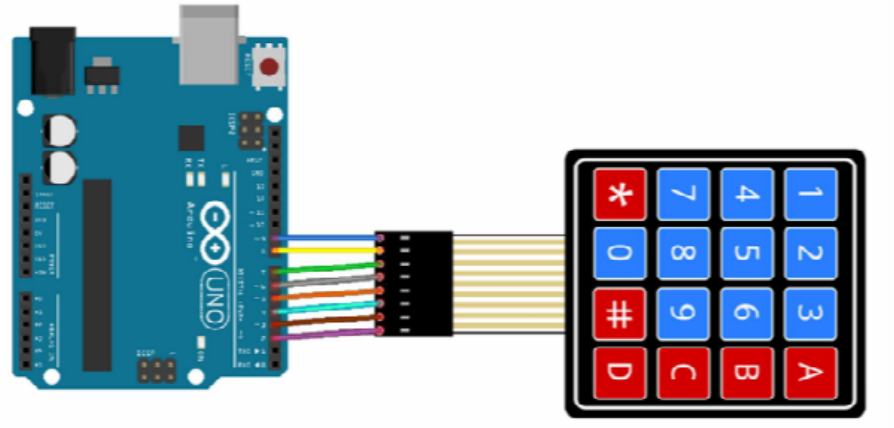

Figure 4: *pin assignment (Source: ECE MBD: Laboratory session)*

The given template for the keypad was included into the Simulink model and the settings were adjusted in order to motivate Simulink to deploy the code onto the Arduino.

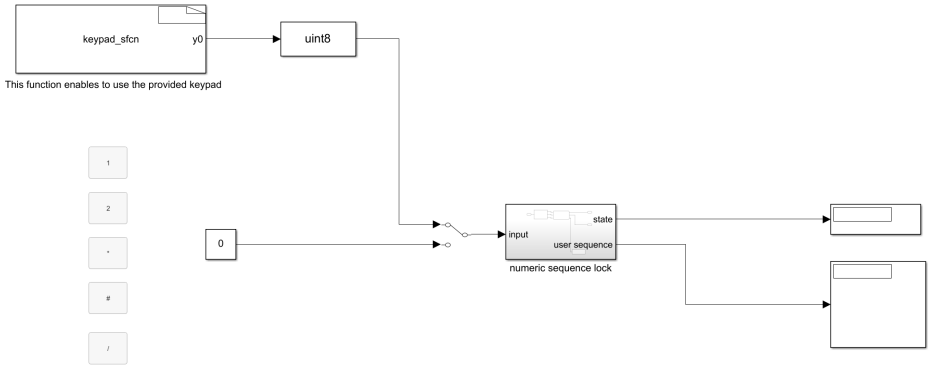

Figure 5: *Implementation in simulink for arduino*

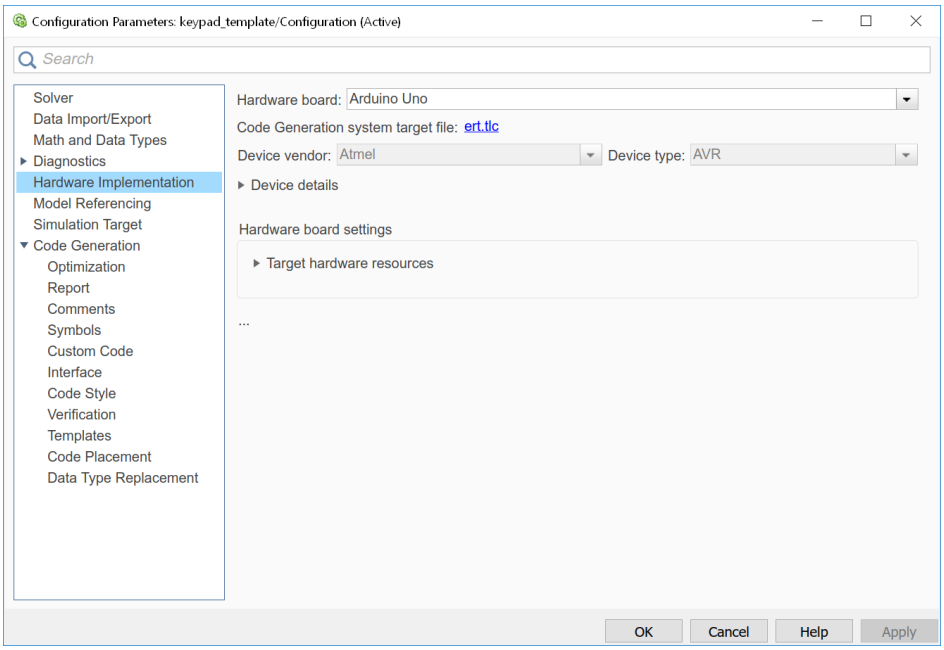

Figure 6: *simulink settings*

In addition, the compile mode has to be set to external. After that the model was deployed on the Arduino and was ready for testing. In order to check the correct working behavior, the same procedure as before was used. Only this time the buttons on the keypad where pressed as opposed to the ones in the Simulink model.

1.6 Voltage Monitoring

2 Conclusion

List of Figures