
# Lab 10

R. Ferrero, P. Bernardi,

A. C. Marceddu

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy



# Exercise 1: Fibonacci sequence

- Create a program which shows Fibonacci sequence (up to number 8) by using LEDs.
- By pressing button KEY1, power off the current led and power on the led corresponding to the previous number of the Fibonacci sequence.
- By pressing button KEY2, power off the current led and power on the led corresponding to the next number of the Fibonacci sequence.

# Implementation

- Use the following mapping: LED 4 = 1, LED 5 = 2, …, LED 11 = 8.
- At the first run, the program must start from number 0, which corresponds to the configuration in which all the LEDs are off.

# Exercise 2: Simon Game

- Implement Simon game.
- Using KEY1, KEY2, INT0 buttons, the player has to repeat a random sequence, displayed using LEDs 4, 5, 6.
- If the sequence inserted by the player is the same as the one to emulate, the player wins. At this point the game repeats, increasing by 1 the length of the sequence.
- Otherwise the player loses, and the game starts again from sequence length 1.

# Sequence shown by LEDs

- To show a sequence of length `n`, the timer has to trigger `n`*2 interrupts, with an interval of 1,5 s.
- In interrupts 1, 3, 5, ..., `n`*2 – 1 (odd), the interrupt handler of the timer generates a random number between 0 and 2 and powers on the corresponding LED: 0 -> LED 4, 1 -> LED 5, 2 -> LED 6.
- In interrupts 2, 4, 6, ..., `n`*2 (even) the interrupt handler powers off the only lit LED.

# Random Number Generation

- To obtain random numbers, you can check the value of the timer when a button is pressed.
- As the user presses buttons one after the other, an array can be used to store the random numbers generated on each push of the button. Remove the values from the array when you use them.
- Computing the modulo 3 of timer counter, it is possible to obtain a random number between 0 and 2.

# Outcome of the game

- If the player has pressed the buttons in the correct sequence, the 4-11 LEDs power on according to the binary number `n` to be displayed (LED 11 is least significant bit). The game starts again with `n` = `n` + 1 as soon as the player pushes the button.
- If instead the `i`-th button is input wrong, on the 4-11 LEDs the `i` value is displayed. The game starts again with `n` = 1 as soon as the player pushes the button.

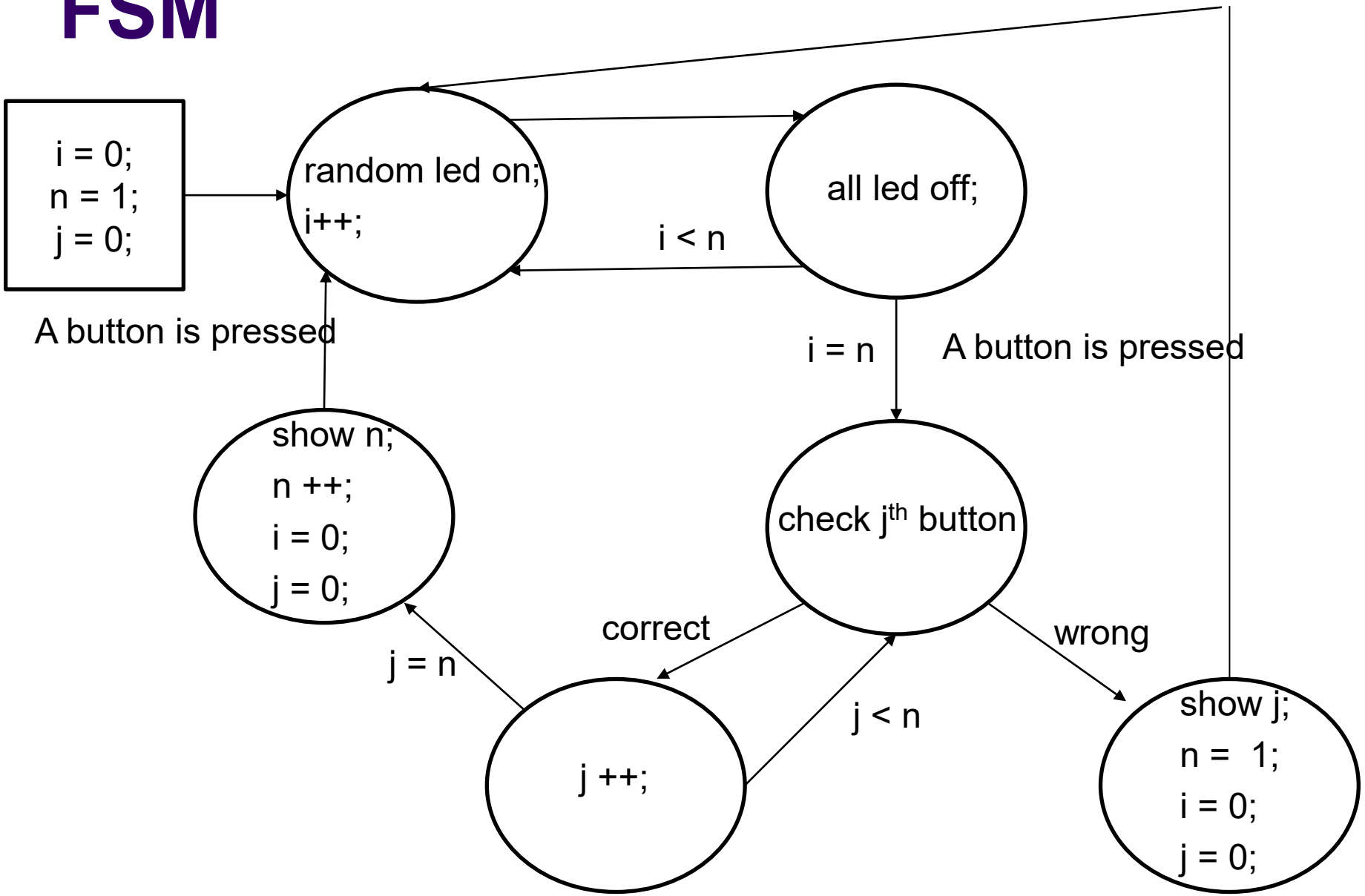
# Suggestions

- To show the binary number `x` on LEDs 4-11, it is sufficient to assign to the low 8 bits of LPC_GPIO2->FIOPIN the value `x`.
- The handler of the timer has to manage their powering on with a random sequence and store that sequence in a statically allocated array (max 256 elements).
- The handler of the buttons has to verify that the pressed button corresponds to the `i`-th element of the sequence stored in memory.

# Suggestions (cont.)

- It is recommended to use a global variable, shared among handlers of timers and buttons, in order to keep track of the current situation of the game.
- A Finite State Machine (FSM) is shown in the next slide.

# FSM


i = 0;
n = 1;
j = 0;
random led on;
i++;
all led off;
i < n
A button is pressed
i = n
A button is pressed
show n;
n ++;
i = 0;
j = 0;
check j^th button
correct
wrong
j = n
j < n
j ++;
show j;
n = 1;
i = 0;
j = 0;