Lab 10: Simon Game

R. Ferrero, M. Russo

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

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`, one timer has to trigger `n`*2 interrupts, with an interval of 1,5 s (timer frequency = 25 MHz, i.e. 25M count/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 a **second timer** timer when a button is pressed, disabling the timer interrupt (why?).
- 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.

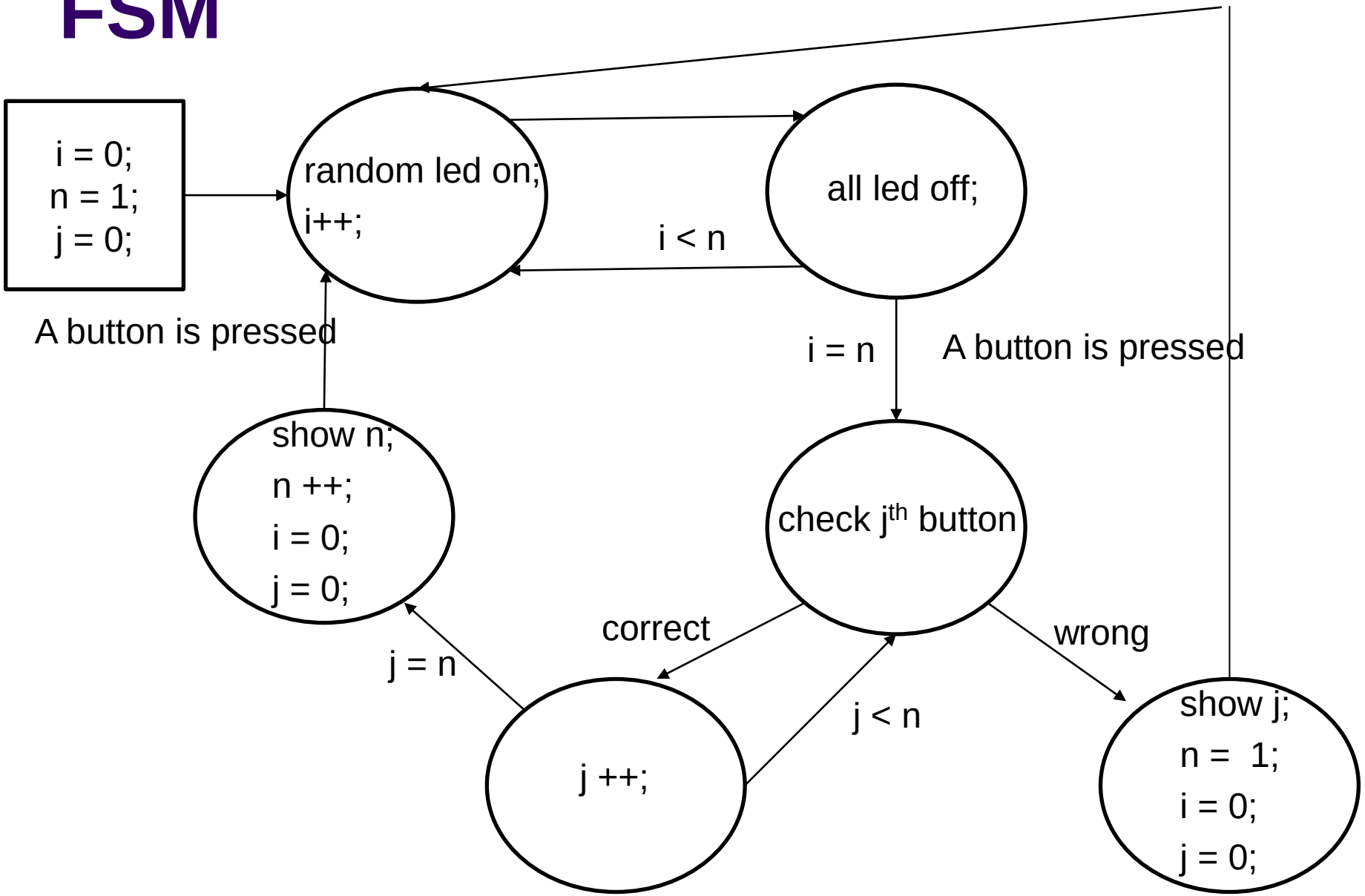
Suggestions

- 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.
  - n=sequence length
  - i=number of LEDs displayed so far
  - j=number of correct buttons pressed so far

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;

More about random number generation

- You will see that initializing timer 0 with a certain value makes us generate always the same numbers (after all, the IRQ of timer 0 is always called at the same point)
- To solve this problem, in the main you could use the current seconds as a seed to initiate timer 0
- LPC_RTC->SEC keeps track of «real world» seconds (the correct time is not calibrated, but we don’t need it here). Remember to modulo 0xFFFFFFFF

Bouncing

- When pressing a button, it could be detected as multiple presses (see page 47 of 13_LPC1768_button.pdf)
- This results in multiple interrupts for just one pressed button: the handler is mistakenly run multiple times
- Button interrupt handlers have to run their code only if 0.2 seconds have passed since the last interrupt (use a third timer without interrupts)

Recap about timers

Useful registers for each timer:

- TC: counts clock cycles %0xFFFFFFFF (**you might want to read its value)**
- MRi: 4 match registers MR continuously compared to TC (**MR0 is set in init_timer depending on the 2nd argument**)
- MCR: 3 bits for each MRi, for setting up what happens when TC=MRi **(has to be manually set)**
  - Interrupt or not?
  - Reset TC or not?
  - Stop TC and disable timer i or not?
- TCR: for enabling the timer (**set in enable_timer)**. Enable it **after** the initialization