Lab Assignment 2-3

Timers
EEE212 - Microprocessors
2021-2022 Fall

You have to use **TIMERS** to create a delay.

Part A - Using Timers (35 pts)

Determine two arbitrary pins to use as ON/OFF switches (e.g., P2.4 and P2.5). According to the state of the switches, generate a square wave at an arbitrary pin with the below frequencies and 50% duty cycle (e.g., P2.6).

Note1: While writing your code, remember to send logic 1 to the pins that you are going to use as input first (to be able to configure the hardware and read the provided external value).

Note2: While building your hardware, **do not directly connect the signal generator/power supply to the input pins.** Use a **10K resistor between the signal source and the 8051 pin(s). For logic 1 use 5V, for logic 0 use GND=0V. Before connecting the signal generator, make sure that amplitude is not greater than 5V.**

Switch Combination	Frequency (Hz.)
00	50
01	100
10	250
11	500

Part B - Sampling (30 pts)

1. Generate a square wave (50% duty cycle) with frequency 1 kHz and amplitude 5V using the signal generator, and feed it to the microcontroller via an input port pin (e.g., P2.4).

Note: While writing your code, remember to make this pin an input first by using setb instruction. In hardware, **do not directly connect the signal generator to the pin.** Use a **10K resistor between the signal source and the port pin.**

2. Your next task is to read that signal with sufficient frequency so that we will be able to transmit it perfectly later using the sampled data. Intuitively, one needs a higher sampling frequency for signals with high frequency since they change a lot (for example, if the signal is known to be f=1Hz ($T = 1/f = 1s$ => [0.5 second] logic 1 + [0.5 second] logic 0), our sampling rate should be at least 2Hz to be able to read both logic levels). Hence, the least sampling frequency we use not to lose any information is the twice of the highest frequency component in that signal (see Nyquist rate).

a. As the first step, using a sampling rate $f_{sampling} = 10kHz$, read the signal fed from the input port and output the sampled signal to an arbitrary output pin (e.g., P2.7) continuously. (For this purpose, first calculate the corresponding sampling period $T_{sampling} = 1/f_{sampling}$ and then, use timers to write a subroutine to generate a delay equal to this period and call it between each read operation.)

b. Then, compare the original one and the sampled waveform at the output via the oscilloscope. Perform this "sample then output" procedure for a $f_{sampling} = 0.5kHz$ sampling frequency also. What do you observe?

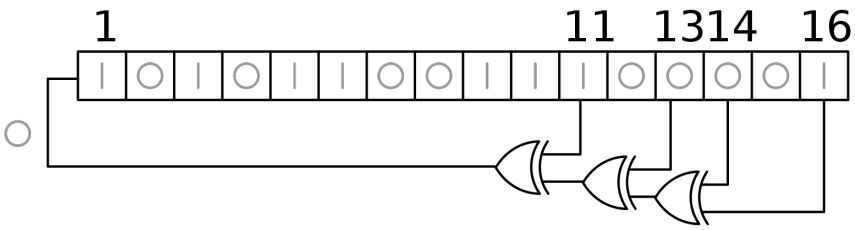

Figure 1: An 16-bit linear-feedback shift register (LFSR) [1]

Part C - Pseudo-random Number Generator (35 pts)

Generating random numbers (e.g., between 0 and 100) is an easy task in high-level programming languages such as Python or C++. However, these languages use different tricks to do so since there is no inherent randomness in hardware. In this part, you will implement a pseudo-random number generator.

First, determine two arbitrary registers (e.g., 40H and 41H) to form a 16-bit linear-feedback shift register (see Figure 1). Then, using this 16-bit register structure, you will generate 8-bit (1 byte) random numbers as described next.

1. Randomly initialize your registers as you wish (e.g., 40H and 41H). Do not use all zeros. Note that for a fixed initialization, the numbers you will generate is also fixed. That is how your TAs will check if your implementation is correct.

2. Save the bit at 16th bit position as a generated random bit. This bit will be used to form an 8-bit random number.

3. Follow the operations in Figure 1. That is, XOR the 16th bit and 14th bit, then XOR the output with 13th bit, and then XOR the output with 11th bit. i.e., $\Big(\big(16 \text{ XOR } 14\big) \text{ XOR } 13\Big) \text{ XOR } 11$. Store that output somewhere.

4. Shift all the bits in your 16-bit register to right by one (i.e., 1 to 2, 2 to 3...). Since you saved the last bit (i.e., 16th bit) in Step 2, you can discard it at this step. LFSR will not use this bit anymore.

5. Write the value found in Step 3 (i.e. the result of the XOR gates) to the least significant bit position of the LFSR.

Now, if you repeat the steps between (and including) Step 2 and Step 5 eight times consecutively, you will generate 8 random bits. If you write the random bits you generate at Step 2 to a register (e.g., 42H) starting from its least significant bit, you will have a 1-byte(=8-bit) random number by the end of this procedure.

Finally, you should repeat this whole 1-byte random number generation procedure 5 times. Each time you generate a 1-byte random number, display it on the LCD. Between generating and printing two random numbers, call a delay subroutine (that uses timers to create a 1 second delay) so that the numbers are visible on the LCD.

HINT Keep in mind that you actually have two separate 8-bit registers and thus should handle the operations such as *shifts* accordingly. Consider using the `RRC` operation to handle shifts.

References

[1] Wikipedia contributors. Linear-feedback shift register. URL https://en.m.wikipedia.org/wiki/Linear-feedback_shift_register. [Online; accessed 20-November-2021].