Lab 2. Mental Math Master

I. Purpose

The goal of this lab is to design Mental Math Master using LEDs and buttons. The purposes of this lab are the followings:

1) Understand how to design a digital circuit with FSM systematically
2) Implement FSM with combinational logic and sequential logic
3) Learn how pseudo-random numbers could be generated on-chip with a linear shift feedback register (LSFR).

II. Problem Statement

Problem 2A. Implement ALU

1) Implement an ALU unit for the following operations - addition, subtraction, multiplication, division, and comparison. Note that a signed representation system must be used for this lab.

Problem 2B. Implement LSFR

1) Implement a linear shift feedback register (LSFR) that generates pseudo-random numbers.

Problem 2C. System integration (Mental math master)

1) Design FSM for mental math master, shown in Figure 1. Specifications for mental math master are the followings (there are 2 LSFRs, one for number and one for operand):

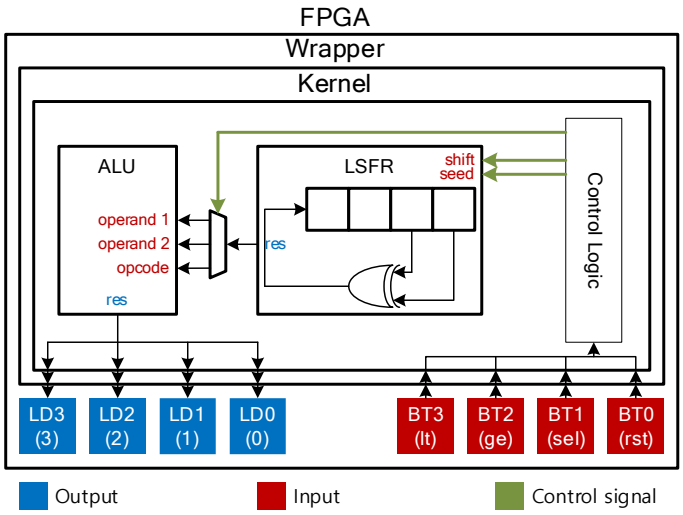


Figure 1. Mental math master

A. Input

i. **BT0** (rst): Button for reset.

ii. **BT1** (sel): Button used to get pseudo-random number with LSFR. This number can be used as operand or opcode, sequentially.

iii. **BT2** (ge): Button to enter the answer. If the accumulated value is greater than or equal to 0, press this button.

iv. **BT3** (lt): Button to enter the answer. If the accumulated value is less than 0, press this button.

B. Output

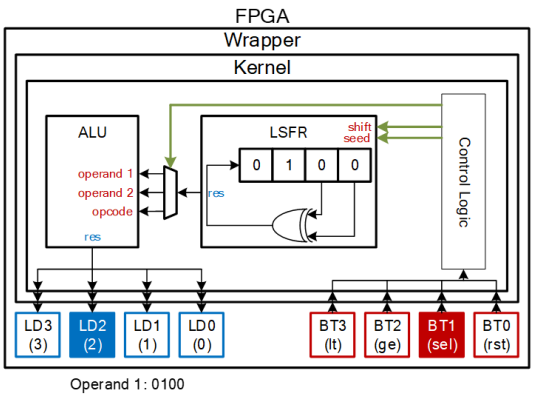
i. **LD3** (MSB 3), **LD2** (2), **LD1** (1), **LD0** (LSB 0): LEDs to represent the 4-bit signed binary number or result of action. In case of signed binary number, each LED is assigned for each digit. LD3 represents MSB, and LD0 represents LSB. For example, assuming 1 as “tuned on” and 0 as “turned off”, LED representation corresponding to 7(0111) would be LD3(0), LD2(1), LD1(1), LD0(1). In case of result of action, color and position of LED light will represent information such as number or result of answer.

C. Operation Scenario *(the values in the figure are just example)*

i. When BT0 (rst) is pressed, the entire mental math master unit should be reset.

ii. After the BT0 (rst) is pressed, the mental math master expects 4 sequential inputs of BT1 (sel). If BT2 (ge) and BT3 (lt) are pressed within this state, they should be ignored so that it would be equal to 4 sequential inputs of BT1 in the result.

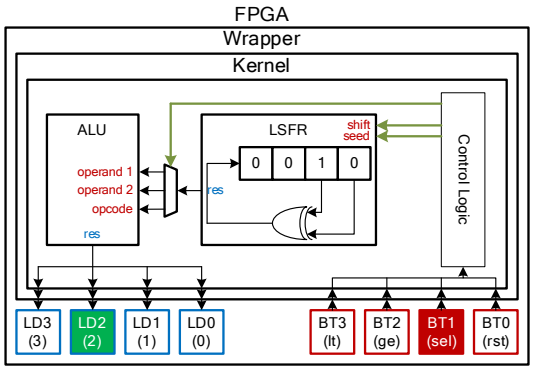
1. First, BT1 (sel) input would generate a pseudo-random number for operand 1. You should generate a pseudo-random number with first LSFR. You should represent number with BLUE LED.



Operand 1 : 0100

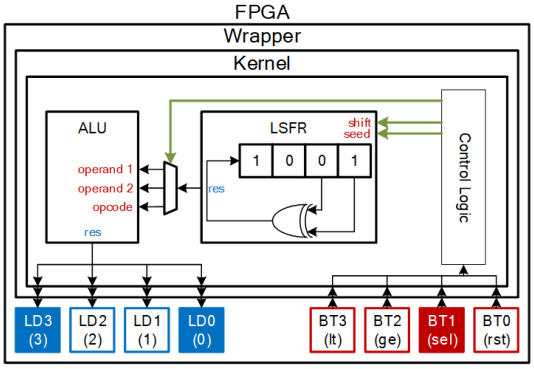
2. Second, BT1 (sel) input would choose the operation mode used for the ALU. You should choose operation mode with second LSFR where the lower 2 bits (so that includes LSB) would be used for mode selection. You should represent number with GREEN LED.

   LED positions and corresponding operand types are as follows:
   LED 0: AND / LED1: MUL / LED2: ADD / LED3: SUB



Operator: Addition

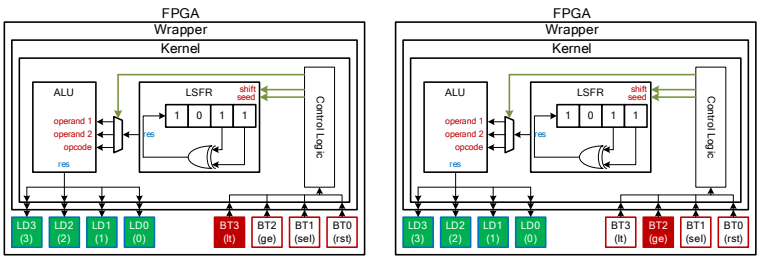
3. Third, BT1 (sel) input would generate a pseudo-random number for operand 2. You should generate a pseudo-random number with first LSFR. After getting the third BT1 input, calculate the result with selected operand 1, operator, and operand 2. You also should represent number with BLUE LED.



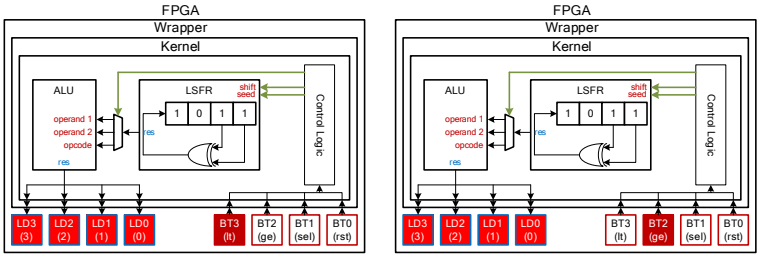
Operand 2: 1001

4. Fourth, BT1 (sel) input would make the machine go into comparison state. This state is for answer the question! If the accumulated value is greater or equal than 0, press BT2 (ge) button. Otherwise, press BT3 (lt) button.
If there exists no button input, every LEDs should be off. When BT2 (ge) or BT3 (le) input has occurred, the led should be work as shown below. When the BT 1 (sel) button input is received, it moves on to the next state.

   A. If the answer is correct, turn on all the LEDs as GREEN



   B. If the answer is incorrect, turn on all the LEDs as RED



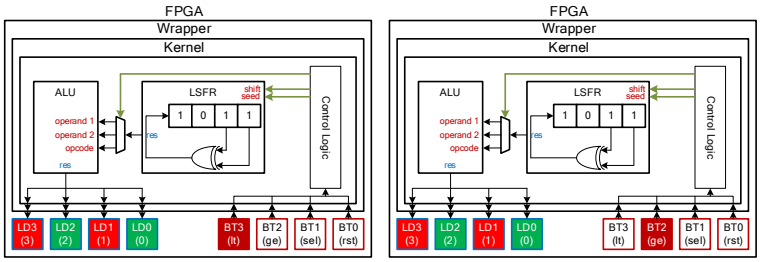
iii. After initialization of 4 sequential BT1 (sel) inputs, operations from **ii-2** to **ii-4** are repeated. But in this case, unintended BT2 (ge) or BT3 (lt) input should be handled.

1. First, BT1 (sel) input would choose the operation mode that is used for ALU. You should choose operation mode with second LSFR where the lower 2 bits (so that includes LSB) would be used for mode selection (**ii-2**). Also, the chosen operation should be displayed through GREEN LED.

2. Second, the next BT1 (sel) input would generate a pseudo-random number for operand 2. You should generate a pseudo-random number with first LSFR. After getting the third BT1 input, calculate the result with selected operator and operand 2 (**ii-3**). Also, the chosen number should be displayed through BLUE LED.

3. Third, the next BT1 (sel) input would make the machine go into comparison state. This state is for answer the question! If the accumulated value is greater or equal than 0, press BT2 (ge) button. Otherwise, press BT3 (lt) button.

   Exception handling) If either BT2 (ge) or BT3 (lt) is pressed before the machine gets into the comparison state, it should be handled since this is an invalid input. If this situation happens, 4 LEDs should display RED-GREEN-RED-GREEN pattern and your FSM should be able to receive remaining BT1 (sel) inputs so that FSM can iterate process of **iii**.



HINT: Be careful with your implementation for the button signals. To generate activation signal for them, our recommendation is using additional FSMs for each signal such as sel, ge, and lt.

III. Backgrounds

What is an arithmetic-logic unit?

An arithmetic-logic unit (ALU) is a combinational digital circuit that computes arithmetic and bitwise operations on binary numbers. It is the fundamental building block of many types of computing devices, including the central processing units (CPUs), floating point units (FPUs), and graphics processing units (GPUs). To enhance the performance of modern computing devices, they contain powerful and complex ALUs.

The conventional CPU consists of an ALU, control unit (CU), and register. The CU controls the execution of the ALU by decoding instructions into the input signals (e.g., operands, opcode, and carry-in). For example, the CU can determine the register address of input operands as well as the operation type (e.g., add, subtract, multiply, and divide). On the other hand, the register is used as the direct sources and destinations for the input and output, respectively, of the ALU.

The ALU, receiving control signals from the control unit, executes arithmetic and logic operations. It receives input operand 1, input operand 2, and ALU control. Then, it sends the result of the execution back to the register. The example of the operations is summarized in Table 1.

Table 1. ALU Operation Type

Opcode	Operations	Type
000	AND	Logic
001	OR	
010	Shift	Shift
011	Add	Arithmetic
100	Subtract	
101	Multiply	
110	Divide	

What is finite state machine (FSM)?

A finite-state machine (FSM) is a type of computation model that can be used to simulate sequential logic (i.e., control execution flow) and some computer programs, where it can be implemented in software or hardware. FSMs can be used to model problems in numerous academic disciplines, such as mathematics, artificial intelligence, games, and linguistics.

FSMs are made of one or more states, where only one single state of the machine can be active at the same time. It means that the machine has to transition from one state to another to perform different actions. Based on inputs, the states will change providing the resulting output.

Figure 2 shows the example of the coin-operated turnstile FSM. The FSM consists of states, transitions, inputs, and outputs. The state represents whether the turnstile is locked or unlocked. The transition of the states occurs when the sequences of inputs are sent to the machine. For instance, when a coin is inserted into the machine's slot, the turnstile is unlocked. When the turnstile's arm is pushed, the customer is allowed to pass, and the turnstile is locked once more.

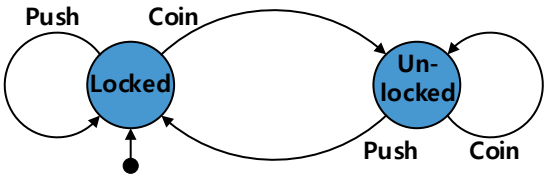
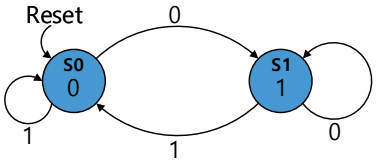
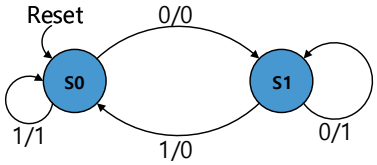


Figure 2. coin-operated turnstile FSM

There are two types of FSMs: the Moore machine and the Mealy machine. The way the output is generated is the main difference between them. The fundamental difference between the Mealy and Moore machines is that the dependency of output is on the current state and input. The current output of the Moore machine is solely determined by its current state, fairly stable without glitch issues. The current output of the Mealy machine is determined by the present state and present external inputs, which are more vulnerable to glitch issues. The detailed differences are summarized in Table 2.

Table 2. Summary of Moore and Mealy machines

	Moore Machine	Mealy Machine
output dependency	present state	present state, present input
number of states	more	less
HW requirement	less	more
output delay	one clock cycle after	same clock cycle
output synchronization	synchronous / output is placed on states	asynchronous / output is placed on transitions
Diagram		

What is linear-feedback shift register?

A linear-feedback shift register (LFSR) is a shift register where its input bit is a linear function of its previous state. Figure 3 illustrates the simple LSFR. The LFSR consists of a 4-bit shift register whose input bit is driven by the XOR gate of the two bits of the shift register values. In the shift register, the initial value, called the seed, is inserted. Then the stream of values produced by the register is generated by its current state (determined by the XOR gate). As a result, the 4-bit LSFR generates a sequence of 16 4-bit data iteratively. As the LFSR with the simple XOR feedback function can only generate a small number of states, it is limited to being used to generate random sequences of data. However, an LFSR with a well-chosen feedback function, such as Fibonacci LFSR and Galois LFSR, can produce a sequence of bits that has a very long cycle, appearing in random patterns.

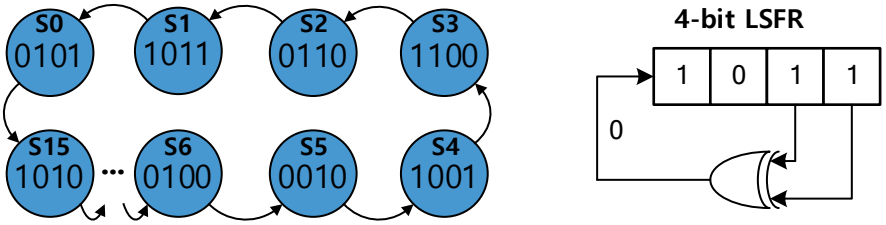


Figure 3. 4-bit XOR based LFSR

IV. Lab Procedure

Problem 2A. Implement ALU

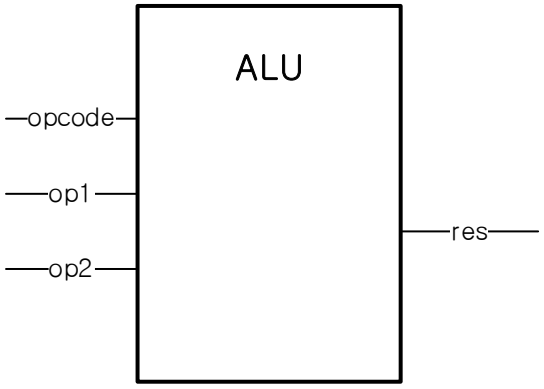


Figure 4. ALU unit

1) Implement an ALU unit for the following operations, including addition, subtraction, multiplication, and AND. Specifications for each operation are the followings:

   1. Input

   A. **opcode**(2): operation code to decide which operation to perform (10: addition, 11: subtraction, 01: multiplication)

   B. **op1, op2**: operand1 and operand2

   2. Output

   A. **res**: results for each operation are followings (the accumulation path can be modified in this lab project):

      i. opcode=10 (Addition): res = (op1+op2) + res

      ii. opcode=11 (Subtraction): res = (op1–op2) + res

      iii. opcode=01 (Multiplication): res = (op1*op2) + res

2) Verify ALU operations with testbench. (Refer the given testbench, and you can modify provided testbench to verify your module.)

Problem 2B. Implement LSFR

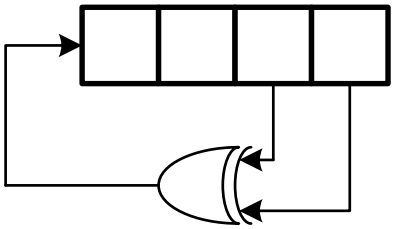


Figure 5. LFSR unit

1) Implement the LSFR unit as shown in Figure 5. Specifications are followings:

   1. Input

   A. **seed**: seed to initialize the LSFR state

   B. **shift**: control signal to transit the LSFR state

   2. Output

   A. **res**: pseudo-random number that generated with LSFR
   (Each state would be output pseudo-random number)

   B. **valid**: represent the result is valid data

2) Verify LSFR with testbench. (Refer the given testbench, and you can modify provided testbench to verify your module.)

Problem 2C. System Integration (Mental math master)

1) Design FSMs for mental math master.
2) Implement mental math master by integrating ALU and LSFR with a controller.
3) Verify mental math master with your own testbench.
4) Implement and map your mental math master on FPGA.
5) Verify the operation of your mental math master with demonstration.

V. Final Report

Discussion for the following question should be included in the report.

1) Explain the two major improvements of your design that makes your circuit design efficient.
2) Explain the effect of the bit length of an accumulation register with the proper experimental result.
3) Describe the throughput of your design, including the average throughput and utilization (avg. throughput / theoretical peak throughput) of the ALU. Discuss how you can increase the utilization of the ALU.

VI. References