Laboratory 1

(Due date: **005**: January 31st, **006**: February 1st)

Objectives

- ✓ Implement a Digital System: Control Unit and Datapath Unit.
- ✓ Describe Algorithmic State Machine (ASM) charts in VHDL.
- ✓ Learn the basics of Microprocessor Design.

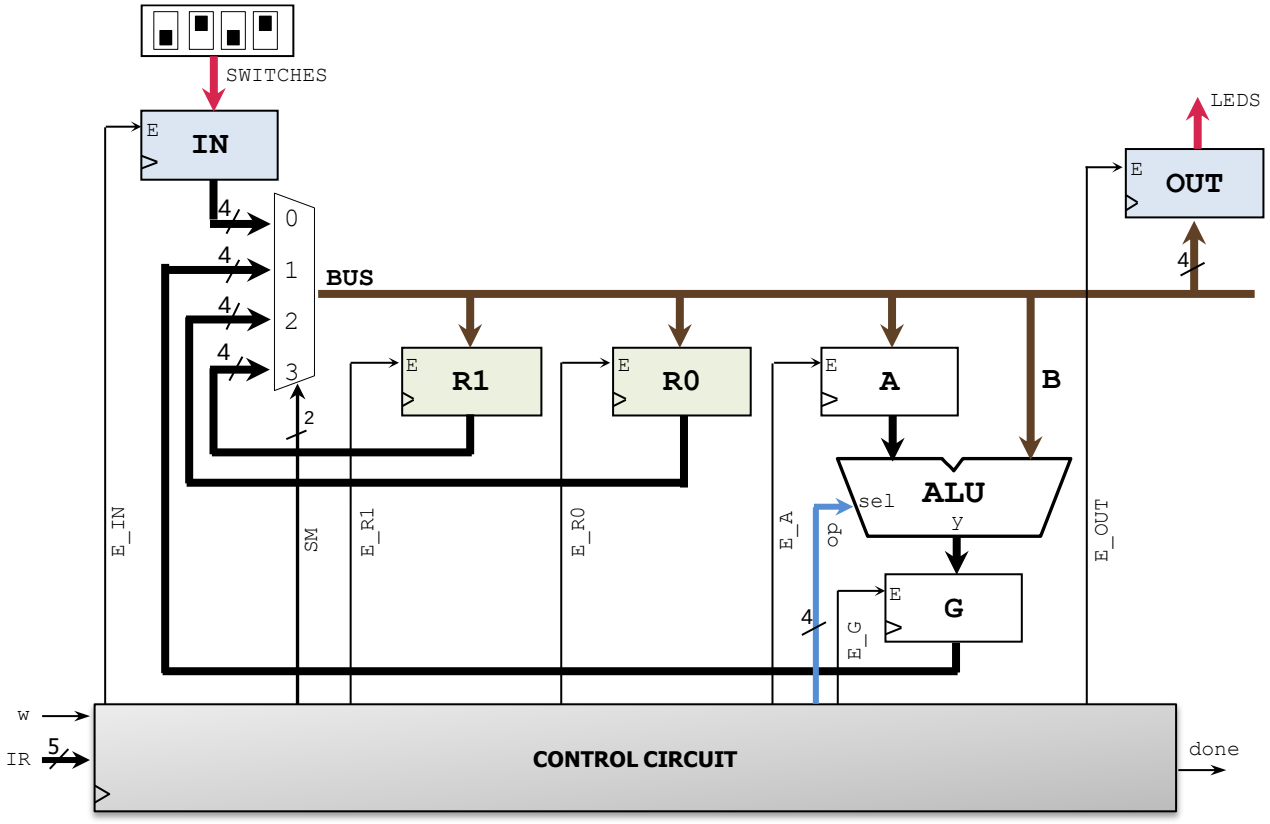
VHDL Coding

- ✓ Refer to the Tutorial: VHDL for FPGAs for parametric code for: Register and ALU.

First Activity: Small Microprocessor – Design and Simulation (70/100)

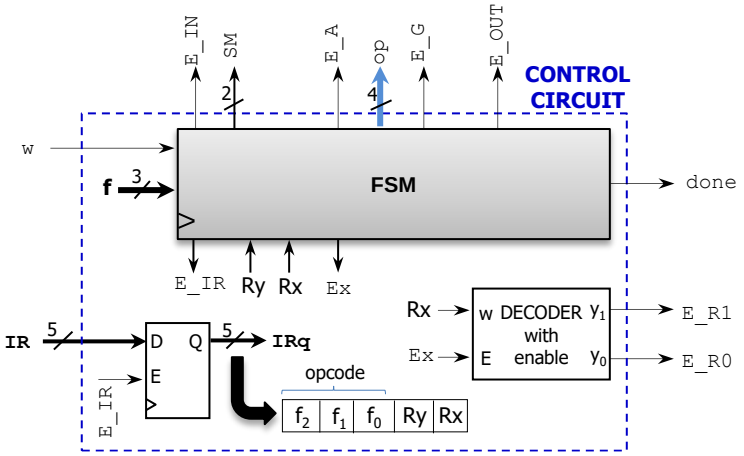
Design Problem

- The figure depicts a 4-bit microprocessor featuring two processor registers (R0 and R1). Instructions: 5-bit wide.



CONTROL CIRCUIT

- It consists of a register (to store incoming 5-bit instructions), an FSM, and a decoder (this helps to simplify the FSM diagram).
- 5-bit instruction: Fed via `IR` (captured on `IRq`)
  - ✓ '`w`' signals an incoming instruction, i.e., an instruction is captured on the register when `w=1`. The instruction is then processed; when it is finished, the signal `done` is asserted. A new instruction can then be provided in the next clock cycle.



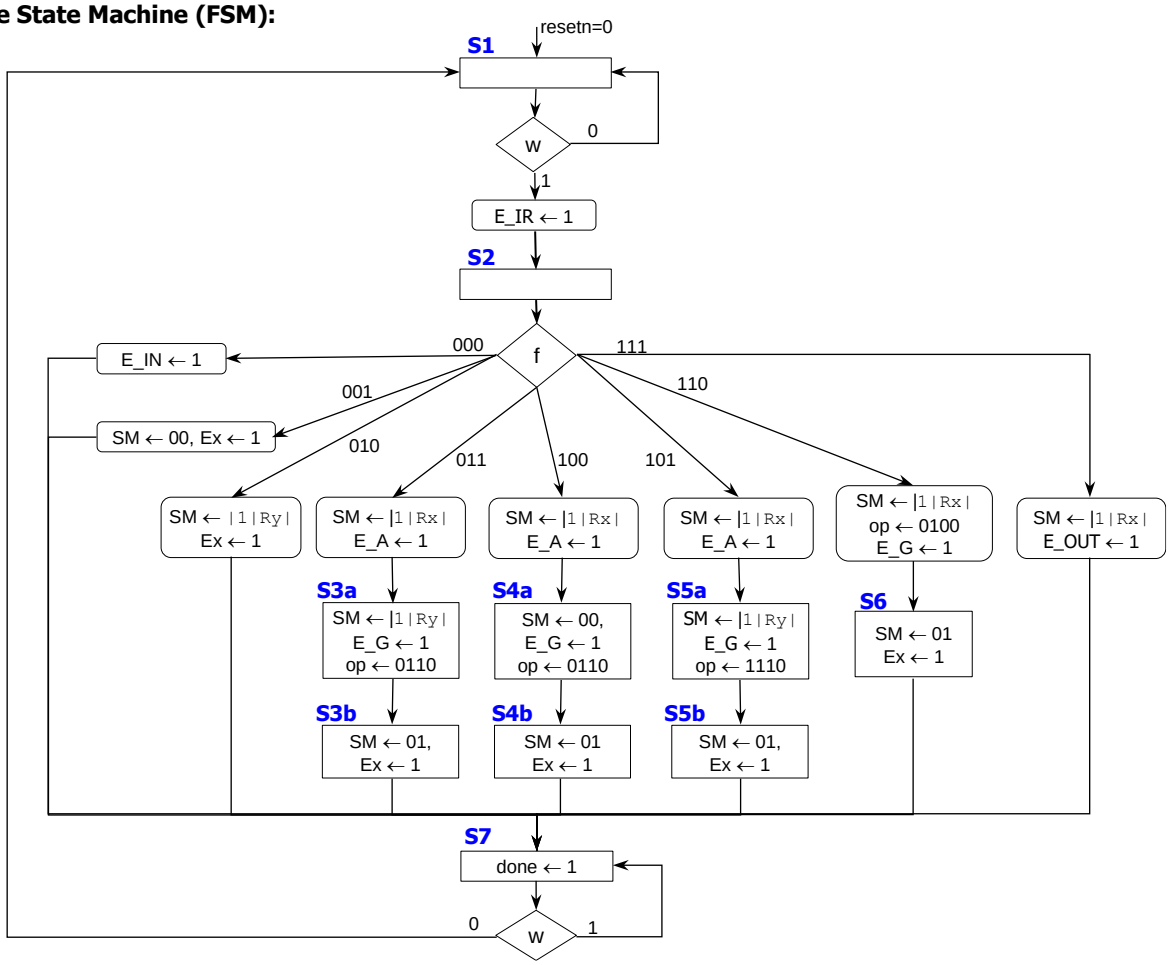
Instructor: Daniel Llamocca
TAs: Alex Fillmore, Luke Nuculaj

- **Instruction Set:**
  Instruction: |f_2|f_1|f_0|Ry|Rx|
  - ✓ Opcode (operation code): **f** = |f_2|f_1|f_0|. These bits specify the operation to be performed.
  - ✓ Operands: |Ry|Rx|. These bits specify the register(s) to be used in the operation.
    - Rx: index of the register where the result of an operation is stored (we also read data from Rx). Rx can be R0 or R1.
    - Ry: index of the register where we only read operands from. Ry can be R0 or R1.

f	Operation	Function
000	load IN	IN ← Switches
001	load Rx, IN	Rx ← IN
010	copy Rx, Ry	Rx ← Ry
011	add Rx, Ry	Rx ← Rx + Ry
100	add Rx, IN	Rx ← Rx + IN
101	xor Rx, Ry	Rx ← Rx XOR Ry
110	inc Rx	Rx ← Rx + 1
111	load OUT, Rx	OUT ← Rx

  - ✓ Instruction examples:
    - load R1,IN ≡ IR = 001X1
    - add R0,R1 ≡ IR = 01110

- **Finite State Machine (FSM):**



- **Decoder with enable**: This circuit simplifies the FSM diagram. A suggested code is listed below:

```
-- E, w: inputs     y: 2-bit output
...
signal Ew: std_logic_vector (1 downto 0);
...
with Ew select
     y <= "01" when "10",
          "10" when "11",
          "00" when others;
```


Instructor: Daniel Llamocca
TAs: Alex Fillmore, Luke Nuculaj

- **Arithmetic Logic Unit (ALU):**

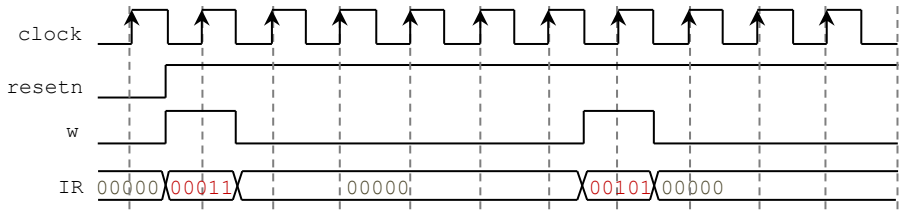
sel	Operation	Function	Unit
0000	y <= A	Transfer 'A'	Arithmetic
0001	y <= A + 1	Increment 'A'	
0010	y <= A - 1	Decrement 'A'	
0011	y <= B	Transfer 'B'	
0100	y <= B + 1	Increment 'B'	
0101	y <= B - 1	Decrement 'B'	
0110	y <= A + B	Add 'A' and 'B'	
0111	y <= A - B	Subtract 'B' from 'A'	
1000	y <= not A	Complement 'A'	Logic
1001	y <= not B	Complement 'B'	
1010	y <= A AND B	AND	
1011	y <= A OR B	OR	
1100	y <= A NAND B	NAND	
1101	y <= A NOR B	NOR	
1110	y <= A XOR B	XOR	
1111	y <= A XNOR B	XNOR	

PROCEDURE

- **Vivado: Complete the following steps:**
  - ✓ Create a new Vivado project. Select the corresponding Artix-7 FPGA device (e.g.: the XC7A50T-1CSG324 FPGA device for the Nexys A7-50T board).
  - ✓ Write the VHDL code for the given circuit. Synthesize your circuit to clear syntax errors.
    - Suggestion: Use the Structural Description: Create a separate `.vhd` file for the components (register, ALU, MUX, Control Circuit) and interconnect them all in a top file. The Control Circuit and ALU have their own components.
  - ✓ Write the VHDL testbench to simulate your circuit.
    - Your testbench must test the following Assembly program (use a 100 MHz input clock with 50% duty cycle):

```
load IN;          IN ← 0101 (SWs = 0101)          IR = 000XX
load R1, IN;      R1 ← 0101                       IR = 001X1
copy R0, R1;      R0 ← 0101, R1 ← 0101
inc R1;           R1 ← 0110
xor R0, R1;       R0 ← 0101 xor 0110 = 0011
add R0, R1;       R0 ← 0011 + 0110 = 1001
load OUT, R0;     OUT ← 1001
```

    - Suggestion: You can use this timing diagram (where the first two instructions are loaded) as a template.



  - ✓ Perform Behavioral Simulation and Timing Simulation of your design. **Demonstrate this to your TA**.
    - Behavioral Simulation: To help debug your circuit, add internal signals (e.g.: state, R0, R1, A) to the waveform view.
  - ✓ I/O Assignment: Create the XDC file associated with your board. (Nexys A7-50T) (Nexys A7-100T)
    - Suggestion (Nexys A7-50T/A7-100T, Nexys 4/DDR):

Board pin names	CLK100MHZ	CPU_RESET	SW8-SW5	SW4-SW0	BTNC	LED4	LED3-LED0
Signal names in code	*clock*	*resetn*	IN	IR	w	done	OUT

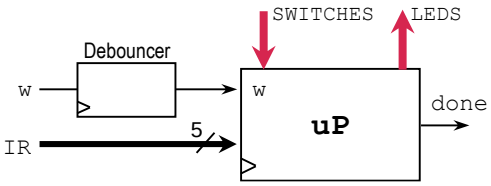
    - Note: synchronous circuits always require a clock and reset signal.
      - **Reset signal**: As a convention in this class, we use active-low reset (*resetn*). Thus, we tie *resetn* to the active-low push button `CPU_RESET` of the Nexys A7-50T/A7-100T, Nexys 4/DDR board.
      - **Clock signal**: Like other signals in the XDC file, uncomment the lines associated with the clock signal and replace the signal label with the name used in your code. In addition, there is parameter `-period` that is set by default to `10.00`. This is the period (in ns) that your circuit should support.
        - Nexys A7-50T: In these lines, replace the label `CLK100MHZ` with the signal name you use in your code (`clock`):

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
```


Instructor: Daniel Llamocca
TAs: Alex Fillmore, Luke Nuculaj

SECOND ACTIVITY: TESTING (30/100)

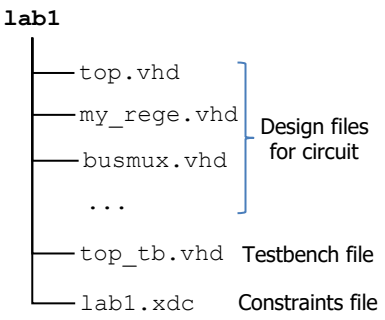
- In order to properly test the microprocessor, we need the avoid mechanical bouncing on the pushbutton for input *w*. Connect the debouncer circuit (use the given files: `mydebouncer.vhd`, `my_genpulse_sclr.vhd`) on the input *w*.
- Note that you do not need to simulate the circuit that includes the debouncer.



  - ✓ Generate and download the bitstream on the FPGA and test the Assembly Program. **Demonstrate this to your TA**.
    - To test the Assembly program, load each instruction via the input `IR` and use the input `w` to feed the instruction.

SUBMISSION

- Submit to Moodle (an assignment will be created):
  - ✓ This lab sheet (<u>as a .pdf</u>) completed and signed off by the TA (or instructor).
    - Note*: This lab assignment has two activities. You get full points of the 1st activity if you demo it by the due date. You can demo the 2nd activity by the due date or late (here, we apply a penalty towards the points of the 2nd activity).*
  - ✓ (<u>As a .zip file</u>) All the generated files: VHDL code, VHDL testbench (for the uP block), and XDC file. **DO NOT submit the whole Vivado Project**.
    - Your .zip file should only include one folder. Do not include subdirectories.
    - It is strongly recommended that all your design files, testbench, and constraints file be located in a single directory. This will allow for a smooth experience with Vivado.
    - You should only submit your source files AFTER you have demoed your work. Submission of work files without demoing will be assigned <u>NO CREDIT</u>.



TA signature: _____ **Date:** _____