ECE 281 Lab 1 – TTL Logic

Read all directions before starting. For this lab, work in teams of two. You may seek help from any ECE 281 instructor and reference any publication in its completion, but may not utilize any student. Normal documentation is required (you do not need to document work with your partner). We expect all the code your own work.

All prelab work should be computer generated or ***neatly*** hand drawn.

Resources:

- "Lab1" or "Data Sheets" pages on Blackboard
- K:\df\dfec\ECE281\Lab1 or K:\df\dfec\ECE281\Datasheets

Prelab (Due Lesson 9):

You are creating a circuit that will take a month as a 4-bit binary input. **January is equivalent to 0001**. For each month that has 31 days, you should turn on the red light (your output value Y is '1'). All other months and the unused binary inputs should produce an output of '0'.

1) **(5 pts)** Draw a truth table to describe these conditions.
2) **(5 pts)** Use a K map to find the simplified equation for Y.
3) **(32 pts)** Using the provided datasheets, draw four separate schematics that can implement your truth table.
    a. The first shall use an 8:1 multiplexer and an inverter(s).
    b. The second shall use a 4:16 "one-cold" (active low output) decoder, inverter(s), and two-input OR gate(s). Using 2-input NANDs is also acceptable.
    c. The third shall use only an inverter(s), two-input OR gate(s), and three-input AND gate(s).
    d. The fourth shall use only NAND gates with either two or four inputs.
4) **(8 pts)** For each schematic, list how many chips of each type you will need to build that schematic and how much it will cost. Keep in mind that **some chips may have multiple gates**; for example, each hex inverter chip contains six inverters. You will need to consult the datasheet for each available part to see how many gates each contains. The available parts list is below, and the datasheets can be found in the resources listed above.

Device	Part Number	Price per chip
8:1 mux	74151	$0.75
4:16 decoder	74154	$4.95
2-input OR	7432	$0.45
4-input NAND	7420	$0.79
3-input AND	7411	$0.49
Hex NOT	7404	$0.65
Double NAND	7400	$0.45

5) **(10 pts)** Answer the following questions:

a. **(2 pts)** If you only have a 4-input NAND gate to use, but your circuit design calls for a 3-input gate, what would you connect the other input to for your circuit to function properly?
b. **(3 pts)** The multiplexer (74151) has a signal called $\overline{STROBE}$. What function does this signal serve?
c. **(2 pts)** The 4-input NAND gate chip (7420) has a connection labeled NC. What does this mean?
d. **(3 pts)** If you were contracted to build a circuit to satisfy the given requirement, what aspects of the design would you need to consider? How would these considerations affect your design?

During class (Due Lesson 11):
You will build **circuit 3a** designed above and **demonstrate** that it works to an instructor, who will sign your schematic.

A. Before you can build your circuit, you must set up the breadboard the same way you did for ICE1 (Parts A, B, and eventually D).
B. Once you have your layout, start connecting your chips.

Hints on wiring and debugging:

- Reference back to ICE1 if you need a refresher on how the breadboard functions. You may also reference the breadboard handout in the Datasheets Library to see how things are connected.
- DO NOT leave floating inputs or control signals. All unused input pins need to be grounded or connected to Vcc.
- If note already done, connect the Vcc and ground tracks on top to the columns on the breadboard. Don't forget to connect the top and bottom half of the columns over the break (if necessary).
- Double check all connections before turning on power
- Do not change your circuit with the power on

After Class (Due Lesson 13)
You will implement the same design in VHDL and demonstrate that it works to an instructor.

1. Implement SSD component in VHDL

Get the VHDL templates

a. You should have them as part of the course download in the Lab1 folder
b. Special templates have been provided for this Lab
    a. thirtyOneDayMonth.vhd
    b. thirtyOneDayMonth_tb.vhd
c. Add your sources to Vivado.
    a. Remember to leave "Copy sources into project" **unchecked**

d. Edit the file headers as needed.

Create your SSD entity

Create your interface (ports) for thirtyOneDayMonth according to Figure 3 below. Figure 3 shows the thirtyOneDayMonth module entity interface in blue and the architecture in purple. Notice that the figure indicates that the input, i_sw, is a bus of 4 wires. This can be created in VHDL by using a **std_logic_vector** signal type. For instance, i_sw could be created in the **port** statement with the following:

```
i_sw    :   in std_logic_vector(3 downto 0);
```

Remember, the "`i_`" follows the naming convention provided in the header that indicates the signal is an input. The statement above "`3 downto 0`" indicates that BIT3 is the MSB and BIT0 is the LSB. You could easily reverse this by swapping the locations of the two numbers. **This has been created for you in the template.**

To access each wire from the bus you created simply refer to the bit number in parentheses. For instance, `i_sw(3)` refers to the MSB of the input, D.

The output, o_led0 is a single wire. This can be created in VHDL by using a **std_logic** signal type. For instance, led could be created in the **port** statement with the following:

```
o_led0 : out std_logic
```

This has been done for you in the template.

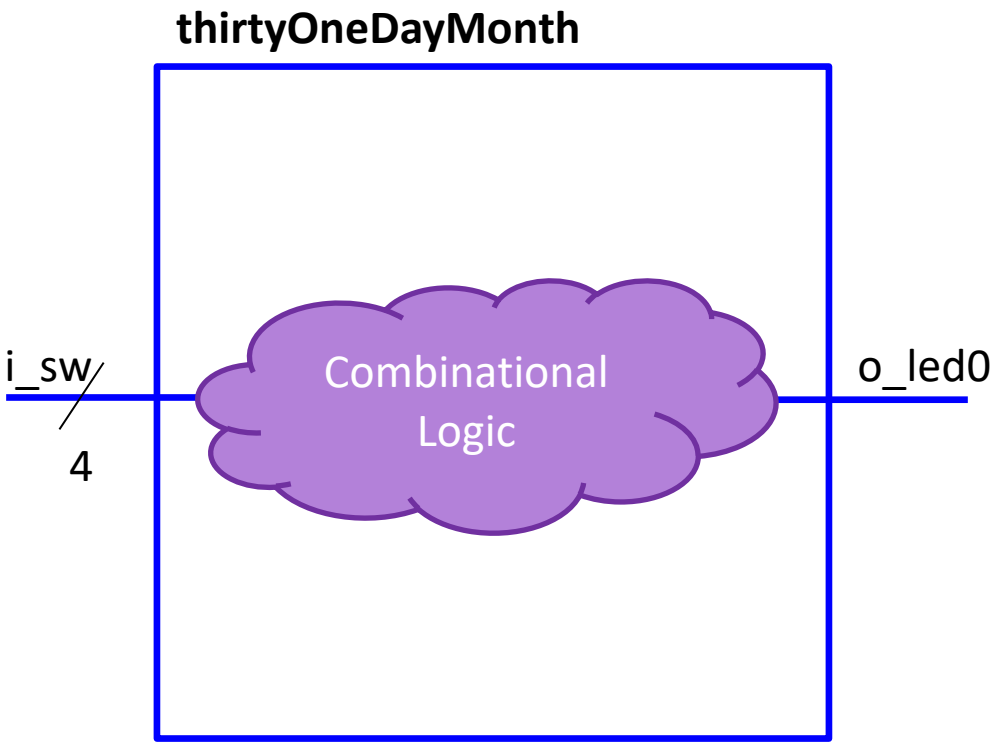


Figure 3 – thirtyOneDayMonth entity and architecture

Create your SSD architecture

After you have created your component interface, you need to describe the architecture. For easy of understanding you can utilize combinational signals (denoted by the "c_" at the beginning) and wire them to the input bus i_sw. For instance, BIT3 of i_sw is wired to c_A.

a. For clarity, declare intermediate combinatorial signals (e.g., c_A) that break out input i_sw and set them to a default value of '0' (off). This should be done *before* **`begin`**.
b. There are no PORT MAPS or PROCESSES needed for this module, so you can delete those sections.
c. In the CONCURRENT STATEMENTS section, map the input i_sw to the signals you created above per Figure 3. For instance:

```
c_A <= i_sw(3);
```

d. If you are wanting to create a vector that is a subset of another vector you can do

```
signal bigvector: std_logic_vector(3 downto 0);
signal smallvector:std_logic_vector(1 downto 0);

begin
    smallvector <= bigvector(3 downto 2);
```

e. You may need to create other intermediate wires to implement your logic.
f. Implement your prelab equations using behavioral modelling

i. Reference the VHDL Reference sheet for assistance in implementing the circuits
ii. Helpful hint: When the example code uses "0001" it is referring to a 4 bit signal. You can instead use x"1" to represent the number in HEX. Ex: "11100011" would become x"E3"

2. Create SSD testbench and simulate

Edit the VHDL template

a. Edit the header as needed.
b. Much of the tb is filled out for you. Read through the provided template to see how it was set up
c. Finish the test process. Remember, you can use hex values with busses: `sw <= x"F";`
d. Fix any syntax errors, and run the simulation.
e. Groups inputs and outputs and change the radix to hex. Makes sure the signals are in the proper order so the hex values make sense. This makes verification easier.
f. Make sure the simulation results match your truth table, and then **include a waveform screenshot in your repo**.

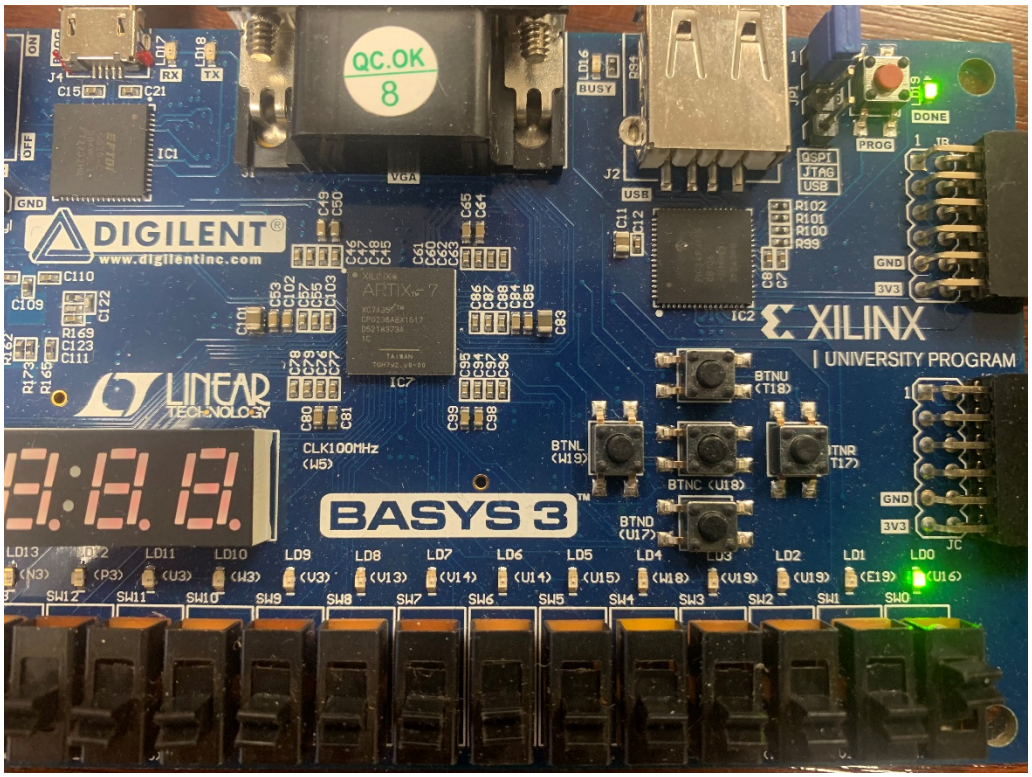
3. Modify the constraints file

a. Add your Basys3Master.xdc file to your project as a constraint and open it.
b. Uncomment the lines you will need to use. This will include the first 4 switch statements (8 lines) and the first led statement (two lines). **This has been done for you in the template.**
c. Rename the section inside the {} to match your signals ex: sw[0]->i_sw[0] and led[0]->o_led0. **This has been done for you in the template.**

4. Implement the design in hardware

a. Generate the bitstream (.bit) file and download it to your FPGA.
   - Do not forget to commit the .bit file to your Git repo.
   - The default location of the file should be: `Lab1\Lab1.runs\impl_1\thirtyOneDayMonth.bit`
b. Verify that your design functions correctly.

Demo the working final product to your instructor.


QC.OK
8
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
CLK100MHz
(W5)
BTNU (T18)
BTNL (W19)
BTNC (U18)
BTNR (T17)
BTND (U17)
LD13 (N3)
LD12 (P3)
LD11 (U3)
LD10 (W3)
LD9 (V3)
LD8 (V13)
LD7 (V14)
LD6 (U14)
LD5 (U15)
LD4 (W18)
LD3 (V19)
LD2 (U19)
LD1 (E19)
LD0 (U16)
SW12
SW11
SW10
SW9
SW8
SW7
SW6
SW5
SW4
SW3
SW2
SW1
SW0
DONE
PROG
QSPI
JTAG
USB
BUSY
VGA
GND
3V3
JC

LAB 1 DELIVERABLES

Table 3 below shows how the points will be distributed for this assignment. Details for how each item will be scored follows.

Table 3: Point distribution for Lab 1

Item	Out of
Prelab	60
TTL Hardware Demo	40
VHDL Hardware Demo	30
Simulation Results Waveform	10
VHDL, XDC, and Bit Files	10
Total	**150**

(60 PTS) PRELAB

- Picture of the graded hardcopy you turned in acceptable

(40 PTS) TTL HARDWARE DEMO

- Demonstrate proper LED behavior on the protoboard as you switch from 0 to 16 (in binary)

(30 PTS) VHDL HARDWARE DEMO

- Demonstrate proper LED behavior on the FPGA as you switch from 0 to 16 (in binary)
- Demo can be performed live with an instructor (preferred) OR documented in the report.
    - If documented in the report, include a brief video (online link is fine) or series of pictures for complete proof of correct functionality.

(10 PTS) SIMULATION RESULTS WAVEFORM

- Clearly shows that that all possible inputs and outputs match the correct Prelab truth table.

(10 PTS) VHDL, XDC, AND BIT FILES

- thirtyOneDayMonth.vhd and thirtyOneDayMonth_tb.vhd included in **code** folder
    - Fill out headers as appropriate
    - Remove extraneous code and comments
- Basys3_Master.xdc file included in **code** folder
- Bitstream (.bit) file used for hardware demo in repo