LAB 2 - Combinatorial Logic

26 August 2021 16:14

LAB 2 – combinatorial logic

This exercise will familiarize you with the basic structure of a VHDL program and some simple combinatorial logic structures.

Tip: Reuse as much as possible the code of Lab 1 - at this point you don't (yet) have to invent/learn anything new - all the keywords and necessary have already been introduced in Lab 1.

Tip: Even if it would be possible to complete all the Labs within a single project - **create a new project for each lab**. (Vivado gets sometimes really lost if you accidentally have several entities with the same name.)

VHDL reference guide, standard, VHDL cheat sheet etc can be found under General -> Files -> Class Materials -> Books

https://tuas365.sharepoint.com/:f:/r/sites/VHDLandFPGA/Class%20Materials/Books?csf=1&web=1&e=Tvr4E0

2-to-4 decoder

Create a design, which controls the green LEDs in a following way:

Inputs		Outputs				
Btn[1]	**Btn[0]**	**led[3]**	**led[2]**	**led[1]**	**led[0]**	**RGB-led (ld4)**
off	off	off	off	off	on	off
off	on	off	off	on	off	Red
on	off	off	on	off	off	Green
on	on	on	off	off	off	Blue

This kind of design is also called a binary decoder – a counter-function for binary encoder (also known as priority encoder). It can be used for example to create an address decoder for 4 memory banks, where the 2 inputs are 2 most significant address bits and the 4 outputs are "enable" signals for the memory devices/banks. Now we are just controlling LEDs.

Source code answer : (build a project with Part: xc7z010iclg225-1L Board: pynq-z2)	lab2	pynq-z2_v1 .0

Master switch (output mux)

Expand the design in a following way:

When switch 0 (sw0) is **off**, RGB-led 5 remains off.

When switch 0 (sw0) is **on**, RGB-led 5 acts as RGB-led 4.

When switch 1 (sw1) is on, it overrides everything so that all channels of RGB-led 5 are on (it turns white).

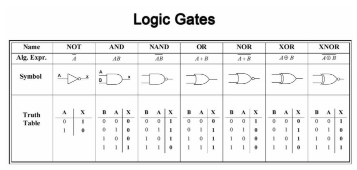
Tip: Think a little bit how to utilize the "others" keyword in a most effective

Playing with some basic logic gates (using logic primitive functions)

Expand the design so that LEDs 0-3 work in a following way:

- When switch 1 is **off,** LEDs 0-3 acts as earlier
- When switch 1 is **on,** LEDs 0-3 acts as in table below:

Inputs		Outputs			
Btn[1]	Btn[0]	led[3]	led[2]	led[1]	led[0]
0	0	1	0	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	0	1	0	1



Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
Truth Table	A X	B A X	B A X	B A X	B A X	B A X	B A X
	0 1	0 0 0	0 0 1	0 0 0	0 0 1	0 0 0	0 0 1
	1 0	0 1 0	0 1 1	0 1 1	0 1 0	0 1 1	0 1 0
		1 0 0	1 0 1	1 0 1	1 0 0	1 0 1	1 0 0
		1 1 1	1 1 0	1 1 1	1 1 0	1 1 0	1 1 1

'0' = off, '1' = on

Note: You need to keep the existing logic (2-to-4-decoder), and add another "logic block" plus a means to select between those. Try to keep as much existing code as possible, just add extra logic to the design. Now it would be a very good moment to draw a simple block diagram of your design.

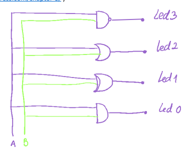
Tip: WITH-SELECT-statement (= selected signal assignment) works of course, but it is a bit "overkill" for a simple 2-to-1 multiplexer. Consider using conditional signal assignment instead:

signal_name <= expression_1 **when** condition_1 **else** expression_2;

Note: Do not use look-up table for above, instead figure out what is the logic function of each led and **use the VHDL "keywords" for basic gates (*and*, *or*, *nand* etc) to implement the logic**. See table of logic gates below:

(source: https://frankcomputerscience.wordpress.com/chapter-3/)

led 3 $\Rightarrow$ $\overline{AB}$ (NAND)
Led 2 $\Rightarrow$ $A+B$ (OR)
Led 1 $\Rightarrow$ $A\oplus B$ (XOR)
Led 0 $\Rightarrow$ AB (AND)



Source code answer : (build a project with Part: xc7z010iclg225-1L Board: pynq-z2)	lab2final	pynq-z2_v1 .0

Questions

1. How many **look-up-tables** (LUTs) does the design use? (See synthesis/utilization report)
   5 look-up-tables (?)

2. How many **registers** are used?
   In this context, register means same as flip-flop, that is, a single bit memory element. In microprocessors, registers mean a 8/16/32/64-bit register, depending on the word length of the computer architecture in question.

1. How many **ports** have you defined in the entity – does it match with the reported I/O-count?

   6 ports were used , 2 input and 4 output

```
btn :      in  STD_LOGIC_VECTOR( 1 downto 0);
sw :       in  STD_LOGIC_VECTOR(1 downto 0);
led :      out  STD_LOGIC_VECTOR (3 downto 0);
led4_r :   out STD_LOGIC;
led4_g :   out STD_LOGIC;
led4_b :   out STD_LOGIC
```

1. Compare the elaborated design to the synthesized design at the schematics level – what is the difference?

5. Does the design fulfil all the timing requirements? What were those?

Once you have **written** answers to all of the questions above, call teacher to check your design (that is, show the demo) and the answers to the questions.