# Lab 4: Sequential Logic Design (Simple (almost practical) Vending machine)

**Objectives:**

- Design and implement **Simple Vending machine**
- Practice writing testbench for sequential circuits
- Use Structural Verilog style to connect multiple modules to implement a digital circuit.

**Start:** Week 8 (Feb 27, 2023) – attend your scheduled lab session

**Pre-lab 4:** Show pre-lab to your TA/ULA within the first 2 hours of your lab session during the week of Oct 9 – 13, 2023.

- (15 points) Write *Verilog code* and *testbench* for Button synchronizer or Level to Pulse converter. Use the state diagram given in section 3.8 of zyBooks: Participation activity 3.8.7 and 3.8.8 to write Verilog code.
- (8 points) Draw a *state diagram* of a 25-cent vending machine.

**Demo Due:**

- Demo part (A) during the week of Oct 9-13, 2023
- Demo part (B) by the end of your lab session during the week of Oct 16-20, 2023

**Code submission:**

Submit the files that you/your group create or modify on D2L Assignment Dropbox by 11.59 pm on Friday October 20, 2023.

**References:** Lecture Notes (on D2L): FSM (in Unit 2 folder) and Verilog_lecture2_seq (in Unit: Verilog folder)

**Lab Overview:** in this lab assignment,

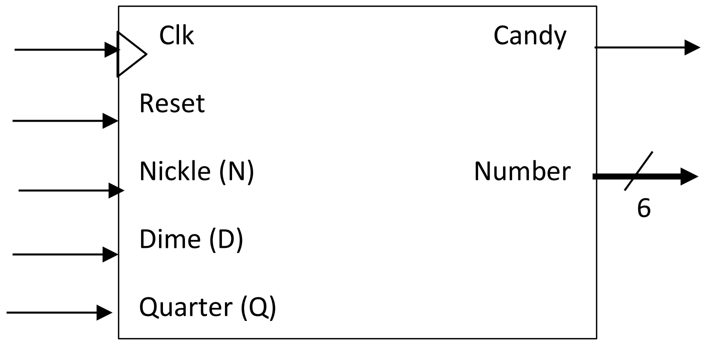
Part A) You will

- Design (state diagram, Verilog code, testbench) and perform post-synthesis functional simulation of your designed ***Vending machine***.
- Design (state diagram, Verilog code, testbench) and perform post-synthesis functional simulation of Button synchronizer or Level to Pulse converter.
- Demo post-synthesis functional simulations of both circuits (one at a time) to TA

Part B) You will then

- Write a top-level Verilog code to connect several modules together to create a circuit that can work on the FPGA board,
- Modify the constraint Nexys4DDR_Master.xdc file such that it can be used with the top-level code to do pins assignment.
- Generate bitstream to program the Nexys4 DDR FPGA board. Demo to TA by the end of your lab session of the 2nd week of lab 4 (Mar 13-17, 2023).

**The 25-cent Vending machine:**
The purpose of the lab is to implement a Vending machine on Nexys 4 DDR board.



You are to design the **electronic vending machine** which sells a ***candy*** for **25 cents**.

- The machine has a clock (Clk) signal and Reset (Rst) button such that when Reset = 1, FSM goes back to an initial state.
- It accepts N (Nickel = 5 cents), D (Dime = 10 cents) and Q (Quarter = 25 cents)
- It keeps track of the amount of money that the customer enters using the *6-bit output, Number*. **At each state (when Candy is still 0), it displays amount of money that it has received.**
- It releases the candy by setting Candy = 1 and also returns the change through *Number* if the customer enters more than or equal to 25 cents.
- For example,
  - if the customer enters a nickel, your machine goes to a state representing 5 cents and *Number* = 5 (since 5 cents are just entered and 5 should be displayed).
    If the customer next enters a dime (10 cents), it goes to a state representing 15 cents and *Number* = 15 (since the machine now gets 15 cents). This scenario continues until a user enters a total of exactly 25 cents or more than 25 cents.
  - if the customer enters exactly 25 cents, your machine should go to the state that set Candy to 1 and *Number* to 0 (no change return).
  - if the customer enters a total of 40 cents, it should go to the state that set Candy to 1 and *Number* to 15 (13 cents are returned since the candy cost only 25 cents).

**Note:**

1) **For this lab, once the machine reaches the state(s) that release the candy, make your machine stay at that state** (this is one of the almost-practical parts since we want to be able to see the outputs, we make the machine stay at that state (state(s) that Candy = 1) until the reset button is pushed. Once the reset button is pushed, the machine goes back to the initial state and waits for the next purchase). In practice, the machine will release the candy, return the change (if any), and then go back to the initial state waiting for the next purchase by a customer.
2) In practice, the coin receptor can accept only one coin at a time (there is a mechanism that prevents a customer to enter two coins at once), this means that it is NOT possible to have N = D =1 or N = Q =1, or D = Q =1, or N = D = Q = 1.
   However, on our Nexys4 DDR board, it can happen. For these cases, when drawing a state diagram, make your state diagram stay at the same state (this is another almost-practical part since your machine will take the money if more than one coins inserted at once).
3) In practice, the coin receptor is a mechanical device and thus very slow compared to an electronic circuit. There will very likely be an arbitrary long time between an insertion of a coin => at each state, it is possible to have all inputs N, D, Q = 0 ➔ when drawing the state diagram, when all inputs N,D,Q are 0, make the machine stay at the same state.

**Part A: 25-cent Vending machine**

A1: Draw a **state diagram of 25-cent Vending Machine** (Read the requirement specifications (page 2) carefully before drawing the state diagram)
A2: **Write Verilog code** for your designed Vending Machine.
A3: Write **testbench** to perform simulation of your designed circuit. **Your testbench should cover at least 5 cases.** Below are suggestions of cases that you can use to create a testbench.

- ***No coin*** entered at any state in your state diagram
- ***More than 1 coins*** are entered at any state in your state diagram
- ***Sum of exact 25 cents*** is entered (again, one coin entered at a time). In this case, your machine should set Candy to 1 and Number to 0 (since no change returned).
  It can be 5 Nickels, 2 Dimes and a Nickel, or 1 Quarter.
- ***Sum of 30 cents*** is entered. For 30 cents, your machine should set Candy to 1 and Number to 5 (5 cents are returned since the candy cost 25 cents). It can be 3 Dimes, 2 Nickels and then 2 Dimes or 1 Nickel and 1 Quarter (in this order).
- ***Sum of 35 cents*** is entered. For 35 cents, your machine should set Candy to 1 and Number to 10 (10 cents are returned since the candy cost 25 cents).
  It can be a combination of 1 Dime and 1 Quarter or 2 Nickels and 1 quarter (in this order)
- ***Sum of 40 cents*** is entered. For 40 cents, your machine should set Candy to 1 and Number to 15 (15 cents are returned since the candy cost 25 cents).
  A combination of Nickel and Dime for 15 cents and then 1 Quarter (in this order).
- ***Sum of 45 cents*** is entered. For 40 cents, your machine should set Candy to 1 and Number to 20 (20 cents are returned since the candy cost 25 cents).
  A combination of Nickel and Dime for 20 cents and then 1 Quarter (in this order).

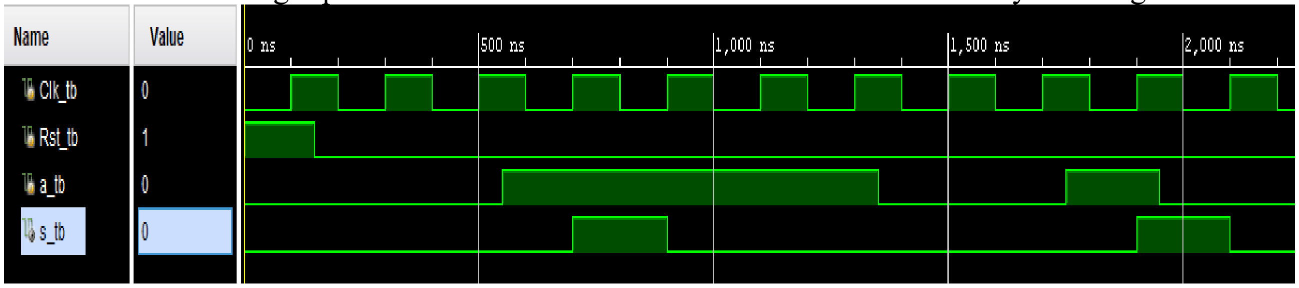
**Show the waveform of Post-Synthesis functional simulation to TA to get credits**

**Button synchronizer or Level to Pulse converter**

This sequential circuit, Button Synchronizer, converts button press to a **single cycle** duration, ***regardless of length of time that the button is actually pressed***.

A4: Use the state diagram given in section 3.8 of zyBooks: Participation activity 3.8.7 and 3.8.8 to write Verilog code.

A5: Use the following input waveform to write a testbench for a simulation of your designed circuit.



@ 1st rising edge of the clock (Clk_tb), Rst_tb = 1, the FSM is at the initial state
@ 2nd and 3rd rising edge of Clk_tb, Rst_tb = 0 and a_tb = 0 -> s_tb = 0
@ 4th rising edge of Clk_tb, a_tb = 1, -> now s_tb = 1.
@ 5th rising edge of Clk_tb, even though a_tb is still 1, s_tb is now back to 0. This is what it means by the statement "it converts a button press to a **single cycle** duration, ***regardless of length of time that the button is actually pressed"***

If working, you should get the same output waveform as shown above.
**Show the waveform of Post-Synthesis functional simulation to TA/ULA/Instructor to get credits**

**Part B:**
To make the simple Vending machine work on the Nexys 4 DDR board, we have to use more components as shown Figure 1 the top-level design below. This **top-level design (.v file)** has

| | |
|---|---|
| **5 inputs**: | CLK100MHZ (connected to pin E3 for 100 MHz signal), |
| | BTNU - Up push button) is Reset |
| | BTNL - Left push button is Nickel |
| | BTNC - Center push button is Dime |
| | BTNR - Right push button is Quarter |
| **3 outputs:** | LED[0] is Candy |
| | CA, CB, CC, CD, CE, CF, CG; //segment a, b, ... g |
| | AN[7:0]; to enable each digit of the 8 digits |

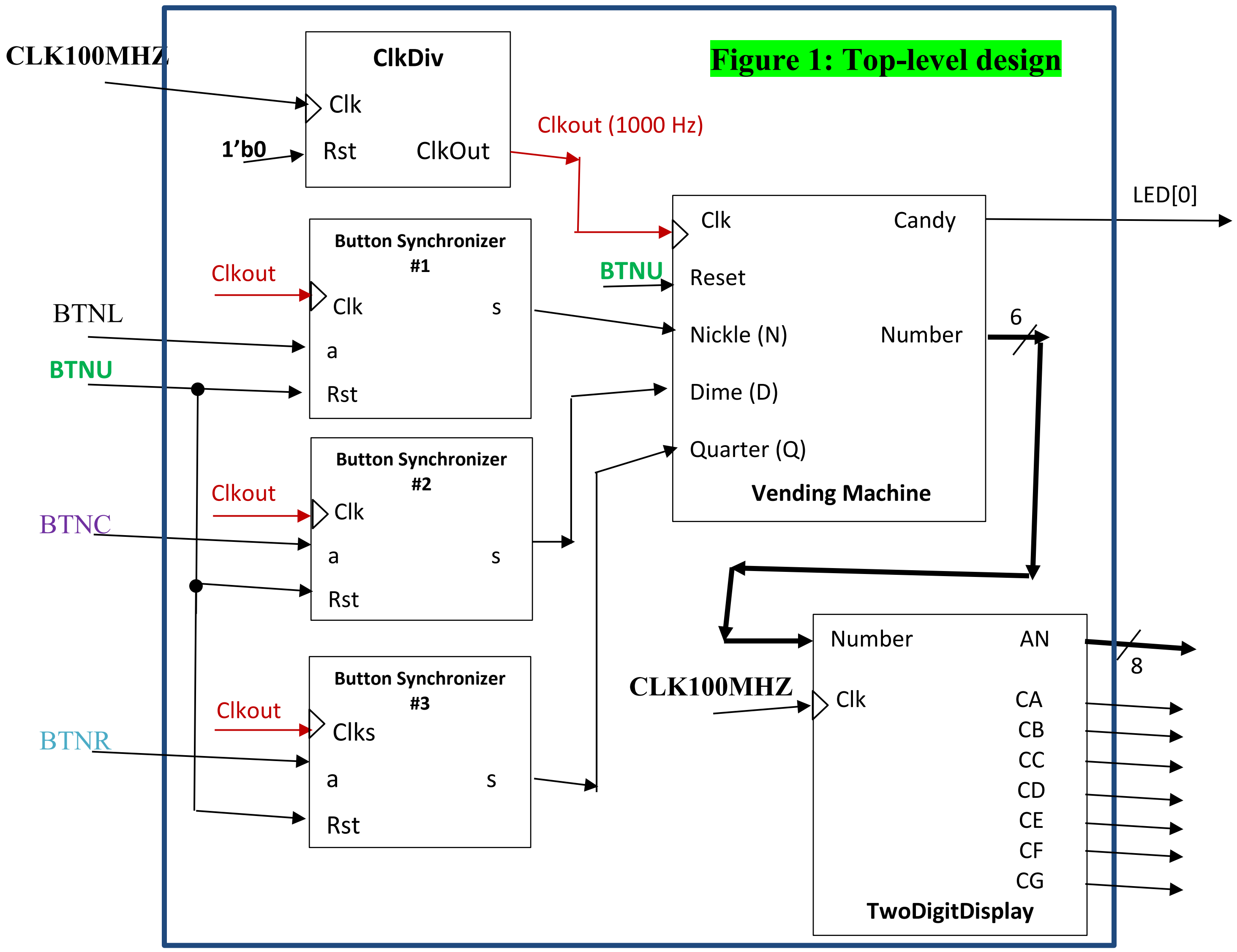


Figure 1: Top-level design

- The ***button synchronizer*** or level to pulse converter is to ensure that the vending machine see only one pulse (one clock cycle) of nickel, dime, or quarter no matter how long the user pushes a button.

Without the button synchronizer, the vending machine will see more than one coin entered since the machine run at 1000 Hz (faster than the speed that we can push and release a button).

- ***TwoDigitDisplay*** which use your SevenSegment code is used to display Number (6-bit output of your Vending machine). Note that TwoDigitDisplay uses CLK100MHZ.

**Action items for Part B**

1) Get the following .v files and put them in the .srcs folder of this lab4 project
   - ClkDiv.v file (from Lab4_code_used_inPart_B zipped folder on D2L lab 4 module)
   - TwoDigitDisplay.v (from Lab4_code_used_inPart_B zipped folder on D2L lab 4 module)
   - sevensegment.v files (from your lab 2)

**Use Add Sources to add the above .v files into your lab4 Vivado project.**

2) **ClkDiv:** It has two inputs: Clk and Rst, and one output: ClkOut.

**MODIFY the DivVal value such that ClkOut has a frequency of 1000 Hz**

From ClkDiv.v code, below is the relationship between frequency of Clk and ClkOut.

$$\text{Frequency of ClkOut (in Hz)} = \frac{\text{Frequency of Clk}}{2*(\text{DivVal}+1)}$$

The oscillator used on the FPGA board is 100MHz = 100*1e6 Hz. This is connected to pin E3.
If DivVal = 49999999 (in the code), then

$$\text{Frequency of ClkOut (in Hz)} = \frac{100*1e6}{2(49999999+1)} = \frac{100*1e6}{2*50*1e6} = 1$$

3) Create new .v file to implement the top-level design.
You may have to set this new .v file as Top module (right-click on the file_name under Design Sources and choose "Set as Top"

)

Write ***Structural*** Verilog code to connect the modules together as shown in Figure 1 (page 4)

Use the following as ***the code template for the top-level code***.
Note: Lab4 is the name that I use for my top-level code. You can use any name that you want BUT it <u>must not</u> be the same name that you use for your vending machine Verilog code.

```
module Lab4(CLK100MHZ, BTNU, BTNL, BTNC, BTNR, LED, CA, CB, CC, CD, CE, CF, CG, AN);
   input CLK100MHZ;
   input BTNU; //BTNU is Reset
   input BTNL, BTNC, BTNR; //BTNL is Nickel, BTNC is Dime, BTNR is Quarter
   output [0:0] LED; //LED[0] is Candy
   output CA, CB, CC, CD, CE, CF, CG; //segment a, b, ... g
   output [7:0] AN; //enable each digit of the 8 digits

  //write your code to connect the modules as shown in Figure 1

endmodule
```

**Note: You do not have to simulate the top-level code.** If you want to, you can perform a simulation as follows: *Before* the simulation, first change DivVal value in ClkDiv to 1 (instead of 50000000 or the number that you find more ClkOut of 1000 Hz. ***After*** the simulation, change DivVal back to the number that you find for 1000 Hz.

4) **Get Nexys4DDR_Master.xdc file** (from Lab4_code_used_inPart_B zipped folder on D2L Lab4 module) and Uncomment the pins that you will use for the inputs and outputs of the top-level code.

5) Generate Bitstream and program the FPGA board. TA will test that your Vending machine works. Note: The reset button must be pushed in order to start over after the machine gets to the state that releases the candy.

**Grading Criteria**

**Part A**

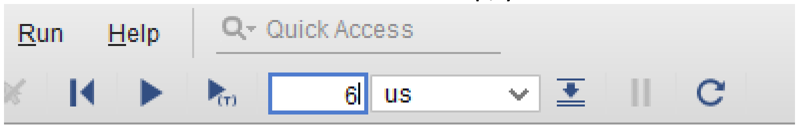
| | |
|---|---|
| - State diagram of Vending Machine | 8 pts |
| - Verilog code for the Vending machine | 22 pts |
| - Testbench for Vending Machine (at least 5 cases) and correct waveform from Post synthesis functional simulation | 15 pts |
| - Verilog and testbench for the Button synchronizer and correct waveform from Post synthesis functional simulation | 15 pts |

**Part B**

| | |
|---|---|
| - Correct value of DivVal in ClkDiv for a frequency of 1000 Hz for ClkOut | 2 pts |
| **-** Structural Verilog code for Top-level design | 18 pts |
| **-** Circuit working on the Nexys 4 DDR board | 20 pts |

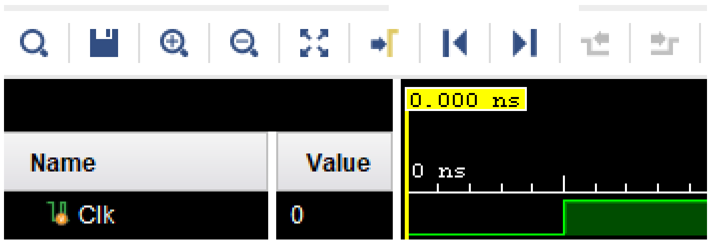
**How to use Vivado features:**

**1) To run a simulation waveform longer than a default 1000 ns**
If you use #100 to generate the clock signal in the testbench, this means that one clock cycle/period is 200 ns. If you want to see the simulation for every scenarios (5 scenarios and each scenarios uses 6 clock cycles) as discussed in Step 3 of part A, you will need at least 5*6*200 = 6000 ns = 6 us (microsecond).
When the simulation waveform show up, you will see below:



1) Click on the box above to change the value to any number, the above shows 6 and use the dropdown menu to change the unit to us (microsecond). 6 us is 6 ns.
   You can change it to any number and unit that you want until you can see the simulation that you desire.
2) Click to clear the waveform.
3) Click to run the simulation for 6 us. Note: if you click again, it will run for another 6 us.
   To put the entire simulation in the simulation window, click to zoom fit (see the explanation below).

**2) Menu bar of the waveform simulation**



- Click to bring the yellow vertical bar to the beginning of the waveform (at time 0.00 ns).

Note: When you click zoom in or zoom out , it does the zoom at the location where the yellow vertical bar is at.

- is to zoom fit. It will fit the entire simulation in the simulation window.

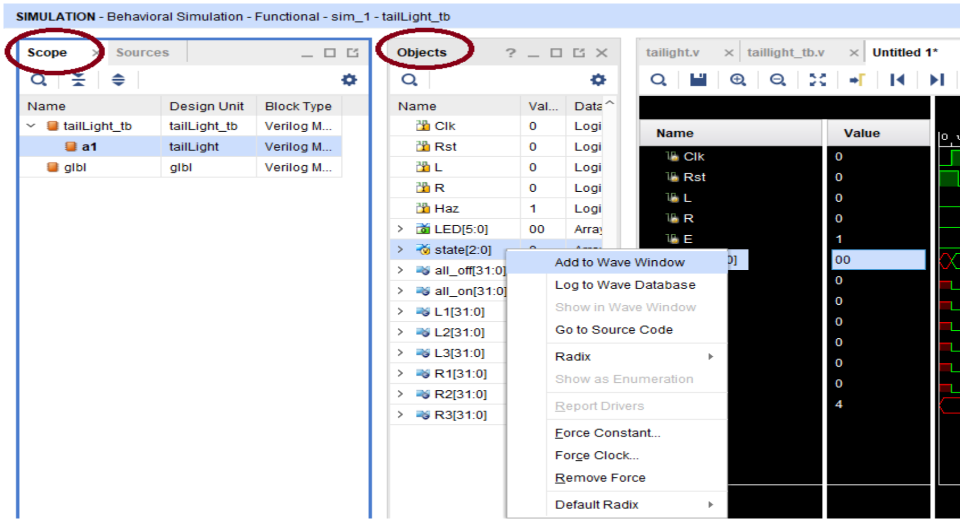
**3) One way to debug your circuit, when you run Behavioral simulation,** is to add ***Internal*** signals (signals that are NOT inputs and outputs of the module) such as state of the sequential circuit.

**To add internal signal (such as state) to the waveform in Behavioral simulation**

On “Scope” window, click > to expand so that the module that you want to add the signal is

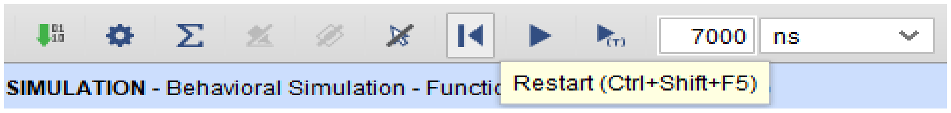
shown

On “Objects” window, Right-click on the signal that you want to add to the waveform and choose ***Add to Wave Window***.



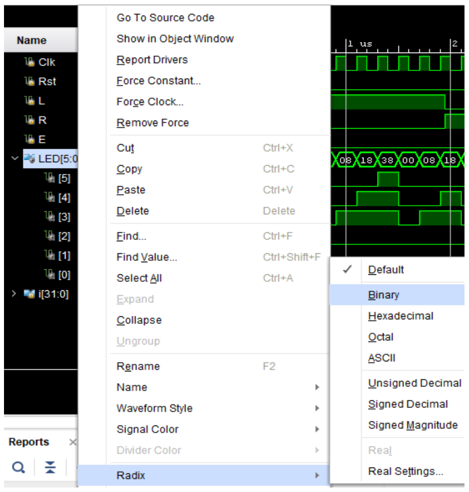
After you add the internal signal such as state, if its waveform does not show up, click on ***Restart*** button (the arrow pointing to the right).

Then type the time to the duration that you want to run the simulation (in the picture below, it shows 7000 ns), then click the ***play button with (T)***. You will see your simulation waveform runs from 0 to 7000ns



**4) To change the base number (Radix) for the vector signal**

By default, the vector signal shows its value in hexadecimal. Right-click on the vector signal name on the waveform, choose Radix, then choose the base-number that you want to use.


Go To Source Code
Show in Object Window
Report Drivers
Force Constant...
Force Clock...
Remove Force
Cut
Copy
Paste
Delete
Find...
Find Value...
Select All
Expand
Collapse
Ungroup
Rename
Name
Waveform Style
Signal Color
Divider Color
Radix
Default
Binary
Hexadecimal
Octal
ASCII
Unsigned Decimal
Signed Decimal
Signed Magnitude
Real
Real Settings...