CSC 343   Spring 2022

Self-Check Laboratory Exercise 4B

Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: ***Professor Izidor Gertner***

What to DO by March 14, 2022:

Create a N-bit register component for your future use. Verify write and read operation using waveforms in Model-Sim

What to Submit:

Please post on direct private channel to Instructor just Figures with **SCREENSHOTS** of waveforms. The waveforms should demonstrate the correctness of N-bit storage operation, write, read. Figure Captions should state **IN ONE SENTENCE** why the N-bit register works correctly. The Register component name should have our last name as a prefix. The file name has to have your last name and title.All signals have to have your last name as a prefix.

No report, nor video is required to submit.

No grade will be given for this Self-Check lab.

A Check Mark **will be assigned. The criteria used for Check Mark** (✓) A **check mark**, **checkmark** or **tick** (✓) is a **mark** used to indicate the concept "yes" (e.g. "yes; this has been verified", "yes; that **is the** correct answer", "yes; this has been completed".

This tutorial review has three parts:

Design N-bit Register

In previous guides we created simple latch and flip flop storage devices. Recall that a basic flip flop can store 1 bit of data and performs this function on a clock edge – either rising (changing from 0 to 1) or falling (changing from 1 to 0) depending on how the device is designed. In this guide we will extend this concept by creating an **N-bit Register**.

Create a new VHDL file, and let's begin designing this register. Note that the full source code will be shown at the end of this section.

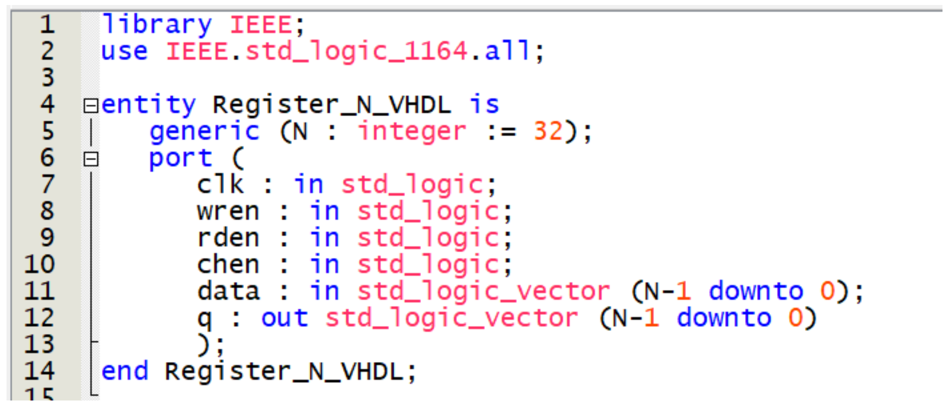
CSC 343   Spring 2022

Self-Check Laboratory Exercise 4B

Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: ***Professor Izidor Gertner***



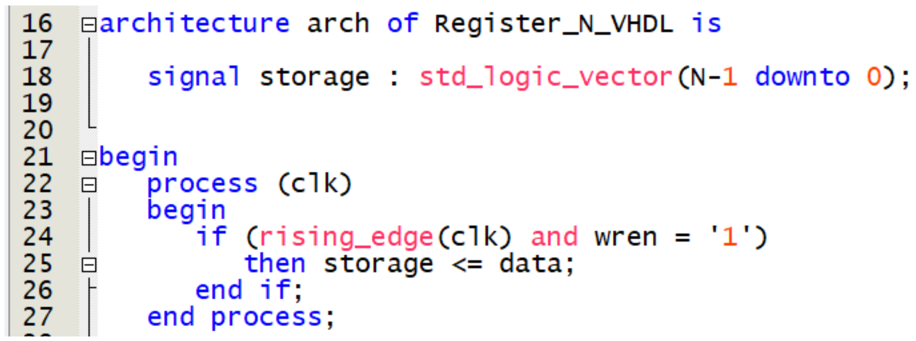
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Register_N_VHDL is
    generic (N : integer := 32);
    port (
        clk : in std_logic;
        wren : in std_logic;
        rden : in std_logic;
        chen : in std_logic;
        data : in std_logic_vector (N-1 downto 0);
        q : out std_logic_vector (N-1 downto 0)
        );
end Register_N_VHDL;
```

In this first section of code, we are defining the inputs and outputs of this device. For future convenience we are using a VHDL feature called **generics**. This permits us to change the size of this register design easily, with the generic value we created called **N.** Above, I have N set to 32, so this time we will experiment with a 32 bit register.

The inputs and outputs are:

1. clk – clock signal for this circuit
2. wren – write enable (if it is 0, the stored data in the register will not change.)
3. rden – read enable (only when it is 1, the stored data will be displayed to the output.)
4. chen – chip enable (if it is 0, the output of this device will be undefined)
5. data – data input
6. q – output

The functionality of our register is straight forward but there are more details left to explore. Let's now define the behavior of this circuit and deepen our understanding.



```vhdl
architecture arch of Register_N_VHDL is

    signal storage : std_logic_vector(N-1 downto 0);


begin
    process (clk)
    begin
        if (rising_edge(clk) and wren = '1')
            then storage <= data;
        end if;
    end process;
```

CSC 343 Spring 2022

Self-Check Laboratory Exercise 4B

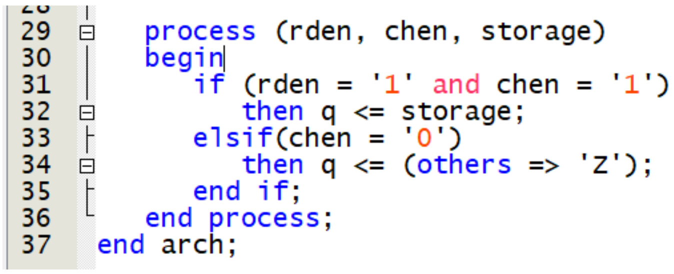
Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: ***Professor Izidor Gertner***

On line 18, we are setting up a storage space within the register that is separate from the output. Note that the output **q** is essentially just a display; the bulk of our work will be coordinating how the data input flows into the storage space and then to the output.

On line 22 we define a process that is sensitive to changes in clock value. When defining a process, the parenthesis that come after it allow us to define our **sensitivity list**. The behavior within should appear familiar to you, to summarize, so far we have defined a 32 bit wide flip flop that is rising edge triggered.



```
   process (rden, chen, storage)
   begin
      if (rden = '1' and chen = '1')
         then q <= storage;
      elsif(chen = '0')
         then q <= (others => 'Z');
      end if;
   end process;
end arch;
```

This second process needs to be responsive to changes in read enable, chip enable, and the storage space. It may not be immediately evident why changes in the storage vector should trigger this process. We will answer this later when we run simulations. 'Z' is a common undefined logic value used for circuits, using the **others** feature allows us to set every bit in **q** as undefined.

Find the full source code on the next page.

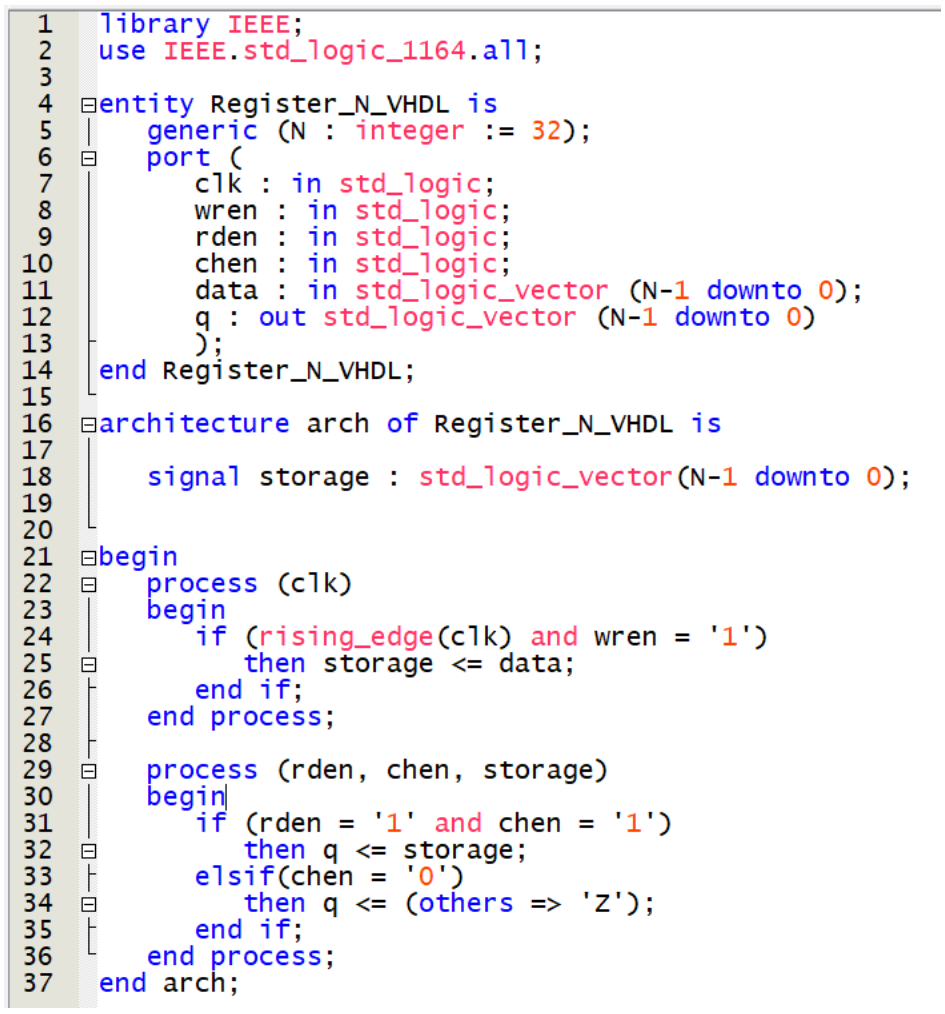
CSC 343 Spring 2022

Self-Check Laboratory Exercise 4B

Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: Professor Izidor Gertner



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Register_N_VHDL is
    generic (N : integer := 32);
    port (
        clk : in std_logic;
        wren : in std_logic;
        rden : in std_logic;
        chen : in std_logic;
        data : in std_logic_vector (N-1 downto 0);
        q : out std_logic_vector (N-1 downto 0)
        );
end Register_N_VHDL;

architecture arch of Register_N_VHDL is

    signal storage : std_logic_vector(N-1 downto 0);


begin
    process (clk)
    begin
        if (rising_edge(clk) and wren = '1')
            then storage <= data;
        end if;
    end process;

    process (rden, chen, storage)
    begin
        if (rden = '1' and chen = '1')
            then q <= storage;
        elsif(chen = '0')
            then q <= (others => 'Z');
        end if;
    end process;
end arch;
```

ModelSim Simulation

In ModelSim, if you are not in a project already, create a new one.

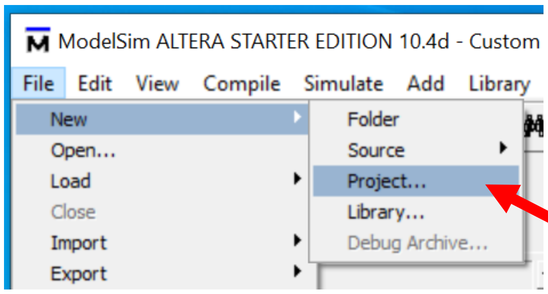
CSC 343   Spring 2022

Self-Check Laboratory Exercise 4B

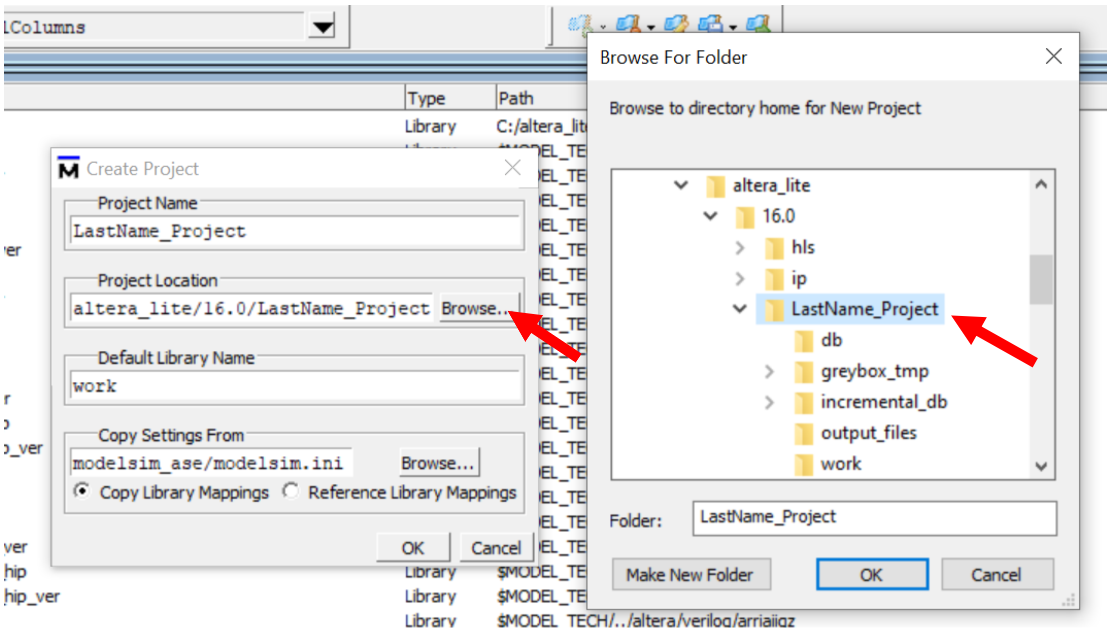
Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: ***Professor Izidor Gertner***



Make sure to **Browse...** for the folder where you have the register VHDL.



Create Project

Project Name: LastName_Project

Project Location: altera_lite/16.0/LastName_Project

Default Library Name: work

Copy Settings From: modelsim_ase/modelsim.ini

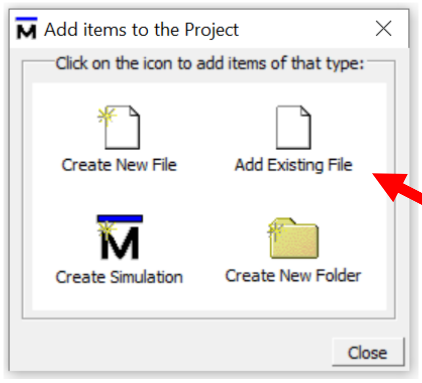
(● Copy Library Mappings) ○ Reference Library Mappings

Browse For Folder

Browse to directory home for New Project

Folder: LastName_Project

Once the project is set up add the register VHDL as an existing file, then right click on it within the **Project** tab and compile it.



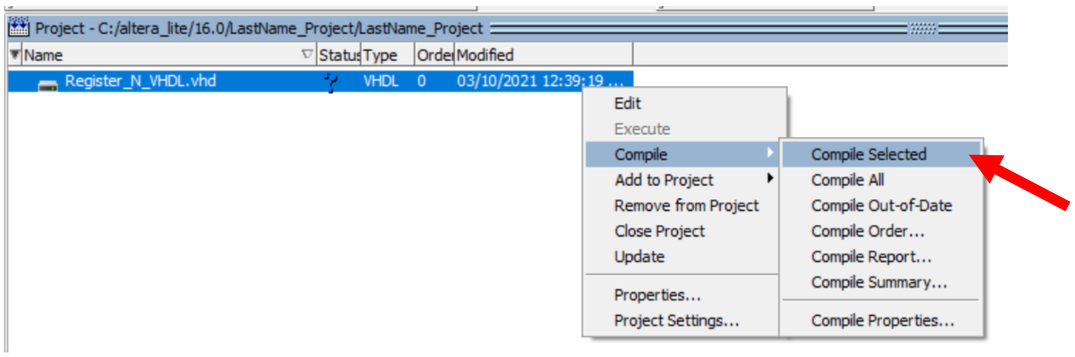
CSC 343 Spring 2022

Self-Check Laboratory Exercise 4B

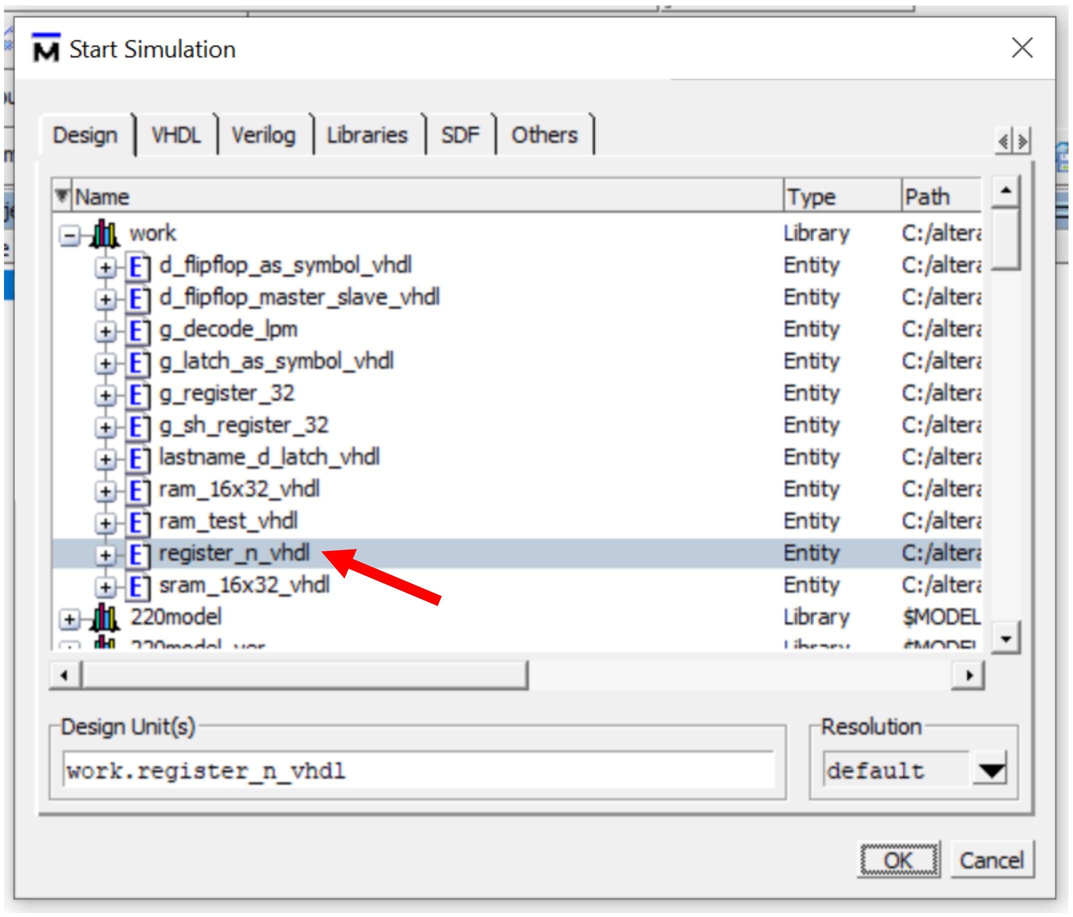
Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: Professor Izidor Gertner



In the top menu find **Simulate > Start Simulation...** and select the VHDL file from the default user library called **work**.



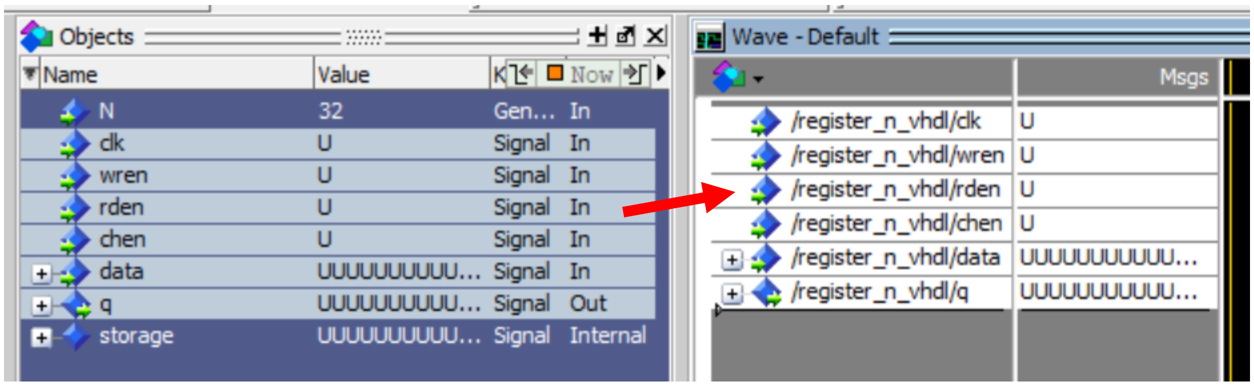
Drag the relevant signals to the wave tab.

Self-Check Laboratory Exercise 4B

Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

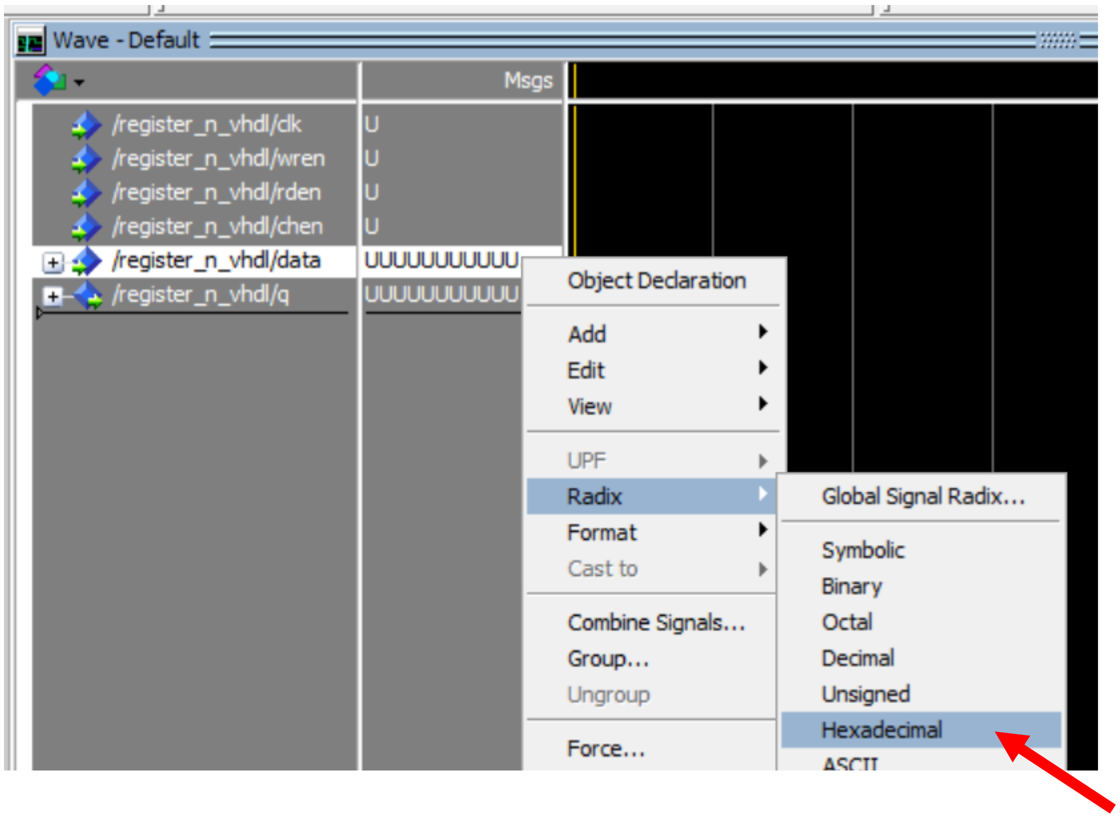
Instructor: Professor Izidor Gertner



Name	Value	Kind	Mode
N	32	Gen...	In
clk	U	Signal	In
wren	U	Signal	In
rden	U	Signal	In
chen	U	Signal	In
data	UUUUUUUUUU...	Signal	In
q	UUUUUUUUUU...	Signal	Out
storage	UUUUUUUUUU...	Signal	Internal

Wave - Default	Msgs
/register_n_vhdl/clk	U
/register_n_vhdl/wren	U
/register_n_vhdl/rden	U
/register_n_vhdl/chen	U
/register_n_vhdl/data	UUUUUUUUUUU...
/register_n_vhdl/q	UUUUUUUUUUU...

For visual convenience we can change the signal value displays to hexadecimal by right clicking on them and changing the radix.



Wave - Default	Msgs
/register_n_vhdl/clk	U
/register_n_vhdl/wren	U
/register_n_vhdl/rden	U
/register_n_vhdl/chen	U
/register_n_vhdl/data	UUUUUUUUUUU
/register_n_vhdl/q	UUUUUUUUUUU

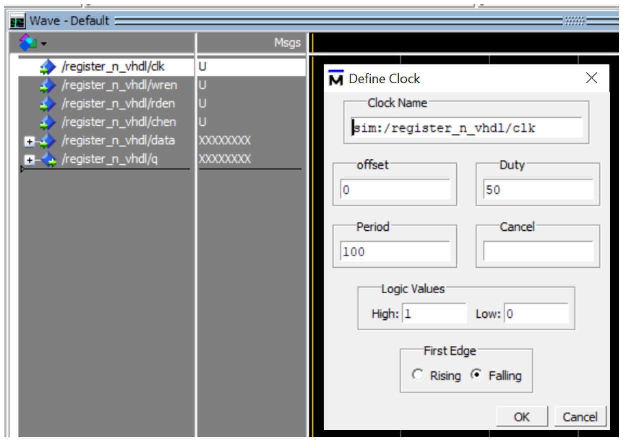
For this first simulation you can follow along with how I set the inputs up below (right click the signal in the wave tab and **Force…** or **Clock…** as needed)

I like to keep the clock at the standard **Clock…** setup but with a falling first edge.

Self-Check Laboratory Exercise 4B
Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: ***Professor Izidor Gertner***



Wave - Default

	Msgs
/register_n_vhdl/clk	U
/register_n_vhdl/wren	U
/register_n_vhdl/rden	U
/register_n_vhdl/chen	U
/register_n_vhdl/data	XXXXXXXX
/register_n_vhdl/q	XXXXXXXX

Define Clock

Clock Name: sim:/register_n_vhdl/clk

offset: 0

Duty: 50

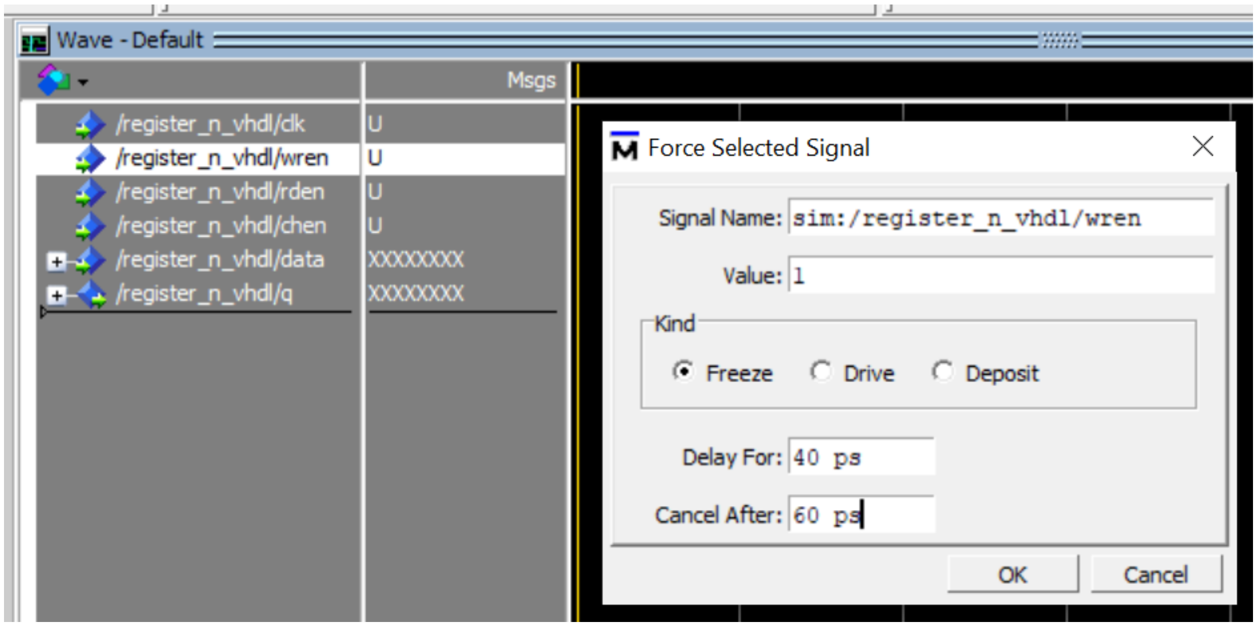
Period: 100

Cancel:

Logic Values — High: 1 Low: 0

First Edge: Rising [✓ Falling]

With the clock set up like that, the rising edges will be at 50 ps, 150 ps, 250 ps, etc. so I have the write enable forced to 1 from 40 ps to 60 ps.



Wave - Default

	Msgs
/register_n_vhdl/clk	U
/register_n_vhdl/wren	U
/register_n_vhdl/rden	U
/register_n_vhdl/chen	U
/register_n_vhdl/data	XXXXXXXX
/register_n_vhdl/q	XXXXXXXX

Force Selected Signal

Signal Name: sim:/register_n_vhdl/wren

Value: 1

Kind: [✓ Freeze] Drive Deposit

Delay For: 40 ps

Cancel After: 60 ps

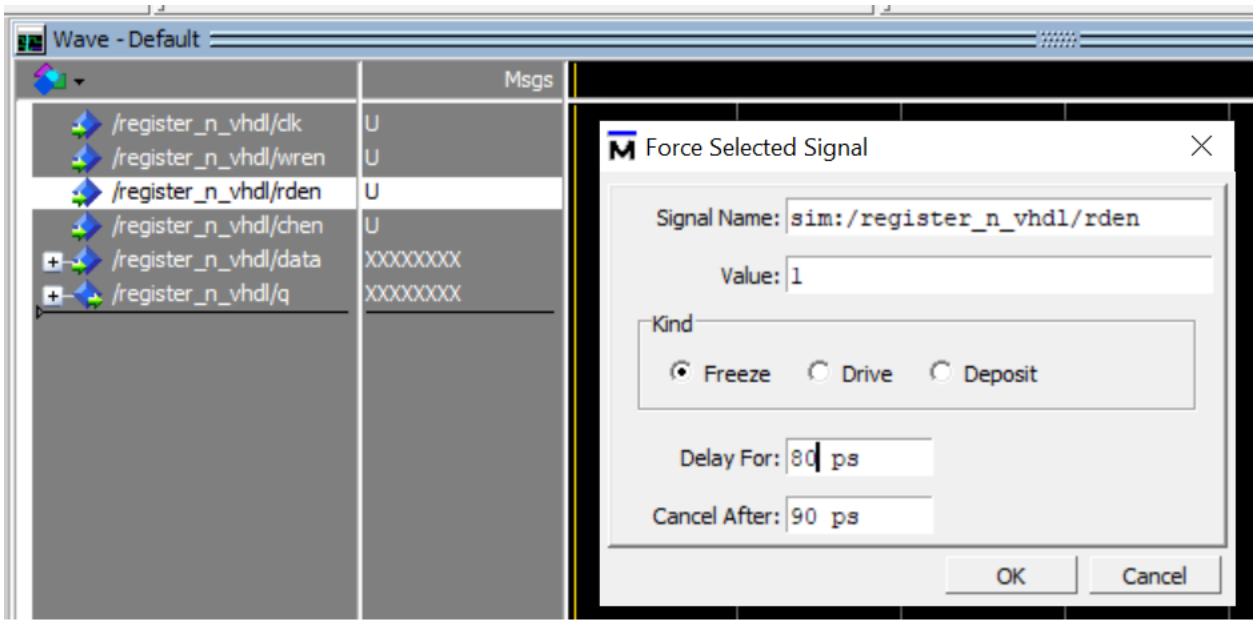
At 80 ps I force the read enable to 1, so the data we just wrote to storage will be displayed in the output.

Self-Check Laboratory Exercise 4B

Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: Professor Izidor Gertner



Wave - Default

	Msgs
/register_n_vhdl/clk	U
/register_n_vhdl/wren	U
/register_n_vhdl/rden	U
/register_n_vhdl/chen	U
/register_n_vhdl/data	XXXXXXXX
/register_n_vhdl/q	XXXXXXXX

Force Selected Signal

Signal Name: sim:/register_n_vhdl/rden

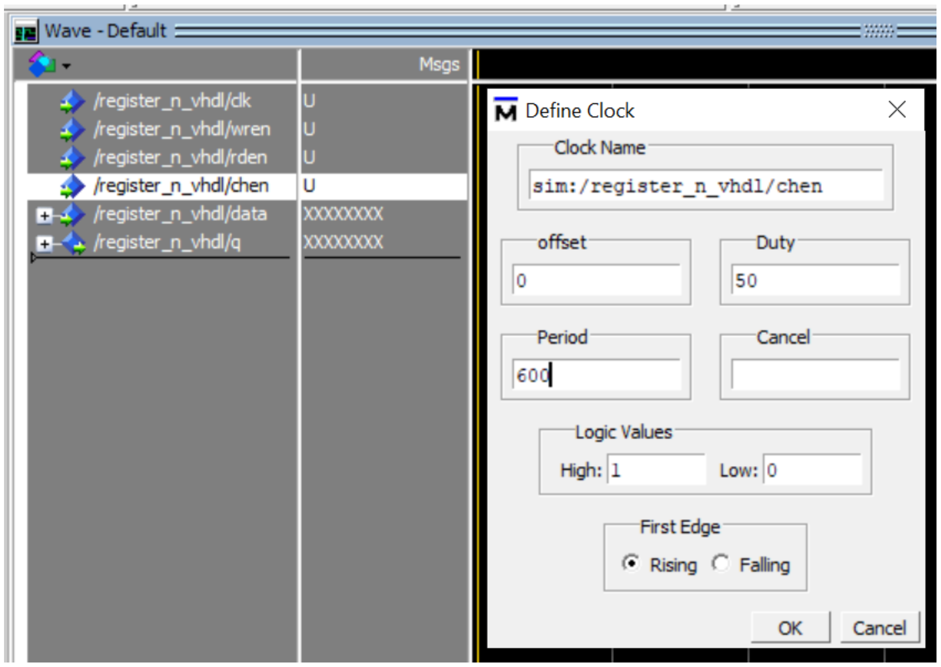
Value: 1

Kind: [✓ Freeze] Drive Deposit

Delay For: 80 ps

Cancel After: 90 ps

Just for demonstration I have the chip enable changing from 1 to 0 at 300 ps. I do this by setting it up as a **Clock…** with a period of 600 ps and a rising first edge.



Wave - Default

	Msgs
/register_n_vhdl/clk	U
/register_n_vhdl/wren	U
/register_n_vhdl/rden	U
/register_n_vhdl/chen	U
/register_n_vhdl/data	XXXXXXXX
/register_n_vhdl/q	XXXXXXXX

Define Clock

Clock Name: sim:/register_n_vhdl/chen

offset: 0

Duty: 50

Period: 600

Cancel:

Logic Values: High: 1 Low: 0

First Edge: [✓ Rising] Falling

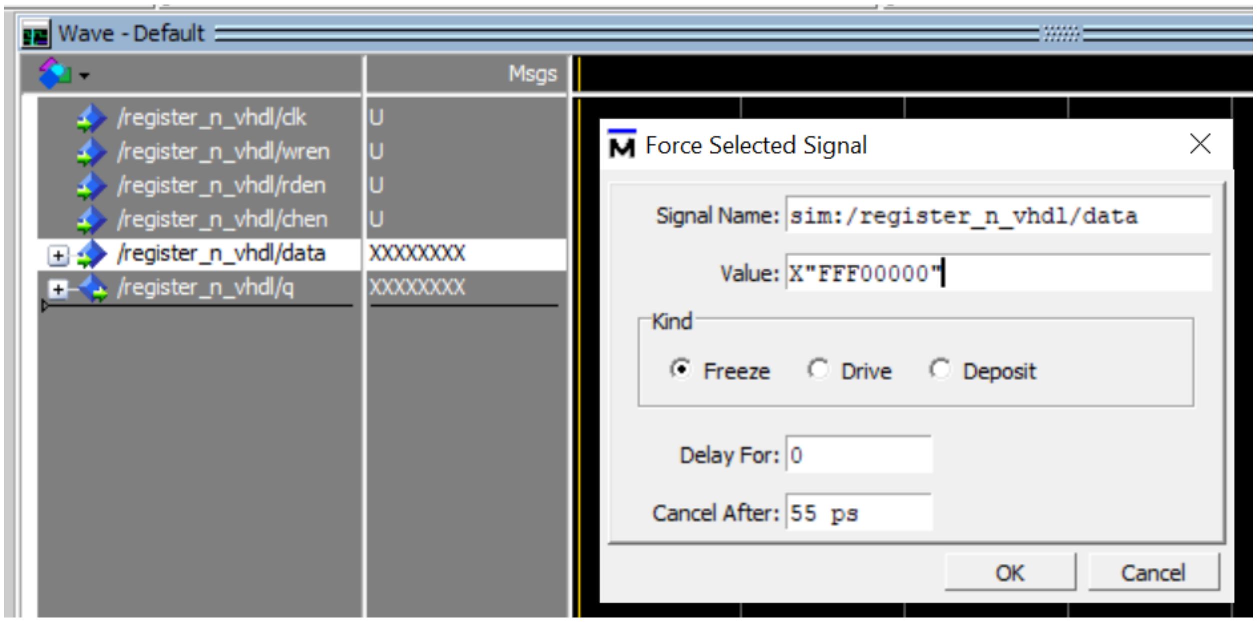
Self-Check Laboratory Exercise 4B

Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

Instructor: ***Professor Izidor Gertner***

I set up the data so it is active during the write we are planning to do at 50 ps. I have the data set to an arbitrary hexadecimal value – FFF00000. The X prefix and quotations tell ModelSim that we are defining a hexadecimal.



```
/register_n_vhdl/clk    U
/register_n_vhdl/wren   U
/register_n_vhdl/rden   U
/register_n_vhdl/chen   U
/register_n_vhdl/data   XXXXXXXX
/register_n_vhdl/q      XXXXXXXX
```

Force Selected Signal

Signal Name: sim:/register_n_vhdl/data

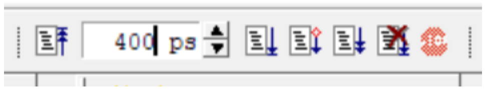
Value: X"FFF00000"

Kind: [✓ Freeze] Drive Deposit

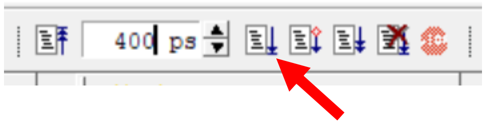
Delay For: 0

Cancel After: 55 ps

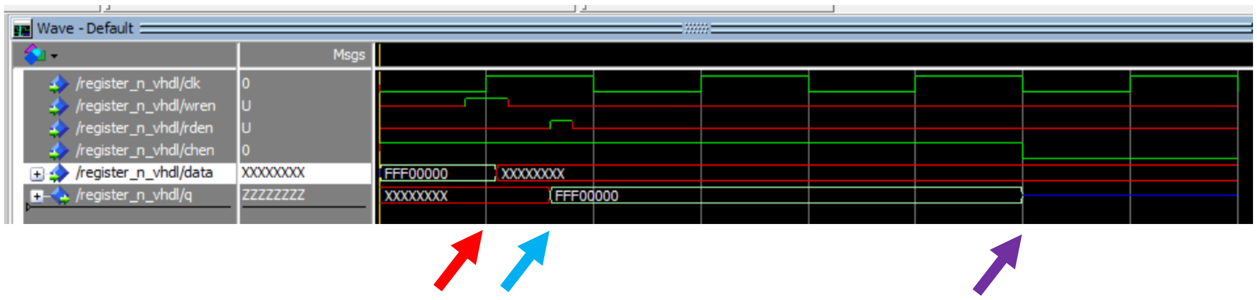
In the toolbar I set the simulation run time for 400 ps, we will observe what we need to by that time.



We can actually run our simulation with this toolbar.



Your results should look just like this.



```
/register_n_vhdl/clk    0
/register_n_vhdl/wren   U
/register_n_vhdl/rden   U
/register_n_vhdl/chen   0
/register_n_vhdl/data   XXXXXXXX
/register_n_vhdl/q      ZZZZZZZZ
```

Self-Check Laboratory Exercise 4B

Creating an N-bit Register Component

Simulation and verification with ModelSim Simulation

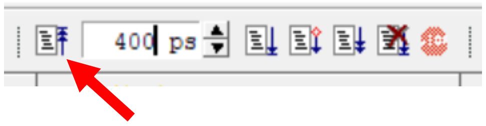
Instructor: ***Professor Izidor Gertner***

At the time of the red arrow, the data input FFF00000 is loaded into the storage space behind the scenes as the clock rises.

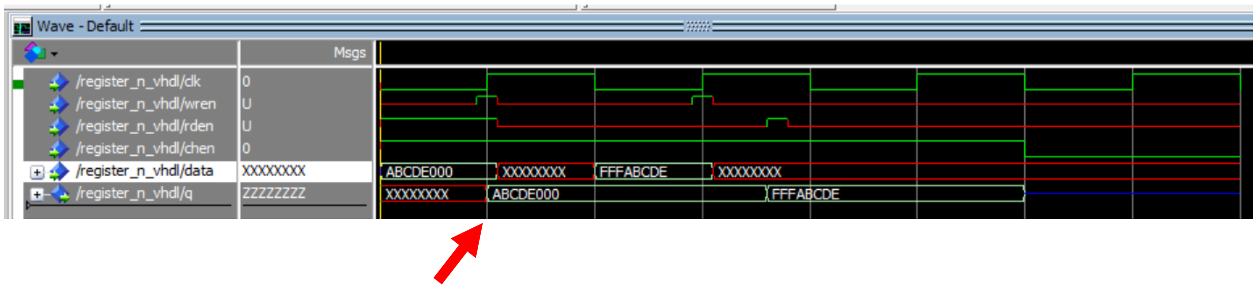
At the time of the blue arrow, the storage which now contains FFF00000 is displayed in the output because we forced read enable to be 1 at that moment.

At the time of the purple arrow, the chip enable moves from 1 to 0, turning the output to undefined ('Z')

We can use this button on the left of the toolbar to start over and run more simulations.



I performed multiple writes and reads this time, for example I have write enable forced to 1 from 45 ps to 55 ps, and also at 145 ps and 155 ps.



Recall that our second **process** in the code had to be responsive to changes in read enable, chip enable, and the storage space. At the time of the red arrow, read enable and chip enable have been forced to 1 since the beginning of the simulation. I perform a write on the rising clock edge there, which changes what is within the storage space. Since we included the storage space in the sensitivity list, even though neither read enable or chip enable changed in value at that moment, the output **q** still immediately reads the new data.

Please experiment to perform another write, another read, then a chip disable.