ModelSim Tutorial for EECS361

Fall 2014 Kaicheng Zhang(kaichengz@u.northwestern.edu)

Introduction

This tutorial will build a 1-bit full adder and a SRAM reader in VHDL. It uses ModelSim as the compiler and the simulator. The purpose of this tutorial has two folds:
1) Give an example on how to create a new project, write VHDL codes, compile the project and do simulations in ModelSim.
2) Give an example on how to utilize the component provided by the course library.

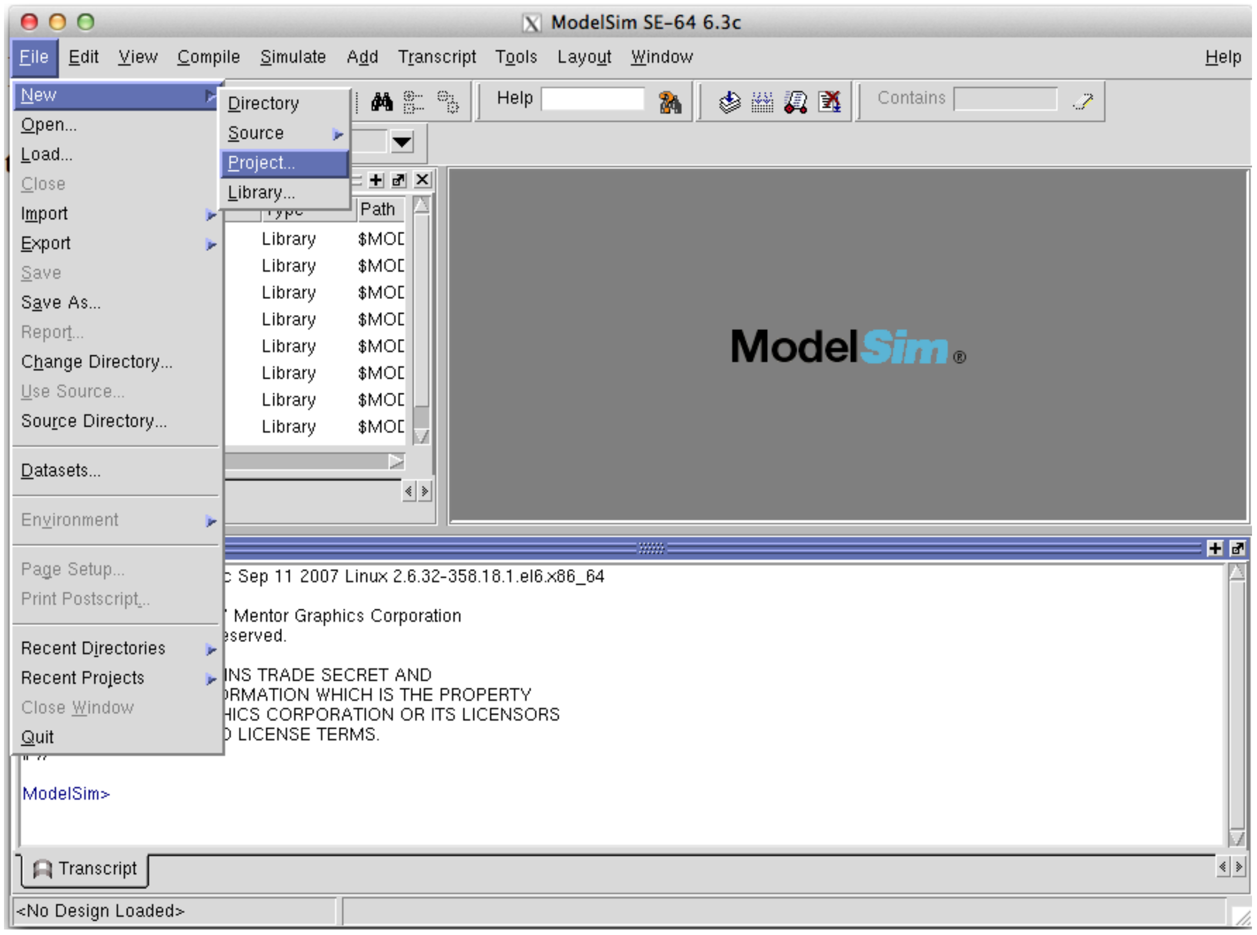
1-bit Full Adder

Start ModelSim

1. Login to one of the linux machines in Wilkinson Lab.
2. Open the terminal.
3. type the following command:
```
source /vol/ece303/ece303.env
```
4. Start ModelSim by typing:
```
vsim
```

Create a new project

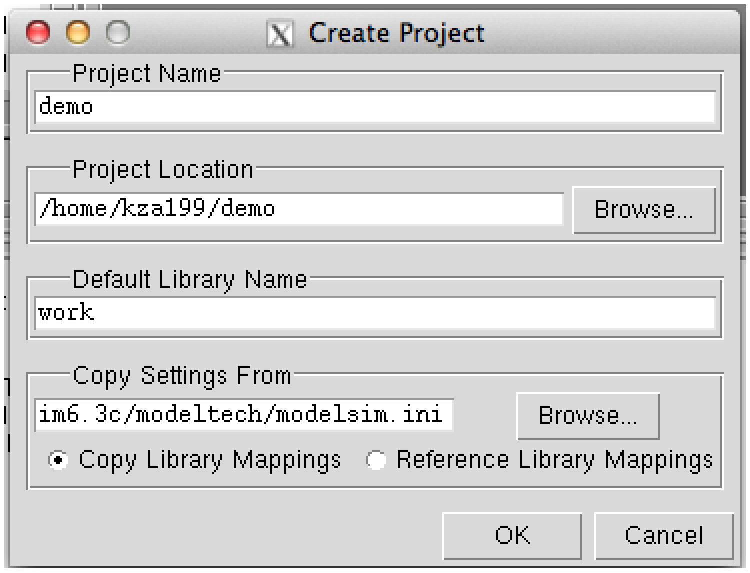
1. Select File -> New -> Project…



```
c Sep 11 2007 Linux 2.6.32-358.18.1.el6.x86_64

' Mentor Graphics Corporation
eserved.

INS TRADE SECRET AND
)RMATION WHICH IS THE PROPERTY
HICS CORPORATION OR ITS LICENSORS
) LICENSE TERMS.

ModelSim>
```

<No Design Loaded>

2. Fill in the “Project Name” and “Project Location”. The “Project Location” is where you want to put your project in. Leave the other fields unchanged. Click OK. If a new dialogue pops out asking you to create the directory, click OK.



Project Name: demo

Project Location: /home/kza199/demo

Default Library Name: work

Copy Settings From: im6.3c/modeltech/modelsim.ini

[✓ Copy Library Mappings] Reference Library Mappings

3. ModelSim will ask you to add items to the project right after your project is created. You can do

it now or do it later. In this tutorial, we simply close this dialogue.

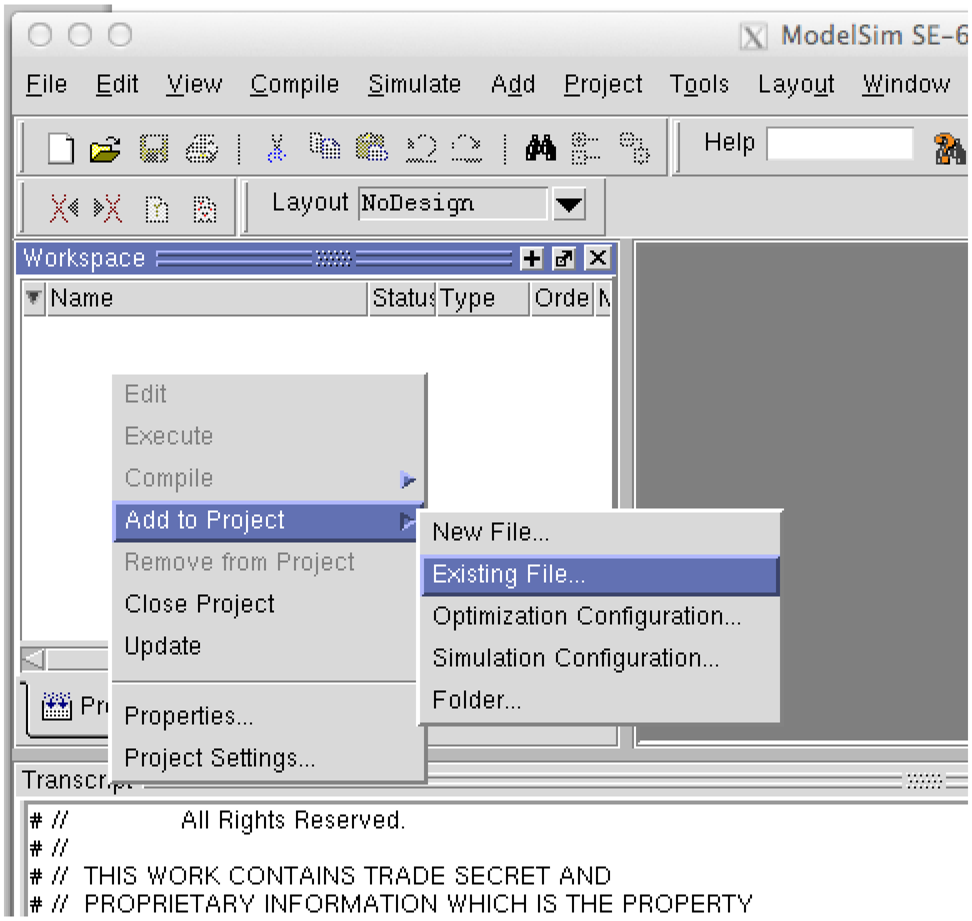
Add EECS 361 library files to the project

We have a number of gates and other components ready for you to use.

1. Copy all the files in “lib” directory of the library package to your own project’s directory. Here is the full list of the files:

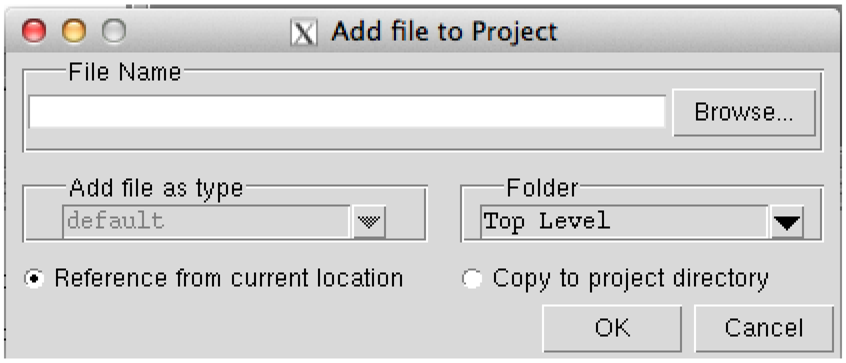
and_gate_32.vhd
and_gate_n.vhd
and_gate.vhd
dec_n.vhd
dffr.vhd
dff.vhd
eecs361_gates.vhd
eecs361.vhd
mux_32.vhd
mux_n.vhd
mux.vhd
nand_gate_32.vhd
nand_gate_n.vhd
nand_gate.vhd
nor_gate_32.vhd
nor_gate_n.vhd
nor_gate.vhd
not_gate_32.vhd
not_gate_n.vhd
not_gate.vhd
or_gate_32.vhd
or_gate_n.vhd
or_gate.vhd
sram.vhd
syncram.vhd
xnor_gate_32.vhd
xnor_gate_n.vhd
xnor_gate.vhd
xor_gate_32.vhd
xor_gate_n.vhd
xor_gate.vhd

2. In the Workspace window, right click, select “Add to Project” -> “Existing File…”.



```
# //              All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
```

2. Click on “Browse…” to open the file selection window.

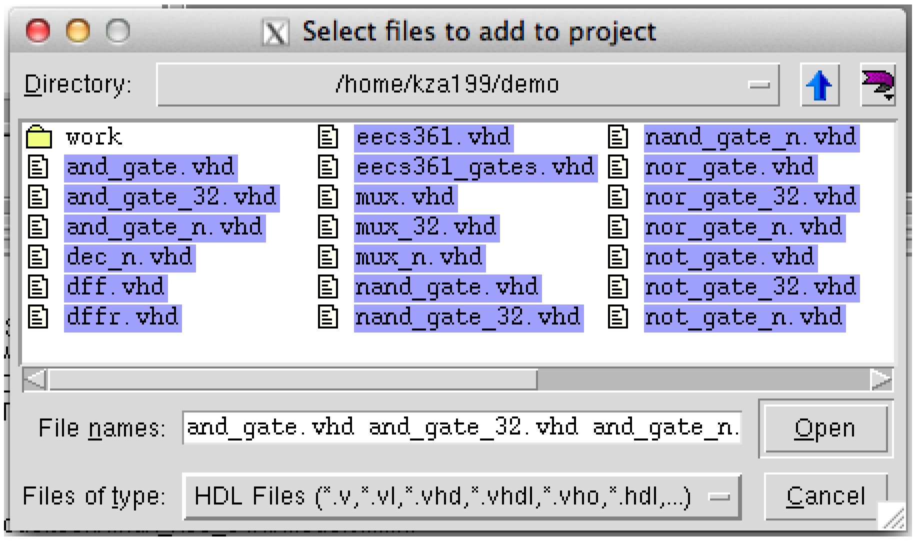


Add file as type: default

Folder: Top Level

[✓ Reference from current location] Copy to project directory

3. The window should start at your project’s location by default. If not, navigate yourself to your project’s location. Use Ctrl and Shift keys to select all the vhd files. Click on Open, and then click on OK.

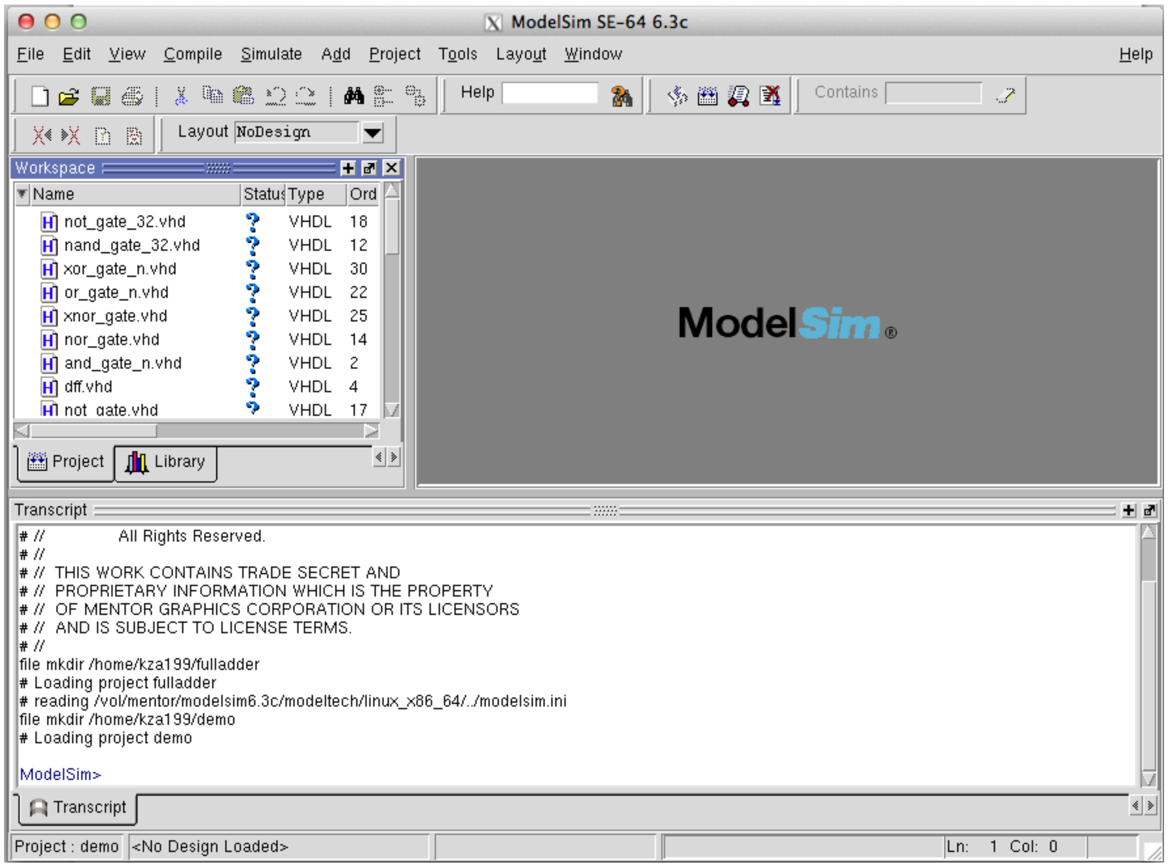


```
work              eecs361.vhd         nand_gate_n.vhd
and_gate.vhd      eecs361_gates.vhd   nor_gate.vhd
and_gate_32.vhd   mux.vhd             nor_gate_32.vhd
and_gate_n.vhd    mux_32.vhd          nor_gate_n.vhd
dec_n.vhd         mux_n.vhd           not_gate.vhd
dff.vhd           nand_gate.vhd       not_gate_32.vhd
dffr.vhd          nand_gate_32.vhd    not_gate_n.vhd
```

File names: and_gate.vhd and_gate_32.vhd and_gate_n.

Files of type: HDL Files (*.v,*.vl,*.vhd,*.vhdl,*.vho,*.hdl,...)

4. You should return to the main window, with the vhd files added to your workspace. Feel free to double click on the file names to check what in the fil



Name	Status	Type	Ord
not_gate_32.vhd	?	VHDL	18
nand_gate_32.vhd	?	VHDL	12
xor_gate_n.vhd	?	VHDL	30
or_gate_n.vhd	?	VHDL	22
xnor_gate.vhd	?	VHDL	25
nor_gate.vhd	?	VHDL	14
and_gate_n.vhd	?	VHDL	2
dff.vhd	?	VHDL	4
not_gate.vhd	?	VHDL	17

```
# //             All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
file mkdir /home/kza199/fulladder
# Loading project fulladder
# reading /vol/mentor/modelsim6.3c/modeltech/linux_x86_64/../modelsim.ini
file mkdir /home/kza199/demo
# Loading project demo

ModelSim>
```

Project : demo   <No Design Loaded>   Ln: 1 Col: 0

Create the 1-bit full adder

1. Select File -> New -> Source -> VHDL to create a new VHDL source file.
2. Write the fulladder VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;
use work.eecs361_gates.all;

entity fulladder is
  port (
    x       : in  std_logic;
    y       : in  std_logic;
    c       : in  std_logic;
    z       : out std_logic;
    cout    : out std_logic
  );
end fulladder;

architecture structural of fulladder is
signal xor0 : std_logic;
signal and0 : std_logic;
signal and1 : std_logic;
begin
  xor0_map : xor_gate port map (x => x, y => y, z => xor0);
  xor1_map : xor_gate port map (x => xor0, y => c, z => z);
  and0_map : and_gate port map (x => x, y => y, z => and0);
  and1_map : and_gate port map (x => xor0, y => c, z => and1);
  or0_map : or_gate port map (x => and0, y => and1, z => cout);
end architecture structural;
```

3. Save the file as fulladder.vhd
4. Add fulladder.vhd to your workspace.

Create testbench for fulladder.

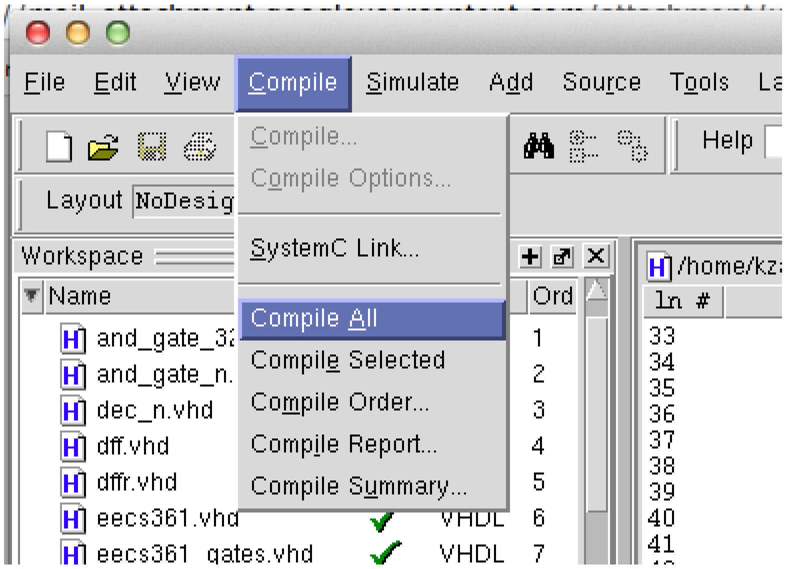
1. As the steps above, create another VHDL file “fulladder_demo.vhd”, write the following content, and add it to the workspace.

```
library ieee;
use ieee.std_logic_1164.all;

entity fulladder_demo is
  port (
    z       : out std_logic;
    cout    : out std_logic
  );
end fulladder_demo;

architecture structural of fulladder_demo is
```

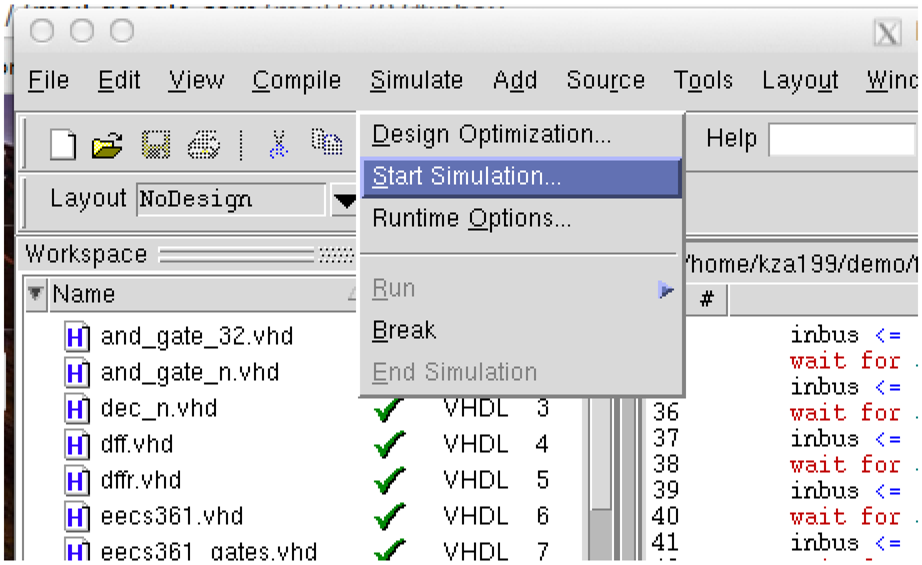
```
component fulladder is
  port (
    x       : in  std_logic;
    y       : in  std_logic;
    c       : in  std_logic;
    z       : out std_logic;
    cout    : out std_logic
  );
end component fulladder;
signal xin : std_logic;
signal yin : std_logic;
signal cin : std_logic;
signal inbus : std_logic_vector(2 downto 0);
begin
  fulladder_map : fulladder port map (x => xin, y => yin, c => cin, z
=> z, cout => cout);
  cin <= inbus(2);
  yin <= inbus(1);
  xin <= inbus(0);

  test_proc : process
  begin
    inbus <= "000";
    wait for 5 ns;
    inbus <= "001";
    wait for 5 ns;
    inbus <= "010";
    wait for 5 ns;
    inbus <= "011";
    wait for 5 ns;
    inbus <= "100";
    wait for 5 ns;
    inbus <= "101";
    wait for 5 ns;
    inbus <= "110";
    wait for 5 ns;
    inbus <= "111";
    wait for 5 ns;
    wait;
  end process;
end architecture structural;
```

Compile and simulate

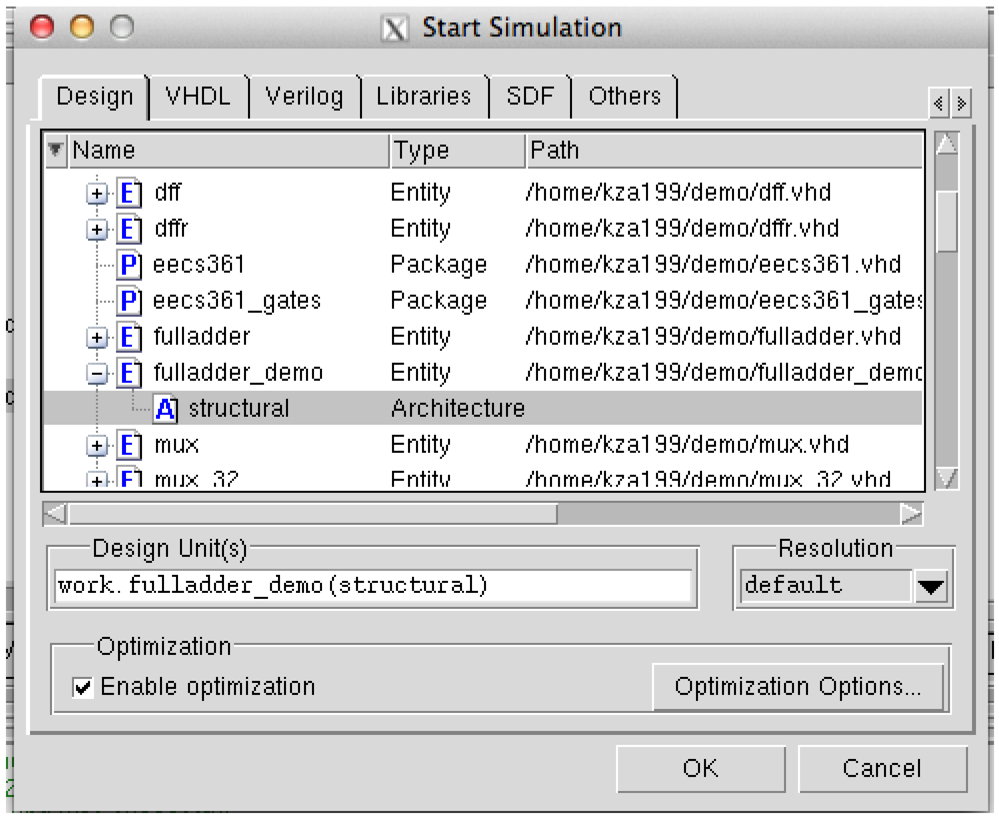
1. Compile all the files by selecting Compile -> “Compile All”. Try a second time or third time if the first compilation generates errors. Because the default order of the files may not match the dependencies between components.



2. After successfully compiled the project, start the simulation by selecting “Simulate” -> “Start Simulation”.



3. Select the structural design of fulladder_demo to simulate. The fulladder_demo is in work library.



Start Simulation

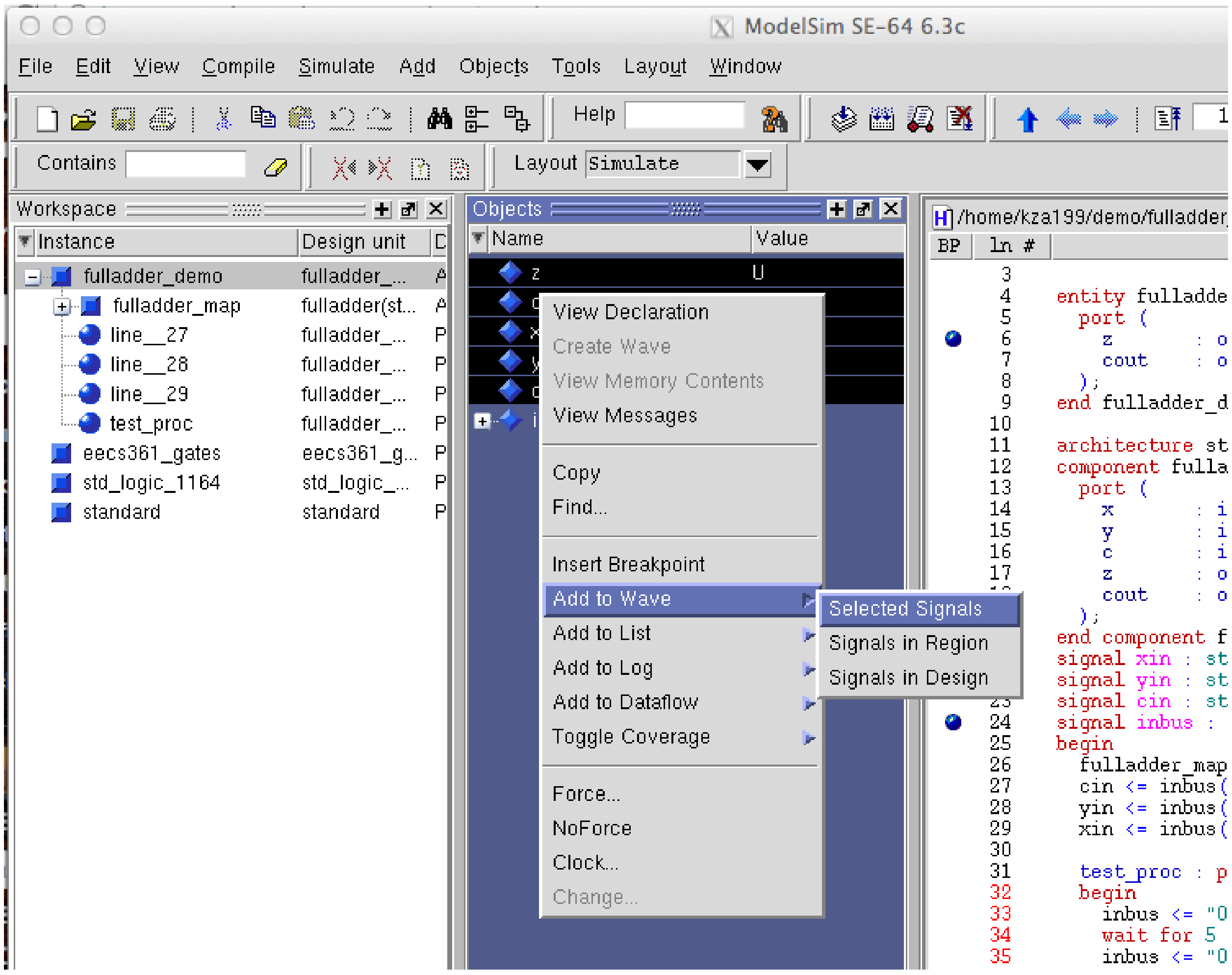
Name	Type	Path
dff	Entity	/home/kza199/demo/dff.vhd
dffr	Entity	/home/kza199/demo/dffr.vhd
eecs361	Package	/home/kza199/demo/eecs361.vhd
eecs361_gates	Package	/home/kza199/demo/eecs361_gates
fulladder	Entity	/home/kza199/demo/fulladder.vhd
fulladder_demo	Entity	/home/kza199/demo/fulladder_demo
structural	Architecture	
mux	Entity	/home/kza199/demo/mux.vhd
mux_32	Entity	/home/kza199/demo/mux_32.vhd

Design Unit(s): work.fulladder_demo(structural)

Resolution: default

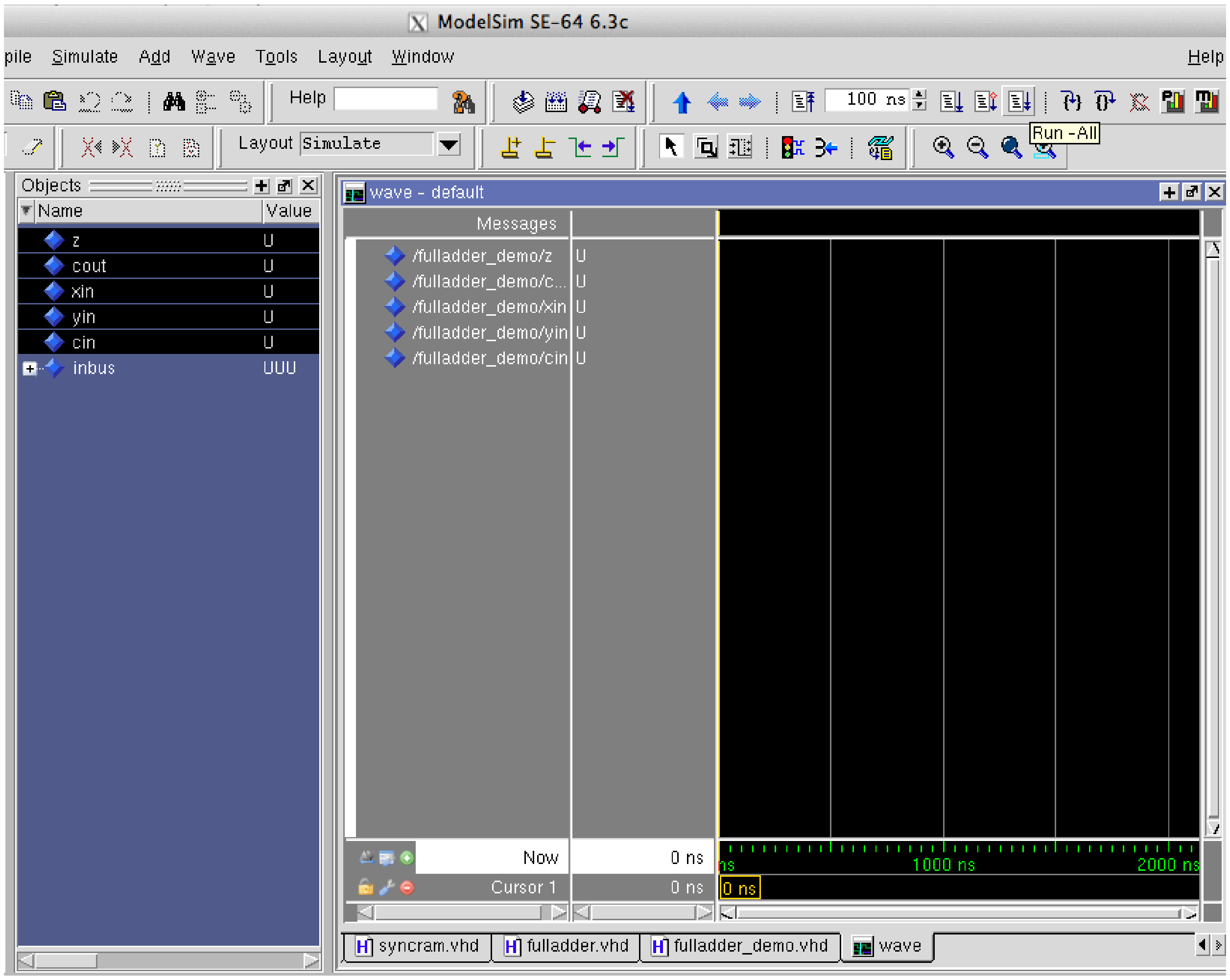
Optimization: [✓] Enable optimization

4. A simulation window will pop out. In the Objects window in the middle, select z, cout, xin, yin, and cin. After that, right click on them, then select “Add to Wave” -> “Selected Signals”.

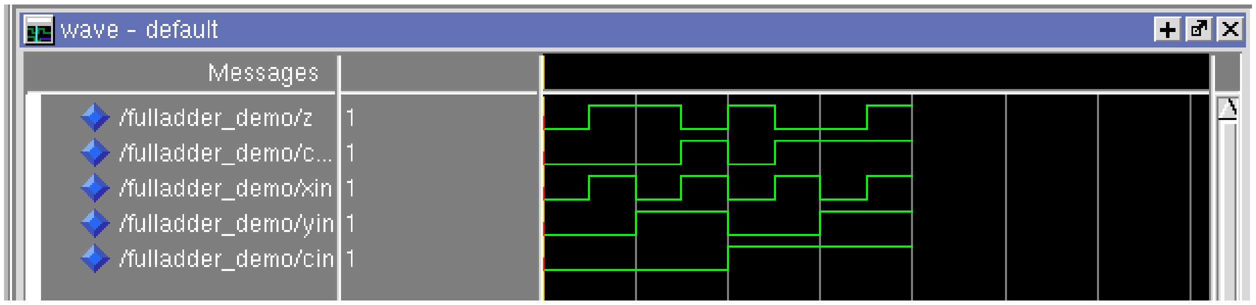


```
entity fulladde
  port (
    z       : o
    cout    : o
  );
end fulladder_d

architecture st
component fulla
  port (
    x       : i
    y       : i
    c       : i
    z       : o
    cout    : o
  );
end component f
signal xin : st
signal yin : st
signal cin : st
signal inbus :
begin
  fulladder_map
  cin <= inbus(
  yin <= inbus(
  xin <= inbus(

  test_proc : p
  begin
    inbus <= "0
    wait for 5
    inbus <= "0
```

5. A wave window will appear in the right. Click “Run -All” button .



6. The simulate will start and finish shortly. The zoom level might not fit the waveform. Zoom in or out to adjust the waveform.



7. The waveform is as our expectation.

Summary of the workflows.

Here is a summary of the workflows:

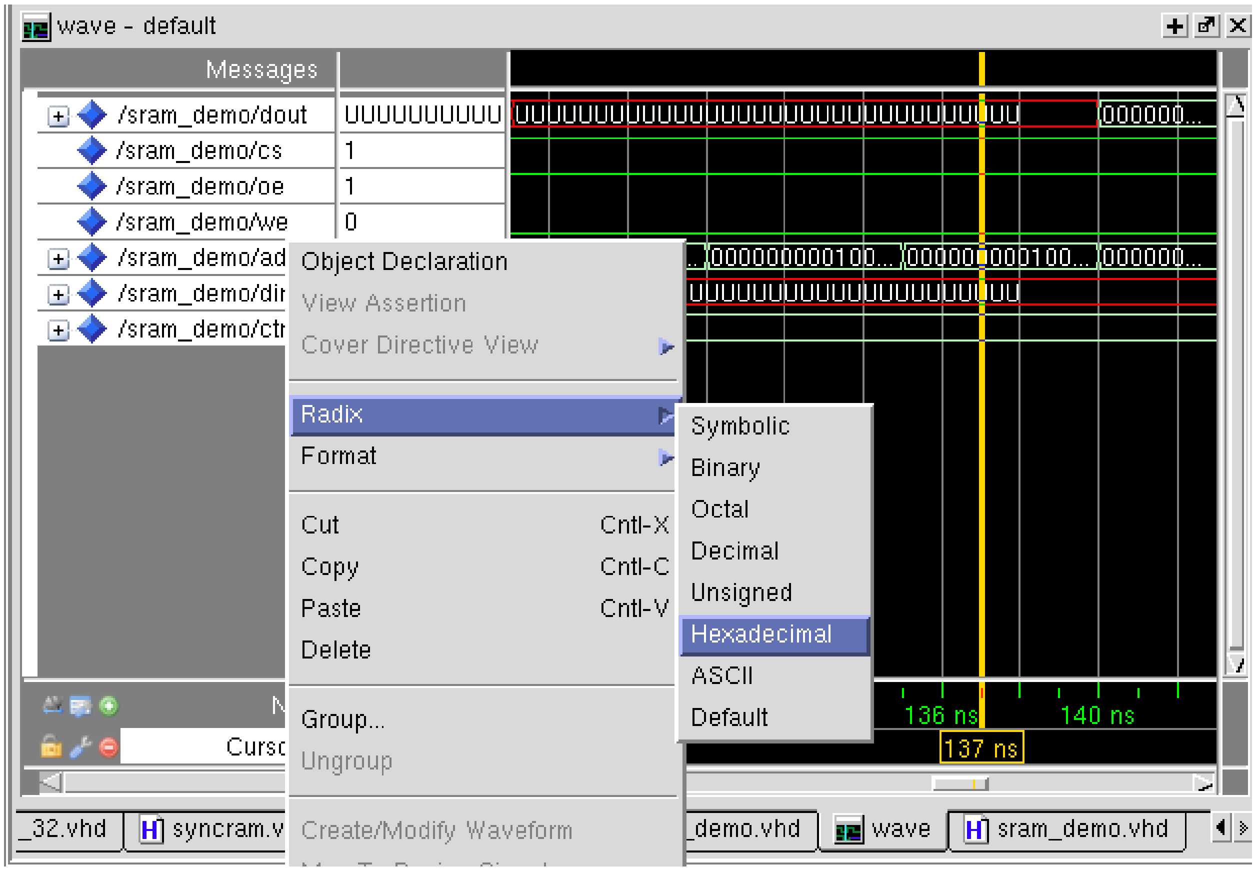
1. Create a project.
2. Add and edit files, including testbenches.
3. Compile all the files.
4. Open the simulation window.

5. Add signals to wave.
6. Run the simulation and check results.

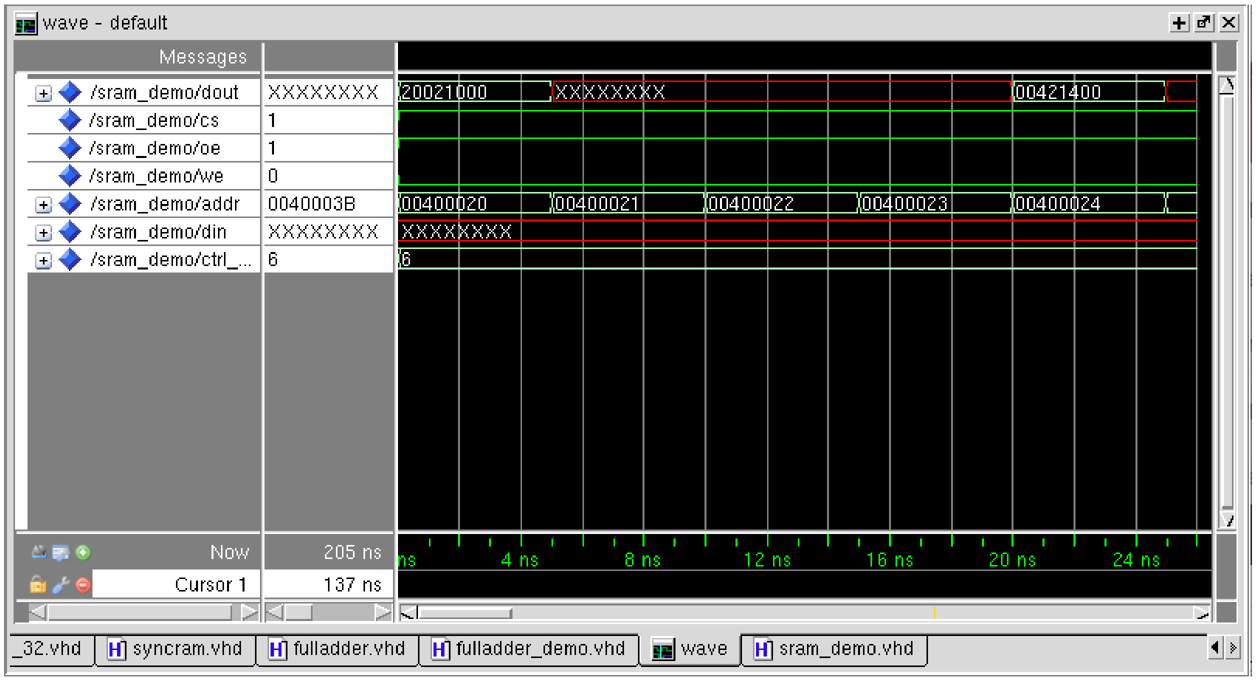
SRAM Reader

The next demo is to give an example on how to use SRAM component provided by the library.
1. Copy the file “data/sort_corrected_branch.dat” in the library package to your project location.
2. Add the SRAM demo file “sram_demo.vhd” and write the following content to it.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.eecs361_gates.all;
use work.eecs361.all;

entity sram_demo is
  port (
    dout : out std_logic_vector(31 downto 0)
  );
end sram_demo;

architecture structural of sram_demo is
signal cs : std_logic;
signal oe : std_logic;
signal we : std_logic;
signal addr : std_logic_vector(31 downto 0);
signal din : std_logic_vector(31 downto 0);
signal ctrl_bus : std_logic_vector(2 downto 0);
begin
  sram_map : sram
    generic map (mem_file => "sort_corrected_branch.dat")
    port map (cs => cs, oe => oe, we => we, addr => addr, din => din,
dout => dout);
  cs <= ctrl_bus(2);
  oe <= ctrl_bus(1);
  we <= ctrl_bus(0);

  process
    variable vaddr : integer range 0 to 2147483647;
  begin
    ctrl_bus <= "110";
    for vaddr in 4194336 to 4194376 loop
      addr <= std_logic_vector(to_unsigned(vaddr, 32));
      wait for 5 ns;
    end loop;
    wait;
  end process;
end;
```
3. Start the simulation of sram_demo with all the signals.
4. You can change the display format of the signal. To do that, select the signals, and right click on them, then select Radix -> Hexadecimal.



5. The waveform is similar to the mentor graphics one, except the VHDL SRAM use ‘1’ to enable the chip.



Messages		
/sram_demo/dout	XXXXXXXX	20021000, XXXXXXXX, 00421400
/sram_demo/cs	1	
/sram_demo/oe	1	
/sram_demo/we	0	
/sram_demo/addr	0040003B	00400020, 00400021, 00400022, 00400023, 00400024
/sram_demo/din	XXXXXXXX	XXXXXXXX
/sram_demo/ctrl_...	6	6
Now	205 ns	
Cursor 1	137 ns	