Nanyang Technological University
School of Electrical and Electronic Engineering

NTU-TUM MSc (IC Design) Programme
Digital IC Design Course: NM6008

Laboratory Manual

Module 3

Verilog Simulations and Synthesis of a Full-Adder

Course Instructors: A/Prof Lim Meng Hiot
A/Prof Gwee Bah Hwee
Dr Cheng Deruo
Dr Cheng Juncheng
Sheng Shirui
Zhang Han

19th Aug – 4th Sep 2024

Section 1: Introduction

This tutorial style laboratory manual document is meant for NTU-TUM Master of Science (MSc) course – Integrated Circuit (IC) Design, NM6008. It is a step-by-step guide for students who are new to the concept of Hardware Description Language (HDL), ultimately to provide students with the necessary understanding to begin designing circuits through HDL modelling. The preparation of the contents of this manual benefits from the contributions of the following individuals:

Assoc Prof Lim Meng Hiot	Assoc Prof Gwee Bah Hwee
Dr Chong Kwen Siong	Mr Farid Ahamed
Mr Ne Kyaw Zwa Lwin	

The following sub-sections state the objectives, the design tools/process used and the experimental design information of this module – *Verilog Simulations and Synthesis of a Full-Adder*.

1.1 Objectives:

The main objective of this module is to provide a quick introduction to Verilog as a Hardware Description Language (HDL) language for modelling and simulating digital ICs. The design example used is a combinational 1-bit full-adder. It will serve to familiarize students with the Synopsys IC design tools (and the associated design methodologies). The specific objectives include:

(i) To construct a behavioural (functional) model of the full-adder using Verilog;
(ii) To construct a test-bench for simulating/validating the full-adder;
(iii) To synthesize the model of the full-adder into a netlist using the STM 65nm CMOS cell library;
(iv) To understand the synthesis design flow;
(v) To re-simulate/re-validate the netlist of the full-adder (with timing annotation) using the same test-bench in (ii);
(vi) To perform power analysis on the full-adder.

1.2 Integrated Circuit (IC) Design Tools and Process Technology

Students will be trained to use the following IC design tools:

- Verilog Compiler Simulator (Synopsys)
- Design Vision (Synopsys)
- Power Compiler (Synopsys)

For more details on Synopsys IC design tools, the following documents are suggested for further reading:

__, HDL Compiler^{TM} for Verilog User Guide, Synopsys.
__, VCS^{TM} User Guide, Synopsys.
__, Design Compiler^{TM} User Guide. Synopsys.
__, Design Vision^{TM} for Verilog User Guide, Synopsys.
__, Power Compiler^{TM} User Guide, Synopsys

The process technology used is the STM 65nm CMOS fabrication process.

1.3 Design Information

An adder is one of the most rudimentary circuit components used for arithmetic functions, including addition, subtraction, multiplication, division and etc. A 1-bit full-adder accepts 3 input signals (A, B and C_{in}) and generates two output signals (S and C_{out}), and the truth-table of the full-adder is tabulated in Table I.

Table I: Truth-table for a Full-Adder

Input			Output	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The logic equations for the outputs can be formulated as follows:

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = (A \cdot B) \oplus (C_{in} \cdot (A \oplus B))$$

Section 2: Procedures

2.1 Behaviour Modelling and Simulations

A. Working Environment:

1. **Go** to your home directory on the desktop, and **choose** the folder “**Module3**”.

2. In the folder “**Module3**”, **choose** the folder “**Behavioral**”.

3. In the folder “**Behavioral**”, you should be able to see the following files.

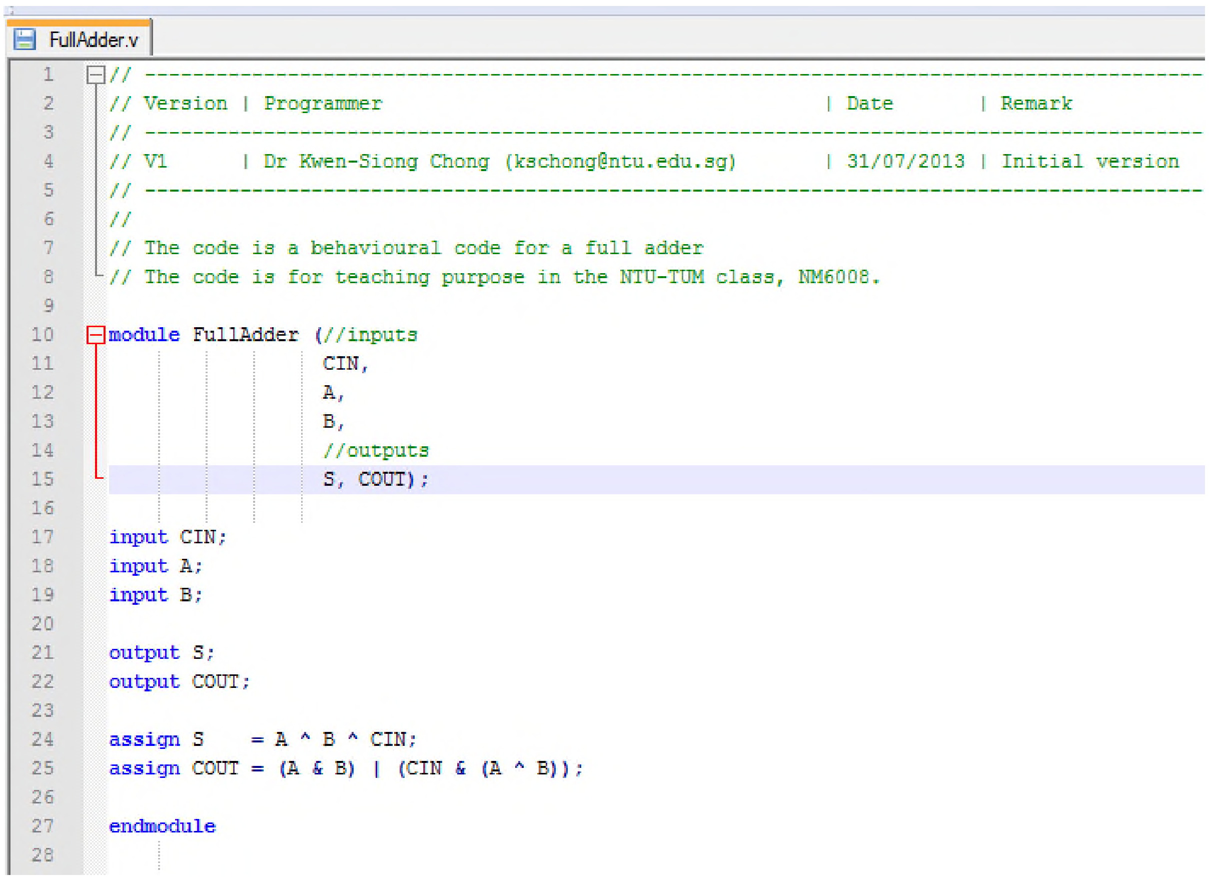
   ```
   ./FullAdder.v
   ./FullAdder_tb.v
   ./synopsys_linux_vj-2014_cshrc
   ./InputVector.txt
   ./run
   ```

   Now, you are ready to start your hands-on.
   [Note: Save all your files in the current folder, i.e. “Behavioural”]

B. Creating a Behavioural Verilog Code

1. **Open** “FullAdder.v” using a Text Editor, i.e. **right-click** the file and **choose** “Text Editor”.

2. **Write** the behavioral Verilog code, shown in Fig. 1, for the full-adder circuit.



FullAdder.v

```
// ------------------------------------------------------------------------------------------
// Version | Programmer                                     | Date       | Remark
// ------------------------------------------------------------------------------------------
// V1      | Dr Kwen-Siong Chong (kschong@ntu.edu.sg)       | 31/07/2013 | Initial version
// ------------------------------------------------------------------------------------------
//
// The code is a behavioural code for a full adder
// The code is for teaching purpose in the NTU-TUM class, NM6008.

module FullAdder (//inputs
                  CIN,
                  A,
                  B,
                  //outputs
                  S, COUT);

input CIN;
input A;
input B;

output S;
output COUT;

assign S    = A ^ B ^ CIN;
assign COUT = (A & B) | (CIN & (A ^ B));

endmodule

```

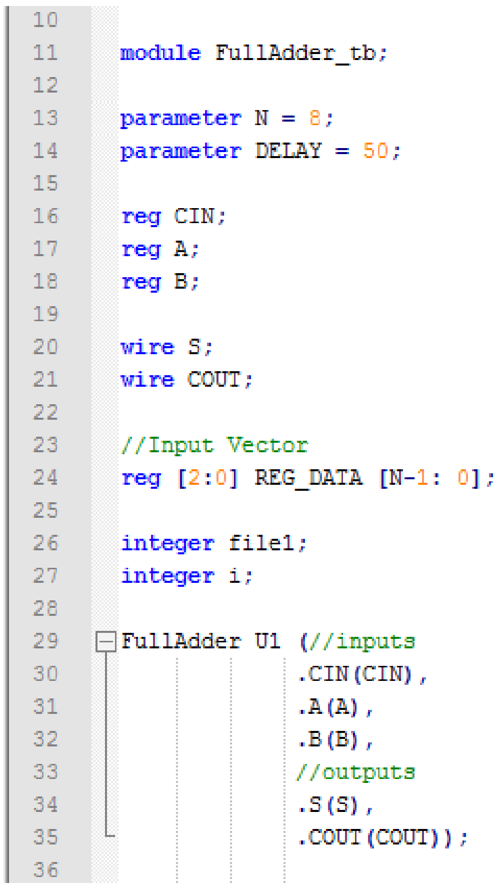
Fig. 1 The Verilog code for the full-adder

3. **Save** the created code.

C. <u>Creating a Test-bench</u>

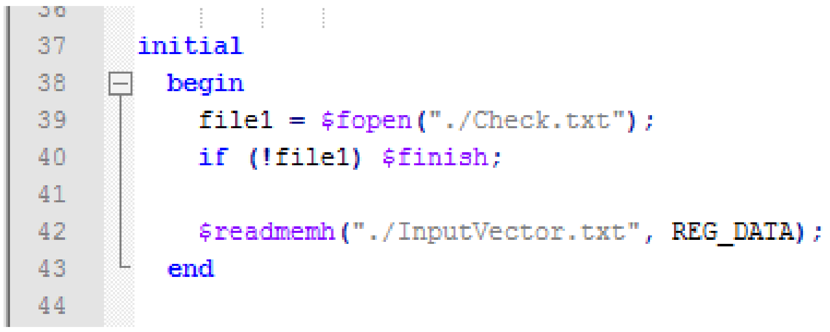
1. **Open** “FullAdder_tb.v” using Text Editor.

2. **Fill in** Parts 1 to 3, as shown in Fig.2 (a) to (c) respectively, in the behavioralVerilog code for realizing a full-adder circuit:



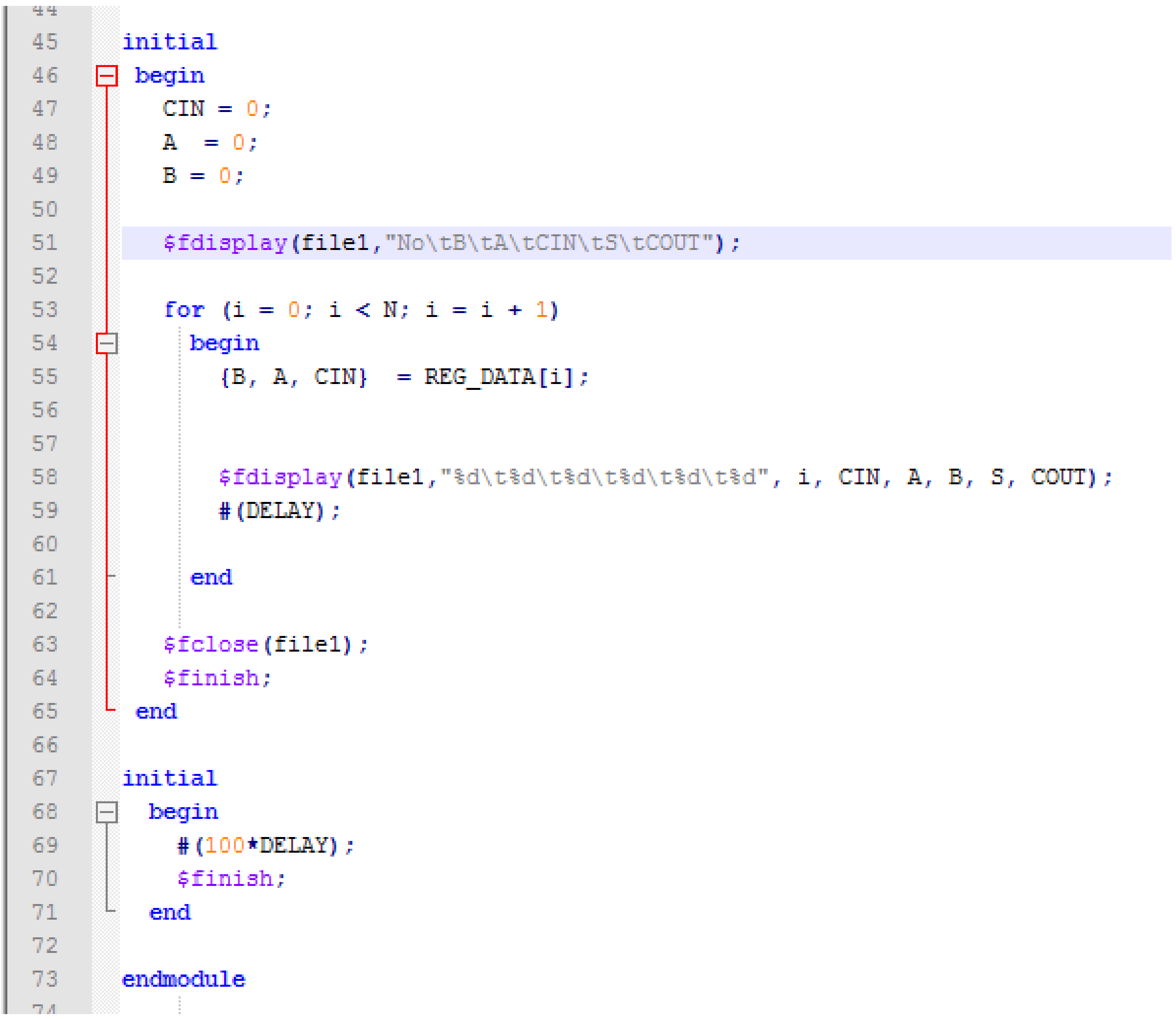
```verilog

module FullAdder_tb;

parameter N = 8;
parameter DELAY = 50;

reg CIN;
reg A;
reg B;

wire S;
wire COUT;

//Input Vector
reg [2:0] REG_DATA [N-1: 0];

integer file1;
integer i;

FullAdder U1 (//inputs
                .CIN(CIN),
                .A(A),
                .B(B),
                //outputs
                .S(S),
                .COUT(COUT));

```

Fig. 2 (a) Part 1: Basic IO/signal declaration and mapping



```verilog
initial
  begin
    file1 = $fopen("./Check.txt");
    if (!file1) $finish;

    $readmemh("./InputVector.txt", REG_DATA);
  end

```

Fig. 2 (b) Part 2: Insert the data vector



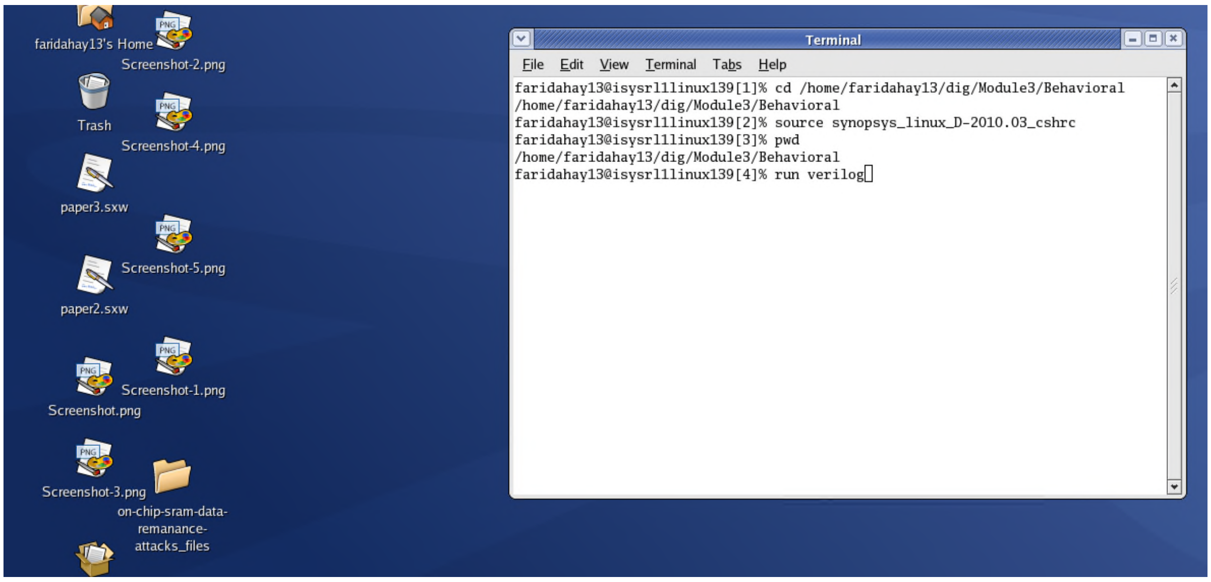
```
initial
 begin
   CIN = 0;
   A  = 0;
   B = 0;

   $fdisplay(file1,"No\tB\tA\tCIN\tS\tCOUT");

   for (i = 0; i < N; i = i + 1)
     begin
       {B, A, CIN}  = REG_DATA[i];


       $fdisplay(file1,"%d\t%d\t%d\t%d\t%d\t%d", i, CIN, A, B, S, COUT);
       #(DELAY);

     end

   $fclose(file1);
   $finish;
 end

initial
  begin
    #(100*DELAY);
    $finish;
  end

endmodule
```

Fig. 2 (c) Part 3: Generate the test vector pattern

3. **Save** the created code.

D. Running Verilog Simulations

1. **Right-click** on the Desktop and **open** a Terminal. The Terminal will probably look like what is shown in Fig. 3.



```
faridahay13@isysrl1linux139[1]% cd /home/faridahay13/dig/Module3/Behavioral
/home/faridahay13/dig/Module3/Behavioral
faridahay13@isysrl1linux139[2]% source synopsys_linux_D-2010.03_cshrc
faridahay13@isysrl1linux139[3]% pwd
/home/faridahay13/dig/Module3/Behavioral
faridahay13@isysrl1linux139[4]% run verilog
```

Fig. 3 A Terminal opened in Linux

2. In the Terminal, **type in** the following commands.

   ```
   >> cd /home/[your home directory name]/Module3/Behavioral
   >> source synopsys_linux_vj-4201_cshrc
   ```

   [Note: The second command is to source the license to invoke the Synopsis tools]

3. In the Terminal, **type in** the following command.

   ```
   >> vcs -debug +v2k ./FullAdder_tb.v ./FullAdder.v
   ```

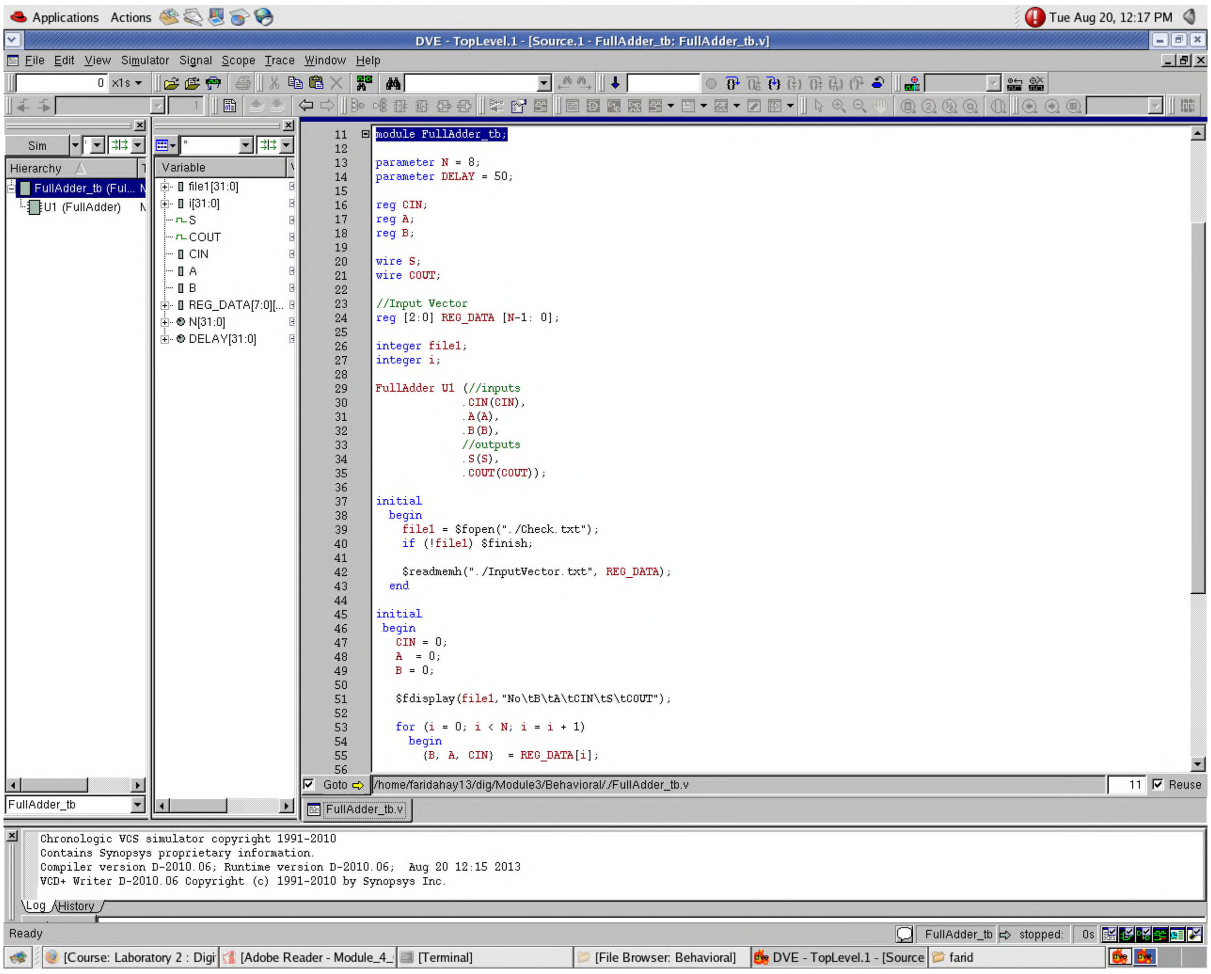
   [Tip: you can copy and save the command in step 3 into the “run” file so that you do not need to re-type the command in the future]

4. **Check** the Terminal if any error message appears. If yes, PLEASE **re-check** your Verilog codes again and repeat step 3. You must **fix all** error messages! If no error message appears, **type in** the following command to invoke the GUI.

   ```
   >> simv -gui
   ```

   [Tip: you can copy and save the command in the step 4 into the “run” file so that you do not need to re-type the command in the future]

   The Discovery Visualization Environment (DVE) GUI will be shown; see Fig. 4. You may explore the functions/tabs in the DVE GUI on your own.



```
module FullAdder_tb;

parameter N = 8;
parameter DELAY = 50;

reg CIN;
reg A;
reg B;

wire S;
wire COUT;

//Input Vector
reg [2:0] REG_DATA [N-1: 0];

integer file1;
integer i;

FullAdder U1 (//inputs
              .CIN(CIN),
              .A(A),
              .B(B),
              //outputs
              .S(S),
              .COUT(COUT));

initial
  begin
    file1 = $fopen("./Check.txt");
    if (!file1) $finish;

    $readmemh("./InputVector.txt", REG_DATA);
  end

initial
 begin
   CIN = 0;
   A  = 0;
   B = 0;

   $fdisplay(file1,"No\tB\tA\tCIN\tS\tCOUT");

   for (i = 0; i < N; i = i + 1)
     begin
       {B, A, CIN}  = REG_DATA[i];
```

```
Chronologic VCS simulator copyright 1991-2010
Contains Synopsys proprietary information.
Compiler version D-2010.06; Runtime version D-2010.06;  Aug 20 12:15 2013
VCD+ Writer D-2010.06 Copyright (c) 1991-2010 by Synopsys Inc.
```

Fig. 4 The Discovery Visualization Environment (DVE) GUI

5. The next step is to **simulate** the full-adder and to **visualize** the input and output waveforms. Before doing so, the file **"inputvector.txt"** has to be written to insert various input test patterns you wish to apply to the full-adder; see Section 2.1.C Part 2.

6. In the generated DVE GUI, **select** all the variables present under the hierarchy of FullAdder_tb; **refer to** the middle column in the DVE GUI in Fig. 4.

7. **Right-click** on the selected variables, and **choose "Add To Waves"** and then **select "New Wave View".** A new DVE waveform GUI will pop up.

8. In the new popped-up DVE waveform window, **choose "simulator"** icon on the toolbar and then **select "Start/Continue".**

9. You can now visualize the input and output waveforms. **Keep** only input and output variables (i.e. A, B, CIN, S, COUT) to get a clearer view of the circuit functionality; **delete** all other variables if any. An example is shown in Fig. 5.

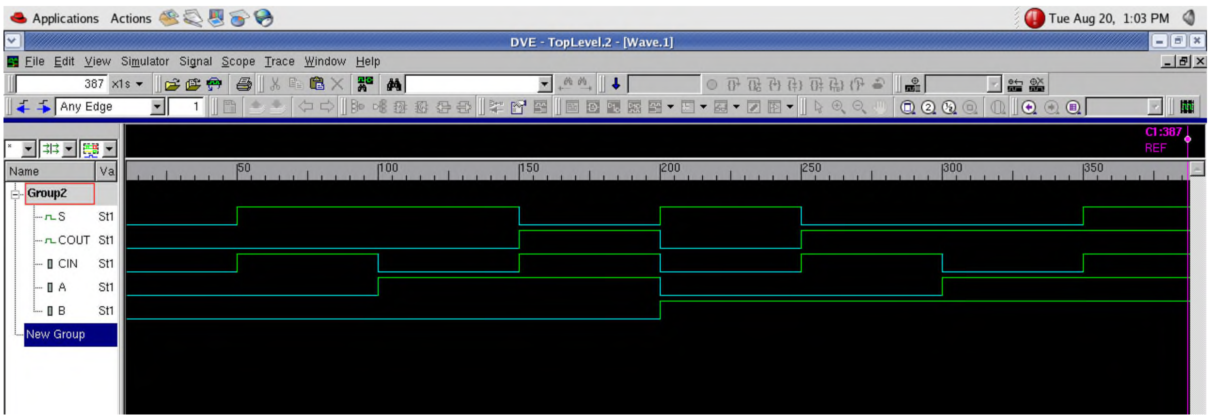


Fig. 5 The DVE waveform window showing the input and output signals

10. **Answer** the question in Section 3.A.1 in page 21.

At this stage, if you have successfully gone through all the steps specified, you have completed the **Behavioral** simulation. Congratulations!

2.2 Synthesis

The synthesis design flow will be carried out based on the Synopsys tool named "**Design Vision"** and on the STM 65nm CMOS process cell library.

A. <u>Working Environment:</u>

1. **Go** to your home directory on the Desktop, and **choose** the folder "**Module3**".

2. In the folder "**Module3**", **choose** the folder "**Synthesis**".

3. In the folder "Synthesis", you should be able to see the following files:

   ./.synopsys_dc.setup [Synopsys configuration file]
   ./ synopsys_linux_vj-2014_cshrc [setup file for the Synopsys tools]
   ./ common_setup_STM.tcl [setup file for the STM process]

4. In the folder "Synthesis", you should be able to see the following folders.

   ./ddc
   ./logs
   ./netlist
   ./scripts
   ./sources

[Note: The arrangement of these folders is the Instructor's preference, and you can choose not to follow this kind of folder/file management if you know the synthesis design flow well]

5. **Copy** the "FullAdder.v" file (that you have designed in Section 2.1) into the folder "sources".

   [Note: In this example, you have only one Verilog file. For multiple Verilog files, you should store all your Verilog files here if you wish to synthesize them]

6. In the folder "scripts", you should be able to see the following files:

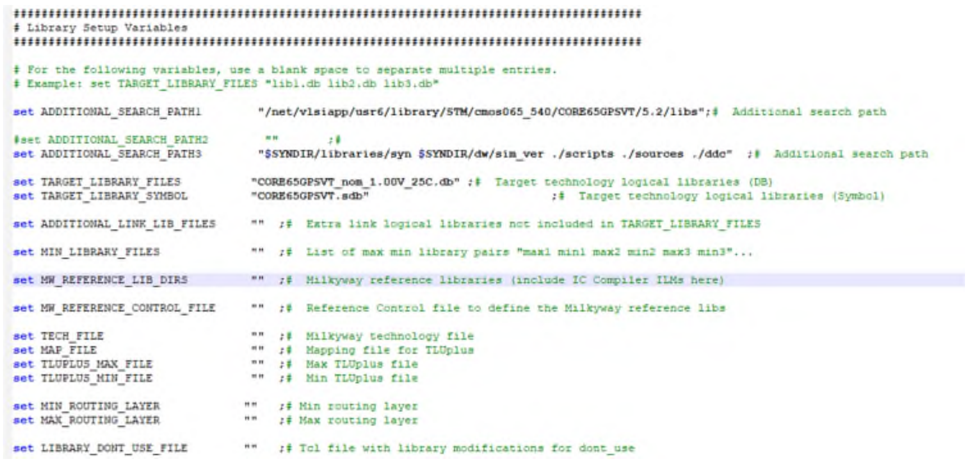
   ./FullAdder.tcl
   ./MyDefaults.con
   ./ReadVerilogFiles.tcl
   ./report.tcl
   ./run_topdown.tcl

   Now, you are ready to start your hands-on.
   [Note: Save all your files in the current folder, i.e. "Synthesis"]

B. <u>Understand the Basic Setups/Files</u>

Attention: Refer to the Synopsys Design Vision Manual or DC Manual if you would like to know more. Sometimes, you may need to explore the chosen Process Design Kit (PDK) in order to know/obtain more information. If possible and applicable, please approach a software application engineer and/or a PDK application engineer for help.

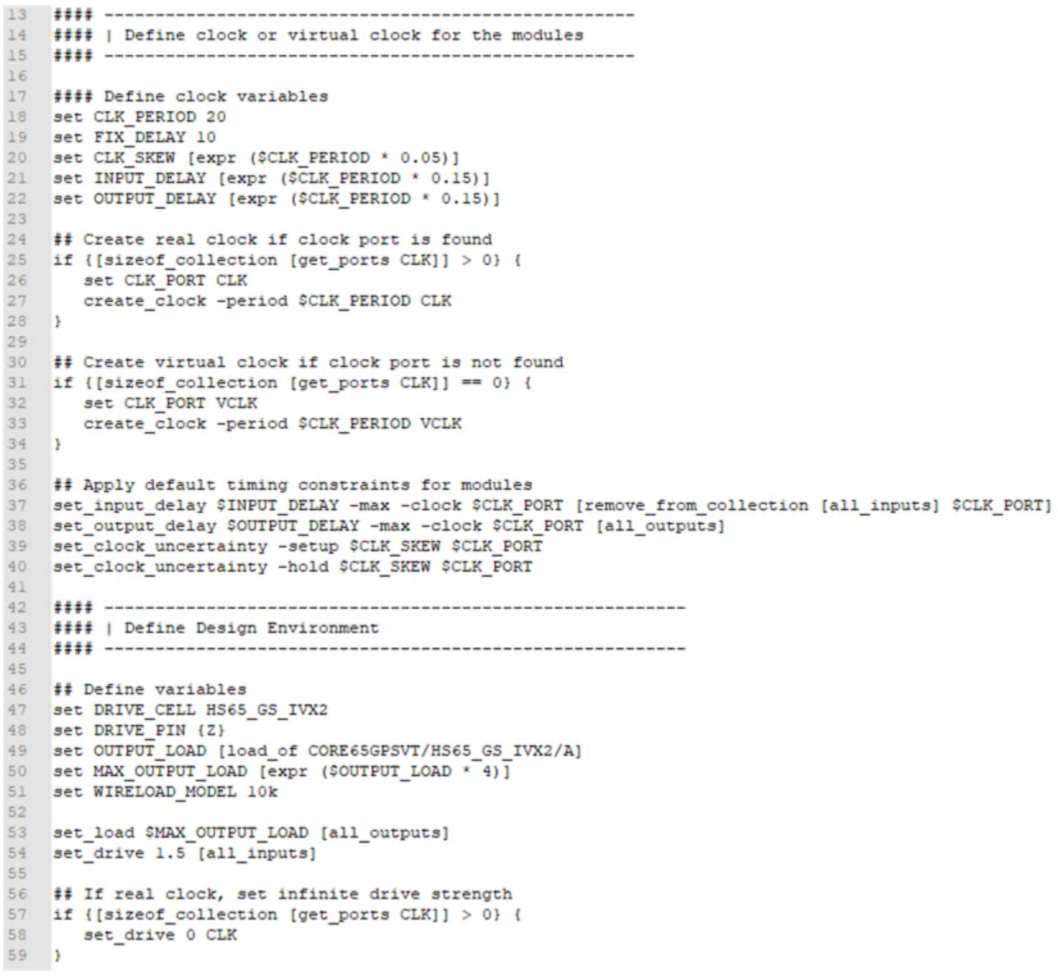
1. **Open** "common_setup_STM.tcl" using Text Editor and **view** it as shown in Fig. 6. The file contains the setups on how the Instructor arranges the folder, and links various STM 65nm CMOS cell libraries. You should be able to obtain the file from any chosen PDK, and if necessary, you could modify the setups/configurations according to your preferred working style. **Close** this file.



```
##################################################################################
# Library Setup Variables
##################################################################################

# For the following variables, use a blank space to separate multiple entries.
# Example: set TARGET_LIBRARY_FILES "lib1.db lib2.db lib3.db"

set ADDITIONAL_SEARCH_PATH1        "/net/vlsiapp/usr6/library/STM/cmos065_540/CORE65GPSVT/5.2/libs";#  Additional search path

#set ADDITIONAL_SEARCH_PATH2        ""        ;#
set ADDITIONAL_SEARCH_PATH3        "$SYNDIR/libraries/syn $SYNDIR/dw/sim_ver ./scripts ./sources ./ddc"  ;#  Additional search path

set TARGET_LIBRARY_FILES          "CORE65GPSVT_nom_1.00V_25C.db" ;#  Target technology logical libraries (DB)
set TARGET_LIBRARY_SYMBOL         "CORE65GPSVT.sdb"                    ;#  Target technology logical libraries (Symbol)

set ADDITIONAL_LINK_LIB_FILES     ""  ;#  Extra link logical libraries not included in TARGET_LIBRARY_FILES

set MIN_LIBRARY_FILES             ""  ;#  List of max min library pairs "max1 min1 max2 min2 max3 min3"...

set MW_REFERENCE_LIB_DIRS         ""  ;#  Milkyway reference libraries (include IC Compiler ILMs here)

set MW_REFERENCE_CONTROL_FILE     ""  ;#  Reference Control file to define the Milkyway reference libs

set TECH_FILE                     ""  ;#  Milkyway technology file
set MAP_FILE                      ""  ;#  Mapping file for TLUplus
set TLUPLUS_MAX_FILE              ""  ;#  Max TLUplus file
set TLUPLUS_MIN_FILE              ""  ;#  Min TLUplus file

set MIN_ROUTING_LAYER            ""   ;# Min routing layer
set MAX_ROUTING_LAYER            ""   ;# Max routing layer

set LIBRARY_DONT_USE_FILE        ""   ;# Tcl file with library modifications for dont_use
```

Fig. 6 An example of the Synopsys configuration file "common_setup_STM.tcl"

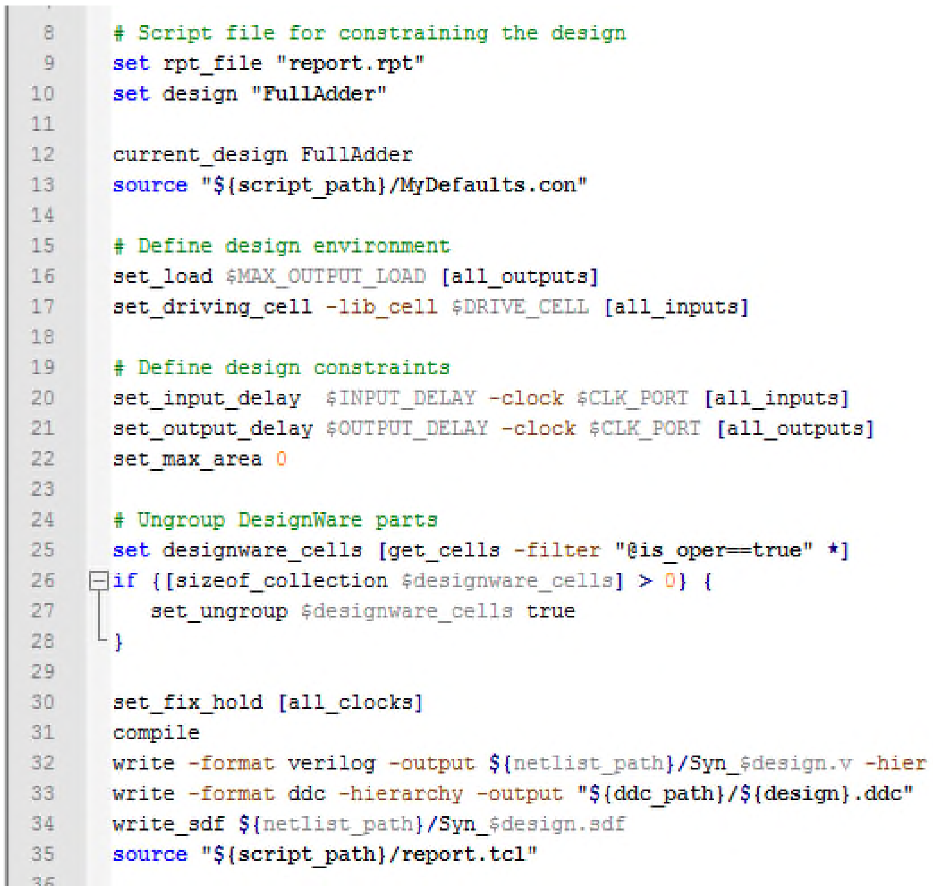
2. **Open** "./scripts/MyDefaults.con" and view it as shown in Fig. 7. The file contains the basic constraints/specifications – these are deemed as default values set by the instructor. Please refer to the Synopsys DC manual if you would like to understand more. You can always create your constraints/specifications here. **Close** this file.



```
#### ----------------------------------------------------
#### | Define clock or virtual clock for the modules
#### ----------------------------------------------------

#### Define clock variables
set CLK_PERIOD 20
set FIX_DELAY 10
set CLK_SKEW [expr ($CLK_PERIOD * 0.05)]
set INPUT_DELAY [expr ($CLK_PERIOD * 0.15)]
set OUTPUT_DELAY [expr ($CLK_PERIOD * 0.15)]

## Create real clock if clock port is found
if {[sizeof_collection [get_ports CLK]] > 0} {
   set CLK_PORT CLK
   create_clock -period $CLK_PERIOD CLK
}

## Create virtual clock if clock port is not found
if {[sizeof_collection [get_ports CLK]] == 0} {
   set CLK_PORT VCLK
   create_clock -period $CLK_PERIOD VCLK
}

## Apply default timing constraints for modules
set_input_delay $INPUT_DELAY -max -clock $CLK_PORT [remove_from_collection [all_inputs] $CLK_PORT]
set_output_delay $OUTPUT_DELAY -max -clock $CLK_PORT [all_outputs]
set_clock_uncertainty -setup $CLK_SKEW $CLK_PORT
set_clock_uncertainty -hold $CLK_SKEW $CLK_PORT

#### ---------------------------------------------------------
#### | Define Design Environment
#### ---------------------------------------------------------

## Define variables
set DRIVE_CELL HS65_GS_IVX2
set DRIVE_PIN {Z}
set OUTPUT_LOAD [load_of CORE65GPSVT/HS65_GS_IVX2/A]
set MAX_OUTPUT_LOAD [expr ($OUTPUT_LOAD * 4)]
set WIRELOAD_MODEL 10k

set_load $MAX_OUTPUT_LOAD [all_outputs]
set_drive 1.5 [all_inputs]

## If real clock, set infinite drive strength
if {[sizeof_collection [get_ports CLK]] > 0} {
   set_drive 0 CLK
}
```

Fig. 7 An example of the constraint file "MyDefaults.con"

3. **Open** "./scripts/FullAdder.tcl" and view it as shown in Fig. 8. The file contains the additional constraints, if different from the default values, and the synthesis commands.



```
 8  # Script file for constraining the design
 9  set rpt_file "report.rpt"
10  set design "FullAdder"
11
12  current_design FullAdder
13  source "${script_path}/MyDefaults.con"
14
15  # Define design environment
16  set_load $MAX_OUTPUT_LOAD [all_outputs]
17  set_driving_cell -lib_cell $DRIVE_CELL [all_inputs]
18
19  # Define design constraints
20  set_input_delay  $INPUT_DELAY -clock $CLK_PORT [all_inputs]
21  set_output_delay $OUTPUT_DELAY -clock $CLK_PORT [all_outputs]
22  set_max_area 0
23
24  # Ungroup DesignWare parts
25  set designware_cells [get_cells -filter "@is_oper==true" *]
26  if {[sizeof_collection $designware_cells] > 0} {
27     set_ungroup $designware_cells true
28  }
29
30  set_fix_hold [all_clocks]
31  compile
32  write -format verilog -output ${netlist_path}/Syn_$design.v -hier
33  write -format ddc -hierarchy -output "${ddc_path}/${design}.ddc"
34  write_sdf ${netlist_path}/Syn_$design.sdf
35  source "${script_path}/report.tcl"
```

Fig. 8 An example of the constraint file "FullAdder.tcl"

The following are some remarks.

(i) In line 10, please make use the name "FullAdder" is the **SAME** as the name of the top module in your Behavioral Verilog file. **The same principle shall apply to other designs in Modules 4 and 5.**

(ii) In line 12, please make use the name "FullAdder" is the **SAME** as the name of the top module in your Behavioral Verilog file. **The same principle shall apply to other designs in Modules 4 and 5.**

(iii) In lines 30 to 35, the scripts are used to compile the design and generate the respective output files.

Close the file.

4. **Open** "./scripts/ReadVerilogFiles.tcl". There is only one command, i.e. to link the Verilog file, as show below. **Close** the file.

   ```
   read_verilog ./sources/FullAdder.v
   ```

   [Note: For other projects which may have many Verilog files, please link all Verilog files in "ReadVerilog.tcl".]

5. **Open** "./scripts/run_topdown.tcl". There are only two commands as shown below**.** The commands link all the required scripts together. **Close** the file.

   ```
   source "${script_path}/ReadVerilogFiles.tcl"
   source "${script_path}/FullAdder.tcl"
   ```

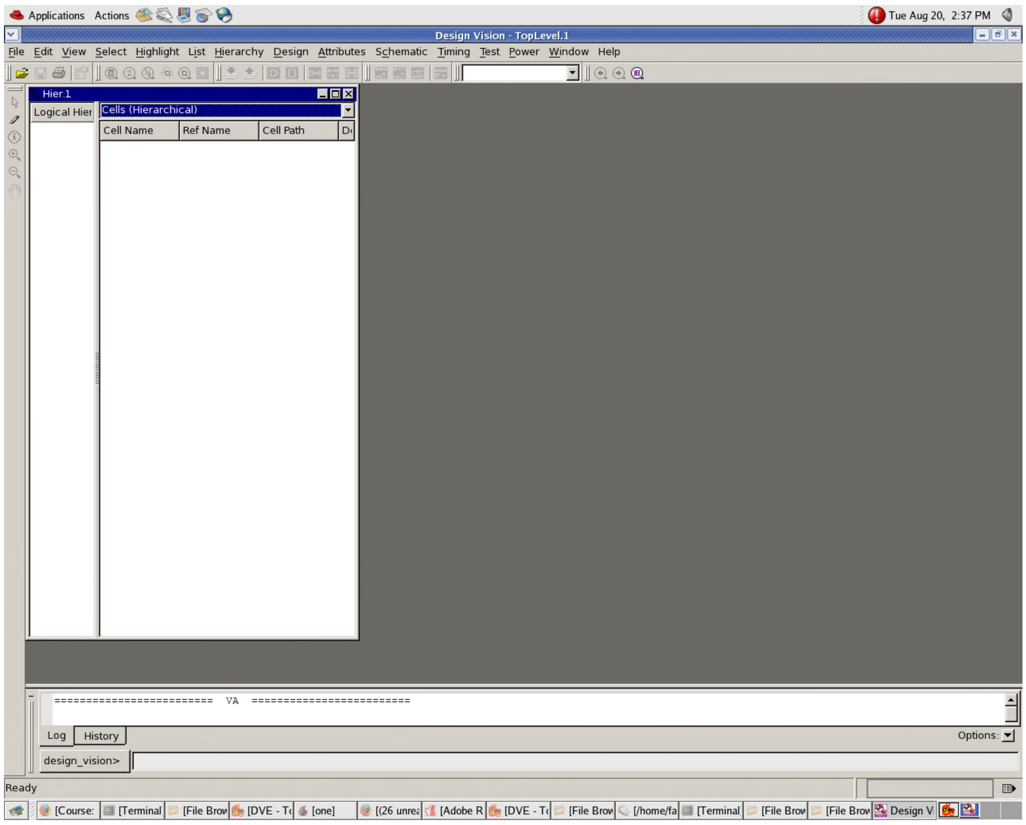
   [Note: For other projects, please make the **NAMES** of the file/folder/module consistent to avoid any possible errors due to missing files/modules]

C. Using the Synopsys Tool – Design Vision

1. **Open** a Terminal. **Type in** the following commands in the Terminal to gain access to the Synopsys Design Vision synthesis tool and STM 65nm process library.

   ```
   >> cd  /home/[your directory name]/Module3/Synthesis
   >> source  sourcefiles
   >> design_vision &
   ```

   The GUI is shown as below in Fig. 9.



```
========================  VA  ========================
```

Fig. 9 the GUI for Design Vision

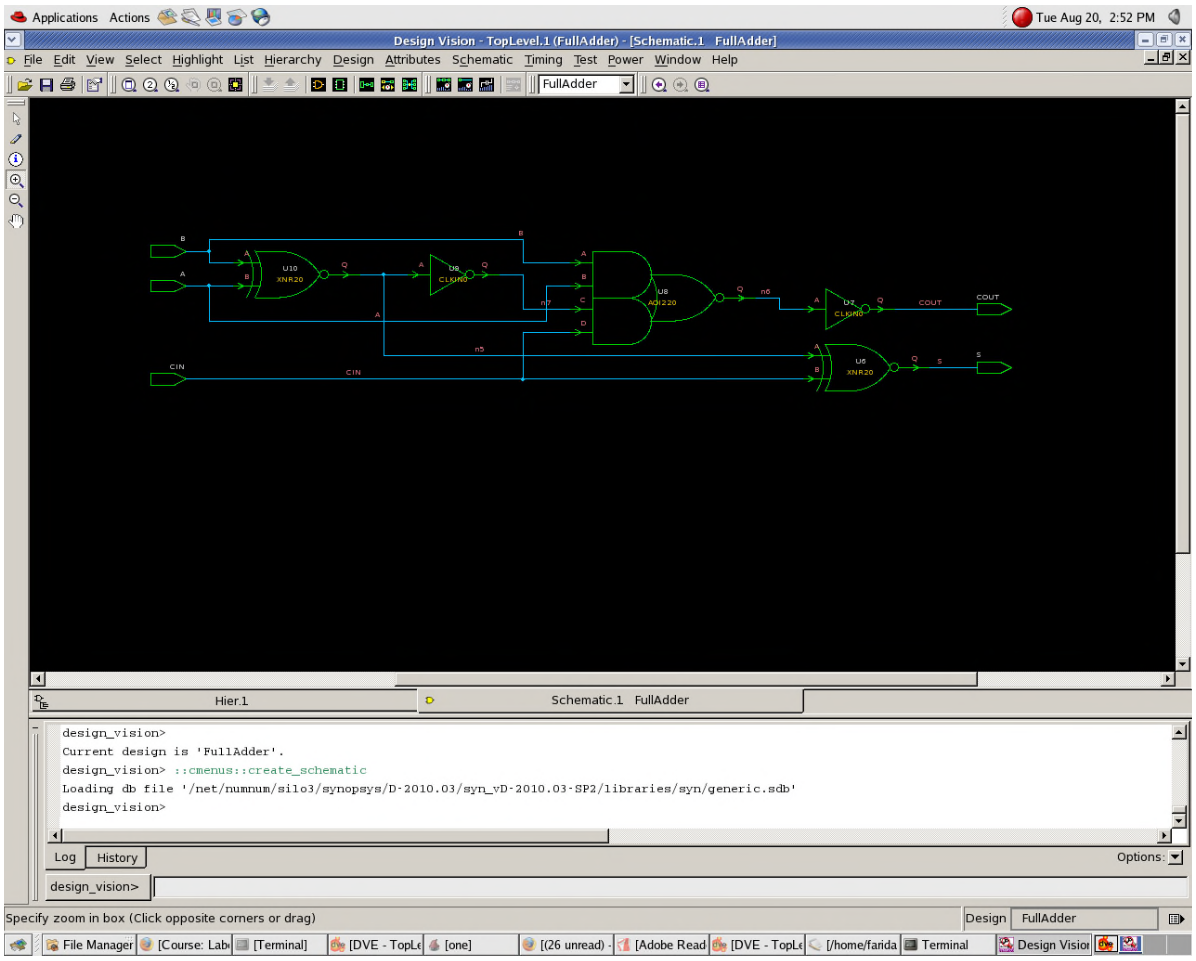
2. **Select "File"** on the toolbar and then **choose "Execute Script".**

3. **Choose** the folder **"scripts"** and then **select** the file **"run_topdown.tcl"**

   [Note 1: For combinational logic synthesis, you will see many error messages. It is probably OK if these error messages relate to a clock signal. If not, please check your setup files and/or Verilog files to make sure that they are error-free.]

   [Note 2: Instead of writing scripts, you could also use the GUI to insert all the commands in order to synthesize your design. However, it is quite common that you might need to re-synthesize your design many times (until it is error-free or you are happy with the result). Hence, writing scripts is still preferred.].

4. **Right-click** on the icon named FullAdder shown under the "**Logical Hierarchy"** tab and then **selects "Schematic View"**. The gate level netlist of the full-adder is obtained and thus you can visualise the structural design of the developed full-adder as shown in Fig. 10.



```
design_vision>
Current design is 'FullAdder'.
design_vision> ::cmenus::create_schematic
Loading db file '/net/numnum/silo3/synopsys/D-2010.03/syn_vD-2010.03-SP2/libraries/syn/generic.sdb'
design_vision>
```

Specify zoom in box (Click opposite corners or drag)

Fig. 10 A viewer to display the full-adder

5. After the synthesis, you should be able to view two generated files in the "**netlist**" folder.

   ./netlist/Syn_FullAdder.v
   ./netlist/Syn_FullAdder.sdf

6. After the synthesis, you should be able to view a generated report in the "**logs**" folder. Take a look what the report tells you about your design.

   ./logs/report.rpt

7. **Answer** the question in Section 3.A.2 in page 21.

At this stage if you have successfully gone through all the steps, you would have completed the **Synthesis**. Congratulations!

2.3 Post-Synthesis Simulations

The synthesized Verilog code has to be re-simulated to account for the delay introduced by the gates based on the STM 65nm CMOS cell library.

A. Working Environment:

1. **Go** to your home directory on the Desktop, and **choose** the folder “**Module3**”.

2. In the folder “**Module3**”, **choose** the folder “**RTL**”.

3. In the folder “**RTL**”, you should be able to see the following files.

   ./synopsys_linux_vj-2014_cshrc
   ./source files
   ./library.f
   ./run

4. **Copy** the synthesized netlist (i.e. Syn_FullAdder.v) and the standard delay format file (i.e. Syn_FullAdder.sdf) in the folder “**RTL**”. Refer back to Section 2.2.

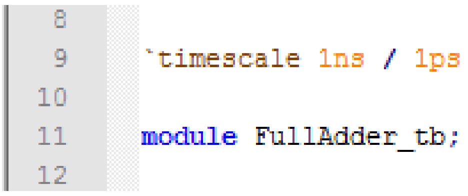
5. **Copy** the test-bench file (i.e. FullAdder_tb.v) and the input vector file (i.e. InputVector,txt) into to “**RTL**” folder. Refer back to Section 2.1.

Now, you are ready to start your hands-on.
[Note: Save all your files in the current folder, i.e. “**RTL**”]

B. Modifying a Behavioural Verilog Code for Creating a Test-bench

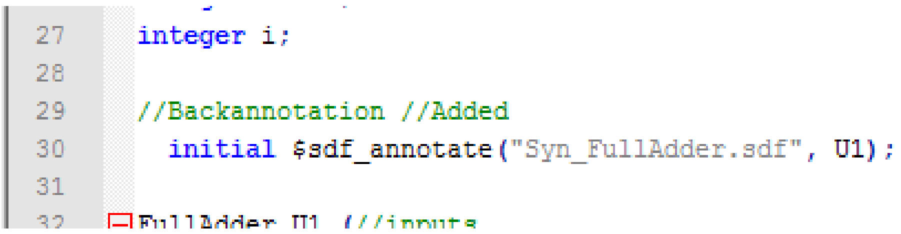
1. **Open** “FullAdder_tb.v” using Text Editor. **Insert** the following commands as follows:

   As shown in line 9, note the “‘timescale 1ns / 1ps” in order to make time format compatible with the STM 65nm CMOS cell library.



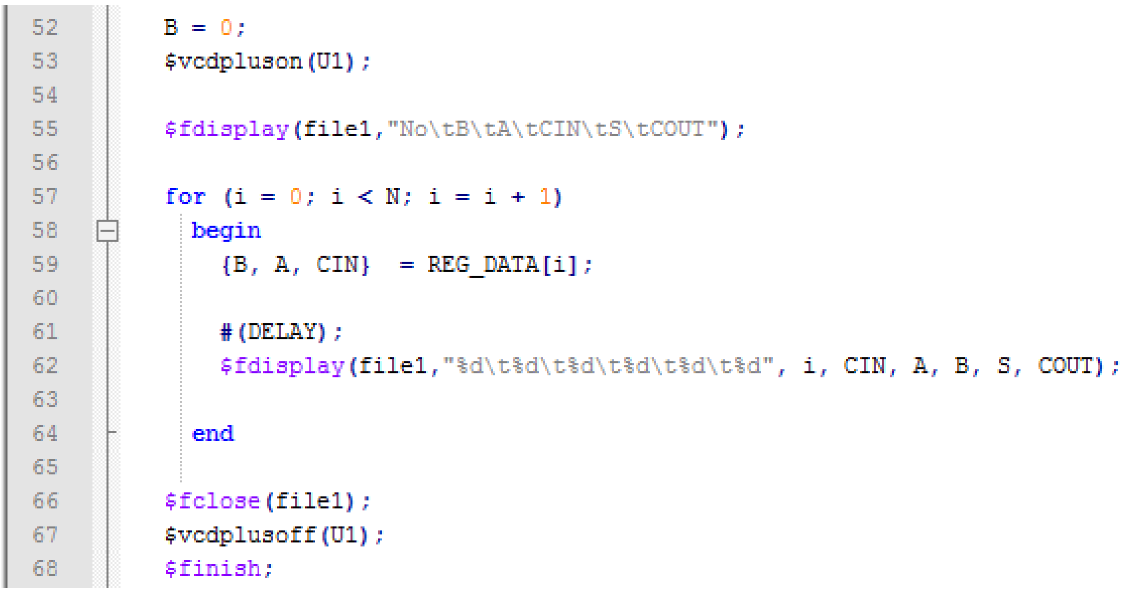
```
 8
 9  `timescale 1ns / 1ps
10
11  module FullAdder_tb;
12
```

As shown in line 30, “initial $sdf_annotate("Syn_FullAdder.sdf", U1);” to annotate the sdf file.



```
integer i;

//Backannotation //Added
  initial $sdf_annotate("Syn_FullAdder.sdf", U1);

FullAdder U1 (//inputs
```

Take note of line 53, “$vcdpluson(U1);” and in line 67, “$vcdplusoff(U1);” to enable the features of recording all the switching profile.



```
52    B = 0;
53    $vcdpluson(U1);
54
55    $fdisplay(file1,"No\tB\tA\tCIN\tS\tCOUT");
56
57    for (i = 0; i < N; i = i + 1)
58      begin
59        {B, A, CIN}  = REG_DATA[i];
60
61        #(DELAY);
62        $fdisplay(file1,"%d\t%d\t%d\t%d\t%d\t%d", i, CIN, A, B, S, COUT);
63
64      end
65
66    $fclose(file1);
67    $vcdplusoff(U1);
68    $finish;
```

2. **Save** the file.

C. Running Verilog Simulations

1. **Right-click** on Desktop and **open** a Terminal.

2. In the Terminal, **type in** the following commands:

```
>> cd /home/[your home directory name]/Module3/RTL
>> source synopsys_linux_vj-2014_cshrc
```

3. In the Terminal, **type in** the following command:

```
>> vcs -debug +v2k ./FullAdder_tb.v ./Syn_FullAdder.v –f ./library.f
```

4. **Check** the Terminal to see if any error message appears. If yes, PLEASE **re-check** your Verilog codes again and repeat the step 3. You must **fix all** errors indicated by the error messages! If no error message appears, **type in** the following command to invoke the GUI.

```
>> simv -gui
```

[Tip: as mentioned earlier, you can copy and save the command in the “run” file so that you can re-run the command as many times as possible]

5. In the generated DVE GUI, **select** the relevant the variables present under the hierarchy of FullAdder_tb.

6. **Right-click** on the selected variables and **choose “Add To Waves”** and then **select “New Wave View”.** A new DVE waveform GUI will pop up.

7. In the new popped-up DVE waveform window, **choose “simulator”** icon on the toolbar and then **select “Start/Continue”.**

8. You can now visualize the input and output waveforms.

9. If you **zoom into** the waveforms you can clearly identify the delay in the generation of output(s) after every input transition. An example is shown in Fig. 11.

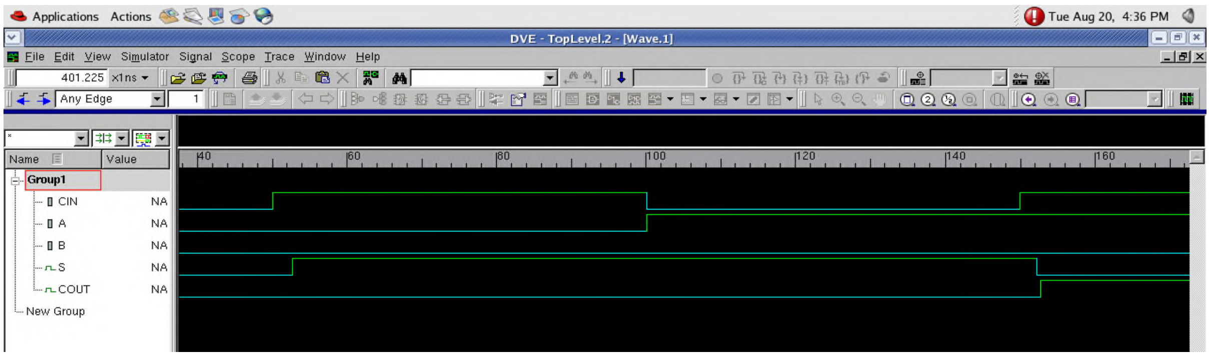


Fig. 11 Waveforms with timing information

10. **Answer** the question in Section 3.A.3 in page 21.

At this stage if you have successfully gone through all the steps, you would have completed the **RTL** simulation. Congratulations!

2.4 Power Simulations

The power simulations are based on the switching profile generated in Section 2.3.

A. Working Environment:

1. **Go** to your home directory on the Desktop, and **choose** the folder “**Module3**”.

2. In the folder “**Module3**”, **choose** “**RTL**”.

3. In the folder “**RTL**”, **choose** the folder “**PowerAnalysis**”.

4. In the folder “**PowerAnalysis**”, you should be able to see the following files:

   ./.synopsys_dc.setup
   ./run

Now, you are ready to start your hands-on.
[Note: Save all your files in the current directory, i.e. “**PowerAnalysis**”]

B. Simulating the Power Dissipation:

1. **Right-click** on Desktop and **open** a Terminal.

2. In the folder /Module3/RTL/PowerAnalysis/, modify common_setup_STM.tcl

   In the line of set ADDITIONAL_SEARCH_PATH1, change the path from “/net/vlsiapp/usr6/library/STM/cmos065_540/CORE65GPSVT/5.2/libs” to “home1/msc01/Desktop/TUM2024/Module3/Library” (modify this path as your own case)

3. T**ype in** the following commands:

```
>> cd /home/[your home directory name]/Module3/RTL/PowerAnalysis
>> source ../sourcefiles
>> vpd2vcd ../inter.vpd inter.vcd
>> vcd2saif -i ./inter.vcd -o inter.saif
>> dc_shell-xg-t
```

   [Tip: you can save these commands into the “run” file].

4. **Type in** the following commands:

```
>> read_verilog ../Syn_FullAdder.v
>> current_design FullAdder
>> link
>> read_saif -input ./inter.saif -instance FullAdder_tb/U1
>> report_power >> "rpt_power.txt"
>> report_power -hier >> "rpt_power_hier.txt"
>> report_power -hier -nosplit >> "rpt_power_hier_nosplit.txt"
>> report_power -cell >> "rpt_power_cell.txt"
```

[Tip: you can save these commands into the "run" file. For other project, please take note the file/module **NAMES**].

5. Four different reports namely "rpt_power.txt", "rpt_power_cell.txt", "rpt_power_hier.txt", "rpt_power_hier_nosplit.txt" will be generated upon executing the power simulation. Examine these reports, and answer the question in Section 3.A.4 in page 21.

At this stage if you have successfully follow through all the steps, you would have completed the **Power** simulation. Congratulations!

Section 3: Assignment Questions

A. Basic Questions

1. In Section 2.1, fill in the truth table based on your simulated waveforms.

Time	Inputs			Output	
	CIN	A	B	COUT	S

2. In Section 2.2, refer to the file "report.rpt" and fill in the following:

   What are the gates used for realizing the full-adder?

   ____
   ____
   ____

   The COUT delay of the full-adder

   ____

   The S delay of the full-adder

   ____

3. In Section 2.3, based on the given simulated waveform, what are the worst-case delays for the COUT and S signals?

   ____
   ____

4. Compare the power dissipation generated by the "report.rpt" in Section 2.2 and by the reports in Section 2.4. Comment on these results.

   ____
   ____
   ____
   ____
   ____

B. Optional Questions

1. Write a Test-bench Verilog Code to generate all the different possible input pattern transitions?

   [Hint: There are 56 patterns in total]

2. Carrying on from B1, find out the critical delay(s) for the full-adder. What are the possible value(s) of inputs (both the present inputs to the next inputs) that cause the critical delay(s)?