

**The University of Texas at Dallas**  
**Dept. of Electrical Engineering**  
**EEDG/CE 6301: Advanced Digital Logic**  
**EEDG/CE 6303: Testing and Testable Design**  
**Instructor: Mehrdad Nourani**

**A Quick Tool Setup and Tutorial  
for Simulation & Synthesis Using SYNOPSIS Toolset**

**DISCLAIMER:** *The following steps aim at your setting SYNOPSIS toolset and running a sample VHDL program. SYNOPSIS is a large commercial CAD tool suite with many additional options that are not explained here. What we explain here is the minimum to run few tools. With this new version of SYNOPSIS that we installed, there might be problems in this document and some commands may not work exactly as explained. Proceed cautiously and use it at your own risk. We encourage you to assign enough time to familiarize yourself with this tool to be able to use it efficiently. Please report problems, corrections and suggestions about this document to [nourani@utdallas.edu](mailto:nourani@utdallas.edu)*

## **1. Setting up SYNOPSIS in your account**

Most of the CAD tools, including SYNOPSIS tools required in this course, are accessible in various labs with UNIX workstations or PCs in ECS building but we recommend using machines in the ECSN 4.324 (Solarium Lab) that run faster speed and have larger RAM. You may also use machines in the ECS Open Lab ECSS 2.104 to remote login to solarium lab. However, due to frequent upgrades and resets, you may sometimes experience difficulties. Many of these tools, including SYNOPSIS, provide shell commands to allow users run them without graphic user interface. This means that you can telnet to one of these machines remotely and run them. You need to be familiar with the basic UNIX commands and one UNIX text editor (VI, EMACS, GVIM, EDIT, etc.). The “Quick UNIX Guide” also posted in the course website can be a good start.

- First you need to connect to a Unix machine/server. To do this, you need a PC that runs NX Client. Set up the Nomachine and connect to “engnx” server. Please follow the instructions posted in the link below:

<https://personal.utdallas.edu/~xxx110230/nx/>

For off-campus connection, you need VPN (check <https://www.utdallas.edu/oit/vpn/>). You may also need to download the nomachine remote access software (check <https://www.nomachine.com/download>)

- Type in the following command at the terminal window in order to enable the environment variables:

**source /proj/cad/startup/profile.synopsys\_2018\_vcs\_2021**

- Note that if you source differently (i.e. the way that you source CAD files for other courses), you will need to close the console and open a new one to source as explained above. This would allow

the environment variables to be taken into effect correctly. Having done these, you are ready for playing with VHDL and Vcs.

- If you have VPN in your home laptop/PC, you can do the same from home. For how to install VPN, please visit the information resource page in UTD website:  
<https://oit.utdallas.edu/howto/vpn/>

## 2. Steps for compilation with vcs

Main steps in the SYNOPSYS VHDL System Simulator (VSS) environment are:

1. VHDL Code Development: You can use any text editor for this step.
2. VHDL Analyzer to compile your code using the command **vhdlan**.
3. Create a simulation executable using **vcs**.
4. Simulating the design using **simv**.

Helps and detailed explanations are available with many options including:

- Manual pages are available for all the commands, e.g. **man vhdlan** provides information for analyzer
- When you get to each environment, help is often provided by **help command**

## 3. Compiling your code

To effectively manage all content created, a separate directory may be created for all work done with Synopsys, for example: **ee6303**. Create another directory called **work** inside this **ee6303** directory. A simple adder circuit has been constructed in this tutorial and two VHDL files (adder.vhd and tbadder.vhd) are available in the course webpage and also at the end of this document. Save these two files in your **ee6303/work** directory. Create another directory under **ee6303/work**, named **WORK**. Files generated from the tool would be automatically saved in **WORK**. The following steps demonstrate how this adder circuit can be compiled and simulated with **vcs**. Additionally, a **Verilog** version of the adder circuit has been included at the end of this document as well (adder.v and tbadder.v).

The first step as mentioned above is to compile the code. In the following command line, more than one code file is compiled at the same time which can also be done individually.

*Note:* The order of compilation does matter here.

```
$ vhdlan adder.vhd tbadder.vhd
```

The files would be compiled successfully if there were no errors. The next step is to create the simulator executable using **vcs** which is done by the following command line.

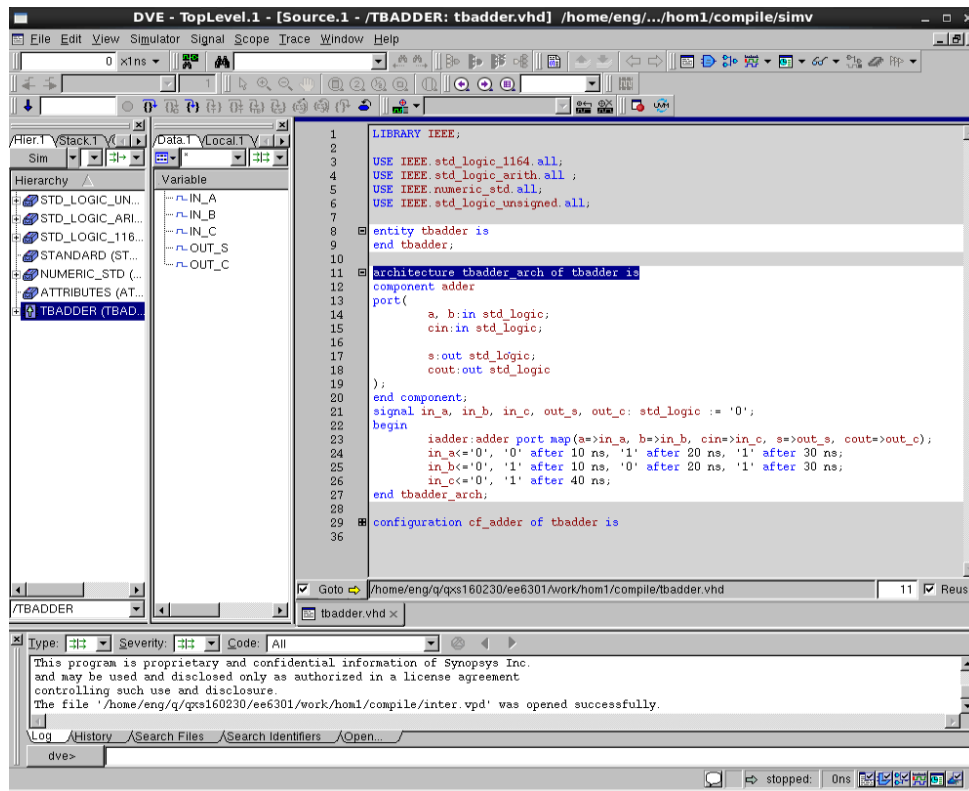
```
$vcs cf_adder -debug
```

Note that cf\_adder is the configuration that is present in the tbadder.vhd file.

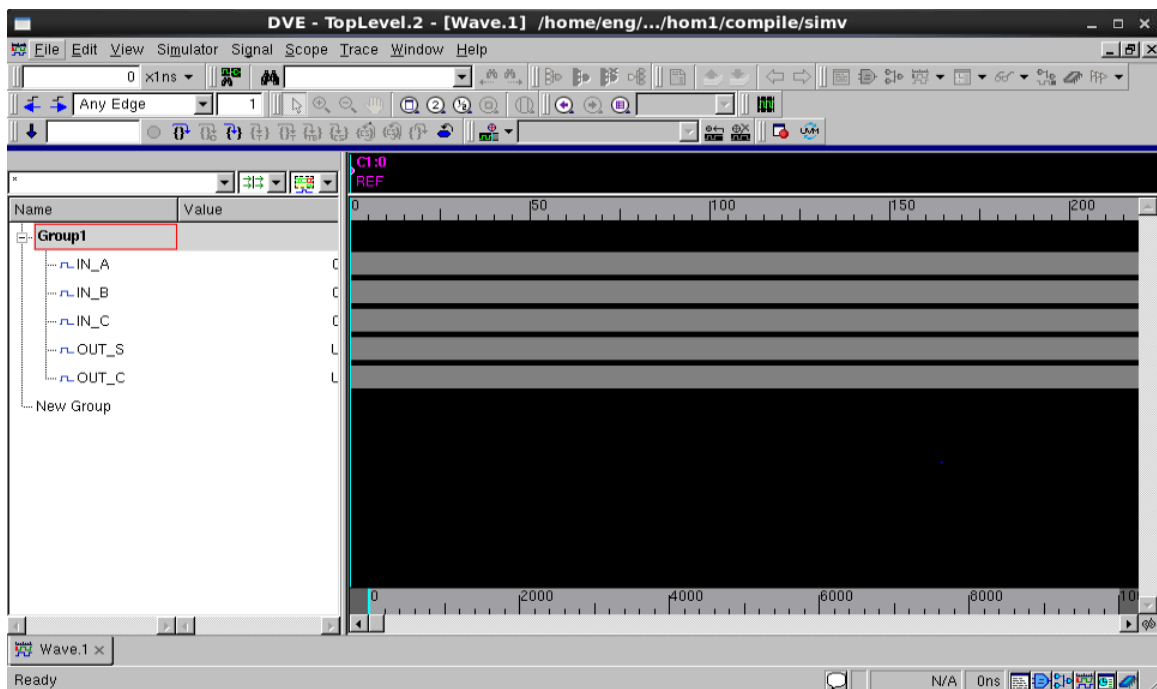
## 4. Simulation

Although a text mode is also available for simulation (explained later), for the sake of more intuitive introduction, the graphical mode is explained first.

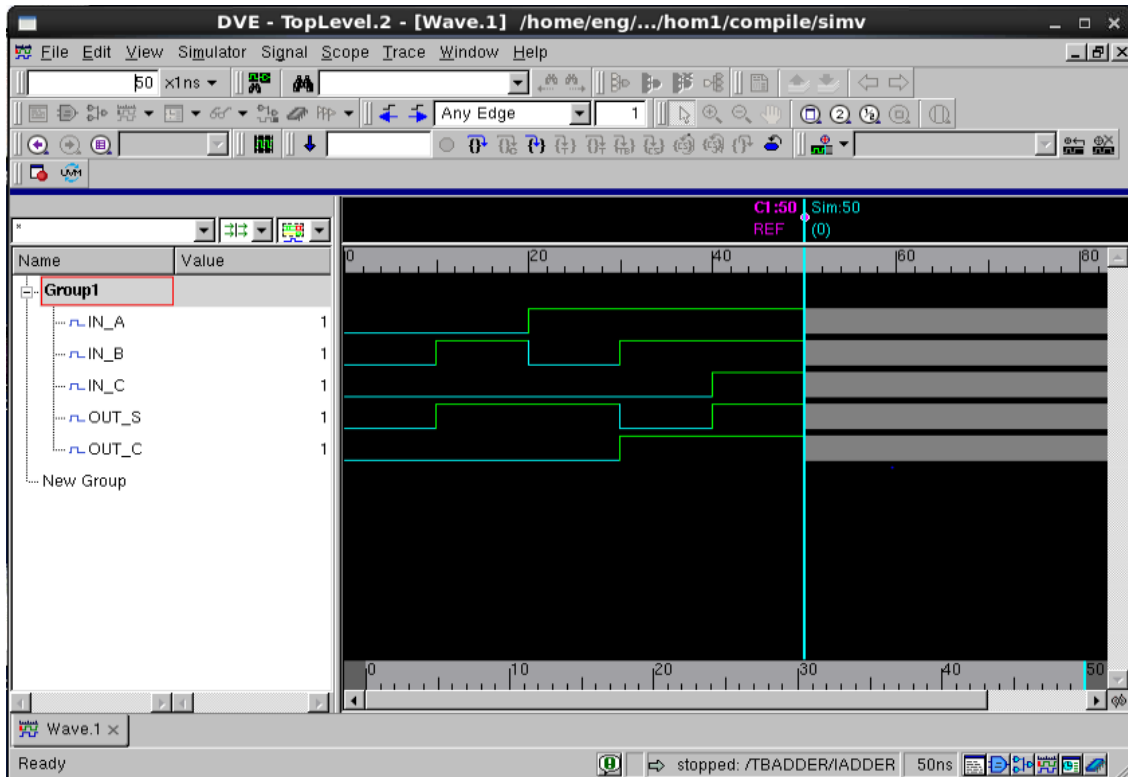
1. Launch the simulation tool Vcs by typing **./simv -gui &** at the prompt. Following window should pop-up after a few seconds.



2. Select all the variables and add to the new wave view. (Select all variables -> Right click -> Add to Waves -> New Wave View)



3. In the top-left textbox, type the number **50** and press **Enter** to run your simulation for 50 ns. Alternatively, type **run 50** in the Sets Current Time in the Waveform window and press **Enter** to run the simulation all in one shot for 50 ns.



## 4.2 Using Vcs with Verilog

Instead of using Vhdlan, at the Command Prompt Type

**\$ vlogan <source>**

This should analyze the Verilog Source file. And the following steps should be same as the commands for the vhdl file.

## 5. Synthesis (script based)

- (a) First open up a shell (terminal) window and source accordingly to enable environment settings:  
**source /proj/cad/startup/profile.synopsys\_2018\_vcs\_2021**
- (b) All design must be done in the directory that was created for simulation and NOT outside. For this example we will use our working directory (e.g. **ee6303/work**) indicated as “.” In the file. Verify that `.synopsys_dc.setup` file is present in this directory.

Content of this file is as follows:

```
define_design_lib WORK -path ./WORK
lappend search_path { } + search_path
set link_library \class.db
set target_library \class.db
set symbol_library \class.sdb
```

**Note 1:** If you do not have this file, in a text editor, copy-paste the above text, and save the file as `.synopsys_dc.setup` under `ee6303/work`.

**Note 2:** Be extra cautious about `.synopsys_dc.setup` file. It appears that if you make mistakes (even syntax or command errors) in this file, synopsys does not give any error message. Instead, it ignores the file and runs with other default options without explicitly saying it. To make sure about the correct library in `dc_shell` you can use: **which target\_library\_filename, which link\_library\_filename** and **which symbol\_library\_filename** to see if it reports the correct one.

**Note 3:** If you change a library (e.g. from `class` to **gtech** or **lsi\_10k** to be able to use special cells), you better change all three (link, target and symbol) libraries. In particular, if the symbol library (`*.sdb`) is missing, synopsys tools cannot provide timing and area statistics/reports.

(c) **cd ~/ee6303**

(d) **cd work**

(e) **dc\_shell**

After entering the environment it gives you a prompt.

(f) **analyze -f vhdl adder.vhd**

Note: **analyze** is required for each file in the hierarchical design starting from the bottom up. For a specific library we can also use **analyze -format vhdl -library WORK adder.vhd**

(g) **elaborate adder**

(h) Note: **elaborate** works on the entity name of the file (not the file name itself) read in step (e). We need to do it only for the top level file. (**elaborate** has other options with **-arch** and **-update** to identify the architecture declared in the file.)

(i) **list\_designs**

(j) **uniquify**

(k) **compile**

(l) **report\_cell**

(m) **report\_area**

Note: The area is reported based on 2-input NAND gate.

(n) **report\_timing**

(o) **write -hierarchy**

Note: This commands writes all designs in the hierarchy. The default format is **ddc** format which is the Synopsys internal database format. Using **write -h -f format** the format can be chosen among **ddc** (Synopsys internal database format) **edif** (Electronic Design Interchange Format)

**vhdl** (VHDL netlist based on technology files) and **verlog** (Verilog netlist based on technology files). For example you can save the compiled results using:

**write -f vhdl -output adder\_synopsys.vhdl**

(p) **quit**

(q) All of the commands can be combined into one file (e.g. **adder.dc\_shell**). Then, using command: **include adder.dc\_shell**. it reads in and executes the file as **dc\_shell** commands. Include files can contain commands and comments. Using include files is an easy and proper way to design, keep the history of activities and communicate with others. For instance, for our example, this could be the content of **adder.dc\_shell** file:

(r) To execute the command file, use **dc\_shell -f adder.dc\_shell**

```
/* ----- */
/* adder.dc_shell (script) file */
/* ----- */
analyze -format vhdl -library WORK adder.vhd
elaborate adder
report_hierarchy
uniquify
compile
report_cell > adder.report /* > writes to a file */
report_area >> adder.report /* >> appends to a file */
report_timing >> adder.report
write -f vhdl -output adder_synopsys.vhdl
exit
```

(s) Other commands to explore: **current design, report area, file name.txt, report port, report timing, list - commands, help dc\_shell command, history**. Also, using **sh unix command** you are able to execute UNIX shell command while you are in the **dc\_shell** environment.

## 6. Viewing Logic (GUI based)

- Open up a shell (terminal) window and source accordingly to enable environment settings:  
**source /proj/cad/startup/profile.synopsys\_2018\_vcs\_2021**
- Make sure you have parsed you desired HDL file with no errors using:  
**vhdlan "your\_design".vhd** at the prompt.
- **design\_vision** is just a GUI that is built on top of **dc\_shell**. Every command in **dc\_shell** can be done with the file menu. Remember to place the **.synopsys\_dc.setup** file in the work directory. The content of this file is available in the previous section (for people who skipped section 5).

(a) **cd ~/ee6303**

(b) **cd work**

(c) **design\_vision &**

(d) **File > Analyze > ADD adder.vhd Select>OK**

(e) **File > Elaborate > library: click on WORK; Design: click on adder(adder\_arch) > OK**

(f) **File > Save**

(g) **File > read > adder.ddc > OK**

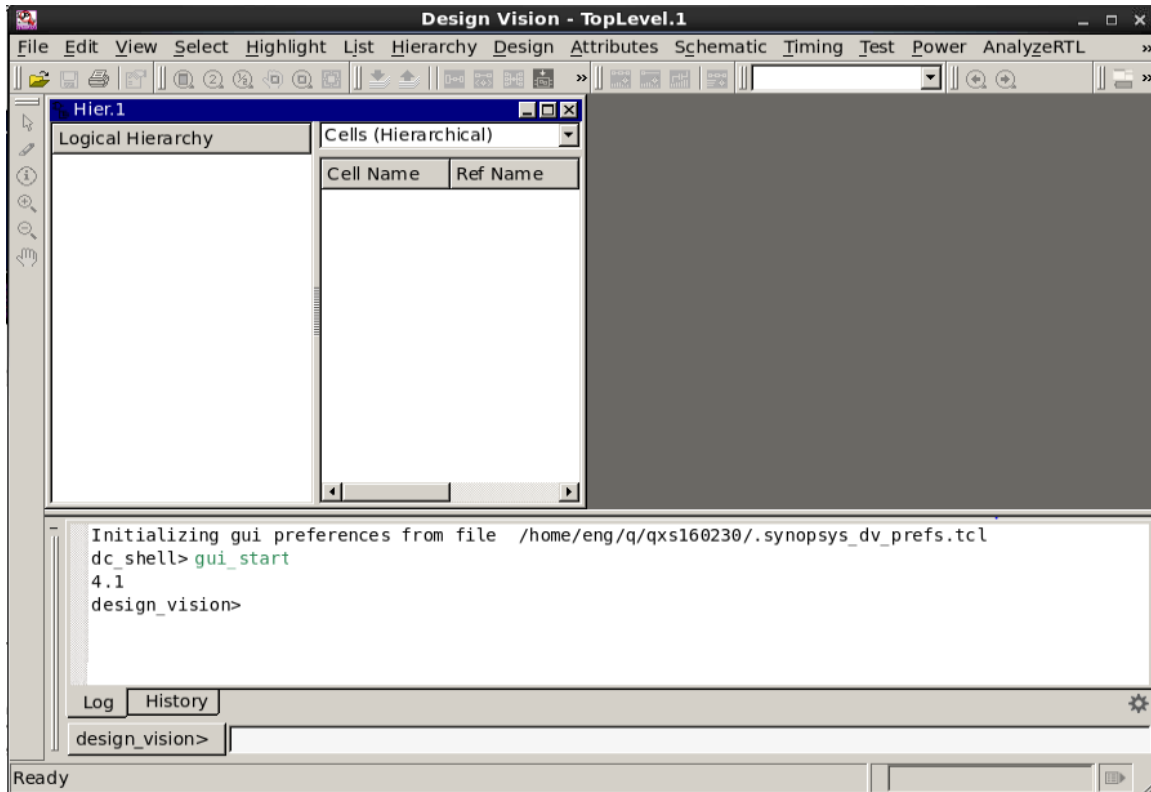
(h) When the system level box is shown, double click on icon until logic is displayed.

(i) **File > Print Schematic**

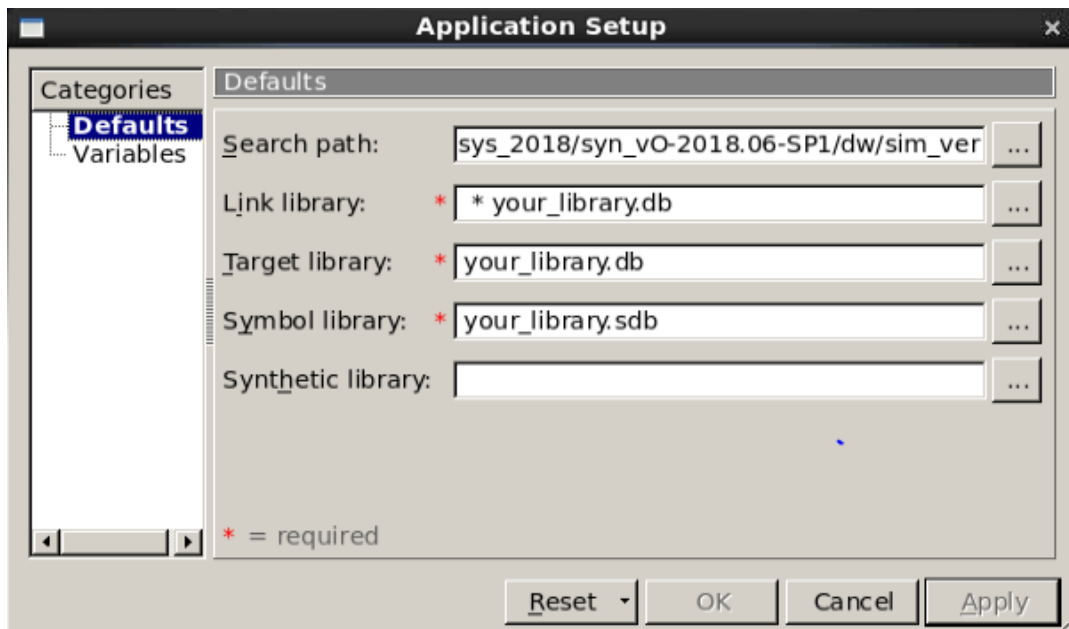
- (j) **Design > compile design**
- (k) You can choose different reports, including Area and Timing. **Design > Report Area**, or **Timing > Report Timing Paths**. Then click on **OK**. The area is reported based on 2-input NAND gate.
- (l) If you wish to synthesize another design, **File > Remove All designs**, parse it using **vhdlan** “your\_design”.vhd at the prompt, and then go through the steps from (d) for your other design(s).
- (m) **To quit, File > Exit > OK**

Here is a graphical view of the main steps above:



- When you type in **design\_vision** at the prompt. The GUI looks like this:



- You can click on **File->Setup** to change the link, target and symbol library. By default, it should contain the libraries set in the .synopsys\_dc.setup file.



In place of your\_library, type in `class.db`, `class.db`, and `class.sdb` `generic.sdb` in front of Link, Target and Symbol Library, respectively. If `gtech` library is required, replace `class.db` with `gtech.db` and `class.sdb` `generic.sdb` with `gtech.db` `generic.sdb`. Then, click **OK**.

- **File->Analyze.** You would need to **ADD** your desired file with the desired format.
- **File->Elaborate** by clicking on the desired design and directory.
- Double click on the gate symbol  under Logical Hierarchy to enable the schematic buttons: 
- Other steps such as **Hierarchy->Uniquify** can be done by clicking and selecting the schematic block.
- To compile the design, click **Design->Compile Design**.
- Save your design, with desired format.
- Imp: Always close design vision by: **File -> Exit**. If not exited, this will not release the license for next use.

## 7. Post-Synthesis Simulation

While automatic synthesis saves a lot of designer's time, it often generates unpredictable, and sometimes even surprising, results due to various interpretations of the VHDL constructs by the synthesis tools. Over time designers and VHDL programmers learn how to avoid pitfalls and difficulties which are tool-dependent. Therefore, it is common to do post-synthesis simulation to make sure that major specifications



and constraints, e.g. functionality, timing behavior, etc. all are satisfied. One way of post-synthesis simulation is to generate a VHDL **structural** description of the gate-level circuit after synthesis and re-simulate it (perhaps using the same test vectors used before) and compare the result with the simulation result prior to synthesis. Suppose we have generated **adder.ddc** in the synthesis phase:

- (a) **cd ~/ee6303**
- (b) **design\_vision &**
- (c) **File > read > adder.ddc > OK**  
When the system level box is shown, double click on icon until logic is displayed
- (d) **File > Save As**  
Choose the name (e.g. **adder\_post.vhd**) and select **File Format: VHDL**. Then: **> OK**
- (e) Now that the gate level structural file **adder\_post.vhd** is created you can resimulate it and compare the results. To simulate it, just follow the same simulation procedure discussed in Section 4.

### **Files needed for running this simulation (VHDL):**

The content of the adder.vhd file is as follows:

```
-----  
  
LIBRARY IEEE;  
  
USE IEEE.std_logic_1164.all;  
USE IEEE.std_logic_arith.all ;  
USE IEEE.numeric_std.all;  
USE IEEE.std_logic_unsigned.all;  
  
entity adder is  
port(  
    a, b:in std_logic;  
    cin:in std_logic;  
  
    s:out std_logic;  
    cout:out std_logic  
);  
end adder;  
  
architecture adder_arch of adder is  
begin  
    s<=(a xor b) xor cin;  
    cout<=(a and b) or (cin and a) or (cin and b);  
end adder_arch;  
  
-----
```

The content of the tbadder.vhd (test bench) file is as follows:

```
-----  
  
LIBRARY IEEE;  
  
USE IEEE.std_logic_1164.all;  
USE IEEE.std_logic_arith.all ;  
USE IEEE.numeric_std.all;  
USE IEEE.std_logic_unsigned.all;  
  
entity tbadder is  
end tbadder;  
  
architecture tbadder_arch of tbadder is  
  component adder  
  port(  
    a, b:in std_logic;  
    cin:in std_logic;  
  
    s:out std_logic;  
    cout:out std_logic  
  );  
end component;  
signal in_a, in_b, in_c, out_s, out_c: std_logic := '0';  
begin  
  iadder:adder port map(a=>in_a, b=>in_b, cin=>in_c, s=>out_s, cout=>out_c);  
  in_a<='0', '0' after 10 ns, '1' after 20 ns, '1' after 30 ns;  
  in_b<='0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns;  
  in_c<='0', '1' after 40 ns;  
end tbadder_arch;  
  
configuration cf_adder of tbadder is  
  for tbadder_arch  
    for iadder:adder  
      use entity WORK.adder (adder_arch);  
    end for;  
  end for;  
end cf_adder;  
  
-----
```

**Files needed for running this simulation (Verilog):**

The content of the adder.v file is as follows:

```
-----  
module adder(  
    input a, input b, input cin,  
    output s, output cout  
);  
  
assign s = (a ^ b) ^ cin;  
assign cout = (a & b) | (a & cin) | (b & cin);  
  
endmodule  
-----
```

The content of the tbadder.v (test bench) file is as follows:

```
-----  
`timescale 1ns/1ps  
module tbadder(  
);  
reg in_a, in_b, in_c;  
wire out_s, out_c;  
adder iadder(in_a, in_b, in_c, out_s, out_c);  
  
initial begin  
    in_a <= 0;  
    in_b <= 0;  
    in_c <= 0;  
    #10;  
    in_b <= 1;  
    #10;  
    in_a <= 1;  
    in_b <= 0;  
    #10;  
    in_b <= 1;  
    #10;  
    in_c <= 1;  
end  
  
endmodule  
  
config cf_adder;  
    design WORK.tbadder;  
    default liblist WORK;  
endconfig  
-----
```