Getting Started with Xilinx WebPack 13.1

B. Ackland

June 2011

(Adapted from S. Tewksbury notes WebPack 7.1)

This tutorial is designed to help you to become familiar with the operation of the WebPack software by entering a simple VHDL design (a full adder), compiling the VHDL and simulating the design using the Xilinx ISim simulator.

1. Entering the Design

After you have downloaded and installed WebPack (separate instructions are provided for this) and activated the license, you can proceed through the following simple digital system design.

The software we will be using is the Xilinx "ISE Project Navigator", in which you can enter the VHDL design, verify its syntax, and simulate its outputs for a specified sequence of inputs. To start up, go to

start⇒All Programs⇒Xilinx ISE Design Suite 13.1⇒ISE Design Tools⇔Project Navigator

The screen shown in Figure 1 appears.

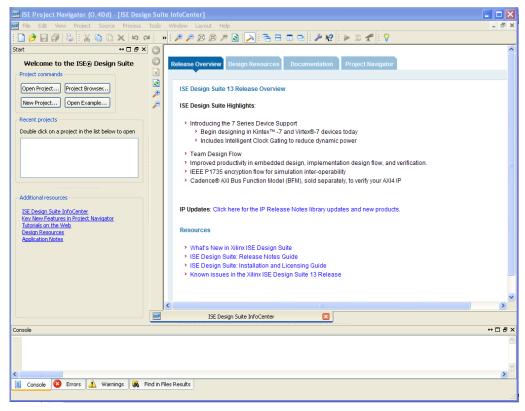


Figure 1

At this point, you have not yet created any project so we start there. A "project" is a set of files, including VHDL code files, that define the circuit components and overall circuit, along with auxiliary files related to simulations and other parts of the design performed. You can either click *New Project* or go to File -> New Project. This launches the New Project Wizard as shown in Figure 2. Enter a project name and location. I would suggest you create the project in the "My Documents" folder, rather than the default (C:\ project_name) to avoid adding lots of new files into your main

directory. The project name for this tutorial is *full_adder*. Click *Next* - this will take you to the **Design Properties** pop-up as shown in Figure 3.

ISE	New Project Wiza	rd	×
•	C reate New Project Specify project loc	ation and type.	
	-Enter a name, locatio	ons, and comment for the project	
	Name:	full_adder	
	Location:	C:\Documents and Settings\Bryan\My Documents\MyVHDL\full_adder	
	Working Directory:	C:\Documents and Settings\Bryan\My Documents\MyVHDL\full_adder	
	Description:		
	-Select the type of to	p-level source for the project	
	Top-level source typ	e:	
	HDL	×	
			2
C	More Info	Next > Cancel	

Figure 2

Design Properties		
Name:	full_adder	
Location:	C:\Documents and Settings\Bryan\My Documents\MyVHDL\full_adder	
Working directory:	C:\Documents and Settings\Bryan\My Documents\MyVHDL\full_adder	
Description:		
Project Settings		
Property Name	Value	~
Top-Level Source Type		-
Evaluation Development Board	None Specified	~
Product Category	All	~
Family	CoolRunner2 CPLDs	~
Device	XC2C256	~
Package	TQ144	~
Speed	-7	♥ =
Synthesis Tool	XST (VHDL/Verilog)	~
Simulator	ISim (VHDL/Verilog)	~
Preferred Language	VHDL	~
Property Specification in Project File	Store all values	~
Manual Compile Order		
VHDL Source Analysis Standard	VHDL-93	~
Enable Message Filtering		~
and a consider morning		
	OK Cancel Hel	2

Figure 3

In the **Design Properties** window, you can specify the Device Family and the Device. This is not critical for this tutorial. I selected the CoolRunner2 family of CPLDs and the XC2C256 device. If you were using the project to generate

the configuration files for a particular CPLD or FPGA, then you would be entering the device family and device for the specific device you would be using. This is needed when you reach the "synthesis" stage in the design and implementation since the synthesis stage needs to know the internal design of the specific programmable device. We will not be synthesizing the design in this tutorial. When you click *OK*, a **Project Summary** window will pop-up. Click *Finish*.

We are now ready to begin entering our VHDL code. From the Project Navigator's pull-down menus, select **Project -> New Source**. This launches the **New Source Wizard** as shown in Figure 4. This window can be used to add a variety of sources to your design. Select "VHDL Module" and add the name of your circuit. The box "Add to project" should be checked. The location should appear as the directory you selected above. Click *Next* and the screen shown in Figure 5 appears.

📧 New Source Wizard	×
Select Source Type Select source type, file name and its location.	
 Schematic User Document Verilog Module Verilog Test Fixture VHDL Module VHDL Library VHDL Package VHDL Test Bench 	File name: full_adder Location: Settings\Bryan\My Documents\MyVHDL\full_adder
More Info	Next > Cancel

Figure 4

The screen in Figure 5 allows you to enter the "ENTITY" ports (inputs and outputs). For the full adder, we have three inputs: *a*, *b* and *cin* (carry-in) and two outputs *sum* and *cout*. Enter these ports and select *in* or *out* as appropriate. Click on *Next* and the wizard will supply a summary as shown in Figure 6. We have also named the *architecture* that we will be creating – in this case we have called it *gate_level*.

🚾 New Source W	/izard					×
Define Module Specify ports	for module.					
Entity name	full_adder					
Architecture name	gate_level					
	Port Name	Direction	Bus	MSB	LSB	^
а		in	✓			
Ь		in	✓			
cin		in	✓			
sum		out	✓			=
cout		out	✓			
		in	✓			
		in	✓			
		in	✓			
		in	✓			
		in	v			
More Info		<	Back	Next >	Cance	



151	New Source	e Wizard			X
	Summary Project N	avigator will crea	te a new skeleton source with the	e following specifications.	
	Add to Project Source Directo Source Type: V Source Name:	ry: C:\Document: 'HDL Module	s and Settings\Bryan\My Docume	nts\MyVHDL\full_adder	
	Entity name: fu Architecture na Port Definitions	ame: gate_level			
		a b cin	Pin Pin Pin	in in in	
		sum cout	Pin Pin	out out	
	More Info			< Back Finish Cancel	

Figure 6

Clicking on finish in Figure 6, the **Project Navigator** screen reappears and you will see that the wizard has already generated some VHDL code, including the *entity* declaration and a skeleton *architecture* (along with some comments) as shown in Figure 7. You will see that the wizard has also included the *library* and *use* commands to give you access to the to the IEEE *std_logic* library. Note that the template has declared the ports to be all of type *std_logic*, which works well for our tutorial example. If you wanted them to be a different type, then you can simply edit this part of the VHDL source code.

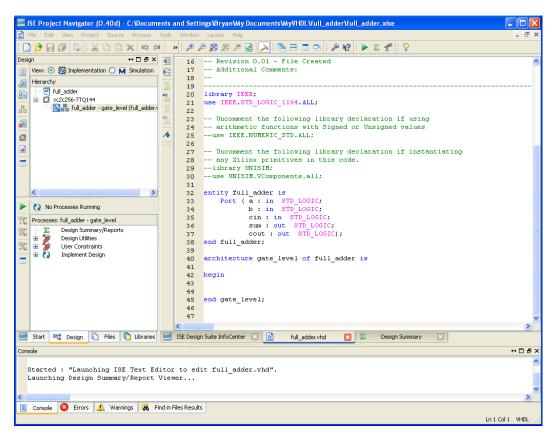


Figure 7

For this example, we will be using the VHDL code shown below. (I have removed the comment information from the window shown in Figure 7 above when generating this file listing). This is the full adder implementation shown in Figure 4.3 of Yalamanchili.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full adder is
  Port (a : in STD_LOGIC;
      b: in STD_LOGIC;
      cin: in STD LOGIC;
      sum : out STD_LOGIC;
      cout : out STD_LOGIC);
end full adder:
architecture gate_level of full_adder is
signal s1,s2,s3:std_logic;
begin
   s1 \ll (a \text{ xor } b) \text{ after } 5 \text{ ns};
   s2 <= (cin and s1) after 3 ns;
   s3 <= (a and b) after 3 ns;
   sum <= (s1 xor cin) after 5 ns;</pre>
   cout <= s2 or s3 after 3 ns;
end gate level;
```

Enter this code into the *full_adder.vhd* window as shown in Figure 8.

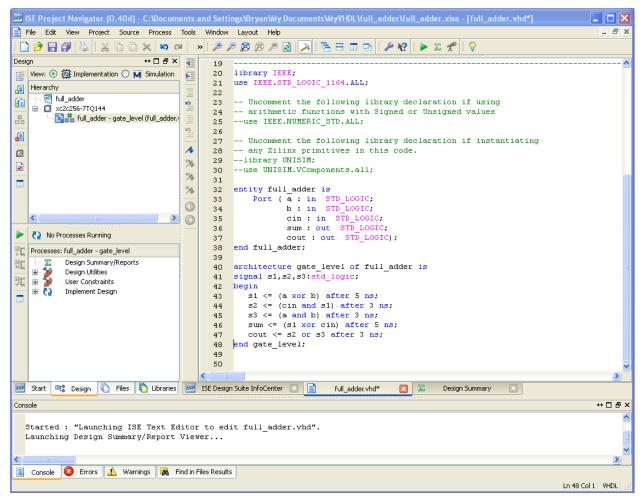


Figure 8

Before proceeding, we should check to ensure that we have made no syntactical errors. Save your VHDL file using **File -> Save**. Set the *View:Simulation* radio button at the top of the *Design panel* (see Figure 9). Select the *full_adder-gate_level* file in the *Hierarchy* window. Then double-click *on Behavioral Check Syntax* in the *Processes* window. This will find any syntax errors. If your design is syntax error-free, you will be notified that your files parsed correctly in the *console* sub-window as shown in Figure 9. If not, there will be a series of error messages in this window.

To illustrate what happens when an error occurs, change the VHDL code to create an error. Looking at the VHDL program window in Figure 8, change the reference to signal *a* in line 43 to be *aa* as shown in Figure 10. Save the file and then run *Behavioral Check Syntax*. The compiler will now generate error statements in the console window as shown in Figure 10 (you may have to scroll the console window to see these). If you click on one of these errors (click on the pink filename), a yellow pointer will appear next to line 43 showing you where the error occurred. Correct the error, save the file and run Behavioral Check Syntax to make sure your code is, once again, correct.

🔤 ISE Project Navigator (0.40d) - C:\Documen	s and Settings\Bryan\Wy Documents\WyVHDL\full_adder\full_adder.xise - [full_adder.vhd]	
File Edit View Project Source Process To	ils Window Layout Help	_ 8 ×
🗠 🖉 🗶 🗇 🖉 🐇 🗳 📢 🔄 🗄	» 🔑 🔑 😥 🖉 🔎 💽 💫 🗄 🚍 🖽 📼 🖙 🕍 🛠 😫 🕨 🖾 🛠 🗄 🖓	
		····· ·
G Behavioral	21 use IEEE.STD LOGIC 1164.ALL;	
	22	
Hierarchy		
Ful_adder	24 arithmetic functions with Signed or Unsigned values 25use IEEE.NUMERIC STD.ALL;	
	27 Uncomment the following library declaration if instantiating	
	28 any Xilinx primitives in this code.	
2	29 library UNISIM;	
	30 use UNISIM.VComponents.all;	_
	51	
12	33 Port (a: in STD LOGIC;	
	34 b : in STD LOGIC;	
	35 cin : in STD_LOGIC;	
No Processes Running	36 sum : out STD_LOGIC;	
	37 cout : out STD_LOGIC);	
Processes: full_adder - gate_level	38 end full_adder; 39	
👷 🖨 🎾 ISim Simulator	40 architecture gate level of full adder is	=
Behavioral Check Syntax	41 signal s1,s2,s3:std logic;	
Simulate Behavioral Model	42 begin	
	43 s1 <= (a xor b) after 5 ns;	
	44 s2 <= (cin and s1) after 3 ns; 45 s3 <= (a and b) after 3 ns;	
	45 s3 <= (a and b) after 3 ns; 46 sum <= (s1 xor cin) after 5 ns;	
	47 cout <= s2 or s3 after 3 ns;	
	48 end gate_level;	
	49	
	50	~
		>
🐖 Start 🔍 Design 🚺 Files 🚺 Libraries 🛃	🛿 ISE Design Suite InfoCenter 🙁 📄 full_adder.vhd 🛛 🗵 Design Summary 🔣	
Console		↔□₽×
D INFO: Project Momt: 659 - Parsing des	ign hierarchy completed successfully.	^
INFO:HDLCompiler:1061 - Parsing VH	DL file "C:/Documents and Settings/Bryan/My Documents/MyVHDL/full_adder/full_a	adder.vhd″ i 🔤
INFO:ProjectMgmt:659 - Parsing des	ign hierarchy completed successfully.	≡
		~
		>
📋 Console 😢 Errors 🛕 Warnings 🕅 Find	in Files Results	
	L.	143 Col 31 VHDL 🚲

Figure 9

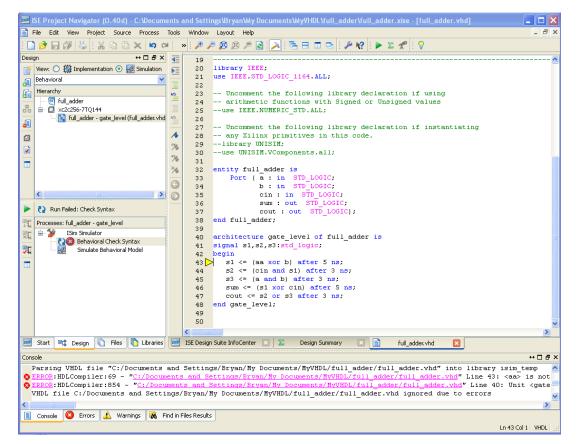


Figure 10

2. Simulating the circuit

Now that we have a design that has been checked, we can proceed to a simulation of the circuit designed. In order to simulate out circuit, we first need a set of signals to drive our inputs *a*, *b* and *cin*. We do this by constructing a separate VHDL source file called a **testbench**. This VHDL code defines the signals and then applies them to the entity under test.

Select from the **Project Navigator** window **Project -> New Source**. The same **New Source Wizard** (Figure 11) appears that was used earlier to enter our *full_adder* source. This time, instead of selecting *VHDL Module*, select *VHDL Test Bench*. Name the file *full_adder_tb* (to distinguish it from the actual full_adder file) and click *Next*. A new window will pop-up asking you which file describes the circuit you want to test. There is only one choice – *full_adder*. Select this file and then click *Next*. The New Source Wizard now shows a summary page. Click *Finish*.

🔤 New Source Wizard	
Select Source Type Select source type, file name and its location.	
Schematic Implementation Constraints File User Document Verilog Module Verilog Test Fixure VHDL Module VHDL Library VHDL Package VHDL Test Bench	File name: [full_adder_tb Location: Ind Settings\Bryan\My Documents\MyVHDL\full_adder
More Info	Next > Cancel

Figure 11

Clicking on *Finish*, the **Project Navigator** screen reappears and you will see that the wizard has already generated some more VHDL code, for a new entity called *full_adder_tb* as shown in Figure 12. There may be some compiler errors that appear in the console window. Ignore them because we are going to edit this testbench file. It is not necessary for you to understand the semantics of this testbench entity at this stage. Simply make the following changes.

- 1. Remove the "constant <clock>_period" line.
- 2. Remove all the code from "--Clock process definitions" to the end of the file and replace with:

```
a_waveform: process is

begin

a<='0', '1' after 20ns;

wait for 40ns;

end process;

b_waveform: process is

begin

b<='0', '1' after 40ns;

wait for 80ns;

end process;

cin_waveform: process is

begin

cin<='0', '1' after 80ns;

wait for 160ns;

end process;
```

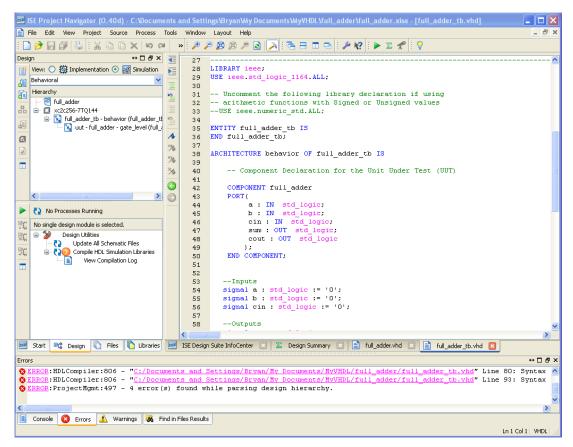


Figure 12.

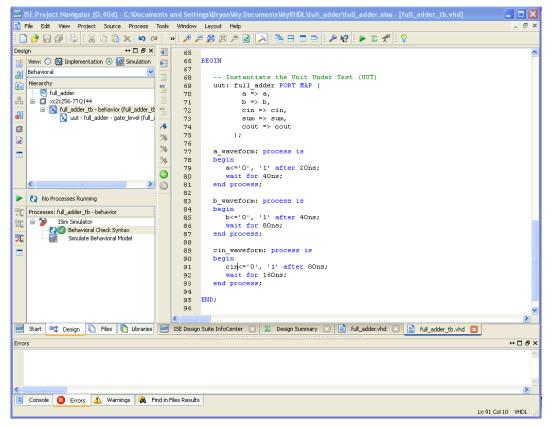


Figure 13.

The code window of the testbench should now appear as shown in Figure 13 (this window is scrolled to show the end of the file). Save the file. Select *full_adder_tb* in the Hierarchy window and then double click on *Behavioral Check Syntax* to check for errors. Correct any errors and then save and check again. Once the code is free of errors, right-click on *Simulate Behavioral Model* and select *Process Properties*. This will generate the **Isim Properties** pop-up window as shown in Figure 14. Set the *Simulation Run Time* to 200 ns and click OK.

Switch Name Property Name Value	
Switch Name Property Name Value	
Use Custom Simulation Command File	
Custom Simulation Command File	
Run for Specified Time	
Simulation Run Time 200 ns	
Waveform Database Filename ettings\Bryan\My Documents\MyVHDL\full_adder\full_adder_tb_isim_be	h.wdb 🛄
Use Custom Waveform Configuration File	
Custom Waveform Configuration File	
Specify Top Level Instance Names work.full_adder_tb	
Load glbl	
Property display level: Standard 🔽 🗹 Display switch names	Default
Property display level: Standard V Display switch hames	
OK Cancel Apply	Help

Figure 14

We are now ready to run the simulation. Double-click on Simulate Behavioral Model. This brings up the **ISim** simulation window as shown in Figure 15. Waveforms are plotted in the top right-hand window. Note that the simulator very conveniently plots the last 3ps of the simulation! In order to see the whole simulation, click the Zoom to Full View icon in the tool-bar immediately above the simulation window. The waveforms will now appear as shown in Figure 16. Check for correct functionality of the full adder. Note the timing delays as outputs respond to inputs according to the delays we specified in the VHDL code.

Experiment with some of the icons on the toolbars of the simulator to see their effect. Then answer the following:

- Note that the simulation does not show the internal signals *s1*, *s2* and *s3*. How can you add these signals to the waveform window (hint: start by right-clicking on *full_adder_tb* in the *Instance and Process Name* window). Once you have added these signals to the waveform window, you can re-run the simulation by clicking *Restart* and then *Run for the Time Specified on the Toolbar* icons on the toolbar immediately above the waveform window.
- 2. Change the timing in the source VHDL so that all gates have a delay of 2ns, and then re-run the simulation. You can do this in one of two ways. Either:
 - a. Exit the simulator (not WebPack just the simulator and you do not need to save the *Default.wcfg* file). Go back and change the source in the original WebPack VHDL file by double-clicking *full_adder* in the Hierarchy window, making the source code changes, reselecting the test-bench *full_adder_tb* and then double-clicking on *Simulate Behavioral Model*. **OR**
 - b. You can edit the VHDL right in the simulator. Go to the pull-down menus in the simulator and select View->Panels->Source Files. A list of the source files will appear in the left hand window of the simulator. Double click on *full_adder.vhd*. The source code will appear in the right hand window. Modify the source and then click on the *Re-launch* icon above the right-hand window. Save the source file when asked. This will save and compile the modified source and re-run the simulation. Click on the

Default.wcfg tab below the right hand window to see the new simulation results. Note that this method does not change the original source back in the WebPack project. So, if you really want to change your design you should use method (a). But this technique (b) can be useful for quick experimentation without having to create a new simulation window each time the code is modified.

3. Reverse the order of the 5 assignment statements in the architecture of the full adder as below:

cout <= s2 or s3 after 3 ns; sum <= (s1 xor cin) after 5 ns; s3 <= (a and b) after 3 ns; s2 <= (cin and s1) after 3 ns; s1 <= (a xor b) after 5 ns;

In what way, if any, does this change the simulation result? Why?

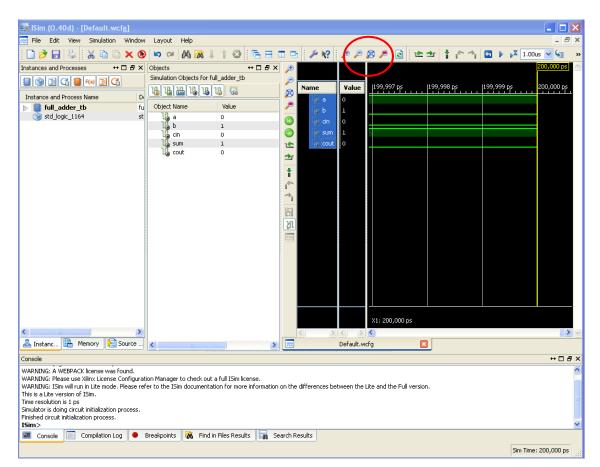


Figure 15

📷 lSim (0.40d)													
👷 File Edit View	Simulation Window	. Layout He	lp										- 8 ×
i 🗋 🎓 🛃 😓 i	X 10 16 × ®) L) (2 (4) 🕅			🗖 🖻 i 🔑	k? : 🏓) ,= <u>1</u> 9 ,=	2	* † r^	⇒ 🖬 🕨	🔉 200ns 💌	€ ≣ »
Instance ↔ □ & ×		⇔⊡₽×	Ð										00 ns
INIC »	Simulation Objects fo	or full_adder_tb											
Instance and Process		6 16 🔛	8	Name	¥alue	0 ns		50 ns		100 ns	150) ns	·
Instance and Process in full_adder_tt	Object Name	Value	۶.	Цеа Цер	0								
💿 std_logic_1164	la a La b	0		цер Цесіn	1 0								
	la b La cin	1	5	Ue sum	1								
	la sum	1	<u>*</u>		0								
	1 cout	0	-										
			+										
			÷										
			1 1										
			Я										
	2												
						X1: 200.000 n	s						
<				<	< >	<							>
🟯 Instanc 📴 🕨		>	222	Default.w	cfg*	🗵 📄 fi	ull_adder_tb	.vhd 🗵	📄 full <u></u>	_adder.vhd	×		
Console												+	• 🗆 🗗 🗙
Finished circuit initializati	on process.												^
ISim> # restart													
ISim>													
# run 200ns Simulator is doing circuit	initialization process.												
Finished circuit initializati ISim>	on process.												~
	ompilation Log 🛛 😐	Breakpoints 👔	🕺 Fin	ıd in Files Resul	ts 📊	Search Results							
												Sim Time: 200,0	00 os
					_								55 ps [.;

Figure 16