appendix

C

Tutorial 2—Implementing Circuits in Altera Devices

In this tutorial we describe how to use the physical design tools in Quartus II. In addition to the modules used in Tutorial 1, the following Quartus II modules are introduced: Fitter, Floorplan Editor, and Timing Analyzer. To illustrate the procedures involved, we will first implement the *example_verilog* project created in Tutorial 1 in a Cyclone II FPGA.

C.1 Implementing a Circuit in a Cyclone II FPGA

Select File > Open Project and browse to the directory *designstyle*2, which contains the Verilog design example used in Tutorial 1. As depicted in Figure C.1, select the *example_verilog* project (Quartus II project files have the filename extension *.qpf*) and click Open.

C.1.1 Selecting a Chip

In Tutorial 1 we used the Compiler to perform the synthesis operations, which generated the information needed for functional simulation. Now, we will implement the design in an FPGA and then use timing simulation.

To specify which chip to use, select Assignments > Device to open the window shown in Figure C.2. Click on the pull-down menu in the box labeled Family and select Cyclone II. Note that in some cases Quartus II will display the message "Device family selection has changed. Do you want to remove all pin assignments?" Click Yes to close this pop-up box.

In the Target device box you can specify that Quartus II should automatically select a device during compilation. The ability to have a chip chosen automatically is sometimes convenient for the designer. However, in this case we wish to select a specific chip, so click on Specific device selected in 'Available devices' list.

The various chips in the Cyclone II family are displayed in the box labeled Available devices. One available chip is the EP2C35F672C6 (if this device is not listed, change the Speed Grade item in the Filter box to Any). The meaning of the chip name is as follows: The EP2C means that the chip is a member of the Cyclone II family, and the 35 gives

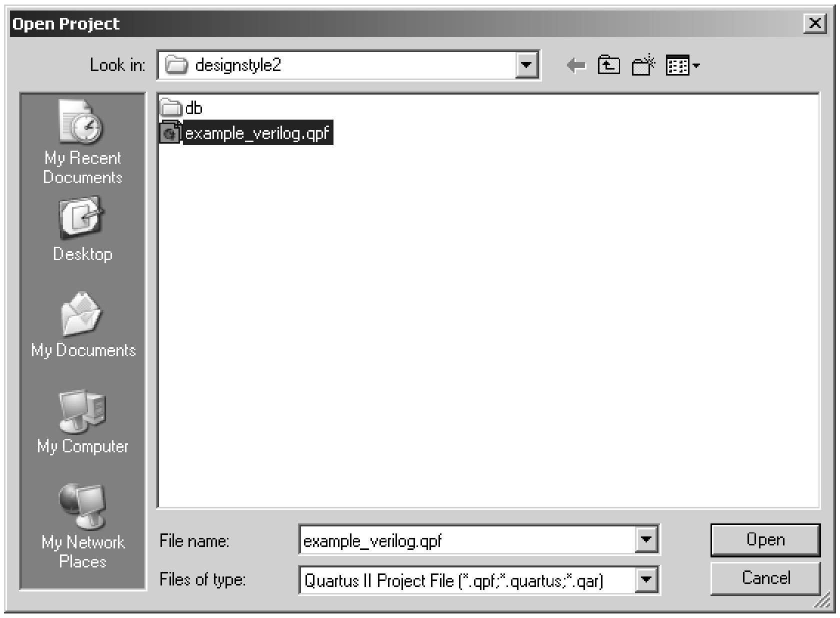


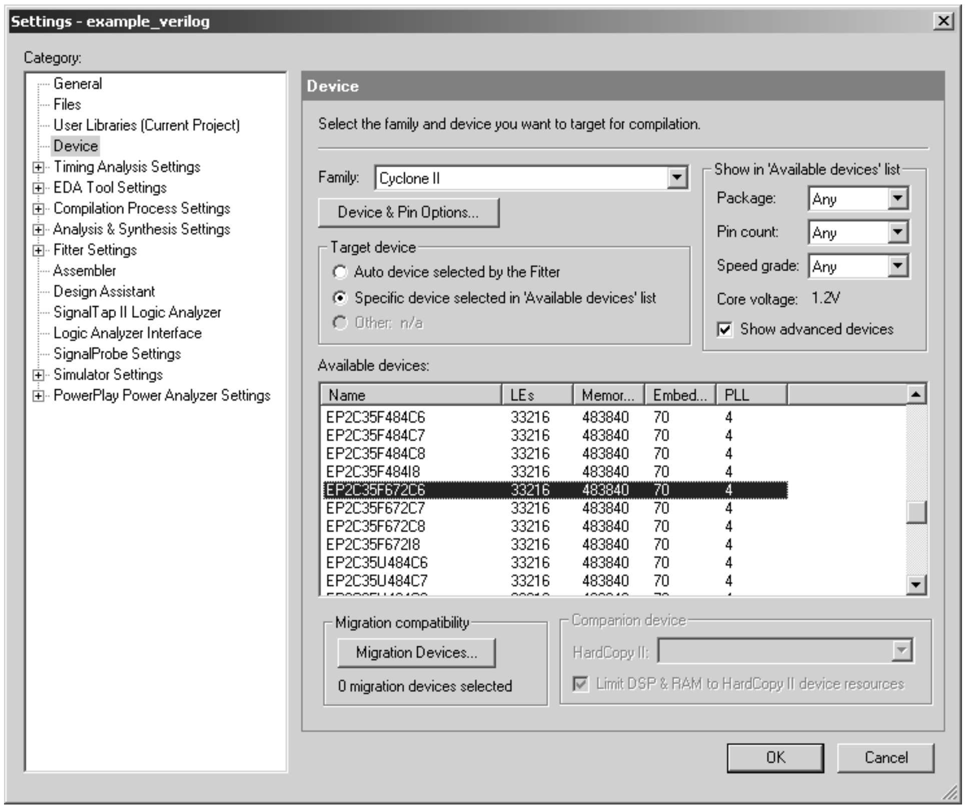
Figure C.1 Opening the *example_verilog* project.

an indication of the number of logic elements in the chip. The designator F672 indicates a Fineline 672-pin ball grid array package; we describe package types in section 3.6.3. The C6 gives the *speed grade*. We discuss speed grades in Appendix E. As indicated in Figure C.2, choose the EP2C35F672C6 device, and then click OK to close the Settings window. We have chosen this chip because it is provided on an Altera development board that is discussed in Appendix D.

C.1.2 COMPILING THE PROJECT

In Appendix B we ran just the synthesis tools in Quartus II, by using the command Processing > Start > Start Analysis & Synthesis. Now, we wish to run in sequence the four modules in the Quartus II software that we showed in Figure B.16: Synthesis, Fitter, Assembler, and Timing Analyzer. Before invoking these tools, open the menu under Tools > Options and then in the category General > Processing click to select Automatically generate equation files during compilation. This setting causes the Quartus II Compiler to record in its Report File the logic expressions generated during the compilation process.

To invoke the tools, select Processing > Start Compilation, or use the toolbar icon that looks like a solid purple triangle. As we saw in Tutorial 1, the compilation progress through each Quartus II module is displayed in the Status window on the left side of the Quartus II display. After the Analysis & Synthesis module converts the Verilog code into a circuit that comprises Cyclone II logic elements, the Fitter module chooses locations on the


Settings - example_verilog

Category:
- General
- Files
- User Libraries (Current Project)
- Device
- Timing Analysis Settings
- EDA Tool Settings
- Compilation Process Settings
- Analysis & Synthesis Settings
- Fitter Settings
- Assembler
- Design Assistant
- SignalTap II Logic Analyzer
- Logic Analyzer Interface
- SignalProbe Settings
- Simulator Settings
- PowerPlay Power Analyzer Settings

Device

Select the family and device you want to target for compilation.

Family: Cyclone II

Device & Pin Options...

Target device
- Auto device selected by the Fitter
- [✓ Specific device selected in 'Available devices' list]
- Other: n/a

Show in 'Available devices' list
- Package: Any
- Pin count: Any
- Speed grade: Any
- Core voltage: 1.2V
- [✓] Show advanced devices

Available devices:

Name	LEs	Memor...	Embed...	PLL
EP2C35F484C6	33216	483840	70	4
EP2C35F484C7	33216	483840	70	4
EP2C35F484C8	33216	483840	70	4
EP2C35F484I8	33216	483840	70	4
EP2C35F672C6	33216	483840	70	4
EP2C35F672C7	33216	483840	70	4
EP2C35F672C8	33216	483840	70	4
EP2C35F672I8	33216	483840	70	4
EP2C35U484C6	33216	483840	70	4
EP2C35U484C7	33216	483840	70	4

Migration compatibility
Migration Devices...
0 migration devices selected

Companion device
HardCopy II:
[✓] Limit DSP & RAM to HardCopy II device resources

OK Cancel

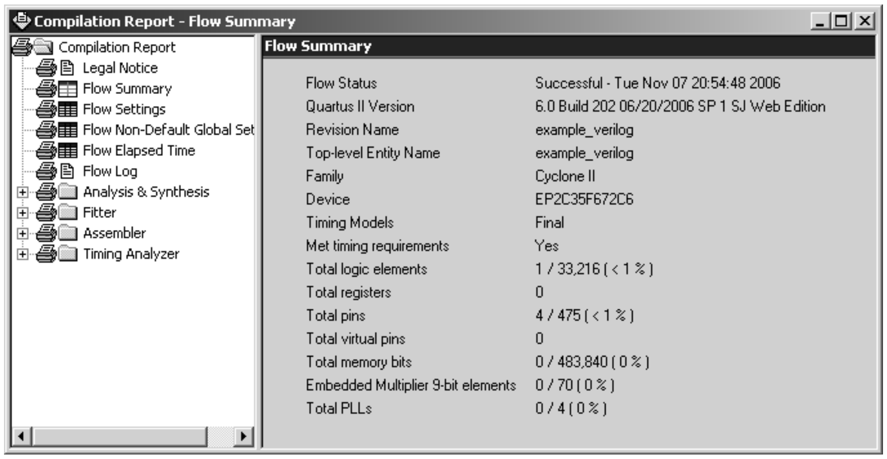
Figure C.2 Selecting a Cyclone II device.

FPGA chip for these logic elements. A detailed discussion of the CAD modules is provided in Chapter 12.

When compilation is finished, the compilation report displayed in Figure C.3 is produced. Click on the small + symbol to expand the Fitter section of the report, and then click on the Equations section to reach the display in Figure C.4. Scroll through this part of the report to see the logic expressions implemented by our circuit. At the bottom of the report the output f is given as

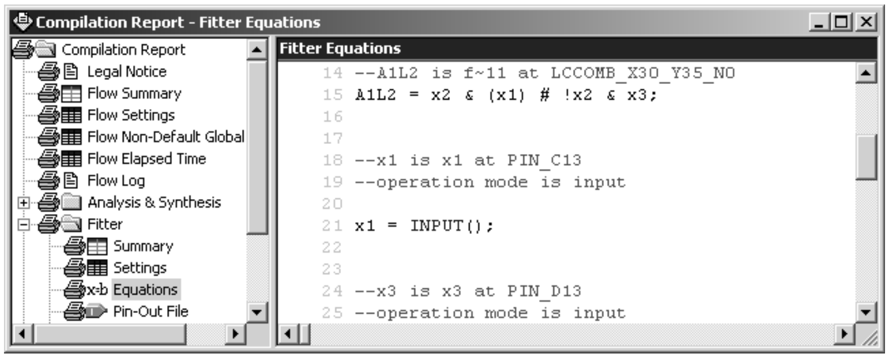
$$\text{f} = \text{OUTPUT(A1L2);}$$

This means that f appears on an output pin, and that output is defined by the logic expression called A1L2, which is realized as indicated near the top of the Fitter Equations section in Figure C.4. This expression properly implements our logic function $f = x_1x_2 + \overline{x}_2x_3$. Note that the # symbol is used by Quartus II to denote the OR operator.



```
Flow Status                          Successful - Tue Nov 07 20:54:48 2006
Quartus II Version                   6.0 Build 202 06/20/2006 SP 1 SJ Web Edition
Revision Name                        example_verilog
Top-level Entity Name                example_verilog
Family                               Cyclone II
Device                               EP2C35F672C6
Timing Models                        Final
Met timing requirements              Yes
Total logic elements                 1 / 33,216 ( < 1 % )
Total registers                      0
Total pins                           4 / 475 ( < 1 % )
Total virtual pins                   0
Total memory bits                    0 / 483,840 ( 0 % )
Embedded Multiplier 9-bit elements   0 / 70 ( 0 % )
Total PLLs                           0 / 4 ( 0 % )
```

Figure C.3 The compilation summary.



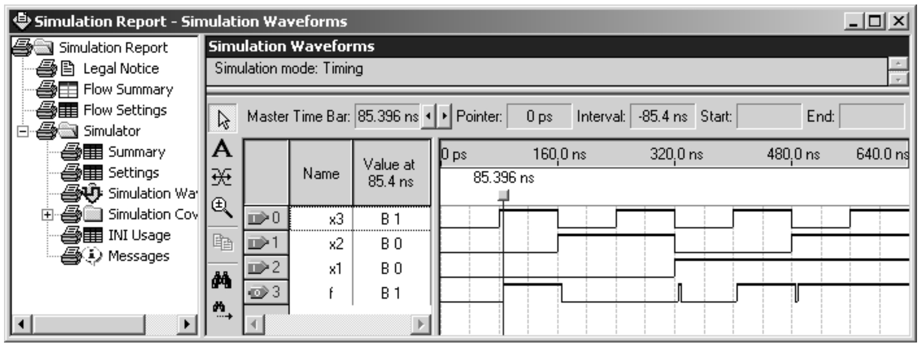
```
--A1L2 is f~11 at LCCOMB_X30_Y35_N0
A1L2 = x2 & (x1) # !x2 & x3;


--x1 is x1 at PIN_C13
--operation mode is input

x1 = INPUT();


--x3 is x3 at PIN_D13
--operation mode is input
```

Figure C.4 The Fitter Equations section.

C.1.3 PERFORMING TIMING SIMULATION

Timing simulation is done by using the same procedure that we described in Tutorial 1 for functional simulation. Select Assignments > Settings and click on the Simulator item, as shown in Figure B.24. Open the drop-down list next to Simulation mode and change this setting from Functional to Timing. Use the Edit > End Time command to set the duration of the simulation to 640 ns. Then, turn on grid lines at 40-ns intervals by selecting Edit > Grid Size and setting the Time period to 40 ns.

Use similar input waveforms for x_1, x_2, and x_3 that were drawn with the Waveform Editor in Tutorial 1 as inputs for the timing simulation. Select Processing > Start Simulation to run the simulation. When it is completed, the simulation report is displayed. Part of this



Simulation mode: Timing

Master Time Bar: 85.396 ns Pointer: 0 ps Interval: -85.4 ns Start: End:

	Name	Value at 85.4 ns
0	x3	B 1
1	x2	B 0
2	x1	B 0
3	f	B 1

Figure C.5 The Timing Simulation Report.

report is shown in Figure C.5. Select View > Fit in Window to see the complete time range of the waveforms. Compare these waveforms to those shown in Figure B.25. The timing simulation produces the same results as the functional simulation in Tutorial 1 except that the changes in the waveform for f are now determined by the timing characteristics of the Cyclone II 2C35 chip. There are two changes in the waveform for f shown in Figure C.5 that we should mention. At the 320 ns point in the simulation, the inputs $x_1x_2x_3$ change from 011 to 100. Since $f = 0$ for both of these input combinations, we would expect to see no change in the output value produced by the simulation. The waveform in Figure C.5 shows that f does have the correct value (0) after the inputs change to 100, but there is a short period of time when a wrong value of $f = 1$ is produced. This temporary change in the output value, which is usually called a *glitch*, is due to the delay properties of the lookup table based logic element in the Cyclone II FPGA. We discuss lookup table based logic cells in section 3.6.5. A similar glitch occurs at the 480 ns point in the simulation shown in Figure C.5. In practice, glitches like these do not cause a problem, because they only exist for a short time before the output stabilizes at the correct value. We discuss this topic in more detail in Chapter 9.

We can use the vertical reference line in the Simulation Report window to determine the exact time when f changes value. To do this select View > Snap to Transition, so that your mouse pointer will align perfectly with an edge on any waveform. Click and drag the vertical reference line to the point where f first changes to 1, as shown in the figure (you can also move the reference line by using the keyboard arrow keys). The box labeled Master Time Bar now displays 85.396 ns, meaning that it takes about 5.4 ns for the change in $x3$, which occurs at 80 ns, to cause a change in f. This result is a reflection of the timing characteristics of the Cyclone II FPGA.

C.1.4 Using the Chip Planner

In addition to examining the equations in the compilation report, another way to view the implementation results is to use the Chip Planner. Select Tools > Chip Planner to open the

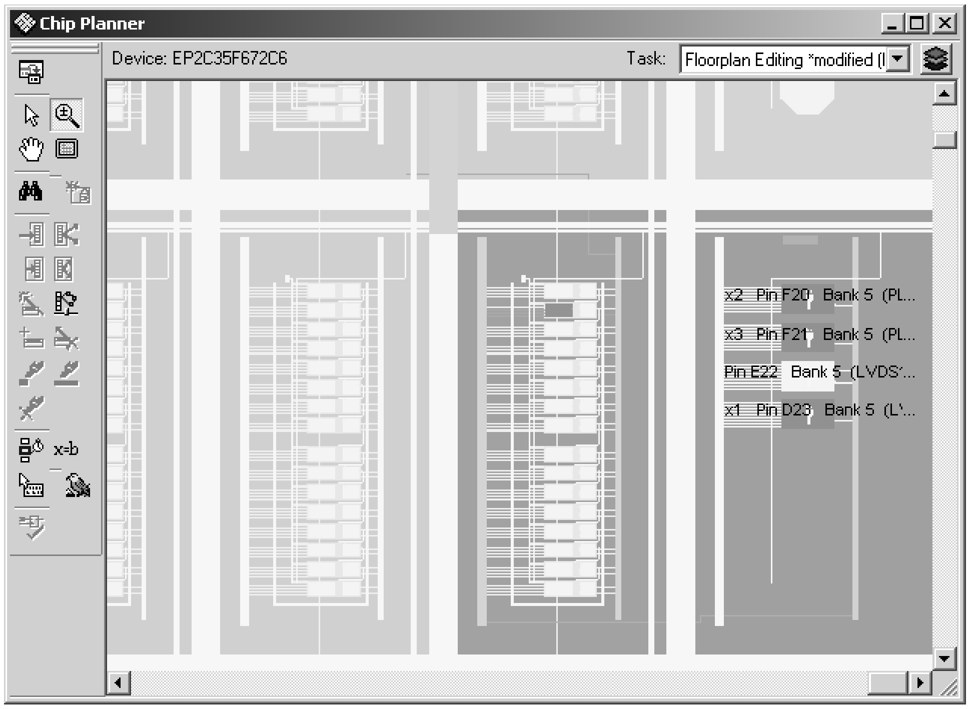
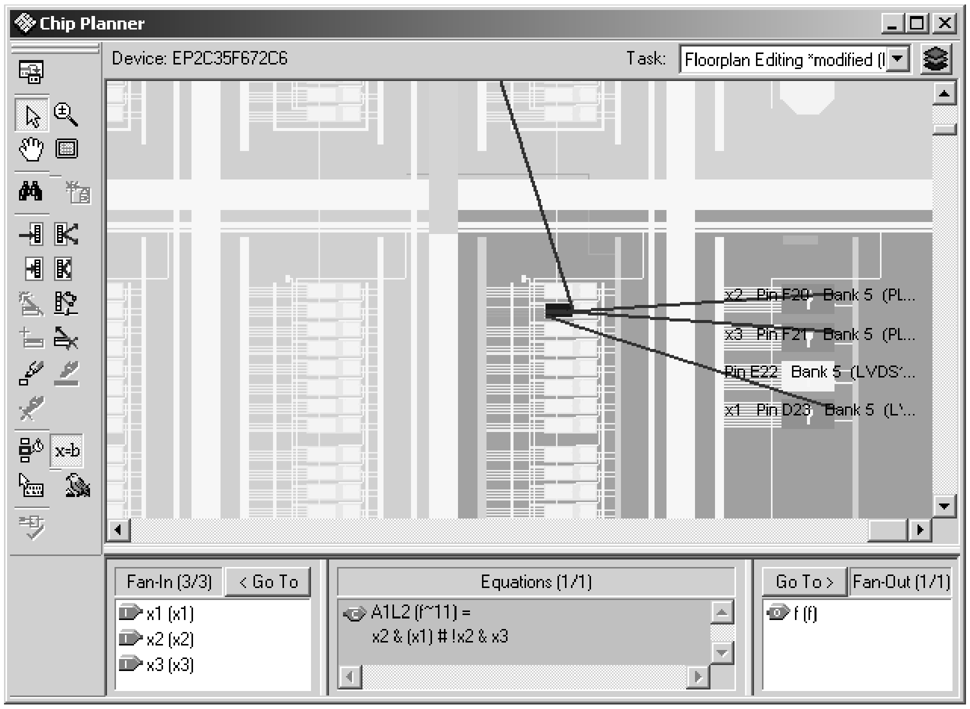


Figure C.6 The Chip Planner display.

window shown in Figure C.6. To make the window look like the one in the figure, it may be necessary to turn off the feature that displays equations in the bottom part of the Chip Planner window. Select View > Equations to toggle off this feature.

Figure C.6 shows some of the logic elements in the Cyclone II 2C35 chip. As shown in Appendix E, each logic element comprises a four-input lookup table. The logic elements are organized into logic array blocks (LABs), where each LAB contains 16 logic elements. Selecting View > Fit in Window in the Chip Planner will display the entire chip. The Chip Planner uses different colors to indicate logic elements and pins that are used in a circuit and those that are unused. For our small example four pins are used for the three inputs and one output, and one logic element (of more than 33,000 in the chip!) is used to implement the function f. To see larger or smaller views of the chip, click on the Zoom Tool button in the Chip Planner toolbar, which looks like a small magnifying glass. Left-click to zoom in and right-click to zoom out. To display different sections of the chip, use the window scroll bars.

Adjust the display so that the logic cell that produces the output f is visible, as depicted in Figure C.7 (your compilation results may use a different logic element and pins from the ones shown in the figure). Make sure the Selection Tool, which looks like an arrowhead, in the Chip Planner is active, and then click on the logic element for f to select it. The Chip Planner can draw lines that indicate which other resources the selected logic element



```
A1L2 (f~11) =
  x2 & (x1) # !x2 & x3
```

Figure C.7 Viewing node fan-in and equations.

is connected to by choosing View > Generate Fan-In Connections and View > Generate Fan-Out Connections. It is also possible to see what logic function is implemented in the selected node by selecting View > Equations. As seen in the figure, this choice displays the logic expressions from the compilation report in the bottom part of the Chip Planner window.

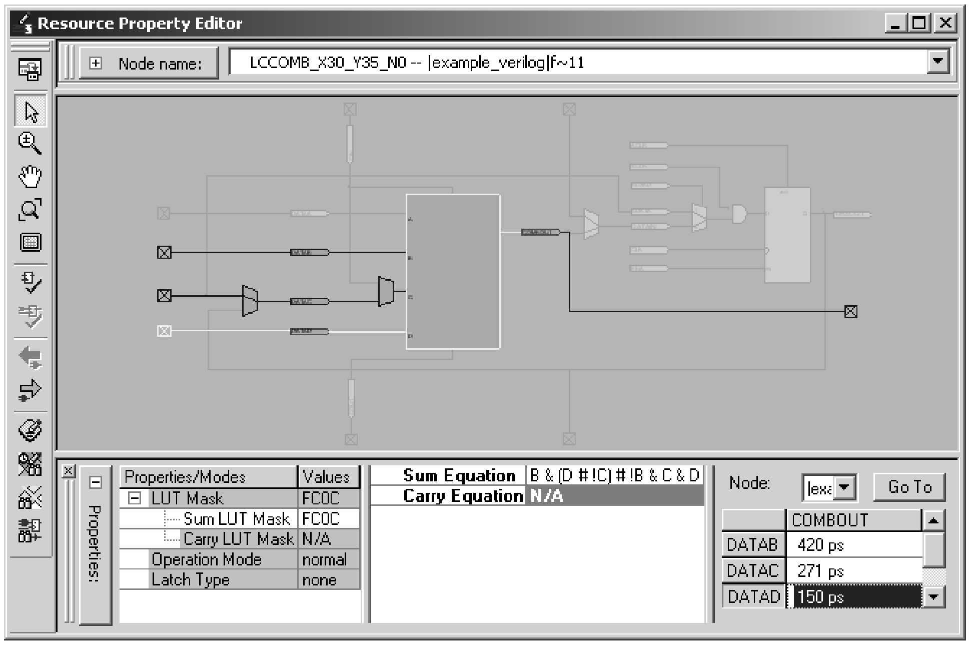
Instead of displaying the whole chip, it is also possible to see more details for individual resources. Right-click on the logic element for f and select Locate > Locate in Resource Property Editor to open the Resource Property Editor tool shown in Figure C.8. Another way to open this tool is to double-click the mouse on the logic element. To make the display look as shown in the figure it may be necessary to select View > View Port Connections to toggle off this feature.

A lot of useful information is available in the Resource Property Editor. It shows that the lookup table inputs called B, C, and D are used for our logic function. Hover the mouse cursor over each of the inputs in turn to see that B is connected to x_3, C to x_2, and D to x_1. The window shows the logic function implemented in the lookup table under the name *Sum Equation*; this terminology is used because it is possible to configure a lookup table such that it implements separate outputs for the sum and carry functions needed in a full-adder (since our logic function is not a full-adder, the expression for Carry Equation is shown as N/A). We should note that the logic expression shown for f in the figure is specified as

$f = x_3(x_1 + \overline{x}_2) + \overline{x}_3 x_2 x_1$. This is not the simplest expression that one may expect, namely $f = x_1 x_2 + \overline{x}_2 x_3$. But both expressions represent the same function and the CAD tools do not always display the simplest form of an equation.

The bottom right corner of the Resource Property Editor window shows the propagation delays through the logic element. Click on the value 150 ps associated with input D, which is connected to x_1. As indicated in Figure C.8 the corresponding path through the logic element is highlighted. The path starting at input C has a delay of 271 ps, and the path through B has the delay 420 ps. The differing delays associated with each input to the lookup table is the reason that we observed glitches in the simulation waveforms for f in Figure C.5; changes in the input x_1 affect the value of the output f more quickly than changes in inputs x_2 and x_3. In larger designs where it is important to optimize the performance of the implemented circuit, the CAD tools make use of the faster inputs through lookup tables for the parts of a circuit that are the most timing critical.

It is possible to explore different parts of the implemented circuit using the Resource Property Editor. To experiment with this feature, right-click on the DATAD input to the lookup table and select Go to source node, as indicated in Figure C.9. This action causes the Resource Property Editor to display the pin used for the input x_1. The Back to previous resource icon in the Resource Property Editor toolbar, which looks like an arrow pointing to the left, can then be used to return the display to the logic element previously viewed. In a similar way, you can right-click on the output of the lookup table and examine the pin used for the output f.



LCCOMB_X30_Y35_N0 -- |example_verilog|f~11

Properties/Modes	Values
LUT Mask	FC0C
Sum LUT Mask	FC0C
Carry LUT Mask	N/A
Operation Mode	normal
Latch Type	none

Sum Equation	B & (D # !C) # !B & C & D
Carry Equation	N/A

	COMBOUT
DATAB	420 ps
DATAC	271 ps
DATAD	150 ps

Figure C.8 The Resource Property Editor display.

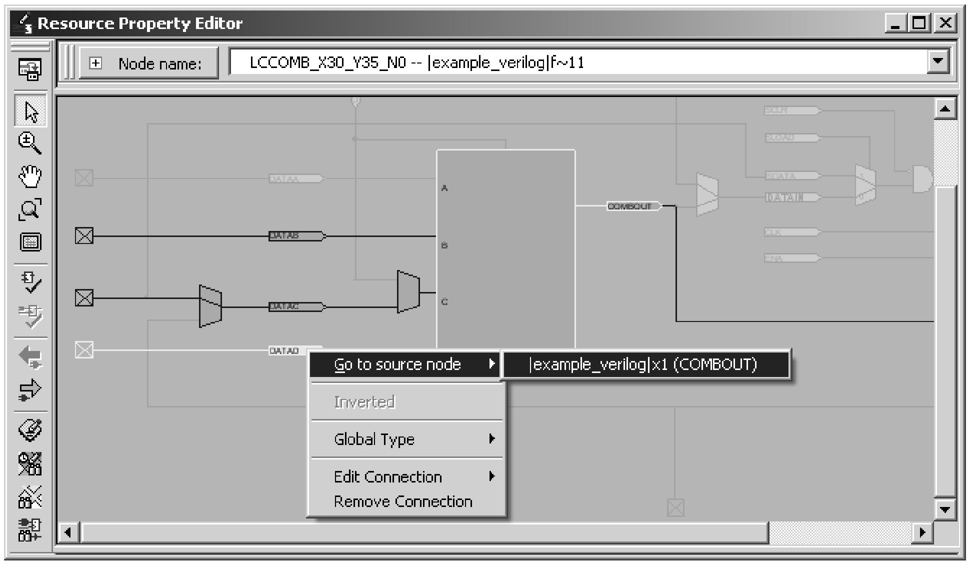


Figure C.9 Using the Resource Property Editor.

We have now completed the implementation of the *example_verilog* project in a Cyclone II FPGA. Close the project.

C.2 Implementing an Adder using Quartus II

In section 5.5 we show how an *n*-bit ripple-carry adder can be specified in Verilog code. In this section we show how the ripple-carry adder can be implemented using the Quartus II system. Create a new project, *adder16*, in a directory *tutorial2\addern*. We will implement the adder circuit in a Cyclone II FPGA. Thus, in the New Project Wizard window shown in Figure B.6, select the Cyclone II family and choose the specific device called the EP2C35F672C6 (if this device is not listed, change the Speed Grade item in the Filter box to Any). We are using this device because it is available in the DE2 Development and Education board provided by Altera (see altera.com), which we discuss in Appendix D.

C.2.1 The Ripple-Carry Adder Code

Verilog code for the *n*-bit adder is given in Figure C.10. It takes as inputs the carry-in signal, *carryin*, plus two *n*-bit numbers, *X* and *Y*, and produces the *n*-bit output sum, *S*, and carry-out signal, *carryout*. The code uses the parameter *n*, so that the adder can be parameterized to work for any number of bits. In this example, *n* is set to 16. In the code the vector *C* is used to represent the intermediate carries between the stages in the adder. A for loop is used to create *n* full-adders that comprise the ripple-carry adder.

```
module adder16 (carryin, X, Y, S, carryout);
   parameter n = 16;
   input carryin;
   input [n−1:0] X, Y;
   output reg [n−1:0] S;
   output reg carryout;
   reg [n:0] C;
   integer k;

   always @(X or Y or carryin)
   begin
      C[0] = carryin;
      for (k = 0; k <= n−1; k = k+1)
      begin
         S[k] = X[k] ^ Y[k] ^ C[k];
         C[k+1] = (X[k] & Y[k]) | (C[k] & X[k]) | (C[k] & Y[k]);
      end
      carryout = C[n];
   end

endmodule
```

Figure C.10 Verilog code for a ripple-carry adder.

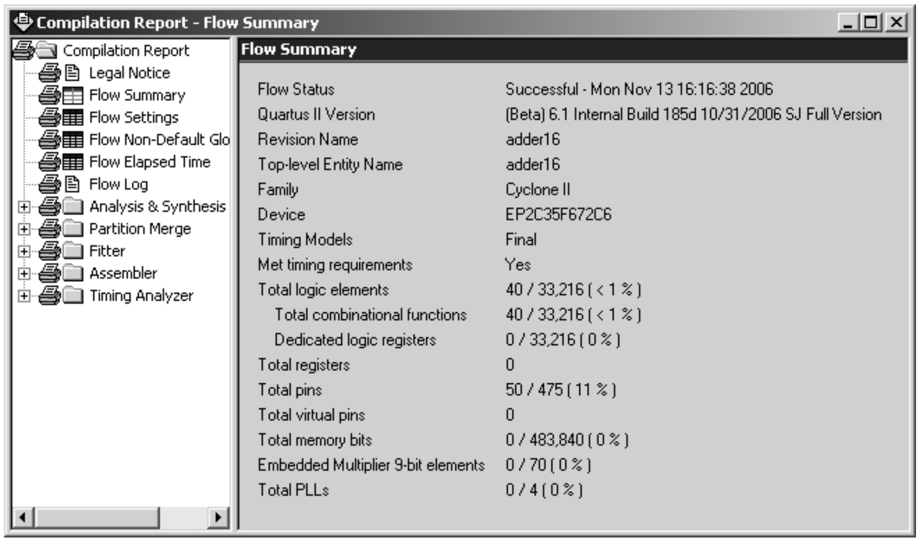
Type the code in Figure C.10 into the Text Editor, as explained in Section B.4.2, and save the file in the *tutorial2\addern* directory using the name *adder16.v*. Compile the circuit. The compilation report is shown in Figure C.11.

C.2.2 SIMULATING THE CIRCUIT

To test the correctness of the circuit, we will perform timing simulation. For brevity only a few test vectors will be used, but in a real design situation more extensive testing would be required.

Create a new Vector Waveform file. Use Edit > End Time to set the desired simulation to run from 0 to 250 ns. Choose the grid lines to be placed at 25-ns intervals. This is done by selecting Edit > Grid Size, which leads to the window in Figure C.12 and setting the Time period to 25 ns. Select View > Fit in Window to display the entire simulation range in the window.

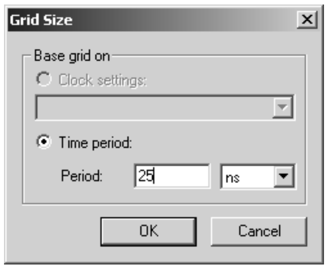
Select Edit > Insert > Insert Node or Bus, and then open the Node Finder utility to reach the window in Figure C.13. Set the filter to Pins: all and click List, which displays the input and output nodes as depicted in the figure. Scroll down the list of displayed nodes until you reach *carryin*. Select this node by clicking on it and then clicking the > sign. Next select the *X* input. Note that this input can be selected either as nodes that correspond



Flow Summary

Flow Status	Successful - Mon Nov 13 16:16:38 2006
Quartus II Version	(Beta) 6.1 Internal Build 185d 10/31/2006 SJ Full Version
Revision Name	adder16
Top-level Entity Name	adder16
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	40 / 33,216 (< 1 %)
Total combinational functions	40 / 33,216 (< 1 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	50 / 475 (11 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure C.11 The compilation report summary.

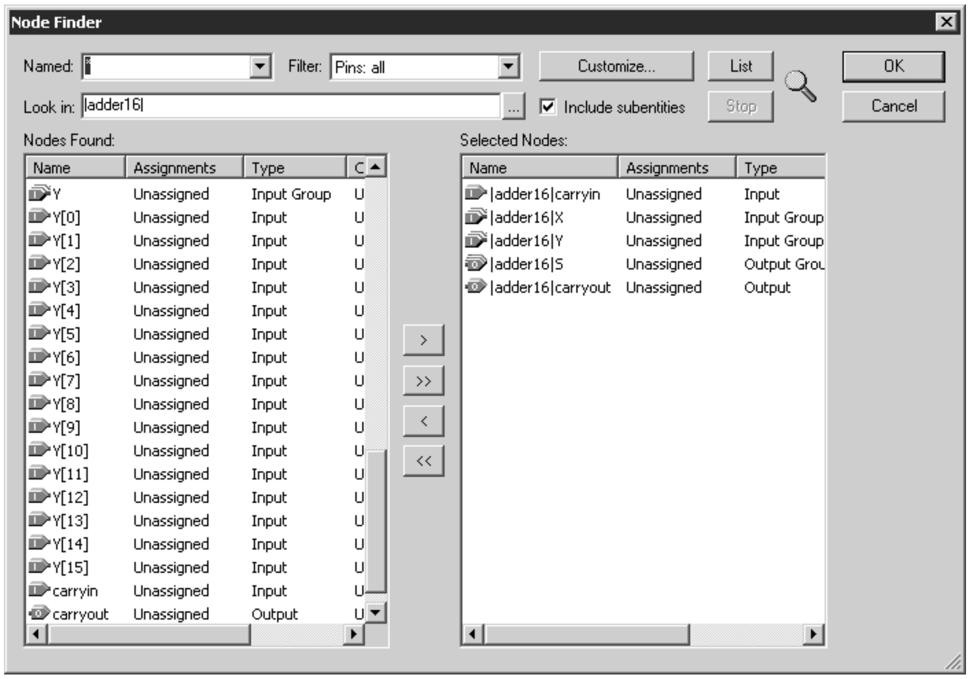


Time period: [✓]
Period: 25 ns

Figure C.12 Setting the spacing of grid lines.

to the individual bits (denoted by bracketed subscripts) or as a 16-bit vector, which is a more convenient form. Then, select the input *Y* and outputs *S* and *carryout*. This produces the image in the figure. Click OK.

The Waveform Editor window now looks like the image in Figure C.14. Vectors *X*, *Y*, and *S* are initially treated as binary numbers. They can also be treated as either octal, hexadecimal, signed decimal, or unsigned decimal numbers. For our purpose it is convenient to treat them as hexadecimal numbers, so right-click on *X* in the Name column and select Properties in the pop-up box to get to the window displayed in Figure C.15. Choose hexadecimal as the radix, make sure that the bus width is 16 bits, and click OK. (Quartus II uses the term *bus* to refer to multibit nodes.) In the same manner, declare that *Y*



Node Finder

Named: * Filter: Pins: all

Look in: |adder16| ☑ Include subentities

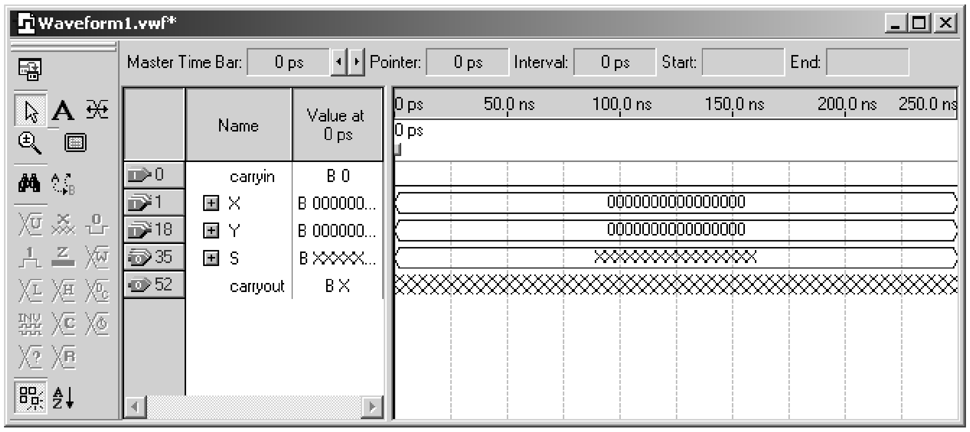
Nodes Found:

Name	Assignments	Type	C
Y	Unassigned	Input Group	U
Y[0]	Unassigned	Input	U
Y[1]	Unassigned	Input	U
Y[2]	Unassigned	Input	U
Y[3]	Unassigned	Input	U
Y[4]	Unassigned	Input	U
Y[5]	Unassigned	Input	U
Y[6]	Unassigned	Input	U
Y[7]	Unassigned	Input	U
Y[8]	Unassigned	Input	U
Y[9]	Unassigned	Input	U
Y[10]	Unassigned	Input	U
Y[11]	Unassigned	Input	U
Y[12]	Unassigned	Input	U
Y[13]	Unassigned	Input	U
Y[14]	Unassigned	Input	U
Y[15]	Unassigned	Input	U
carryin	Unassigned	Input	U
carryout	Unassigned	Output	U

Selected Nodes:

Name	Assignments	Type
|adder16|carryin	Unassigned	Input
|adder16|X	Unassigned	Input Group
|adder16|Y	Unassigned	Input Group
|adder16|S	Unassigned	Output Grou
|adder16|carryout	Unassigned	Output

Figure C.13 The Node Finder window.

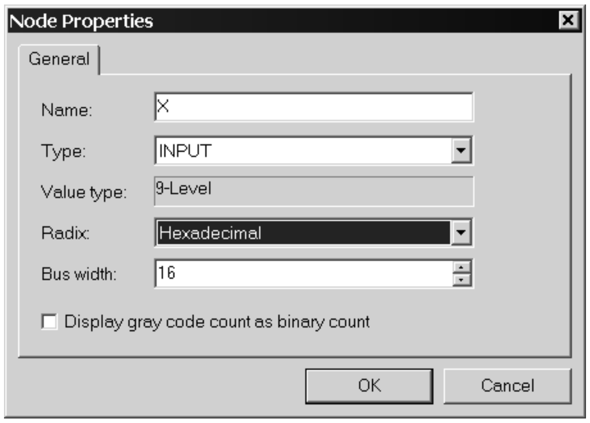


Waveform1.vwf*

Master Time Bar: 0 ps Pointer: 0 ps Interval: 0 ps Start: End:

	Name	Value at 0 ps
0	carryin	B 0
1	⊞ X	B 000000...
18	⊞ Y	B 000000...
35	⊞ S	B XXXX...
52	carryout	B X

Figure C.14 Selected input and output nodes.


Node Properties

General

Name: X

Type: INPUT

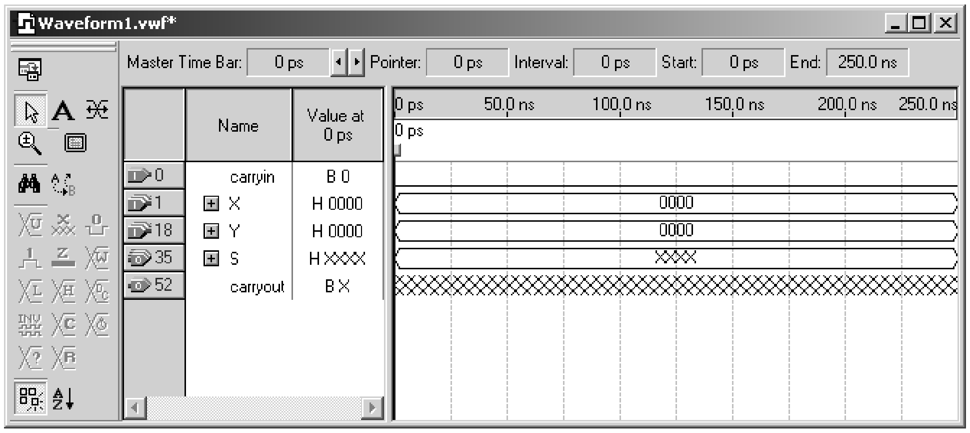
Value type: 9-Level

Radix: Hexadecimal

Bus width: 16

☐ Display gray code count as binary count

Figure C.15 Defining the characteristics of a node.


Waveform1.vwf*

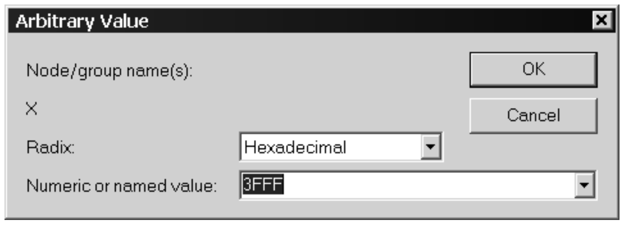
Master Time Bar: 0 ps Pointer: 0 ps Interval: 0 ps Start: 0 ps End: 250.0 ns

	Name	Value at 0 ps
0	carryin	B 0
1	X	H 0000
18	Y	H 0000
35	S	H XXXX
52	carryout	B X

Figure C.16 Using the hexadecimal representation for multibit signals.

and S should be treated as hexadecimal numbers. The resulting waveform display is shown in Figure C.16.

We will now set the test values of X and Y. The default value of these inputs is 0. To assign specific values in various intervals proceed as follows. Select (highlight) the interval from 100 to 175 ns of input X. Press the Arbitrary Value icon in the toolbar (it is labeled by a question mark), to bring up the pop-up window in Figure C.17. Enter the value 3FFF and click OK. Then, set X to the value 7FFF in the interval from 175 to 250 ns. Set Y to 0001 in the interval from 50 to 250 ns. Thus, the input waveforms should be as depicted in Figure C.18. If this were a real design project we would enter additional test values into

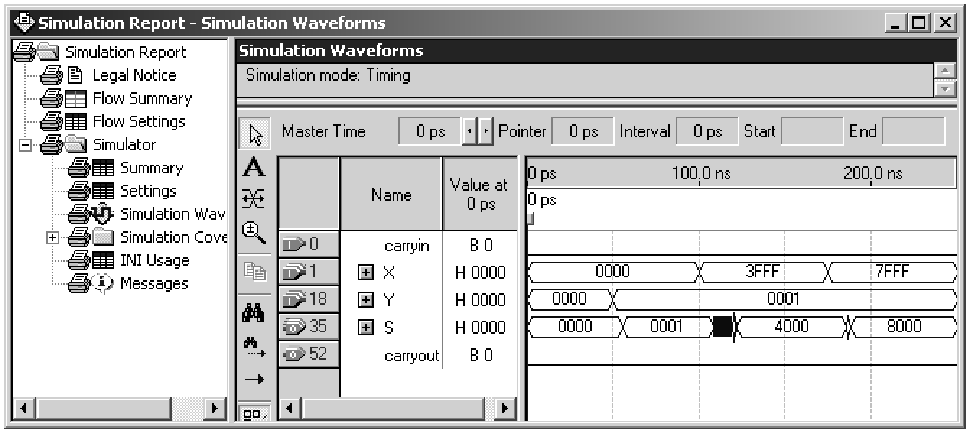


Node/group name(s): X

Radix: Hexadecimal

Numeric or named value: 3FFF

Figure C.17 Assigning the value of a multibit signal.



Simulation mode: Timing

Master Time 0 ps Pointer 0 ps Interval 0 ps Start End

	Name	Value at 0 ps
0	carryin	B 0
1	X	H 0000
18	Y	H 0000
35	S	H 0000
52	carryout	B 0

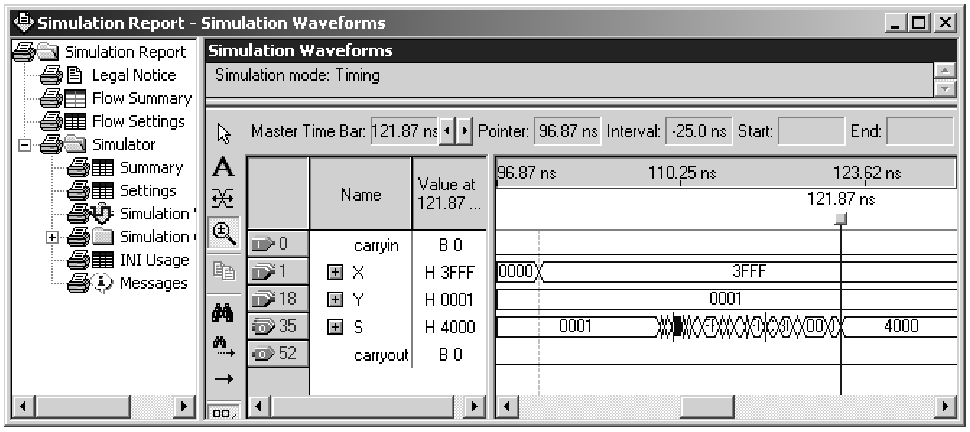
Figure C.18 The result of timing simulation.

the waveforms, but for purposes of this tutorial a few test vectors will suffice. Save the file as *adder16.vwf*.

C.2.3 Timing Simulation

To examine the functionality of the circuit, and determine its speed of operation in the chosen device, we will perform a timing simulation. Select Assignments > Settings > Simulator to reach the window in Figure B.25 and choose Timing as the simulation mode. Run the simulator. The result is given in Figure C.18. It shows considerable delays in producing the correct value S = 4000 because the carries are rippling through the adder stages.

Point to the small square handle at the top of the reference line and drag it to the point where the S value becomes 4000. A more accurate view can be obtained if the waveform image is enlarged using the Zoom Tool. Enlarge the image to look like the display in



Simulation mode: Timing

Master Time Bar: 121.87 ns Pointer: 96.87 ns Interval: -25.0 ns Start: End:

	Name	Value at 121.87 ...
0	carryin	B 0
1	X	H 3FFF
18	Y	H 0001
35	S	H 4000
52	carryout	B 0

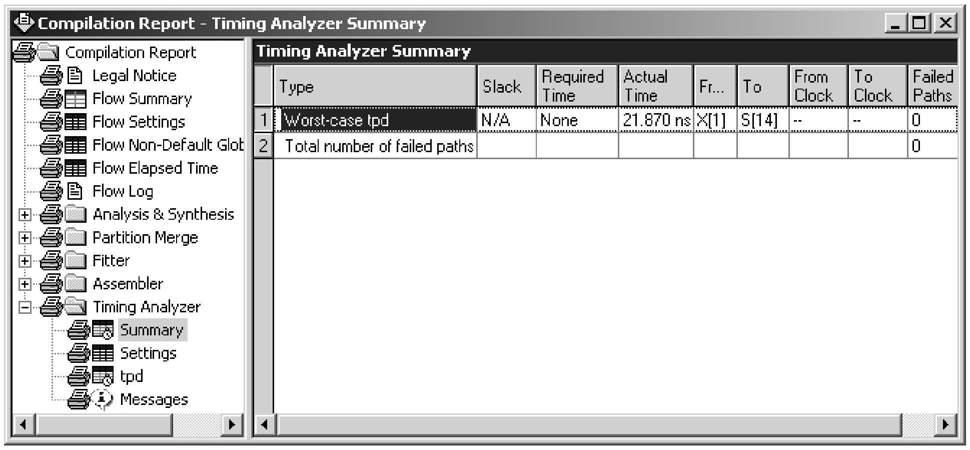
Figure C.19 Detailed results of timing simulation.

Figure C.19. Click on the Selection Tool icon, and drag the reference line as closely as possible to the point where the value 4000 becomes valid.

The change in S from 0001 to 4000 is caused by the X input changing from 0000 to 3FFF, which occurs at 100 ns. As seen in Figure C.19, the output S changes to 4000 at approximately 121.9 ns. Therefore, the propagation delay through the adder, for these particular values of inputs, is estimated to be 21.9 ns. Note that, in this case, the adder performs the operation 3FFF + 1 = 4000 which involves a carry rippling through most of the stages of the adder circuit. For other values of inputs, the propagation delay may be much smaller. In Figure C.18, we see that the operation 0000 + 0001 = 0001 is completed in about 5.76 ns.

When we compile our circuit using Processing > Start Compilation one of the modules executed is the Timing Analyzer. As explained in Chapter 12, this module automatically produces an estimate of the speed of the circuit. Open the compilation report by selecting Processing > Compilation Report or by clicking on its icon. The report includes the derived timing analysis. Click on the small + symbol next to Timing Analyzer to expand this section of the report. Then, click on Summary to get the display in Figure C.20. The summary indicates that the estimated worst case propagation delay from an input to output pin, t_{pd}, is 21.9 ns. This longest path starts at the $X[1]$ input and ends at $S[14]$. More detailed information about the propagation delays along various paths through the circuit can be seen by clicking on tpd on the left side of Figure C.20, which displays the information in Figure C.21. Here, we see that there are several paths along which the propagation delay is close to the maximum, including the one given in the summary in Figure C.20. These longest-delay paths are referred to as *critical paths*.

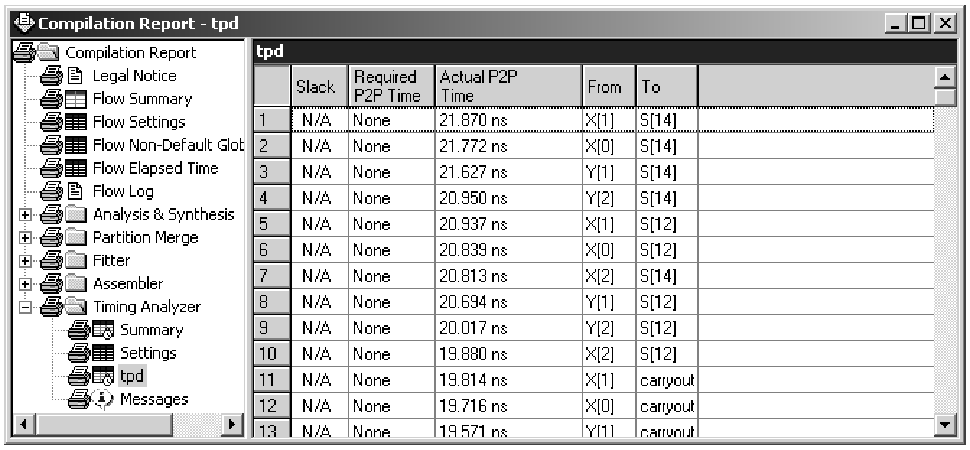
The Timing Analyzer performs several types of timing analysis. The results displayed in Figure C.21 give the delays through a combinational circuit, from input pins to output pins. The other types of analysis are applicable only to circuits that contain storage elements, namely flip-flops. This type of analysis is discussed in section C.4.



Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	Fr...	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	21.870 ns	X[1]	S[14]	--	--	0
2	Total number of failed paths								0

Figure C.20 The worst-case propagation delay.



tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	21.870 ns	X[1]	S[14]
2	N/A	None	21.772 ns	X[0]	S[14]
3	N/A	None	21.627 ns	Y[1]	S[14]
4	N/A	None	20.950 ns	Y[2]	S[14]
5	N/A	None	20.937 ns	X[1]	S[12]
6	N/A	None	20.839 ns	X[0]	S[12]
7	N/A	None	20.813 ns	X[2]	S[14]
8	N/A	None	20.694 ns	Y[1]	S[12]
9	N/A	None	20.017 ns	Y[2]	S[12]
10	N/A	None	19.880 ns	X[2]	S[12]
11	N/A	None	19.814 ns	X[1]	carryout
12	N/A	None	19.716 ns	X[0]	carryout
13	N/A	None	19.571 ns	Y[1]	carryout

Figure C.21 The critical paths.

We have finished working on the *addern* circuit, so close the project.

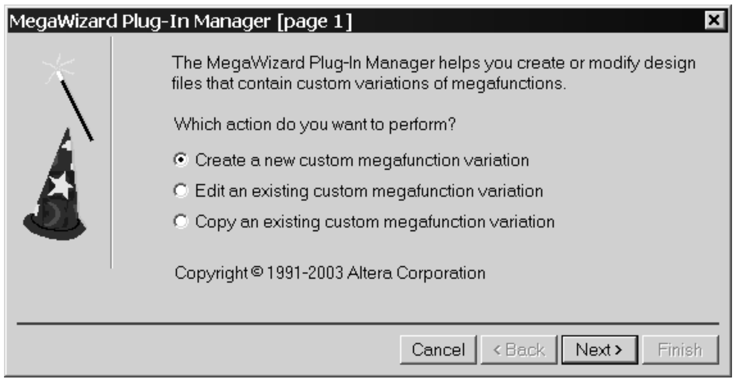
C.3 USING AN LPM MODULE

In section 5.5.1 we discuss how an adder circuit can be implemented by using the *lpm_add_sub* module in the library of parameterized modules (LPM). In this section we compare the adder circuit produced by the *lpm_add_sub* module to the ripple-carry adder implemented in the previous section. Create a new project, *adder16_lpm*, in a directory *tutorial2\adderlpm*. Choose the same FPGA chip as in section C.2.

The easiest way to instantiate an LPM module is by means of a wizard. Select Tools > MegaWizard Plug-in Manager to activate the wizard. A number of pop-up boxes will appear in which we can specify the features of the desired module. In the screen shown in Figure C.22 choose to create a new variation of a megafunction, and click Next. In the screen in Figure C.23 select the LPM_ADD_SUB module. Make sure that the Cyclone II family is indicated at the top right, and also select the entry Verilog HDL as the type of file to create. Let the output file be named *megadd.v*. (The filename extension, *v*, will be added automatically.) Click Next. In Figure C.24, specify that a 16-bit adder circuit is required. Click Next to reach the subsequent screen and accept the default setting that indicates that both inputs can vary. Click Next again to reach the window in Figure C.25 and specify that both carry input and output signals are needed. Observe that the wizard displays a symbol for the adder which includes the specified inputs and outputs. Advance past the next screen, which presents a pipelining option that we will not use. The last screen is given in Figure C.26, which indicates the files generated by the wizard. Click Finish. We are interested only in the *megadd.v* file, so make sure that this is the only file selected by a check mark.

The *megadd* module is shown in Figure C.27. (We have removed the comments to make the figure smaller.) The top-level Verilog code that instantiates this module is given in Figure C.28. Enter this code into a file called *adder16_lpm.v*.

Compile the design. A summary of the timing analysis is shown in Figure C.29. In this design, the worst-case propagation delay is about 13.28 ns. Clearly, the adder implementation by means of an appropriate LPM is superior to our generic specification in Figure C.10. The reason that this adder is much faster than our previously created ripple-carry adder is that the LPM makes use of special circuitry in the FPGA for performing addition. We discuss such circuitry, often called a *carry-chain*, in Sections 5.4 and 12.1. We may conclude that a designer should normally use an LPM if a suitable module exists in the library.



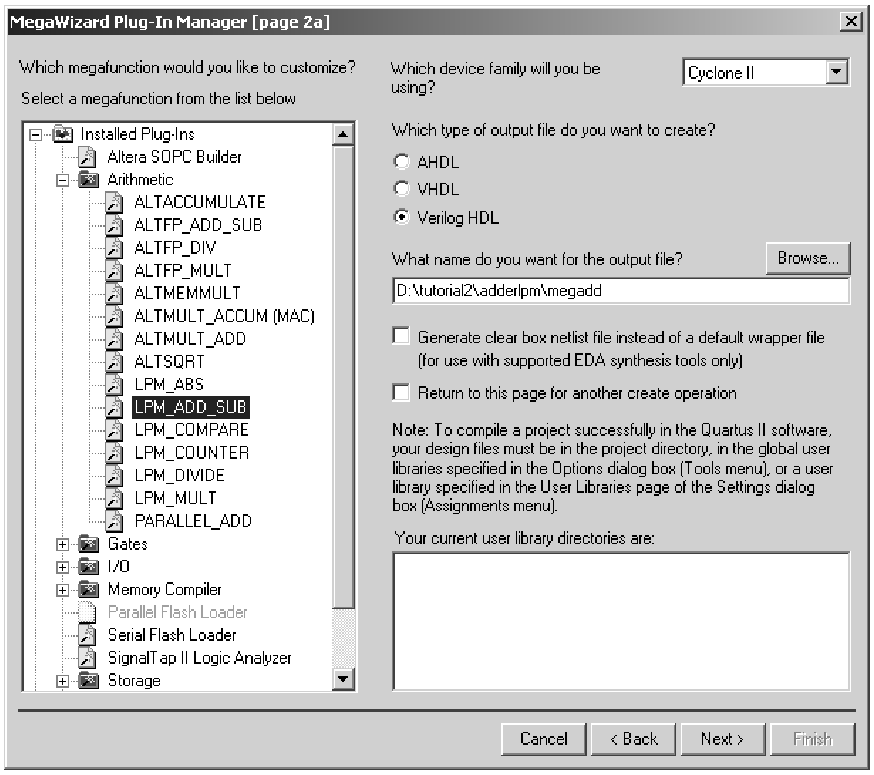
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

Which action do you want to perform?

- [✓ Create a new custom megafunction variation]
- Edit an existing custom megafunction variation
- Copy an existing custom megafunction variation

Copyright © 1991-2003 Altera Corporation

Figure C.22 Choose to create an LPM instance.


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?
Select a megafunction from the list below

- Installed Plug-Ins
  - Altera SOPC Builder
  - Arithmetic
    - ALTACCUMULATE
    - ALTFP_ADD_SUB
    - ALTFP_DIV
    - ALTFP_MULT
    - ALTMEMMULT
    - ALTMULT_ACCUM (MAC)
    - ALTMULT_ADD
    - ALTSQRT
    - LPM_ABS
    - LPM_ADD_SUB
    - LPM_COMPARE
    - LPM_COUNTER
    - LPM_DIVIDE
    - LPM_MULT
    - PARALLEL_ADD
  - Gates
  - I/O
  - Memory Compiler
  - Parallel Flash Loader
  - Serial Flash Loader
  - SignalTap II Logic Analyzer
  - Storage

Which device family will you be using? Cyclone II

Which type of output file do you want to create?
AHDL
VHDL
[✓ Verilog HDL]

What name do you want for the output file? D:\tutorial2\adderlpm\megadd

Generate clear box netlist file instead of a default wrapper file (for use with supported EDA synthesis tools only)

Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

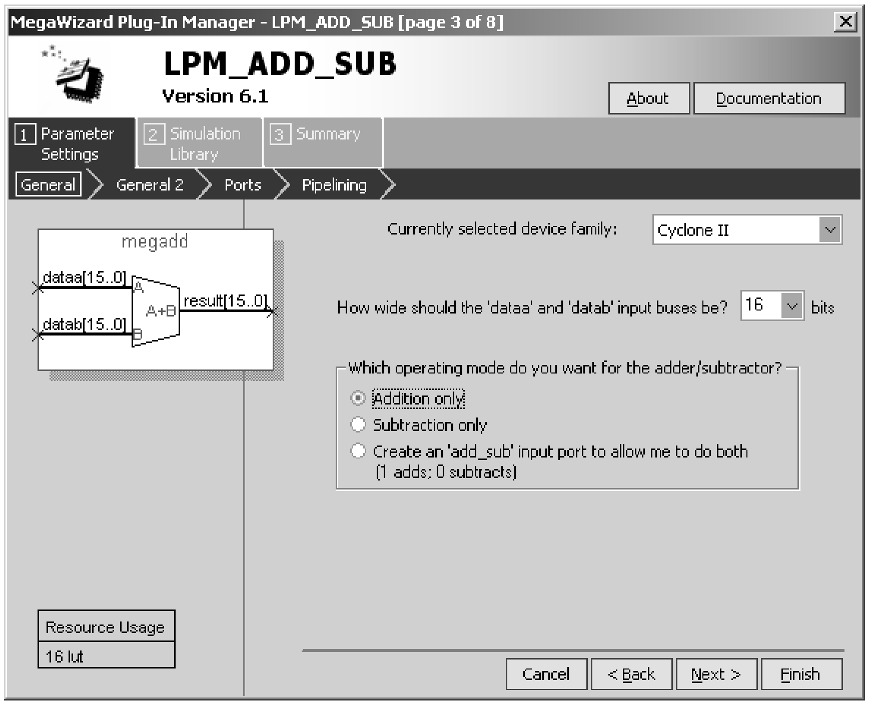
Figure C.23 Select the LPM and its Verilog specification.

To examine the circuit produced by using the LPM adder, open the Chip Planner tool as discussed in section C.1.4. Locate in the Chip Planner the part of the circuit that implements the adder, as indicated in Figure C.30. The logic elements that comprise the adder are connected together vertically by the carry chain wires. As indicated in the figure, select one of the logic elements in the adder and double-click on it to examine its contents in the Resource Property Editor tool. As illustrated in Figure C.31, the logic element is configured into a mode that produces both a sum output as well as a separate carry output that is fed to the next stage of the adder.

The *adder16_lpm* project can now be closed.

C.4 DESIGN OF A FINITE STATE MACHINE

This example shows how to implement a sequential circuit using Quartus II. The presentation assumes that the reader is familiar with the material in Chapter 8. In section 8.1 we show a simple Moore-type finite state machine (FSM) that has one input, *w*, and one output, *z*.


MegaWizard Plug-In Manager - LPM_ADD_SUB [page 3 of 8]

LPM_ADD_SUB
Version 6.1

Currently selected device family: Cyclone II

How wide should the 'dataa' and 'datab' input buses be? 16 bits

Which operating mode do you want for the adder/subtractor?
[✓ Addition only]
Subtraction only
Create an 'add_sub' input port to allow me to do both (1 adds; 0 subtracts)

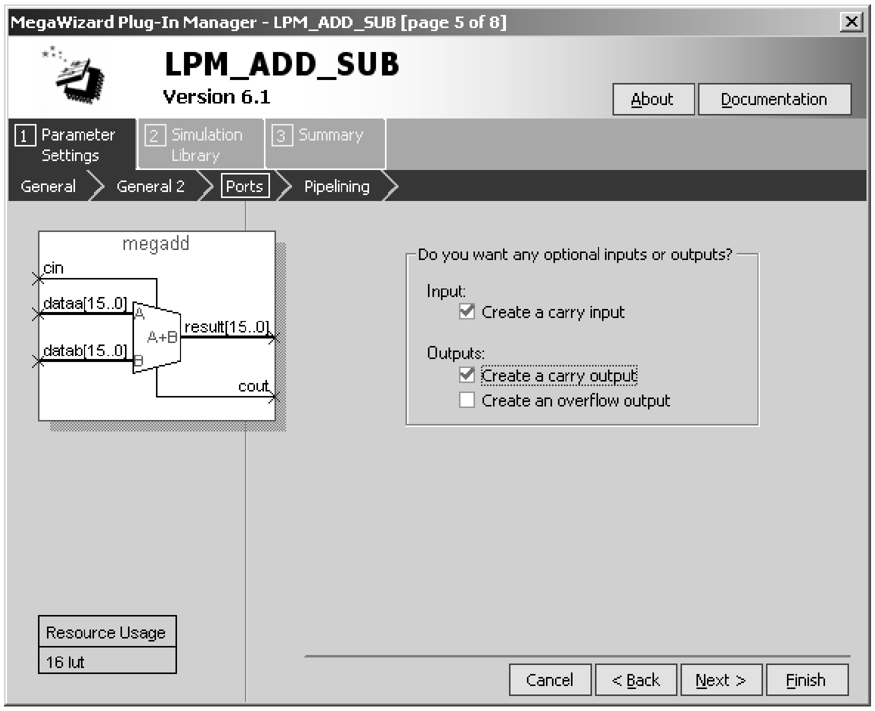
Resource Usage: 16 lut

Figure C.24 Choose the adder option and the number of bits.

Whenever w is 1 for two successive clock cycles, z is set to 1. The state diagram for the FSM is given in Figure 8.3; it is reproduced in Figure C.32. Verilog code that describes the machine appears in Figure 8.33; it is reproduced in Figure C.33. Create a new project, *simple*, in the directory *tutorial2\fsm*. Create a new Text Editor file and enter the code shown in Figure C.33. Save the file with the name *simple.v*.

Select the same Cyclone II device as in previous sections of this tutorial. Before compiling the code we need to make one change in the settings used by the synthesis module in Quartus II. Select Assignments > Settings to open the Settings window, and under Category click on the item Analysis and Synthesis Settings. Then, click on the button More Settings to open the window shown in Figure C.34. In the box called Existing options settings scroll down to the bottom of the list and click on the item State Machine Processing. Change the value of this option from its default (Auto) to the setting User-Encoded. This setting instructs the Synthesis module to use the state assignment specified in the Verilog code in Figure C.33, rather than to choose state codes automatically. Compile the circuit.

Open the Waveform Editor and import the nodes *Resetn*, *Clock*, *w*, and *z*. These nodes are found by setting the Node Finder filter to Pins: all. We also want to see the behavior of the state variables, which are implemented by means of flip-flops. To find these nodes, set the Node Finder filter to Registers: post-fitting and click List. The Node Finder displays two


MegaWizard Plug-In Manager - LPM_ADD_SUB [page 5 of 8]

LPM_ADD_SUB
Version 6.1

Do you want any optional inputs or outputs?

Input:
- [✓] Create a carry input

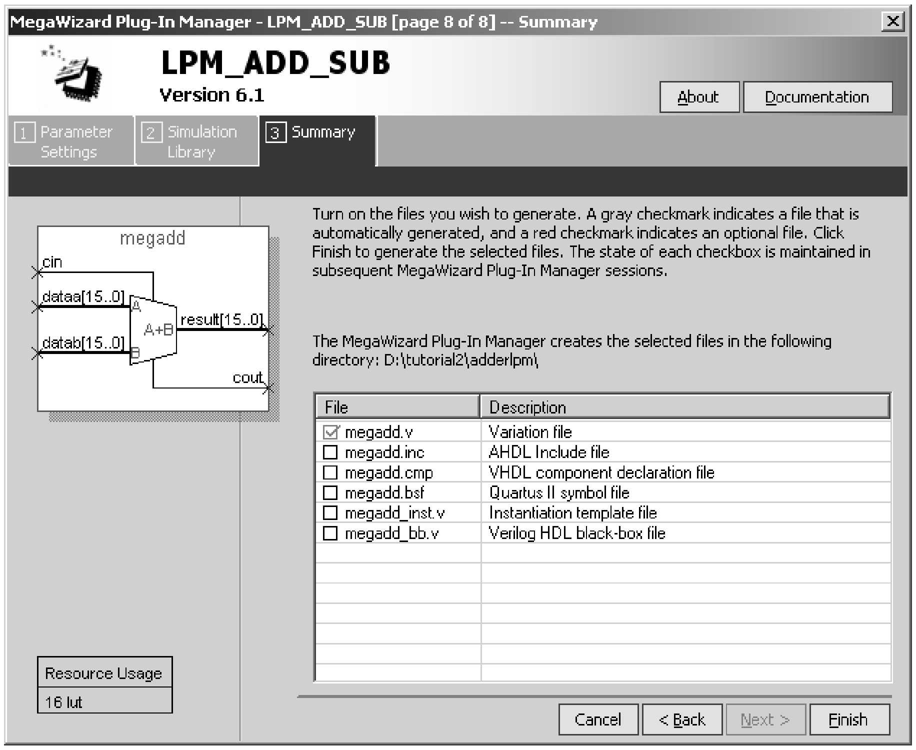
Outputs:
- [✓] Create a carry output
- [ ] Create an overflow output

Resource Usage
16 lut

Figure C.25 Include carry input and output connections.

nodes, as shown in Figure C.35. Import both of these nodes into the Waveform Editor. Set the total simulation time to 650 ns and set the grid size to 25 ns. Set *Resetn* = 0 during the first 50 ns, and then set *Resetn* = 1. To enter the waveform for the clock signal, click on the name of the *Clock* waveform in the Waveform Editor display. With the signal highlighted, click on the *Overwrite Clock* icon in the toolbar (the icon depicts a clock). This causes the pop-up window in Figure C.36 to appear. Set the clock period to be 50 ns, make sure that the offset is 0 and the duty cycle is 50 percent, and click OK. The defined clock signal is now displayed in the Waveform Editor window, as depicted in Figure C.37. Next, draw the waveform for w as indicated in the figure. To make the changes in w occur shortly after the positive clock edge, we temporarily changed the grid size in the Waveform Editor to 5 ns. Specifying the waveform for w in this manner is a reasonable choice, because most signals in a real system are generated by flip-flops that use the same clock signal. Save the file, under the name *simple.vwf*. Run the Timing Simulator to get the result shown in Figure C.38.

The FSM behaves correctly, setting $z = 1$ in each clock cycle for which $w = 1$ in the preceding two clock cycles. Examine the timing delays in the circuit, using the reference line in the Waveform Editor. Observe that changes in the FSM's state occur about 2.95 ns after an active clock edge and that 6.37 ns are needed to change the value of z at its output pin.



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a red checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory: D:\tutorial2\adderlpm\

File	Description
✓ megadd.v	Variation file
☐ megadd.inc	AHDL Include file
☐ megadd.cmp	VHDL component declaration file
☐ megadd.bsf	Quartus II symbol file
☐ megadd_inst.v	Instantiation template file
☐ megadd_bb.v	Verilog HDL black-box file

Figure C.26 Files generated by the wizard.

Open the Timing Analyzer summary in the compilation report, which is displayed in Figure C.39. Row 4 in the table indicates that the maximum frequency, which is often called F_{max}, at which the synthesized circuit can operate is 420.17 MHz. This is a useful indicator of performance. The F_{max} is determined by the longest propagation delay between two registers (flip-flops). The figure also shows the values of some other timing parameters. The worst-case flip-flop setup time, t_{su}, and hold time, t_h, are given. Line 1 in Figure C.39 specifies that the w input can change until up to 0.609 ns *after* the active clock edge occurs (at the clock pin), and still meet the flip-flop setup requirement. Line 3 shows that the worst-case hold time at the w input pin is 0.839 ns after the active clock edge. We explain in section 10.3.2 how flip-flop timing parameters are determined in a target chip. The parameter t_{co} indicates the time elapsed from an active edge of the clock signal at the clock pin until an output signal is produced at an output pin. This delay is 6.37 ns for the z output, which is what we also observed in the waveforms in Figure C.38.

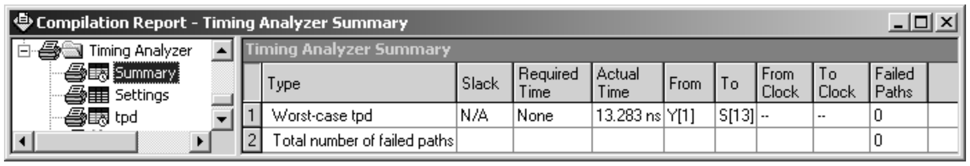
Note that the states of this FSM are implemented using two state variables. The Verilog code in Figure C.33 specified the present state variables as $y[2]$ and $y[1]$. However, Quartus II gave the names $y{\sim}15$ and $y{\sim}14$ to these variables, as we discovered when using the Node

```
module megadd (dataa, datab, cin, result, cout);
   input [15:0] dataa;
   input [15:0] datab;
   input cin;
   output [15:0] result;
   output cout;
   wire sub wire0;
   wire [15:0] sub wire1;
   wire cout = sub wire0;
   wire [15:0] result = sub wire1[15:0];

   lpm_add_sub lpm_add_sub component (
                  .dataa (dataa),
                  .datab (datab),
                  .cin (cin),
                  .cout (sub wire0),
                  .result (sub wire1));
      defparam
         lpm_add_sub_component.lpm width = 16,
         lpm_add_sub_component.lpm direction = "ADD",
         lpm_add_sub_component.lpm type = "LPM_ADD_SUB",
         lpm_add_sub_component.lpm hint = "ONE_INPUT_IS_CONSTANT=NO";

endmodule
```

Figure C.27 Verilog code for the *megadd* module.

```
module adder16_lpm (carryin, X, Y, S, carryout);
   input carryin;
   input [15:0] X, Y;
   output [15:0] S;
   output carryout;

   megadd adder circuit (.cin(carryin), .dataa(X), .datab(Y),
               .result(S), .cout(carryout));
endmodule
```

Figure C.28 Verilog code that instantiates the LPM adder module.



	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	13.283 ns	Y[1]	S[13]	--	--	0
2	Total number of failed paths								0

Figure C.29 The worst-case delay for the *adder16_lpm* circuit.

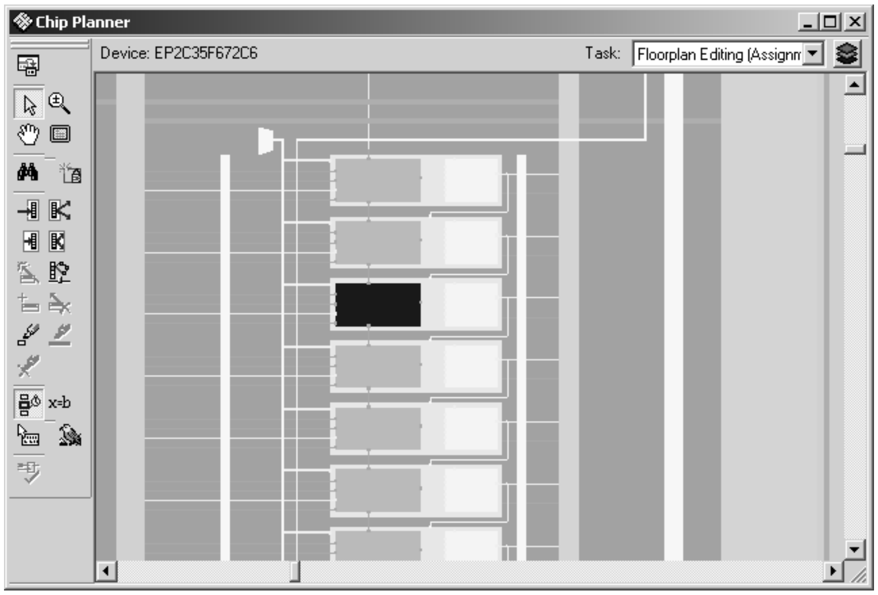


Figure C.30 Examining the 16-bit adder in the Chip Planner.

Finder. Quartus II uses the names of all inputs and outputs as given in the Verilog code, but it may generate different names for internal connections.

Two or more binary signals displayed in the Waveform Editor can be combined into a "group" (corresponding to a vector in Verilog terminology) of signals that can be referred to by a single name. Open the *simple.vwf* file and select the *y*∼15 and *y*∼14 simultaneously, so that their waveforms are highlighted (make sure that *y*∼15 is listed *above* *y*∼14, as shown in Figure C.37). Select Edit > Grouping > Group to reach the pop-up box in Figure C.40. Type *y* as the group name, choose binary as the radix, and click OK. This causes *y* to be used, instead of *y*∼15 and *y*∼14, in the file *simple.vwf*. Perform timing simulation to get the result in Figure C.40. Now, the FSM states are represented by the values of the vector *y* that correspond to our Verilog code.

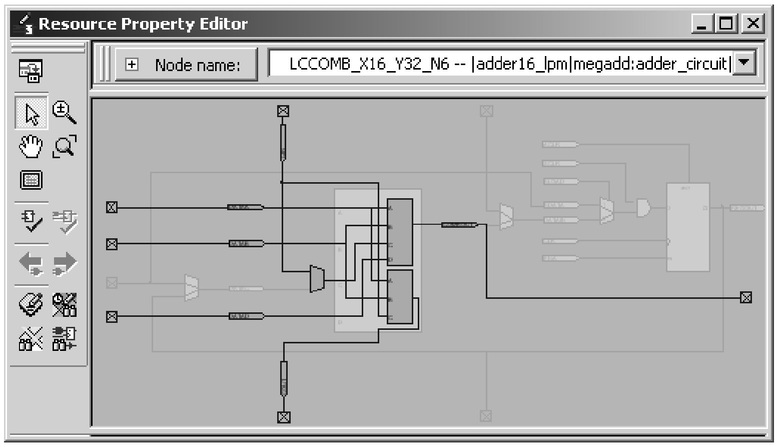


Figure C.31 One stage of the 16-bit adder.

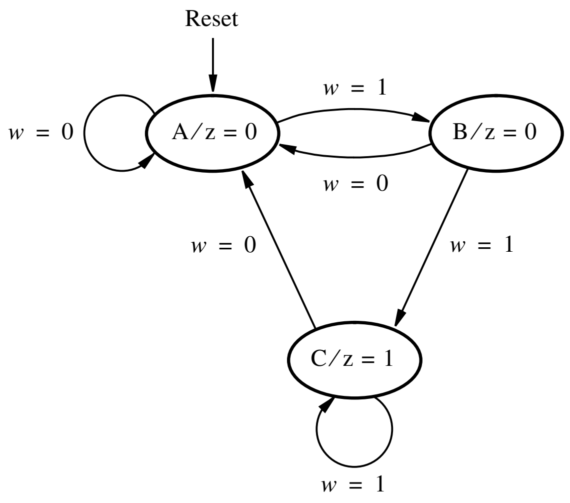


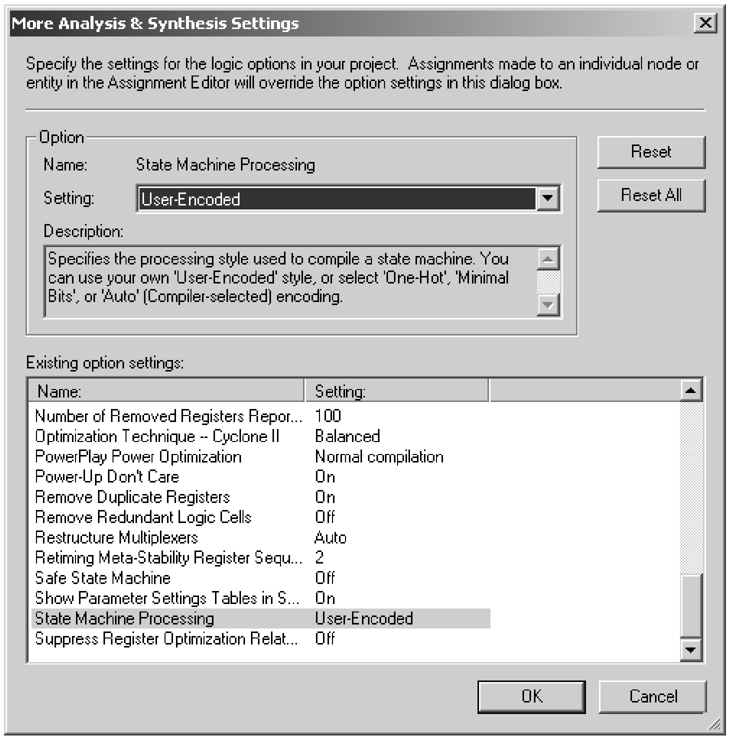
Figure C.32 State diagram of a Moore-type FSM.

```
module simple (Clock, Resetn, w, z);
   input Clock, Resetn, w;
   output z;
   reg [2:1] y, Y;
   parameter [2:1] A = 2 b00, B = 2 b01, C = 2 b10;

   // Define the next state combinational circuit
   always @(w or y)
      case (y)
         A: if (w)   Y = B;
            else     Y = A;
         B: if (w)   Y = C;
            else     Y = A;
         C: if (w)   Y = C;
            else     Y = A;
         default:    Y = 2 bxx;
      endcase

   // Define the sequential block
   always @(negedge Resetn or posedge Clock)
      if (Resetn == 0)  y <= A;
      else  y <= Y;

   // Define output
   assign z = (y == C);

endmodule
```

Figure C.33 Verilog code for the FSM in Figure C.32.

C.5 Concluding Remarks

Having completed this and the preceding tutorial, the reader is familiar with many of the most important features of Quartus II. In the next tutorial we will show how the user can manipulate which pins on the target chip are used for a circuit, and how FPGA programming is done with Quartus II.



More Analysis & Synthesis Settings

Specify the settings for the logic options in your project. Assignments made to an individual node or entity in the Assignment Editor will override the option settings in this dialog box.

Option

Name: State Machine Processing

Setting: User-Encoded

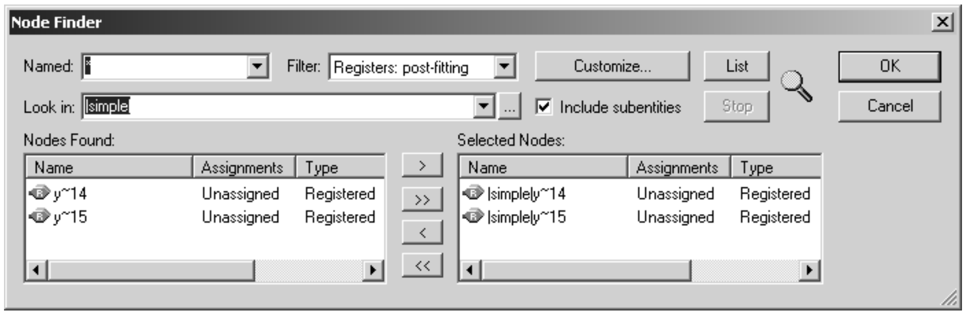
Description:

Specifies the processing style used to compile a state machine. You can use your own 'User-Encoded' style, or select 'One-Hot', 'Minimal Bits', or 'Auto' (Compiler-selected) encoding.

Existing option settings:

Name:	Setting:
Number of Removed Registers Repor...	100
Optimization Technique -- Cyclone II	Balanced
PowerPlay Power Optimization	Normal compilation
Power-Up Don't Care	On
Remove Duplicate Registers	On
Remove Redundant Logic Cells	Off
Restructure Multiplexers	Auto
Retiming Meta-Stability Register Sequ...	2
Safe State Machine	Off
Show Parameter Settings Tables in S...	On
State Machine Processing	User-Encoded
Suppress Register Optimization Relat...	Off

Figure C.34 Setting the state machine processing to User-Encoded.



Node Finder

Named: *

Filter: Registers: post-fitting

Look in: |simple|

☑ Include subentities

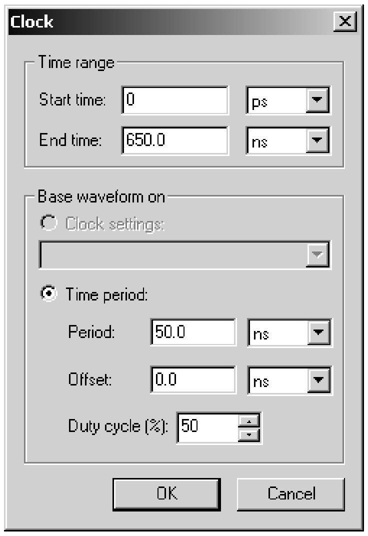
Nodes Found:

Name	Assignments	Type
y~14	Unassigned	Registered
y~15	Unassigned	Registered

Selected Nodes:

Name	Assignments	Type
|simple|y~14	Unassigned	Registered
|simple|y~15	Unassigned	Registered

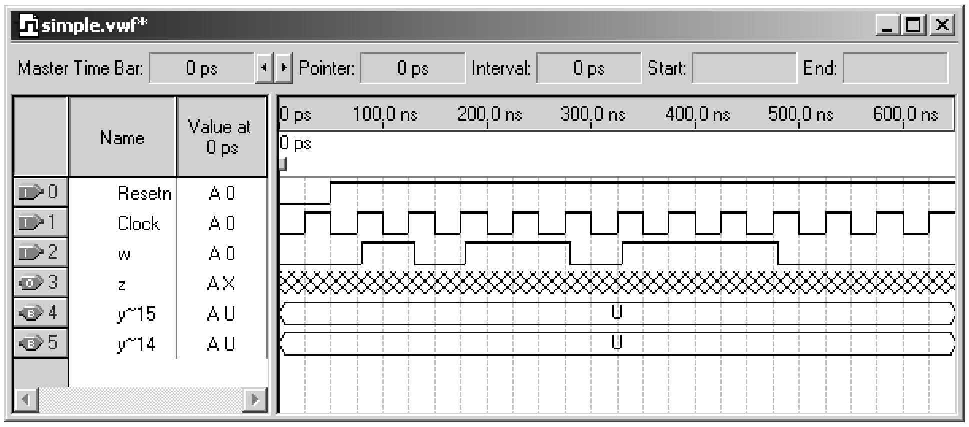
Figure C.35 Nodes that represent the state variables.


Clock

Time range
Start time: 0 ps
End time: 650.0 ns

Base waveform on
Clock settings:
(●) Time period:
Period: 50.0 ns
Offset: 0.0 ns
Duty cycle (%): 50

Figure C.36 Creating the *Clock* waveform.


simple.vwf*

Master Time Bar: 0 ps Pointer: 0 ps Interval: 0 ps Start: End:

	Name	Value at 0 ps
0	Resetn	A 0
1	Clock	A 0
2	w	A 0
3	z	A X
4	y~15	A U
5	y~14	A U

Figure C.37 Input test vectors.

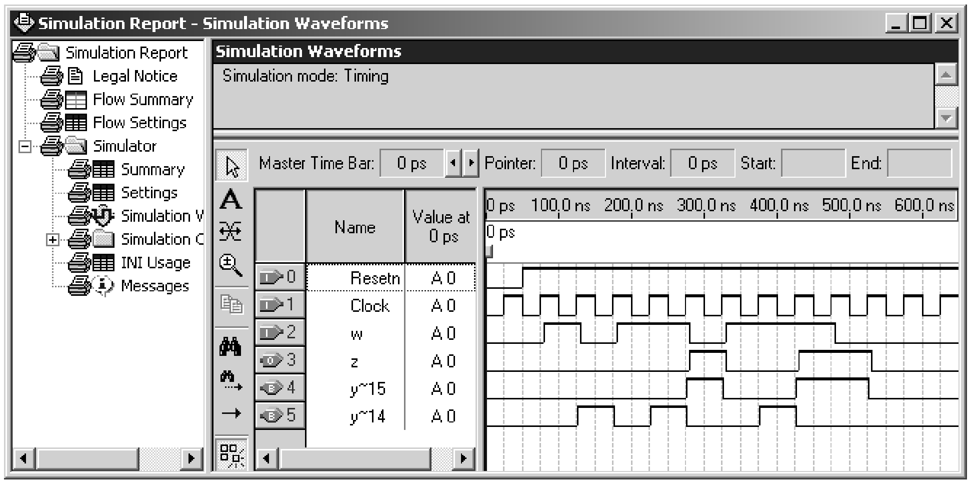
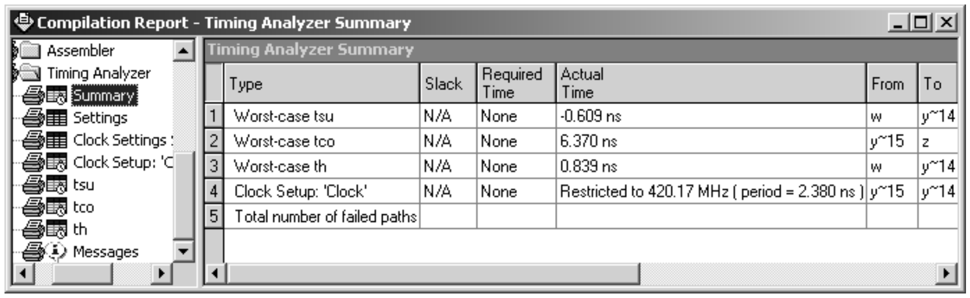
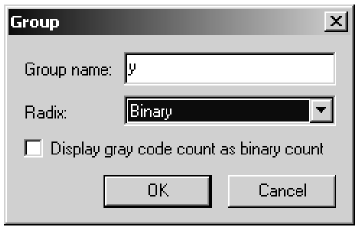


Figure C.38 Timing simulation waveforms.



	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	-0.609 ns	w	y~14
2	Worst-case tco	N/A	None	6.370 ns	y~15	z
3	Worst-case th	N/A	None	0.839 ns	w	y~14
4	Clock Setup: 'Clock'	N/A	None	Restricted to 420.17 MHz (period = 2.380 ns)	y~15	y~14
5	Total number of failed paths					

Figure C.39 Summary of the timing analysis for the FSM circuit.


Group

Group name: y

Radix: Binary

☐ Display gray code count as binary count

Figure C.40 Grouping of signals.

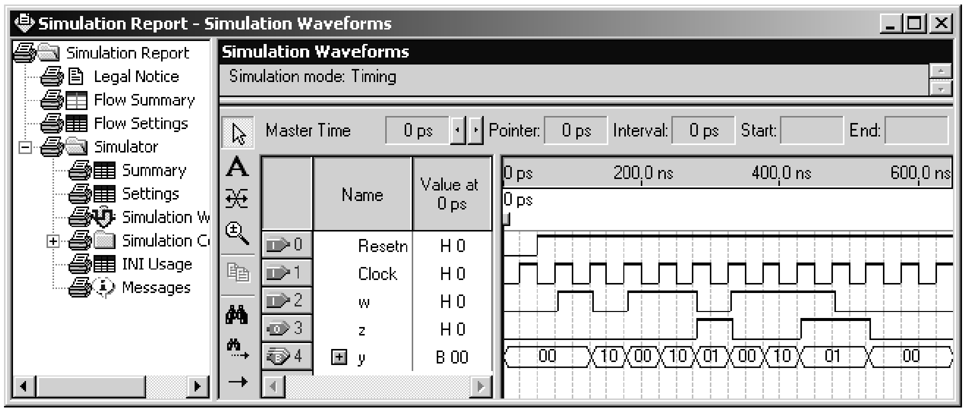


Figure C.41 Waveform displayed as a vector y.