

Quartus II Introduction for Verilog Users

Computer Aided Design (CAD) software makes it easy to implement a desired logic circuit by using a programmable logic device, such as a field-programmable gate array (FPGA) chip. A typical FPGA CAD flow is illustrated in Figure 1.

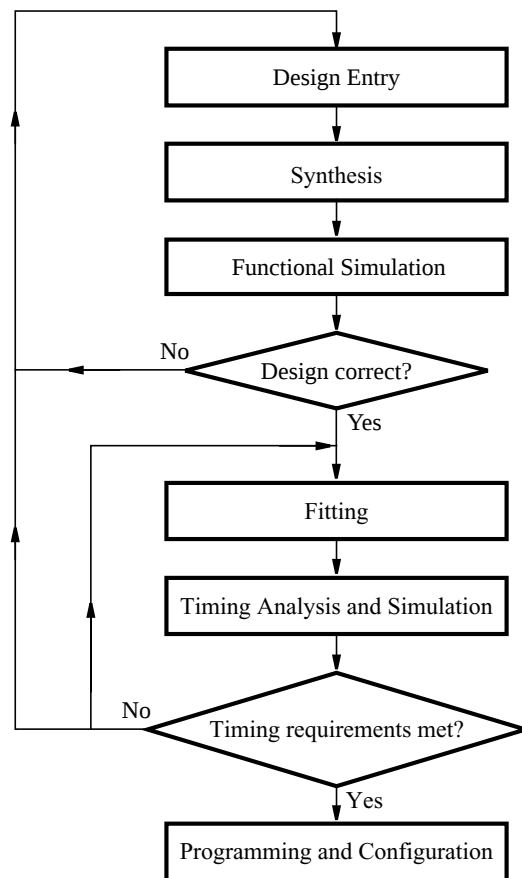


Figure 1. Typical CAD flow.

It involves the following basic steps:

- **Design Entry** – the desired circuit is specified either by using a hardware description language, such as Verilog or VHDL, or by means of a schematic diagram
- **Synthesis** – the CAD Synthesis tool synthesizes the circuit into a netlist that gives the logic elements (LEs) needed to realize the circuit and the connections between the LEs
- **Functional Simulation** – the synthesized circuit is tested to verify its functional correctness; the simulation does not take into account any timing issues

- **Fitting** – the CAD Fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs
- **Timing Analysis** – propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit
- **Timing Simulation** – the fitted circuit is tested to verify both its functional correctness and timing
- **Programming and Configuration** – the designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections

This tutorial introduces the basic features of the Quartus[®] II software. It shows how the software can be used to design and implement a circuit specified using the Verilog hardware description language. It makes use of the graphical user interface to invoke the Quartus II commands. Doing this tutorial, the reader will learn about:

- Creating a project
- Synthesizing a circuit from Verilog code using the Quartus II Integrated Synthesis tool
- Fitting a synthesized circuit into an Altera FPGA
- Examining the report on the results of fitting and timing analysis
- Examining the synthesized circuit in the form of a schematic diagram generated by the RTL Viewer tool
- Making simple timing assignments in the Quartus II software

1 Getting Started

Each logic circuit, or subcircuit, being designed with Quartus II software is called a *project*. The software works on one project at a time and keeps all information for that project in a single directory (folder) in the file system. To begin a new logic circuit design, the first step is to create a directory to hold its files. As part of the installation of the Quartus II software, a few sample projects are placed into a directory called `\qdesigns`. To hold the design files for this tutorial, we will use a directory `\quartus.tutorial`. The running example for this tutorial is a simple adder/subtractor circuit, which is defined in the Verilog hardware description language.

Start the Quartus II software. You should see a display similar to the one in Figure 2. This display consists of several windows that provide access to all the features of Quartus II software, which the user selects with the computer mouse. Most of the commands provided by Quartus II software can be accessed by using a set of menus that are located below the title bar. For example, in Figure 2 clicking the left mouse button on the menu named File opens the menu shown in Figure 3. Clicking the left mouse button on the entry Exit exits from Quartus II software. In general, whenever the mouse is used to select something, the *left* button is used. Hence we will not normally specify which button to press. In the few cases when it is necessary to use the *right* mouse button, it will be specified explicitly. For some commands it is necessary to access two or more menus in sequence. We use the convention Menu1 > Menu2 > Item to indicate that to select the desired command the user should first click the left mouse button on Menu1, then within this menu click on Menu2, and then within Menu2 click on Item. For example, File > Exit uses the mouse to exit from the system. Many commands can be invoked by clicking on an icon displayed in one of the toolbars. To see the list of available toolbars, select Tools > Customize > Toolbars. Once a toolbar is opened, it can be moved using the mouse, and icons can be dragged from one toolbar to another. To see the command associated with an icon, position the mouse over the icon and a tooltip will appear that displays the command name.

It is possible to modify the appearance of the display in Figure 2 in many ways. Section 7 shows how to move, resize, close, and open windows within the main Quartus II display.

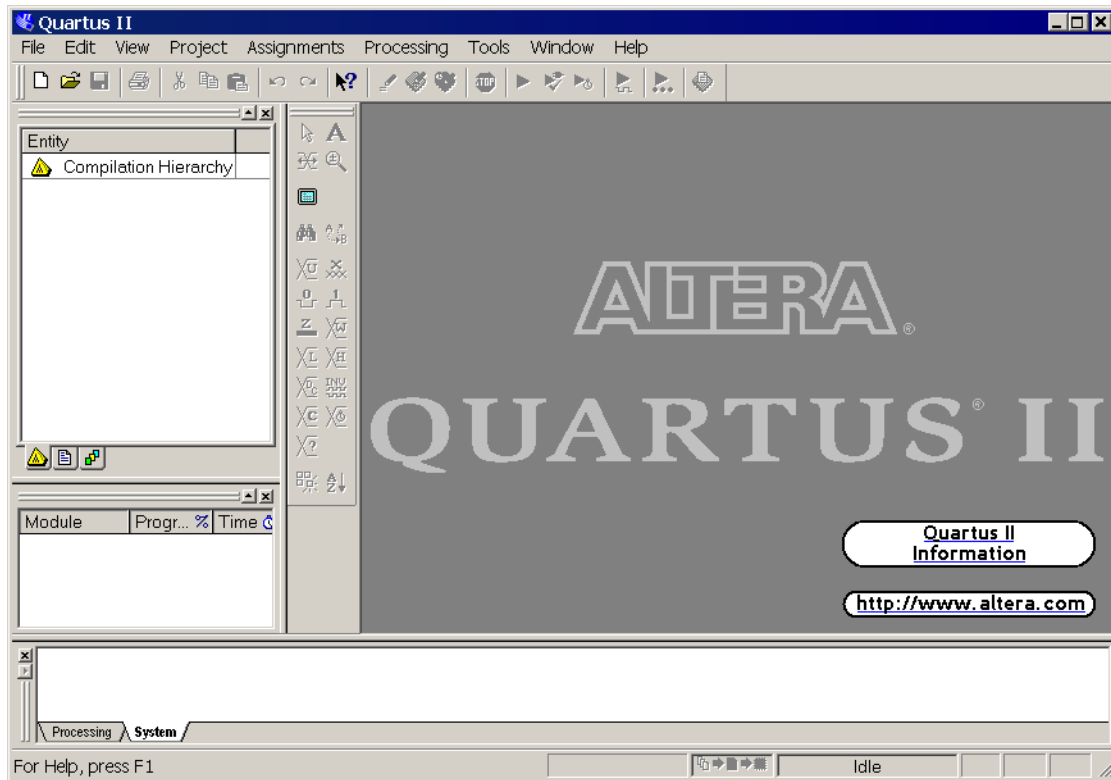


Figure 2. The main Quartus II display.

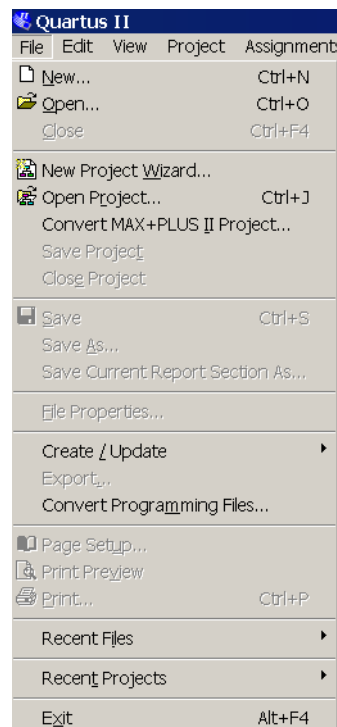


Figure 3. An example of the File menu.

1.1 Quartus II Online Help

Quartus II software provides comprehensive online documentation that answers many of the questions that may arise when using the software. The documentation is accessed from the menu in the **Help** window. To get some idea of the extent of documentation provided, it is worthwhile for the reader to browse through the **Help** menu. For instance, selecting **Help > How to Use Help** gives an indication of what type of help is provided.

The user can quickly search through the Help topics by selecting **Help > Search**, which opens a dialog box into which key words can be entered. Another method, context-sensitive help, is provided for quickly finding documentation for specific topics. While using most applications, pressing the **F1** function key on the keyboard opens a Help display that shows the commands available for the application.

2 Starting a New Project

To start working on a new design we first have to define a new *design project*. Quartus II software makes the designer's task easy by providing support in the form of a *wizard*. Select **File > New Project Wizard** to reach a window that indicates the capability of this wizard. Press **Next** to get the window shown in Figure 4.

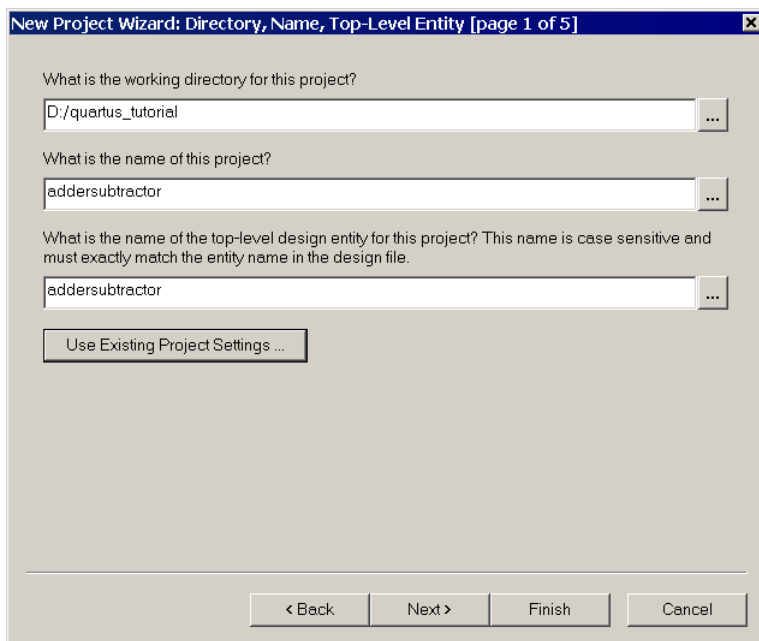


Figure 4. Creation of a new project.

Set the working directory to be *quartus_tutorial*; of course, you can use a directory name of your choice. The project must have a name, which is usually the same as the top-level design entity that will be included in the project. Choose *addersubtractor* as the name for both the project and the top-level entity, as shown in Figure 4. Press **Next**. Since we have not yet created the directory *quartus_tutorial*, Quartus II software displays the pop-up box in Figure 5 asking if it should create the desired directory. Click **Yes**, which leads to the window in Figure 6. This window makes it easy to specify which existing files (if any) should be included in the project. Assuming that we do not have any existing files, click **Next**, which leads to the window in Figure 7. Here, we can specify the type of device in which the designed circuit will be implemented. Choose Stratix™ as the target device family. We can let Quartus II software select a specific device in the family, or we can choose the device explicitly. We will

take the latter approach. From the list of available devices, choose the device called EP1S10F484C5. Press Next, which opens the window in Figure 8. Here, one can specify any third-party tools that should be used. A commonly used term for CAD software for electronic circuits is *EDA tools*, where the acronym stands for electronic design automation. This term is used in Quartus II messages that refer to third-party tools, which are the tools developed and marketed by companies other than Altera; other tutorials show how such tools may be used. Since we will rely solely on Quartus II tools, we will not choose any other tools. Press Next. Now, a summary of the chosen settings appears in the screen shown in Figure 9. Press Finish, which returns to the main Quartus II window, but with *addersubtractor* specified as the new project, in the display title bar, as indicated in Figure 10.

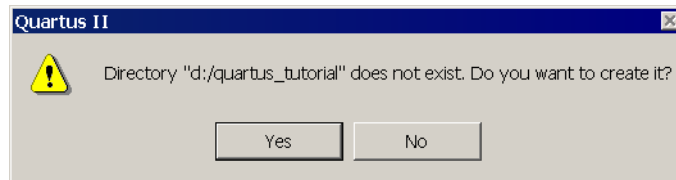


Figure 5. Quartus II software can create a new directory for the project.

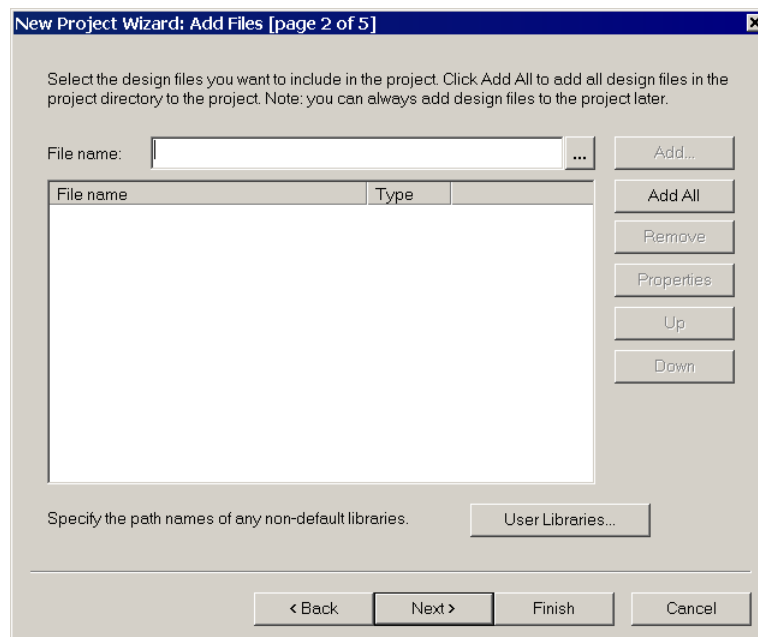


Figure 6. The wizard can include user-specified design files.

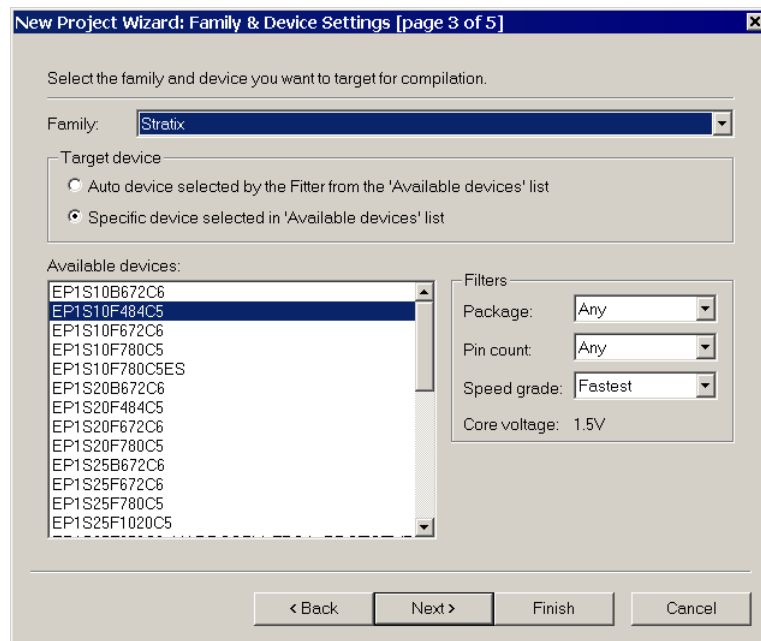


Figure 7. Choose the device family and a specific device.

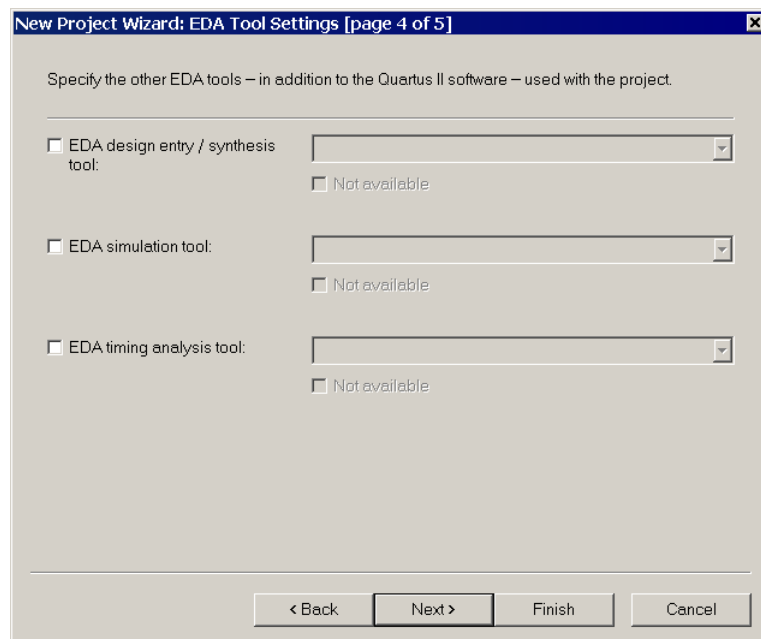


Figure 8. Other EDA tools can be specified.

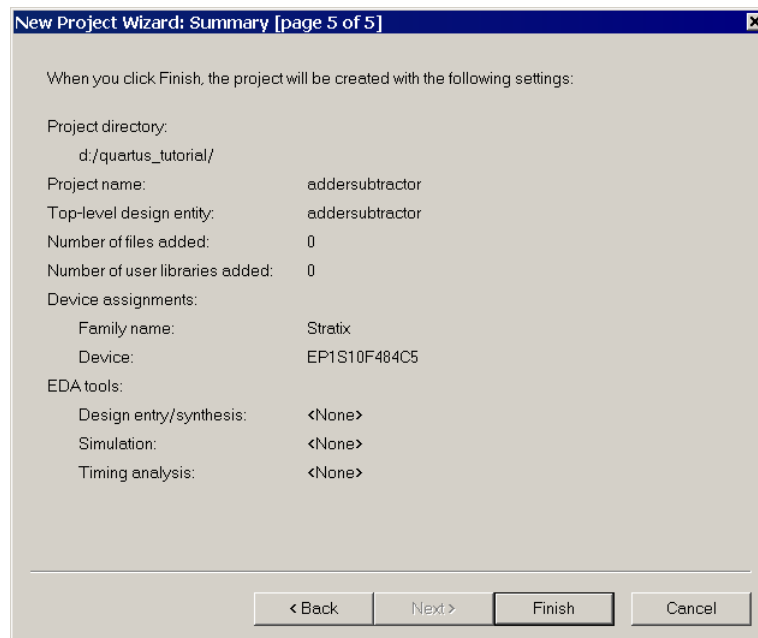


Figure 9. Summary of the project settings.

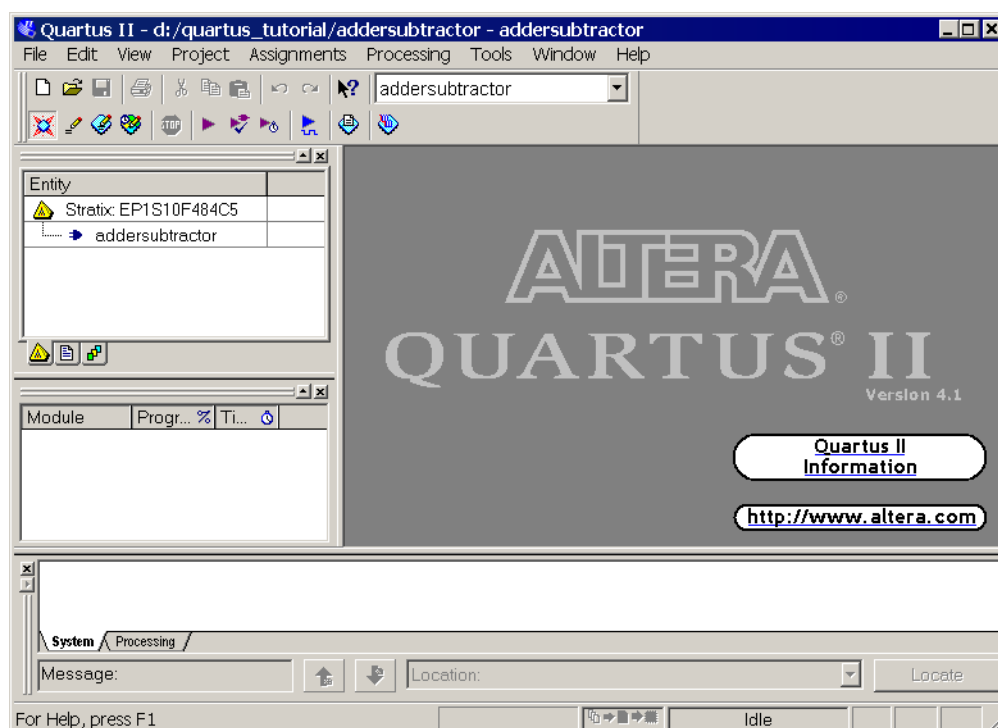


Figure 10. The Quartus II display for the created project.

3 Design Entry Using Verilog Code

As a design example, we will use the adder/subtractor circuit shown in Figure 11. The circuit can add, subtract, and accumulate n -bit numbers using the 2's complement number representation. The two primary inputs are numbers $A = a_{n-1}a_{n-2} \cdots a_0$ and $B = b_{n-1}b_{n-2} \cdots b_0$, and the primary output is $Z = z_{n-1}z_{n-2} \cdots z_0$. Another input is the *AddSub* control signal which causes $Z = A + B$ to be performed when *AddSub* = 0 and $Z = A - B$ when *AddSub* = 1. A second control input, *Sel*, is used to select the accumulator mode of operation. If *Sel* = 0, the operation $Z = A \pm B$ is performed, but if *Sel* = 1, then B is added to or subtracted from the current value of Z . If the addition or subtraction operations result in arithmetic overflow, an output signal, *Overflow*, is asserted.

To make it easier to deal with asynchronous input signals, we will load them into flip-flops on a positive edge of the clock. Thus, inputs A and B will be loaded into registers *Areg* and *Breg*, while *Sel* and *AddSub* will be loaded into flip-flops *SelR* and *AddSubR*, respectively. The adder/subtractor circuit places the result into register *Zreg*.

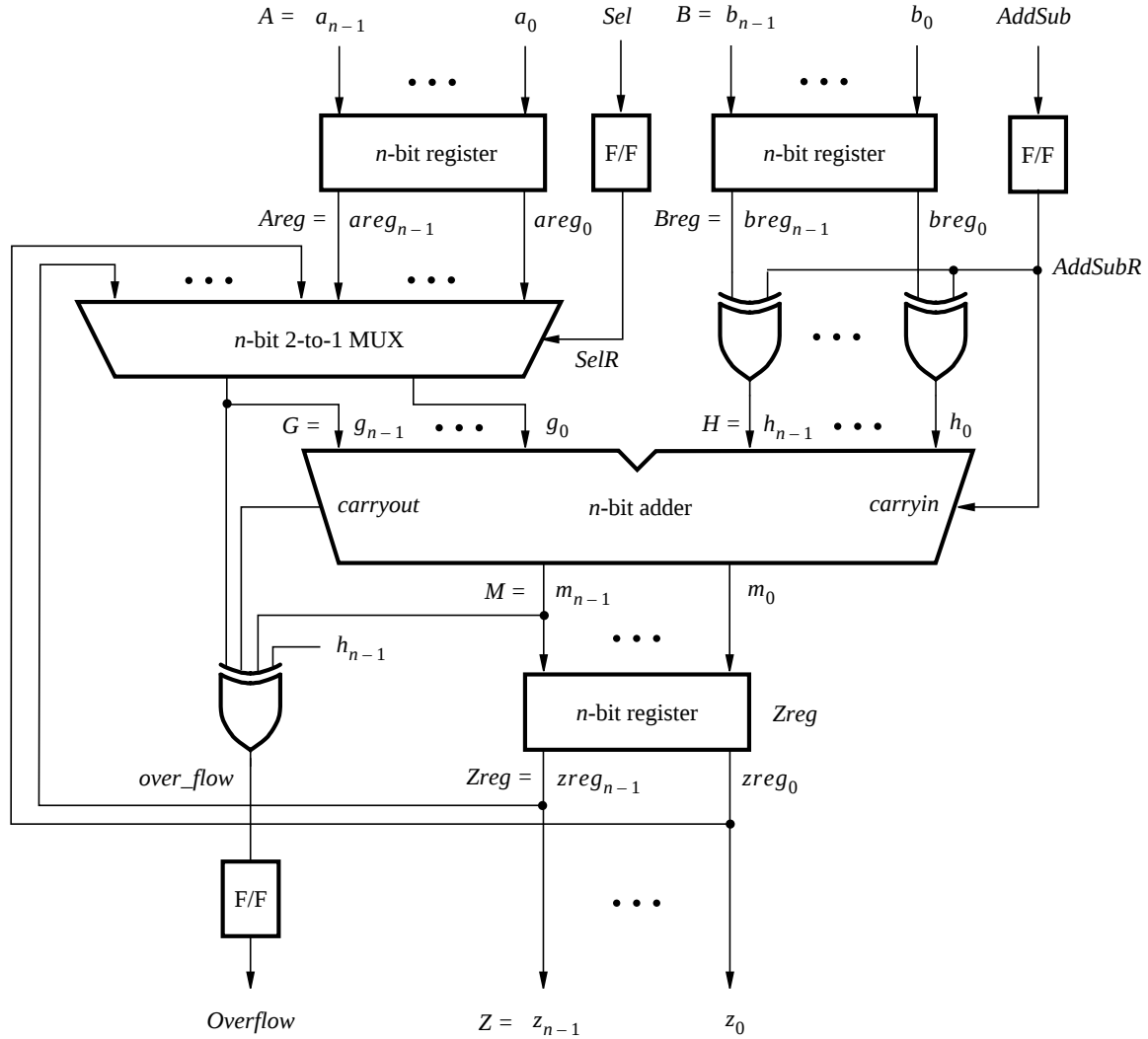


Figure 11. The adder/subtractor circuit.

The required circuit is described by the Verilog code in Figure 12. For our example, we will use a 16-bit circuit as specified by $n = 16$.

```

// Top-level module
module addersubtractor (A, B, Clock, Reset, Sel, AddSub, Z, Overflow);
    parameter n = 16;
    input [n-1:0] A, B;
    input Clock, Reset, Sel, AddSub;
    output [n-1:0] Z;
    output Overflow;
    reg SelR, AddSubR, Overflow;
    reg [n-1:0] Areg, Breg, Zreg;
    wire [n-1:0] G, H, M, Z;
    wire carryout, over_flow;

// Define combinational logic circuit
    assign H = Breg ^ {n{AddSubR}};
    mux2to1 multiplexer (Areg, Z, SelR, G);
    defparam multiplexer.k = n;
    adderk nbit_adder (AddSubR, G, H, M, carryout);
    defparam nbit_adder.k = n;
    assign over_flow = carryout ^ G[n-1] ^ H[n-1] ^ M[n-1];
    assign Z = Zreg;

// Define flip-flops and registers
    always @(posedge Reset or posedge Clock)
        if (Reset == 1)
            begin
                Areg <= 0; Breg <= 0; Zreg <= 0;
                SelR <= 0; AddSubR <= 0; Overflow <= 0;
            end
        else
            begin
                Areg <= A; Breg <= B; Zreg <= M;
                SelR <= Sel; AddSubR <= AddSub; Overflow <= over_flow;
            end
endmodule

// k-bit 2-to-1 multiplexer
module mux2to1 (V, W, Sel, F);
    parameter k = 8;
    input [k-1:0] V, W;
    input Sel;
    output [k-1:0] F;
    reg [k-1:0] F;

    always @(V or W or Sel)
        if (Sel == 0) F = V;
        else F = W;
endmodule

... continued in Part b

```

Figure 12. Verilog code for the circuit in Figure 11 (Part a).

```
// k-bit adder
module adderk (carryin, X, Y, S, carryout);
    parameter k = 8;
    input [k-1:0] X, Y;
    input carryin;
    output [k-1:0] S;
    output carryout;
    reg [k-1:0] S;
    reg carryout;

    always @(X or Y or carryin)
        {carryout, S} = X + Y + carryin;
endmodule
```

Figure 12. Verilog code for the circuit in Figure 11 (Part b).

Note that the top Verilog module is called *addersubtractor* to match the name given in Figure 4, which was specified when the project was created. This code can be typed into a file by using any text editor that stores ASCII files, or by using the Quartus II text editing facilities. While the file can be given any name, it is a common designers' practice to use the same name as the name of the top-level Verilog module. The file name must include the extension *v*, which indicates a Verilog file. So, we will use the name *addersubtractor.v*. For convenience, we provide the required file in the directory *qdesigns*. Copy this file into the project directory *quartus tutorial*.

3.1 Using the Quartus II Text Editor

This section shows how to use the Quartus II Text Editor. You can skip this section if you prefer to use some other text editor to create the *addersubtractor.v* file, or if you have chosen to copy the file from the *qdesigns* directory.

Select File > New to get the window in Figure 13, choose Verilog HDL File, and click OK. This opens the Text Editor window. The first step is to specify a name for the file that will be created. Select File > Save As to open the pop-up box depicted in Figure 14. In the box labeled Save as type choose Verilog HDL File. In the box labeled File name type *addersubtractor*. Put a checkmark in the box Add file to current project. Click Save, which puts the file into the directory *quartus tutorial* and leads to the Text Editor window shown in Figure 15. Maximize the Text Editor window and enter the Verilog code in Figure 12 into it. Save the file by typing File > Save, or by typing the shortcut Ctrl-s.

Most of the commands available in the Text Editor are self-explanatory. Text is entered at the *insertion point*, which is indicated by a thin vertical line. The insertion point can be moved either by using the keyboard arrow keys or by using the mouse. Two features of the Text Editor are especially convenient for typing Verilog code. First, the editor can display different types of Verilog statements in different colors, which is the default choice. Second, the editor can automatically indent the text on a new line so that it matches the previous line. Such options can be controlled by the settings in Tools > Options > Text Editor.

3.1.1 Using Verilog Templates

The syntax of Verilog code is sometimes difficult for a designer to remember. To help with this issue, the Text Editor provides a collection of *Verilog templates*. The templates provide examples of various types of Verilog statements, such as a **module** declaration, an **always** block, and assignment statements. It is worthwhile to browse through the templates by selecting Edit > Insert Template > Verilog HDL to become familiar with this resource.

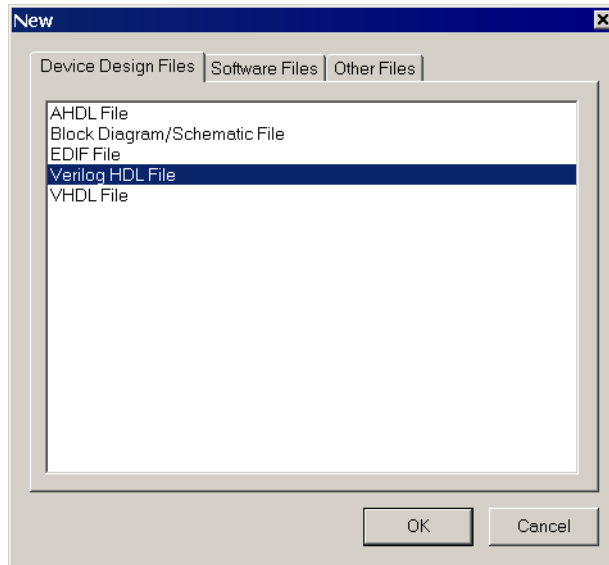


Figure 13. Choose to prepare a Verilog file.

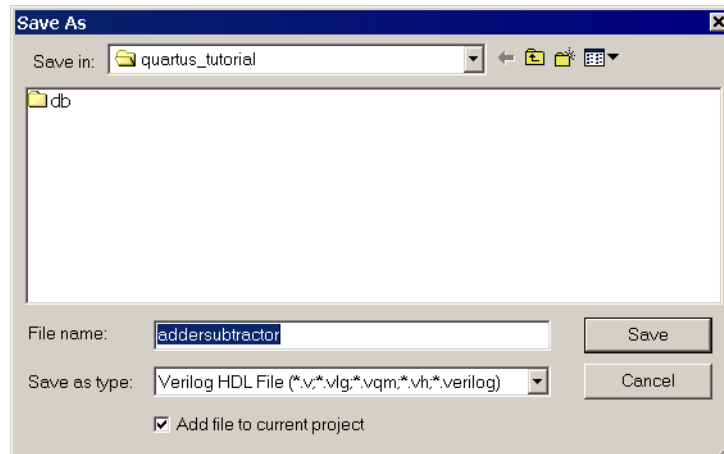


Figure 14. Name the file.

3.2 Adding Design Files to a Project

As we indicated when discussing Figure 6, you can tell Quartus II software which design files it should use as part of the current project. To see the list of files already included in the *addersubtractor* project, select **Assignments > Settings**, which leads to the window in Figure 16. As indicated on the left side of the figure, click on the item **Files**. An alternative way of making this selection is to choose **Project > Add/Remove Files in Project**.

If you used the Quartus II Text Editor to create the file and checked the box labeled **Add file to current project**, as described in Section 3.1, then the *addersubtractor.v* file is already a part of the project and will be listed in the window in Figure 16. Otherwise, the file must be added to the project. If not already done, place a copy of the file *addersubtractor.v* into the directory *quartus_tutorial*, by getting it from the directory *qdesigns* or by using a file that you created using some other text editor. To add this file to the project, click on the **File name: ...** button in Figure 16 to get the pop-up window in Figure 17. Select the *addersubtractor.v* file and click **Open**. The selected file is now indicated in the **Files** window of Figure 16. Click **OK** to include the *addersubtractor.v* file

in the project. We should mention that in many cases the Quartus II software is able to automatically find the right files to use for each entity referenced in Verilog code, even if the file has not been explicitly added to the project. However, for complex projects that involve many files it is a good design practice to specifically add the needed files to the project, as described above.



Figure 15. Text Editor window.

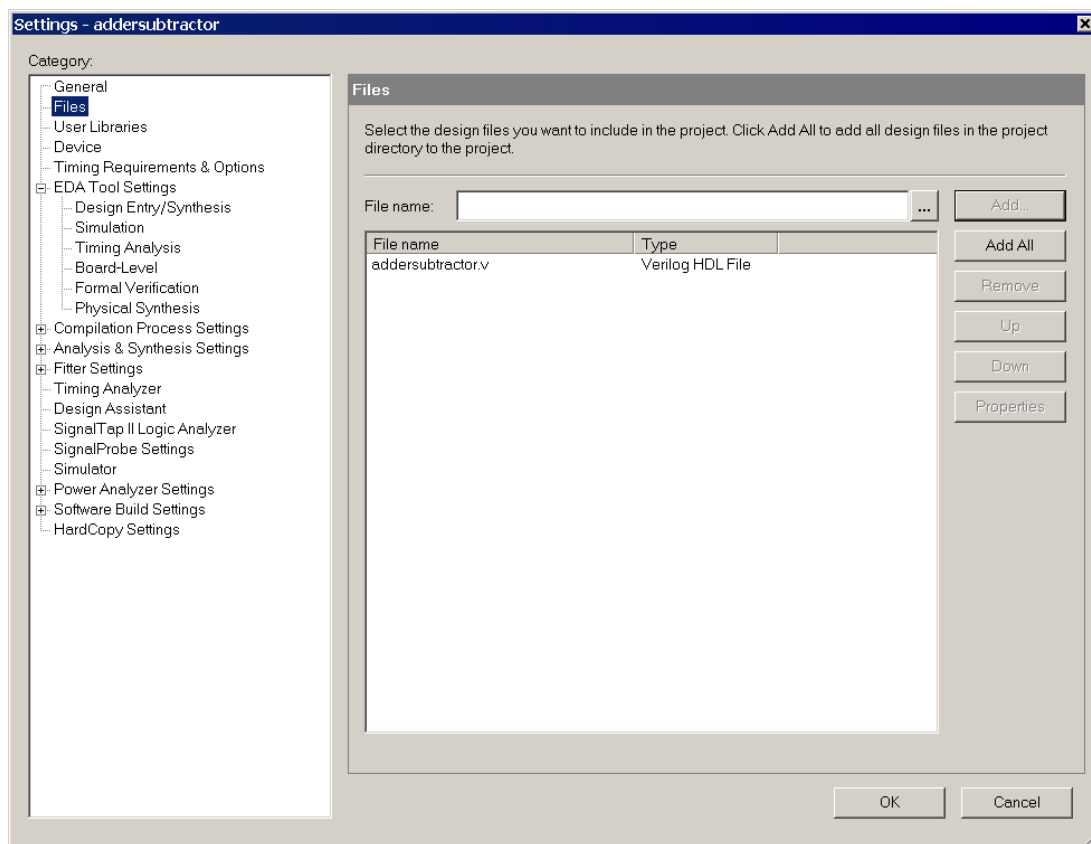


Figure 16. Settings window.

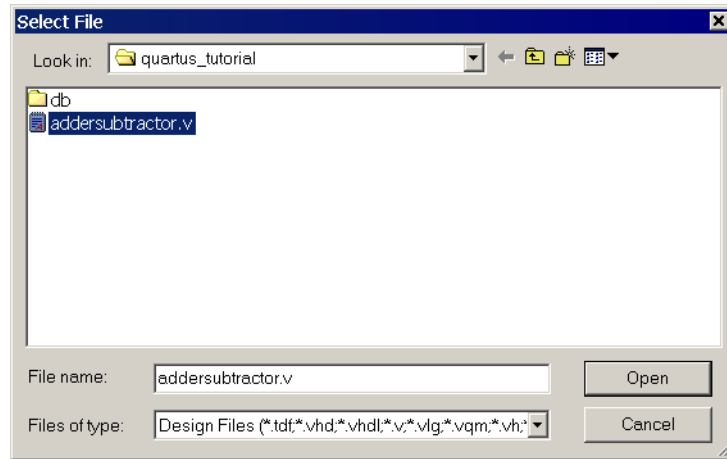


Figure 17. Select the file.

4 Compiling the Verilog Code

The Verilog code is processed by several Quartus II tools that analyze the code and generate an implementation of it for the target chip. These tools are controlled by the application program called the *Compiler*.

Run the Compiler by selecting **Processing > Start Compilation**, or by using the toolbar icon . As the compilation moves through various stages, its progress is reported in the window on the left side. Successful (or unsuccessful) compilation is indicated in a pop-up box. Acknowledge it by clicking OK, which leads to the Quartus II display in Figure 18, in which we have expanded the Entity hierarchy in the top left corner to show all modules in the *addersubtractor* design. In the message window, at the bottom of Figure 18, various messages are displayed. In case of errors, there will be appropriate messages given.

When the compilation is finished, a compilation report is produced. A window showing this report, displayed in Figure 19, is opened automatically. The window can be resized, maximized, or closed in the normal way, and it can be opened at any time either by selecting **Processing > Compilation Report** or by clicking on the icon . The report includes a number of sections listed on the left side of its window. Figure 19 displays the Compiler Flow Summary section, which indicates that only a miniscule amount of chip resources are needed to implement this tiny circuit on the selected FPGA chip.

The Compilation Report provides a lot of information that may be of interest to the designer. It indicates the speed of the implemented circuit. A good measure of the speed is the maximum frequency at which the circuit can be clocked, referred to as *f_{max}*. This measure depends on the longest delay along any path between two registers clocked by the same clock. Quartus II software performs a timing analysis to determine the expected performance of the circuit. It evaluates several parameters, which are listed in the Timing Analyzer section of the Compilation Report. Click on the small + symbol next to Timing Analyzer to expand this section of the report, as shown in Figure 19. Clicking on Timing Analyzer item **Summary** displays the table in Figure 20. The last entry in the table shows that the maximum frequency for our circuit implemented on the specified chip is 218.72 MHz. You may get a different value of *f_{max}*, dependent on the specific version of Quartus II software installed on your computer. To see the paths in the circuit that limit the *f_{max}*, click on the Timing Analyzer item **Clock Setup: 'Clock'** in Figure 20 to obtain the display in Figure 21. This table shows that the critical path begins at bit 8 of register *Zreg* and ends at the flip-flop *Overflow*.

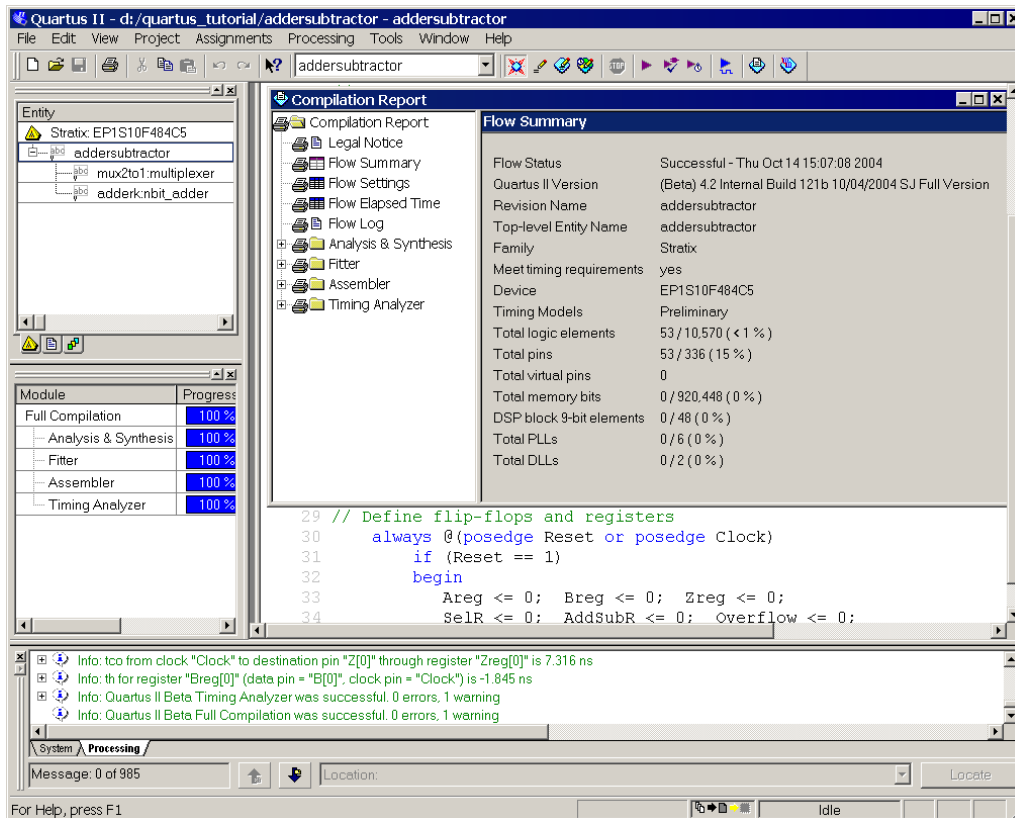


Figure 18. Display after a successful compilation.

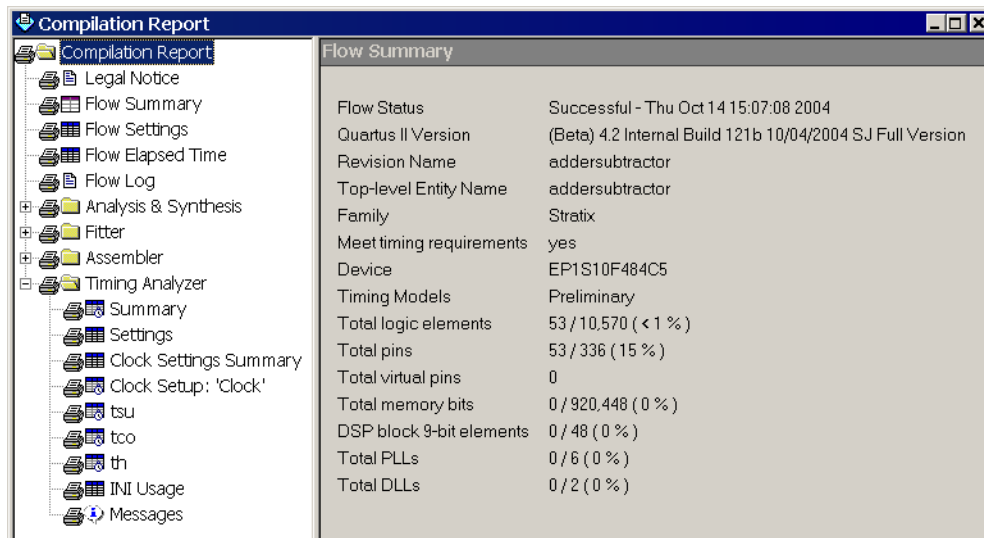


Figure 19. Compilation report.

Timing Analyzer Summary						
	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	2.779 ns	B[6]	Breg[6]
2	Worst-case tco	N/A	None	7.316 ns	Zreg[0]	Z[0]
3	Worst-case th	N/A	None	-1.845 ns	B[0]	Breg[0]
4	Clock Setup: 'Clock'	N/A	None	218.72 MHz (period = 4.572 ns)	Zreg[8]	Overflow~reg0
5	Total number of failed paths					

Figure 20. Summary of timing analysis.

Clock Setup: 'Clock'						
	Slack	Actual fmax (period)	From	To	From Clock	To Clock
1	N/A	218.72 MHz (period = 4.572 ns)	Zreg[8]	Overflow~reg0	Clock	Clock
2	N/A	230.79 MHz (period = 4.333 ns)	SelR	Overflow~reg0	Clock	Clock
3	N/A	237.08 MHz (period = 4.218 ns)	Zreg[10]	Overflow~reg0	Clock	Clock
4	N/A	242.07 MHz (period = 4.131 ns)	AddSubR	Overflow~reg0	Clock	Clock
5	N/A	246.55 MHz (period = 4.056 ns)	Zreg[8]	Zreg[13]	Clock	Clock
6	N/A	246.55 MHz (period = 4.056 ns)	Zreg[8]	Zreg[14]	Clock	Clock
7	N/A	246.55 MHz (period = 4.056 ns)	Zreg[8]	Zreg[15]	Clock	Clock
8	N/A	251.70 MHz (period = 3.973 ns)	Zreg[9]	Overflow~reg0	Clock	Clock
9	N/A	252.72 MHz (period = 3.957 ns)	Zreg[7]	Overflow~reg0	Clock	Clock
10	N/A	259.27 MHz (period = 3.857 ns)	Zreg[8]	Zreg[12]	Clock	Clock
11	N/A	261.99 MHz (period = 3.817 ns)	SelR	Zreg[13]	Clock	Clock
12	N/A	261.99 MHz (period = 3.817 ns)	SelR	Zreg[14]	Clock	Clock
13	N/A	261.99 MHz (period = 3.817 ns)	SelR	Zreg[15]	Clock	Clock
14	N/A	263.37 MHz (period = 3.797 ns)	Zreg[8]	Zreg[11]	Clock	Clock
15	N/A	267.45 MHz (period = 3.739 ns)	SelR	Zreg[8]	Clock	Clock
16	N/A	267.45 MHz (period = 3.739 ns)	SelR	Zreg[9]	Clock	Clock
17	N/A	267.45 MHz (period = 3.739 ns)	SelR	Zreg[10]	Clock	Clock

Figure 21. Critical paths.

The table in Figure 20 also shows the measurements of other timing parameters. While f_{max} is a function of the longest propagation delay between two registers in the circuit, it does not indicate the delays with which output signals appear at the pins of the chip. The time elapsed from an active edge of the clock signal at the clock source until a corresponding output signal is produced (from a flip-flop) at an output pin is denoted as the t_{co} parameter at that pin. In the worst case, the t_{co} in our circuit is 7.316 ns. Click on t_{co} in the Timing Analyzer section to view the table given in Figure 22. The first entry in the table shows that it takes 7.316 ns for a signal to propagate from bit 0 in register $Zreg$ to the output pin z_0 . The other two parameters given in Figure 20 are setup time, tsu , and hold time, th .

tco							
	Slack	Required tco	Actual tco	From	To	From Clock	
1	N/A	None	7.316 ns	Zreg[0]	Z[0]	Clock	
2	N/A	None	7.181 ns	Zreg[10]	Z[10]	Clock	
3	N/A	None	7.161 ns	Zreg[15]	Z[15]	Clock	
4	N/A	None	7.156 ns	Zreg[12]	Z[12]	Clock	
5	N/A	None	7.138 ns	Zreg[7]	Z[7]	Clock	
6	N/A	None	7.077 ns	Zreg[2]	Z[2]	Clock	
7	N/A	None	6.908 ns	Zreg[8]	Z[8]	Clock	
8	N/A	None	6.902 ns	Zreg[11]	Z[11]	Clock	
9	N/A	None	6.889 ns	Overflow~reg0	Overflow	Clock	
10	N/A	None	6.889 ns	Zreg[9]	Z[9]	Clock	
11	N/A	None	6.888 ns	Zreg[14]	Z[14]	Clock	
12	N/A	None	6.867 ns	Zreg[1]	Z[1]	Clock	
13	N/A	None	6.866 ns	Zreg[3]	Z[3]	Clock	
14	N/A	None	6.859 ns	Zreg[5]	Z[5]	Clock	
15	N/A	None	6.827 ns	Zreg[4]	Z[4]	Clock	
16	N/A	None	6.627 ns	Zreg[6]	Z[6]	Clock	
17	N/A	None	6.580 ns	Zreg[13]	Z[13]	Clock	

Figure 22. The *tco* delays.

An indication of where the circuit is implemented on the chip is available by selecting **Assignments > Timing Closure Floorplan**, or by clicking the icon . Figure 23 depicts the result, highlighting in color the logic elements used to implement the circuit. To make the image appear as shown you may have to select **View > Field View**, so that the tool does not show the details of the chip resources. You also may need to select **View > Assignments > Show Fitter Placements** or click the corresponding icon in the toolbar (which is likely to be already selected by default). The floorplan view can be enlarged by maximizing the window and selecting **View > Fit in Window** (shortcut Ctrl-w), and it can be expanded to fill the screen by clicking the Full Screen icon .

A Zoom Tool, activated by the icon , can be used to enlarge parts of the image even more. Figure 24 shows a zoomed-in view of the floorplan that highlights the implemented circuit. To see the details given in the figure you have to select the Floorplan Editor command **View > Interior Cells**. By positioning the cursor on any logic element the designer can see what part of the circuit is implemented in this resource. The floorplan tool has several icons that can be used to view aspects such as fanin and fanout of nodes, connecting paths between nodes, and so on. For more information on using this tool refer to the online help, by selecting **Help > Contents > Viewing the Fit**.

The detailed implementation of the circuit in the form of logic equations is also included in the compilation report. It can be viewed by selecting **Fitter > Fitter Equations**. These equations do not necessarily correspond directly to any logic expressions that may have been given in the Verilog design file, because the synthesized circuit is implemented on the FPGA chip in logic elements that constitute lookup (truth) tables (LUTs).

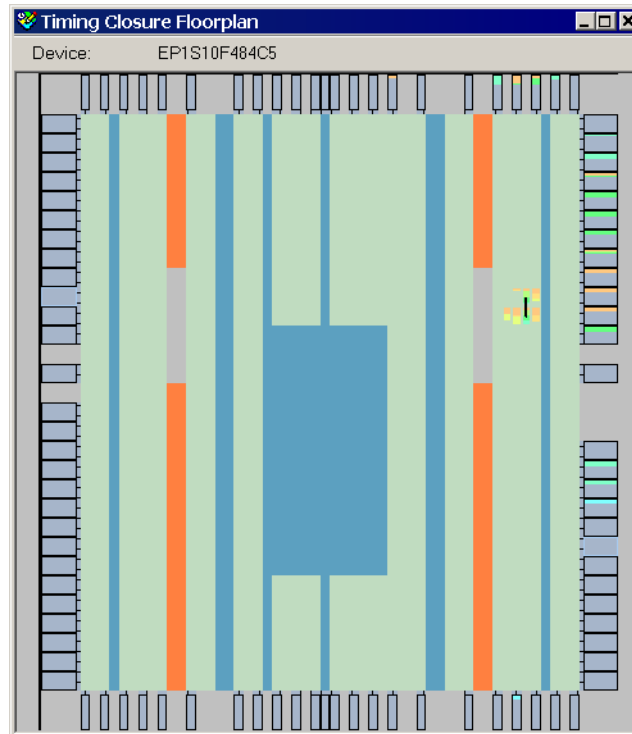


Figure 23. View of the floorplan.

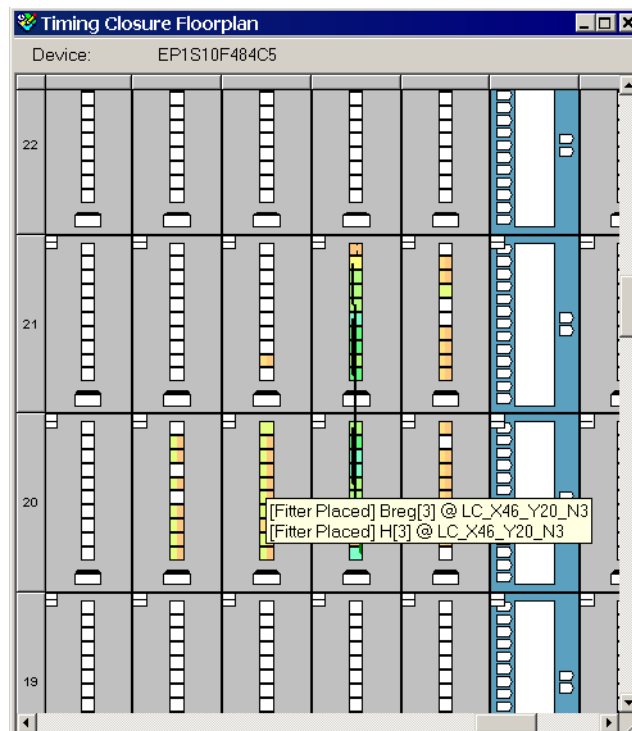


Figure 24. A portion of the expanded view.

4.1 Errors

Quartus II software displays messages produced during compilation in the Messages window. If the Verilog design file is correct, one of the messages will state that the compilation was successful and that there are no errors.

If the Compiler does not report zero errors, then there is at least one mistake in the Verilog code. In this case a message corresponding to each error found will be displayed in the Messages window. Double-clicking on an error message will highlight the offending statement in the Verilog code in the Text Editor window. Similarly, the Compiler may display some warning messages. Their details can be explored in the same way as in the case of error messages. The user can obtain more information about a specific error or warning message by selecting the message and pressing the F1 function key.

To see the effect of an error, open the file *addersubtractor.v*. Line 21 has the statement

```
assign H = Breg ^ {n{AddSubR}};
```

Remove the semicolon in this statement, illustrating a typographical error that is easily made. Compile the erroneous design file. Quartus II software displays a pop-up box indicating that the compilation was not successful. Acknowledge it by clicking OK. The compilation report summary, given in Figure 25, now confirms the failed result. Click on Analysis & Synthesis > Messages in this window to have all messages displayed as shown in Figure 26. Double-click on the first error message, which states that there is a Verilog syntax error. Quartus II software responds by opening the *addersubtractor.v* file and highlighting the statement affected by the error, as shown in Figure 27. Correct the error and recompile the design.

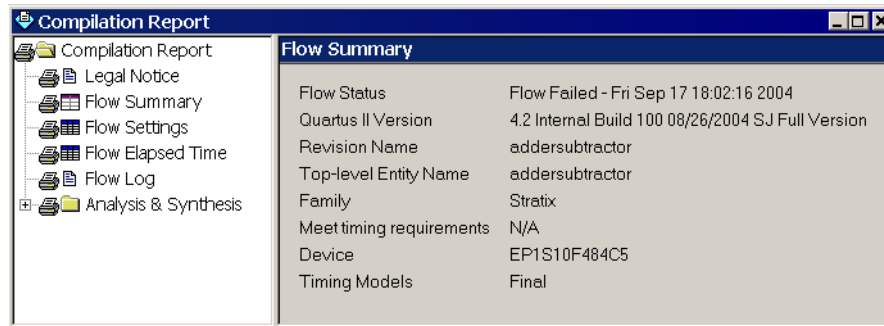


Figure 25. Compilation report for the failed design.

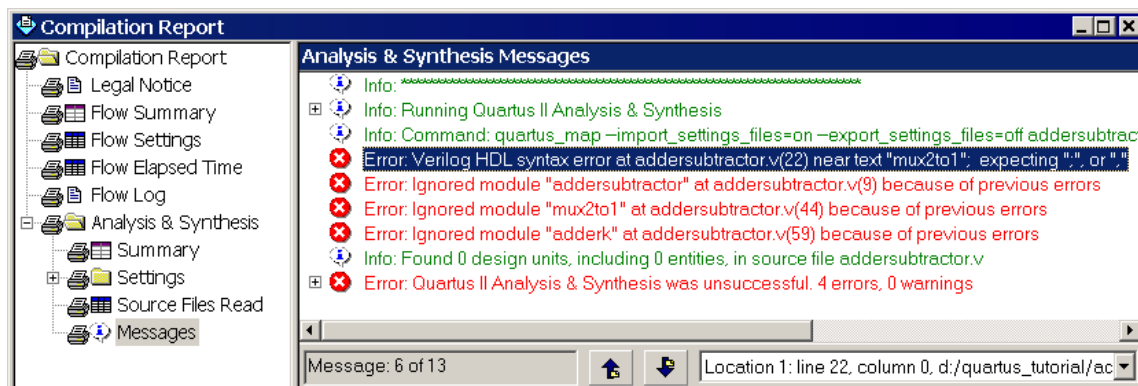


Figure 26. Error messages.

```

addersubtractor.v
16  reg [n-1:0] Areg, Breg, Zreg;
17  wire [n-1:0] G, H, M, Z;
18  wire carryout, over_flow;
19
20  // Define combinational logic circuit
21  assign H = Breg ^ {n{AddSubR}}
22  mux2to1 multiplexer (Areg, Z, SelR, G);
23  defparam multiplexer.k = n;
24  adderk nbit_adder (AddSubR, G, H, M, carryout);
25  defparam nbit_adder.k = n;
26  assign over_flow = carryout ^ G[n-1] ^ H[n-1] ^ M[n-1];
27  assign Z = Zreg;
28

```

Figure 27. Identifying the location of the error.

5 Using the RTL Viewer

Quartus II software includes a tool that can display a schematic diagram of the designed circuit. The display is at the Register Transfer Level of detail, and the tool is called the *RTL Viewer*. Click Tools > RTL Viewer, to reach the window shown in Figure 28.

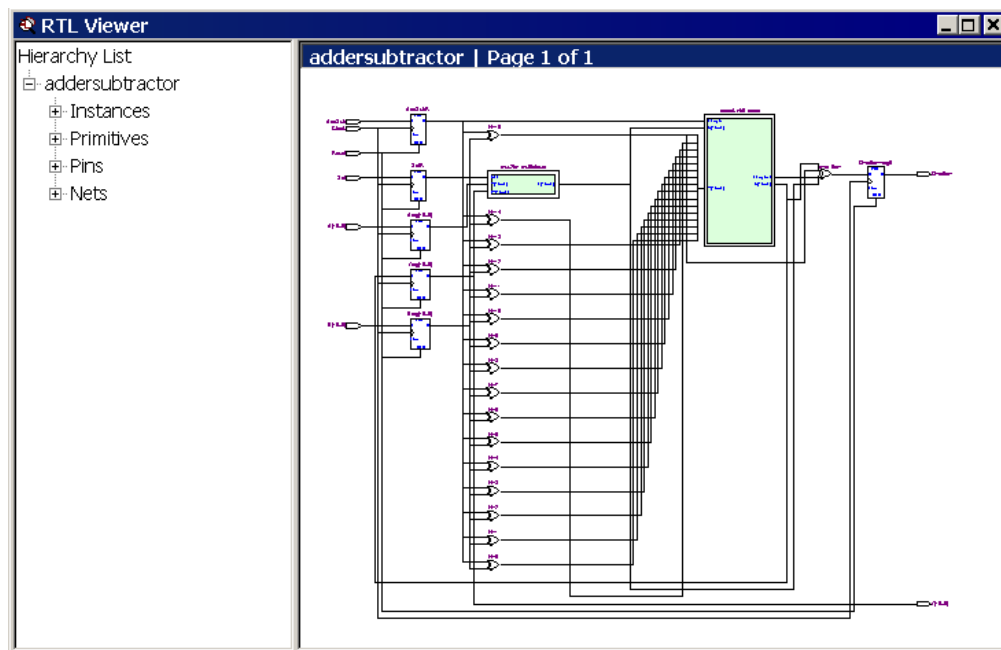


Figure 28. The *addersubtractor* circuit displayed by the RTL Viewer.

The displayed image depicts the structure of the entire *addersubtractor* circuit. The inputs to the circuit, shown on the left side, are registered. The two subcircuits, defined by the *mux2to1* and *adderk* modules, are drawn as shaded boxes. The remainder of the circuit are the XOR gates used to complement the *B* vector when subtraction is performed, and the circuitry needed to generate the *Overflow* signal.

Use the Zoom Tool to enlarge the image and view the upper-left portion of the circuit, as illustrated in Figure 29. Note that individual flip-flops are used for the *AddSub* and *Sel* signals. Sixteen-bit vectors *A* and *B* are denoted by heavy lines connected to the registers, *Areg* and *Breg*, which are indicated as heavily outlined flip-flop symbols. The *Zreg* register is drawn in the same manner.

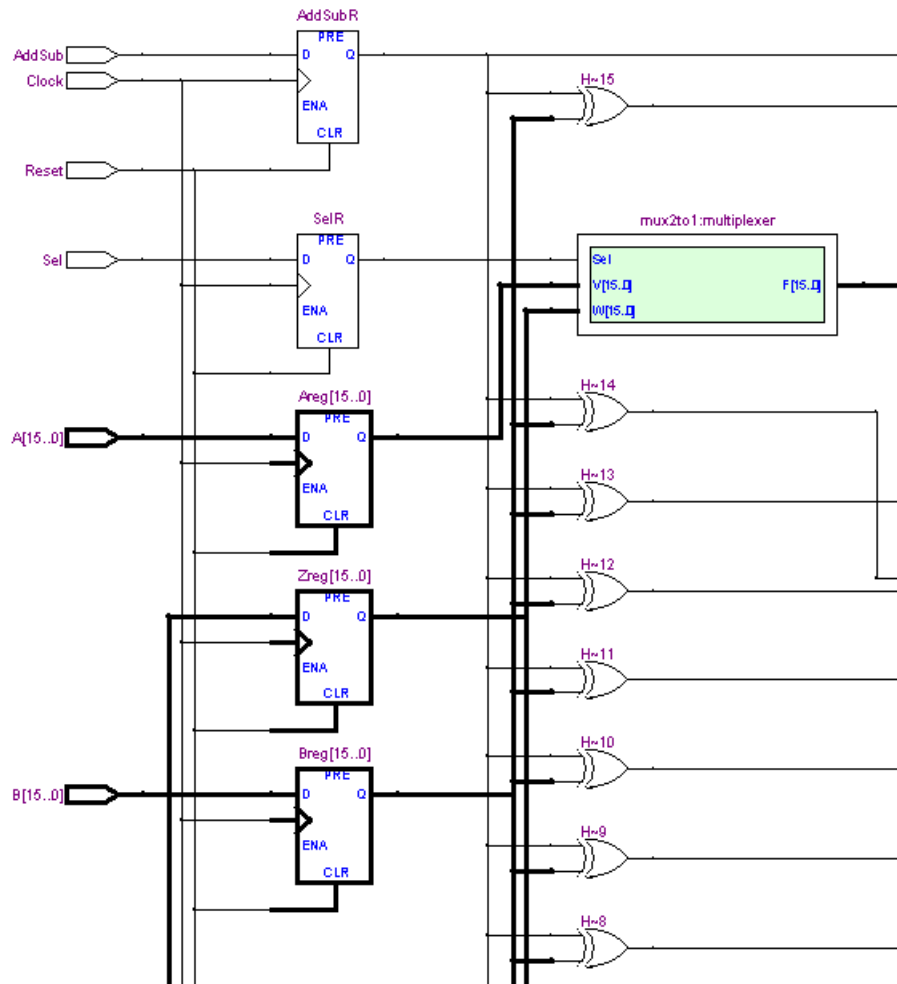


Figure 29. An enlarged view of the circuit.

Details of subcircuits can be seen by clicking on the box that represents a subcircuit. Double click on the *mux2to1* box to obtain the image in Figure 30. It shows the multiplexers used to choose either the *Areg* or *Z* vector as one of the inputs to the adder, under control of the *SelR* signal. Observe that the multiplexer data inputs are labeled as specified in the Verilog code for the *mux2to1* module in part *b* of Figure 12, namely as *V* and *W* rather than *Areg* and *Z*.

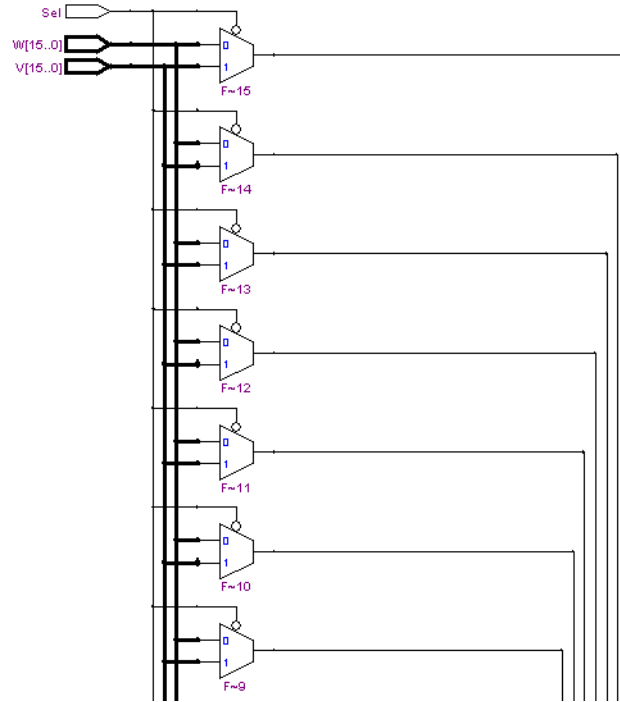


Figure 30. The multiplexer subcircuit.

The RTL viewer is a useful tool. It can be used effectively to facilitate the development of Verilog code for a circuit that is being designed. It provides a pictorial feedback to the designer, which gives an indication of the structure of the circuit that the code will produce. Viewing the pictures makes it easy to spot missing elements, wrong connections, and other typical errors that one makes early in the design process.

6 Specifying Timing Constraints

Quartus II software allows the user to specify timing constraints for the designed circuit. Click **Assignments > Timing Settings** to reach the window in Figure 31. Here, it is possible to indicate the required values of various design parameters and the desired performance of the circuit. Consider the *fmax* indicator. So far, we have not specified the desired value of *fmax*. The compilation in Section 4 produced the *fmax* of 218.72 MHz. Suppose that we need a circuit that can operate at a clock frequency of 240 MHz. Specify this value as the default required *fmax* as shown in Figure 31, and click OK. Note that other parameters, such as t_{su} , t_{co} , and t_h can be specified in this window.

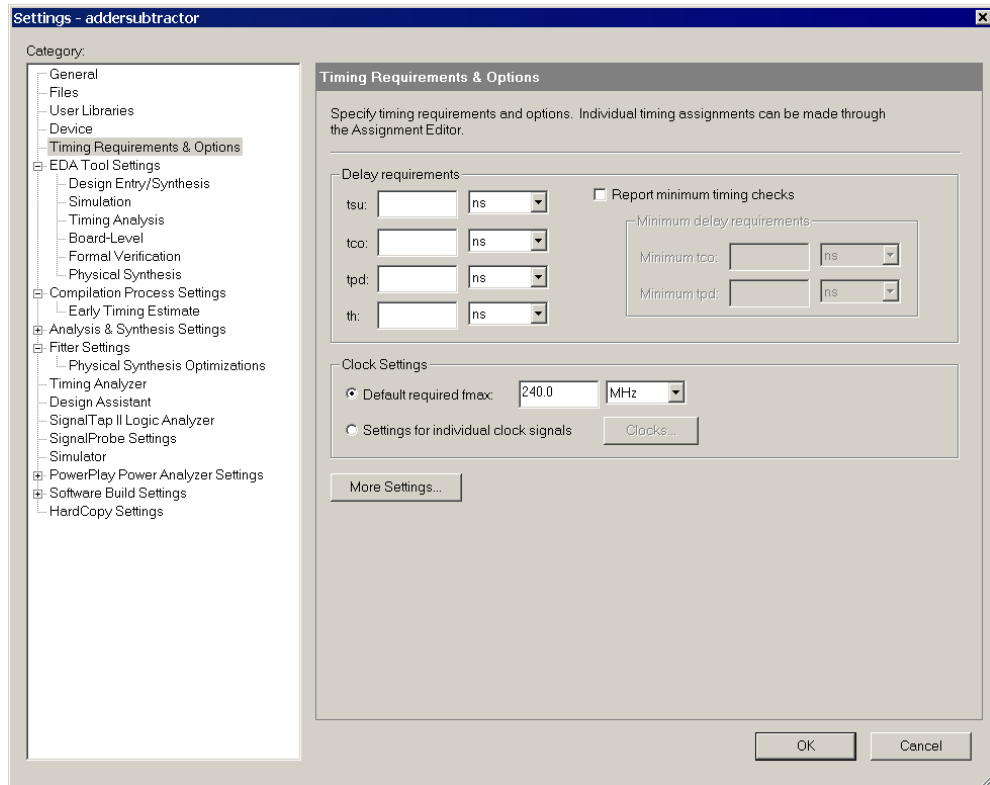


Figure 31. Setting the timing constraints.

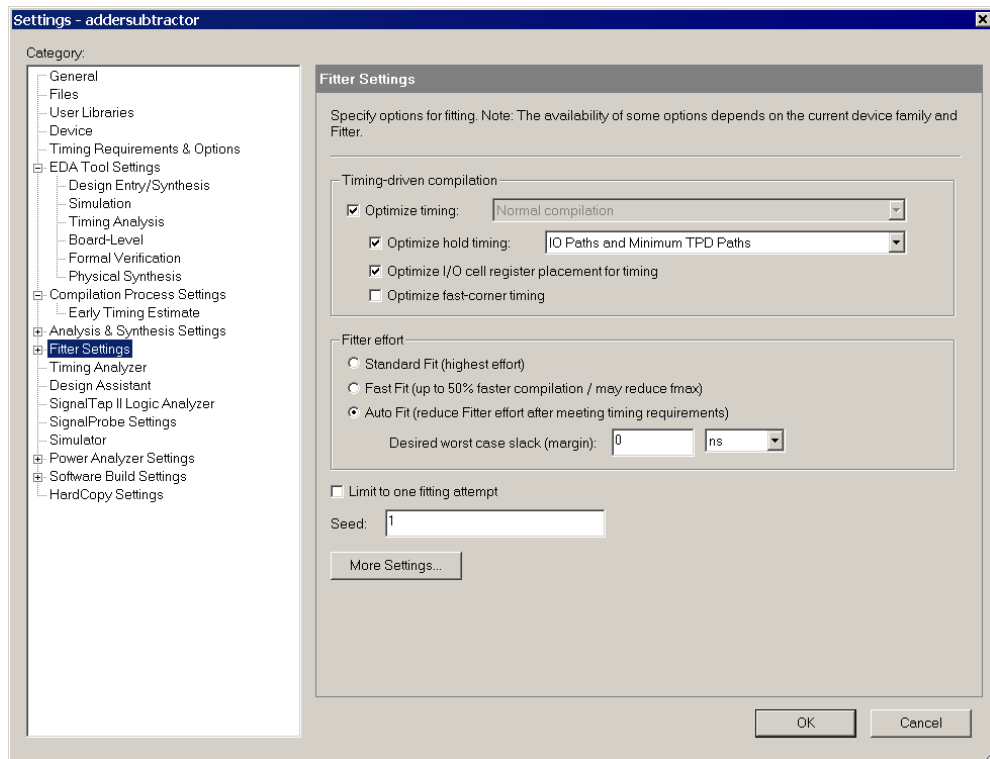


Figure 32. Fitter settings.

The Quartus II Compiler includes a Fitter executable that places the designed circuit into the available logic elements on the chip and generates the necessary wiring connections to realize the circuit. This is a complex process that can take a long time, particularly if the circuit is large and an ambitious value of f_{max} is specified. The time can be reduced if a lower value of f_{max} is acceptable. The user can indicate the level of the Fitter's effort. Click **Assignments > Settings** and then select the category **Fitter Settings** which opens the window in Figure 32. Three different levels of effort can be given. Choose the **Auto Fit** option, which instructs the Fitter to stop as soon as it finds an adequate implementation. The **Fast Fit** option reduces the compilation time, but it may produce a lower f_{max} . The third option, called **Standard Fit**, forces the Fitter to produce the best implementation it can find; at this effort level the Fitter will exceed the user's timing requirements as much as it can, which often results in longer compilation time. Click **OK**, and recompile the circuit.

The new timing results are shown in Figure 33. The new f_{max} is 246.97 MHz, which meets the specified requirement. The critical paths in this implementation are given in Figure 34. Comparing these results with those in Figure 21, we see that now the most critical path begins at bit 3 of register *Z* and ends at the flip-flop *Overflow*. The first column in the figure shows the *slack* for each path, which is the amount of delay that could still be added to a given path without violating the specified timing constraint.

Timing Analyzer Summary						
	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	3.200 ns	B[1]	Breg[1]
2	Worst-case tco	N/A	None	7.818 ns	Zreg[2]	Z[2]
3	Worst-case th	N/A	None	-1.946 ns	B[14]	Breg[14]
4	Clock Setup: 'Clock'	0.117 ns	240.04 MHz (period = 4.166 ns)	246.97 MHz (period = 4.049 ns)	Zreg[3]	Overflow~reg
5	Clock Hold: 'Clock'	1.044 ns	240.04 MHz (period = 4.166 ns)	N/A	Areg[0]	Zreg[0]
6	Total number of failed paths					

Figure 33. New timing results.

Clock Setup: 'Clock'						
	Slack	Actual fmax (period)	From	To	From Clock	To Clock
1	0.117 ns	246.97 MHz (period = 4.049 ns)	Zreg[3]	Overflow~reg0	Clock	Clock
2	0.365 ns	263.09 MHz (period = 3.801 ns)	AddSubR	Overflow~reg0	Clock	Clock
3	0.389 ns	264.76 MHz (period = 3.777 ns)	SelR	Overflow~reg0	Clock	Clock
4	0.633 ns	283.05 MHz (period = 3.533 ns)	Zreg[3]	Zreg[15]	Clock	Clock
5	0.633 ns	283.05 MHz (period = 3.533 ns)	Zreg[3]	Zreg[14]	Clock	Clock
6	0.633 ns	283.05 MHz (period = 3.533 ns)	Zreg[3]	Zreg[13]	Clock	Clock
7	0.669 ns	285.96 MHz (period = 3.497 ns)	Zreg[5]	Overflow~reg0	Clock	Clock
8	0.694 ns	288.02 MHz (period = 3.472 ns)	Breg[6]	Overflow~reg0	Clock	Clock
9	0.711 ns	289.44 MHz (period = 3.455 ns)	Zreg[3]	Zreg[12]	Clock	Clock
10	0.711 ns	289.44 MHz (period = 3.455 ns)	Zreg[3]	Zreg[11]	Clock	Clock
11	0.711 ns	289.44 MHz (period = 3.455 ns)	Zreg[3]	Zreg[10]	Clock	Clock
12	0.711 ns	289.44 MHz (period = 3.455 ns)	Zreg[3]	Zreg[9]	Clock	Clock

Figure 34. New critical paths.

7 Quartus II Windows

The Quartus II display contains several utility windows, which can be positioned in various places on the screen, changed in size, or closed. In Figure 18, which is reproduced in Figure 35, there are four windows.

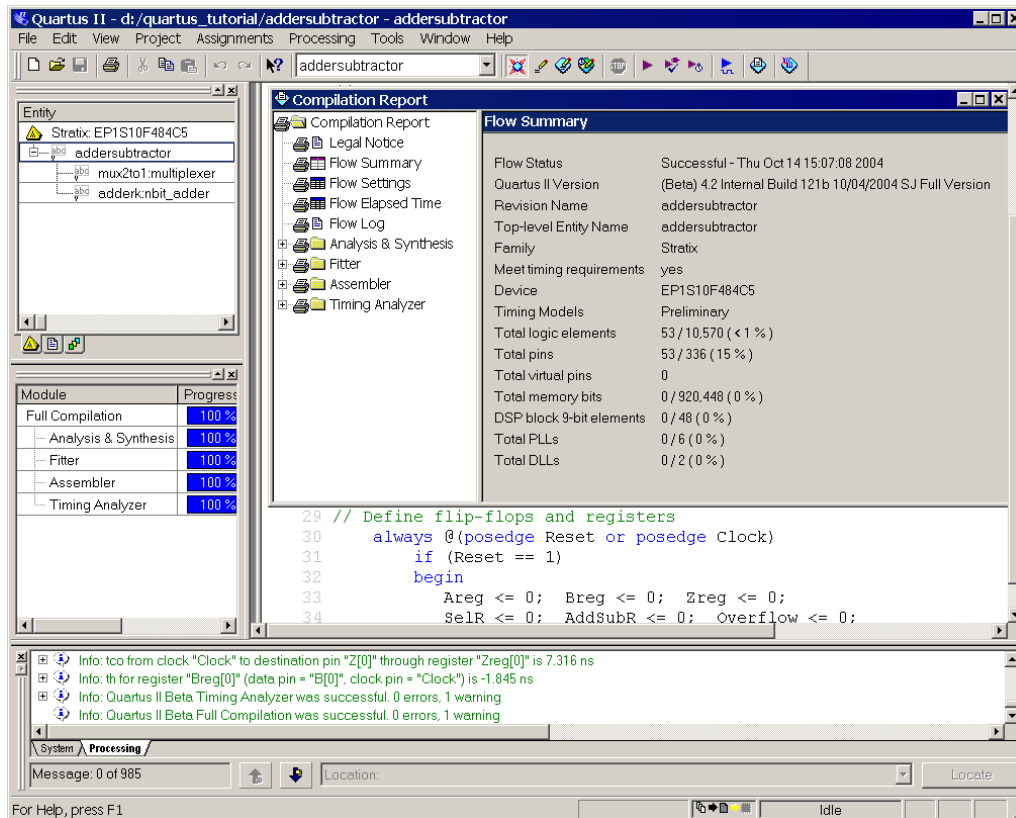


Figure 35. The main Quartus II display.

The Project Navigator window is shown near the top left of the figure. Under the heading Entity, it depicts a tree-like structure of the designed circuit using the names of the modules in the Verilog code of Figure 12. To see the usefulness of this window, open the previously compiled project *quartus tutorial\addersubtractor* to get to the window that corresponds to Figure 35. Now, double-click on the name *adder_k* in the hierarchy. Quartus II software will open the file *addersubtractor.v* and highlight the Verilog module that specifies the adder subcircuit. Next, right-click on the same name and choose *Locate > Locate in Timing Closure Floorplan* from the pop-up menu that appears. This causes Quartus II software to display the floorplan, as in Figure 24, and highlight the part that implements the adder subcircuit.

The Status window is located below the Project Navigator window in Figure 35. As you have already observed, this window displays the compilation progress as a project is being compiled.

At the bottom of Figure 35 there is the Message window, which displays user messages produced during the compilation process.

The large area on the right side of the Quartus II display is used for various purposes. As we have seen, it is used by the Text Editor. It is also used to display various results of compilation and simulation.

A utility window can be moved by dragging its title bar, resized by dragging the window border, or closed by clicking on the X in the top-right corner. A particular utility window can be opened by using the *View > Utility Windows* command.

Copyright ©2004 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.