Quartus® II Introduction for VHDL Users

This tutorial presents an introduction to the Quartus® II software. It gives a general overview of a typical CAD flow for designing circuits that are implemented by using FPGA devices, and shows how this flow is realized in the Quartus® II software. The design process is illustrated by giving step-by-step instructions for using the Quartus® II software to implement a simple circuit in an Altera® FPGA device.

The Quartus® II system includes full support for all of the popular methods of entering a description of the desired circuit into a CAD system. This tutorial makes use of the VHDL design entry method, in which the user specifies the desired circuit in the VHDL hardware description language. Another version of this tutorial is available that uses Verilog hardware description language.

Contents:

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.

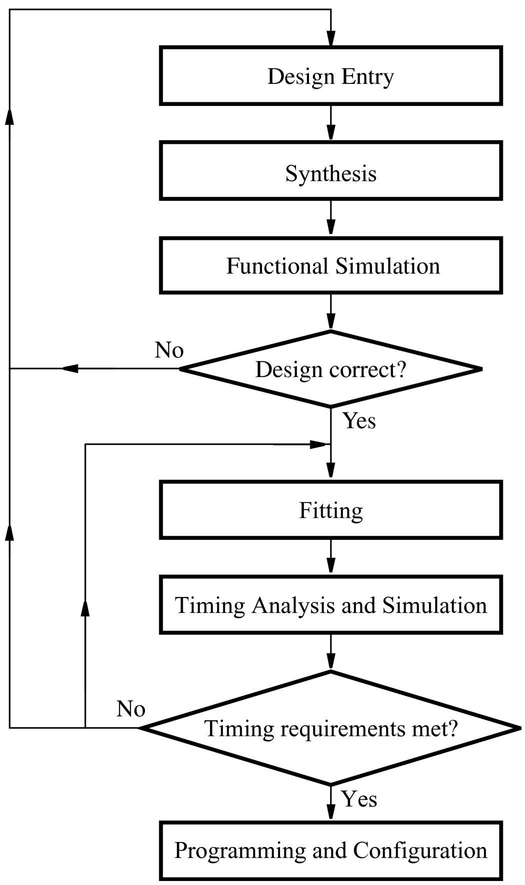


Figure 1: Typical CAD flow.

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 VHDL hardware description language. It makes use of the graphical user interface to invoke the Quartus® II commands. During this tutorial, the reader will learn about:

- Creating a project
- Synthesizing a circuit from VHDL 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 the 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. To hold the design files for this tutorial, we will use a directory called *quartus_tutorial*. The running example for this tutorial is a simple adder/subtractor circuit, which is defined in the VHDL 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 the Quartus® II software, which the user selects with the computer mouse. Most of the commands provided by the 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 the 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.... Once a toolbar is opened, it can be moved using the mouse. 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.

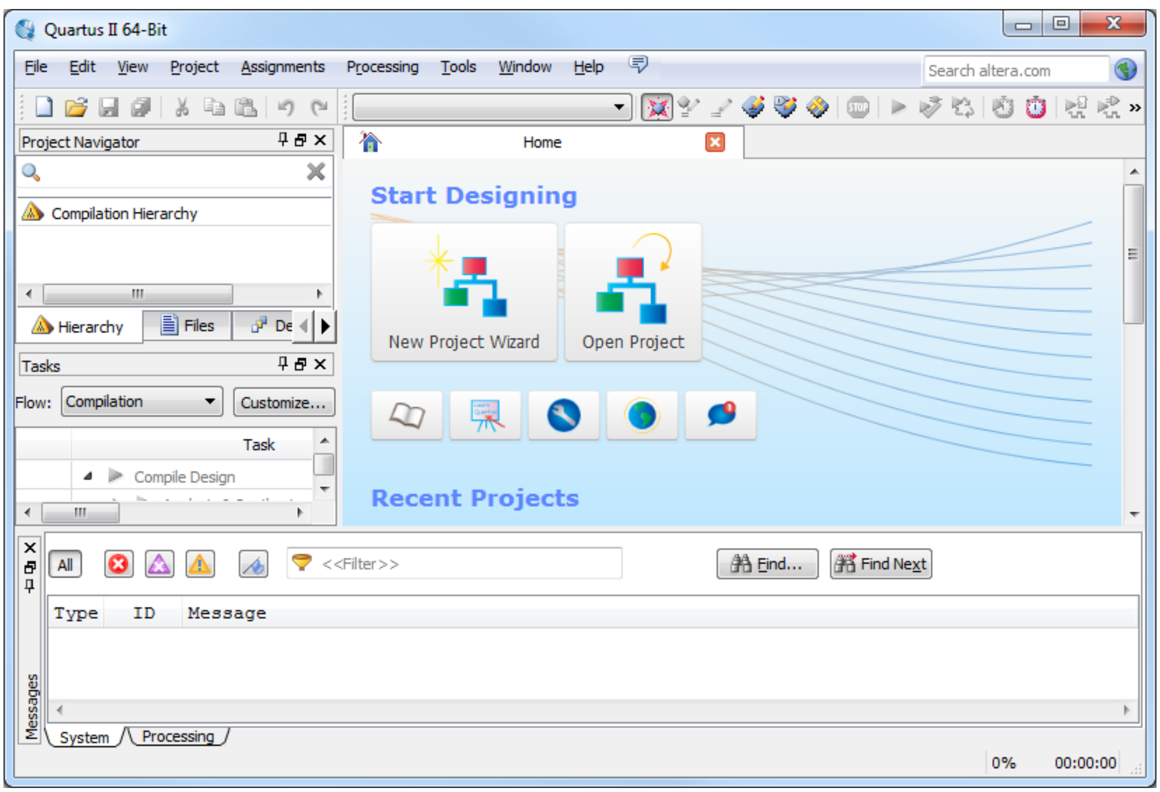


Figure 2: The main Quartus® II display.

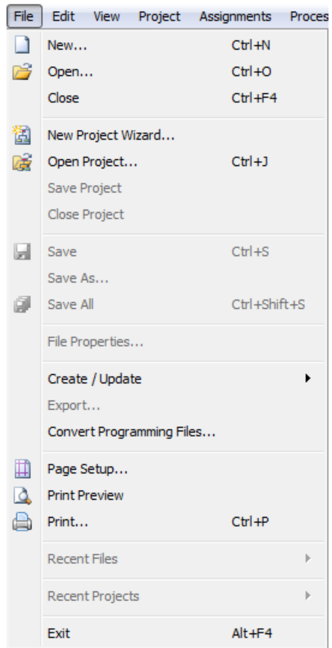


Figure 3: An example of the File menu.

1.1 Quartus® II Online Help

The 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.

The user can quickly search through the Help topics by selecting Help > Search, which opens a web-based interface with a dialog box into which keywords can be entered. Another method, context-sensitive help, is provided for quickly finding documentation about 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*. The Quartus® II software makes the designer's task easy by providing support in the form of a *wizard*.

1. Select File > New Project Wizard to reach a window that indicates the capability of this wizard. Press Next. This will bring up the wizard screen as shown in Figure 4.

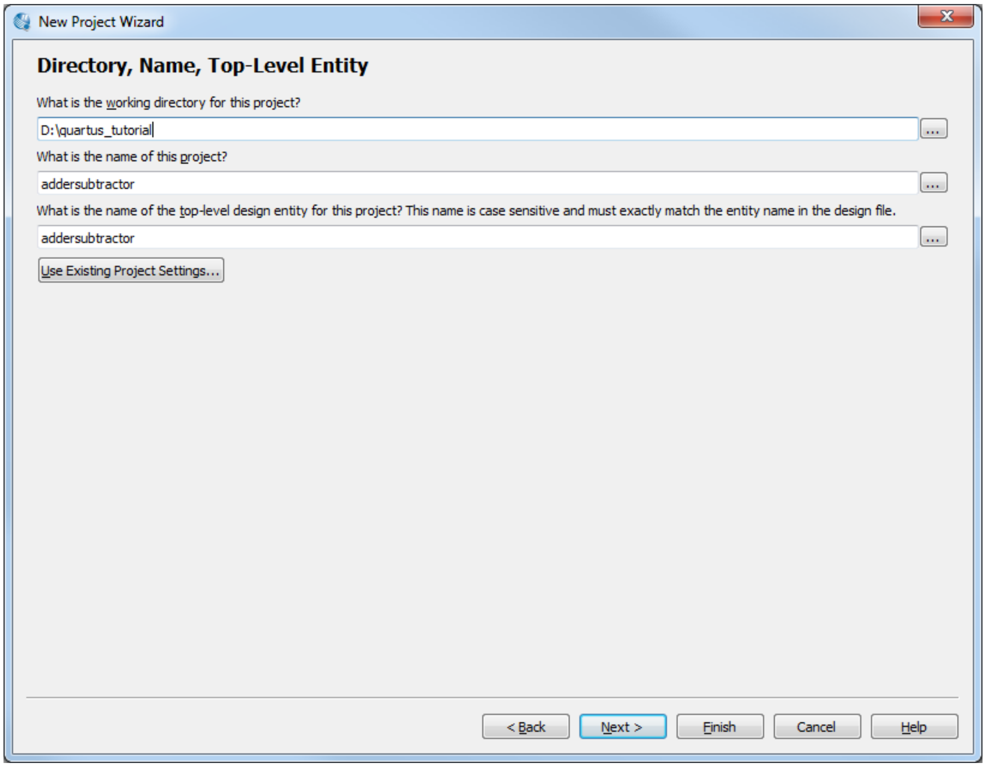


Figure 4: Creation of a new project.

2. 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*, the 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.

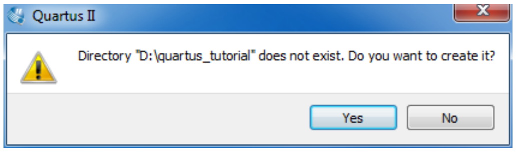


Figure 5: The Quartus® II software can create a new directory for the project.

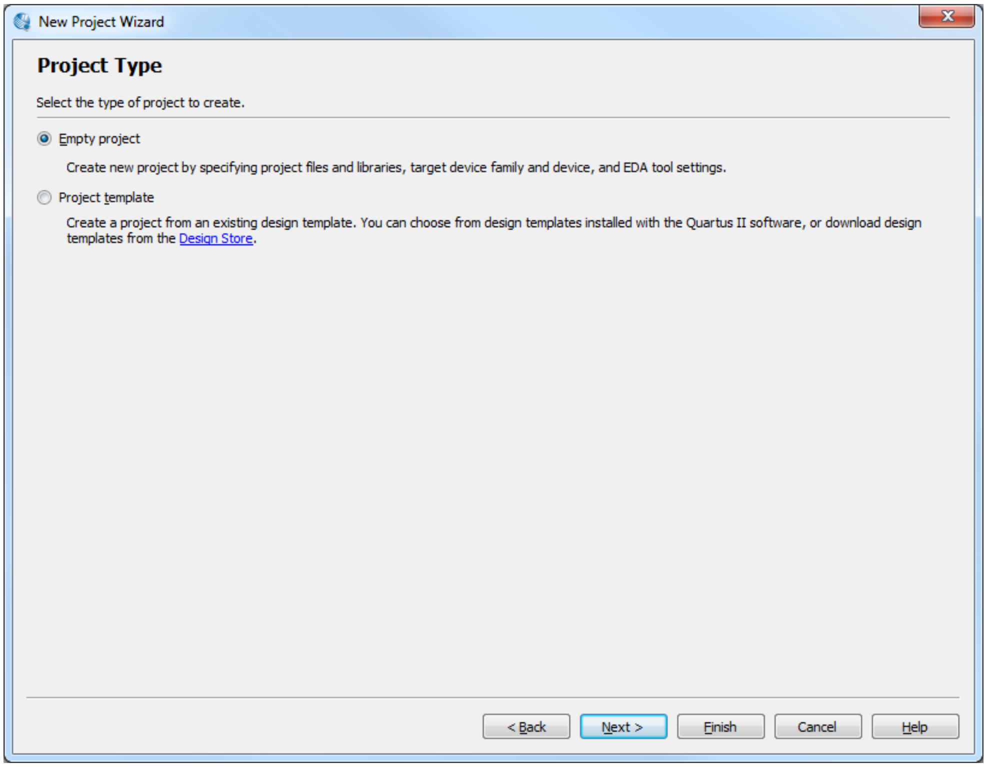


Figure 6: Choosing the project type.

3. The Project Type window, shown in Figure 6, allows you to choose from the Empty project and the Project template options. For this tutorial, choose Empty project as we will be creating a project from scratch, and press Next which leads to the window in Figure 7.

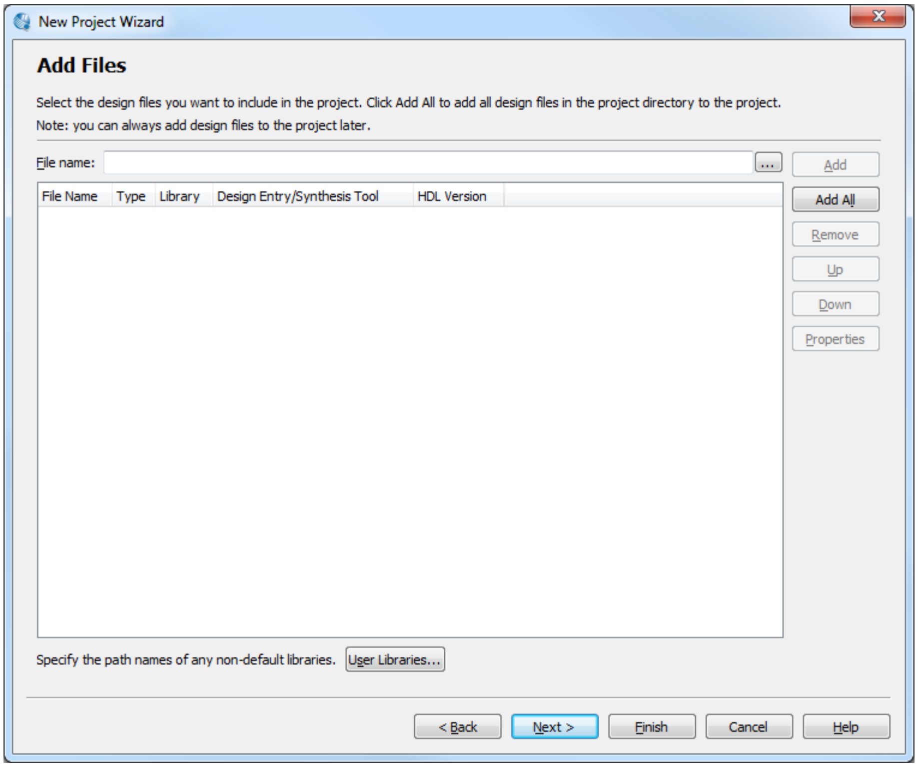


Figure 7: The wizard can include user-specified design files.

4. 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 8.

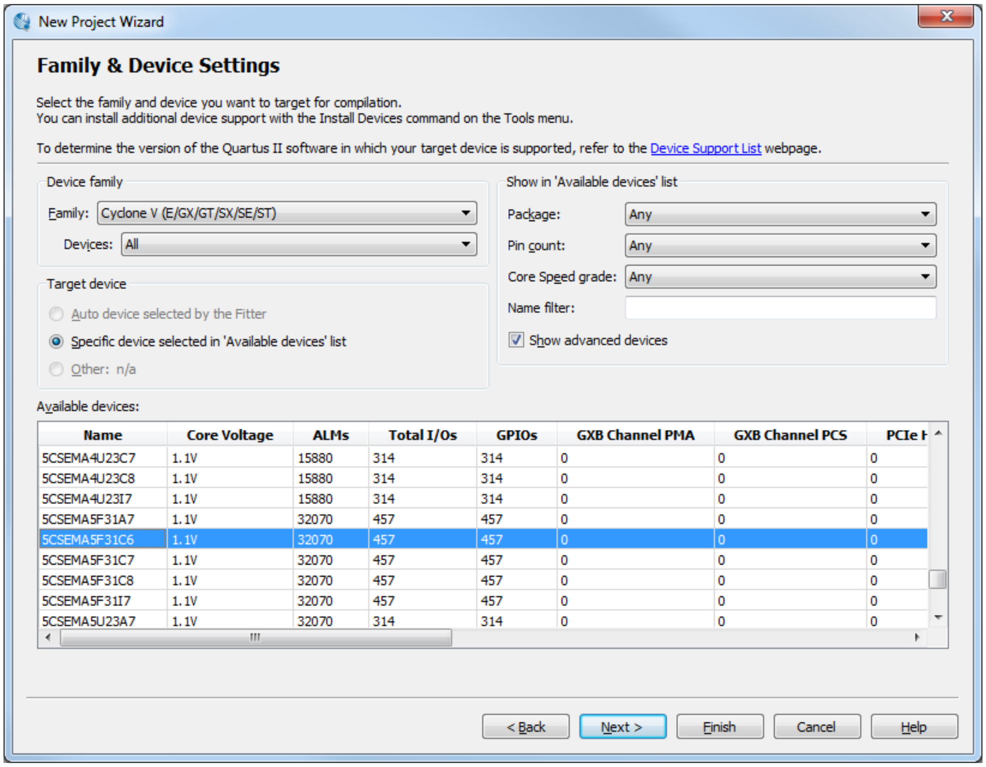


Figure 8: Choose the device family and a specific device.

5. In this window, we can specify the type of device in which the designed circuit will be implemented. Choose the Cyclone V®(E/GX/GT/SX/SE/ST) menu item as the target device family. We can let the 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 5CSEMA5F31C6. Press Next, which opens the window in Figure 9.

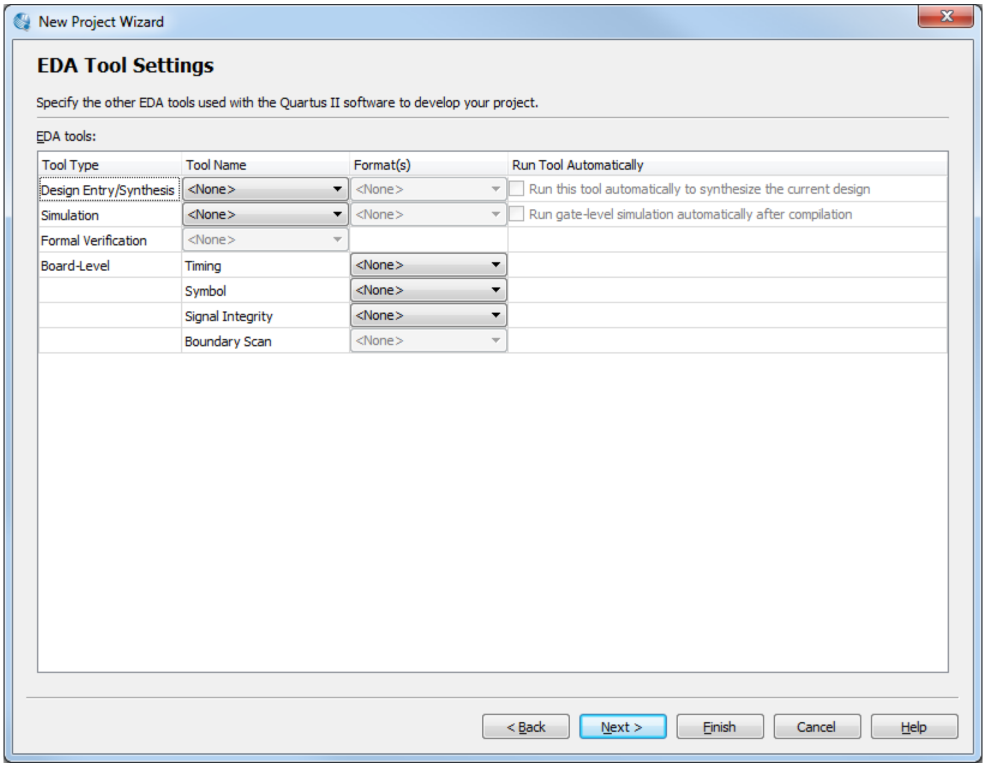


Figure 9: Other EDA tools can be specified.

6. In this window we 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 the 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 the 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 10. Press Finish, which returns to the main Quartus® II display. Note that *addersubtractor* is now specified as the current project, as indicated in the title bar at the top of the display. The screen should look similar to that of Figure 11.

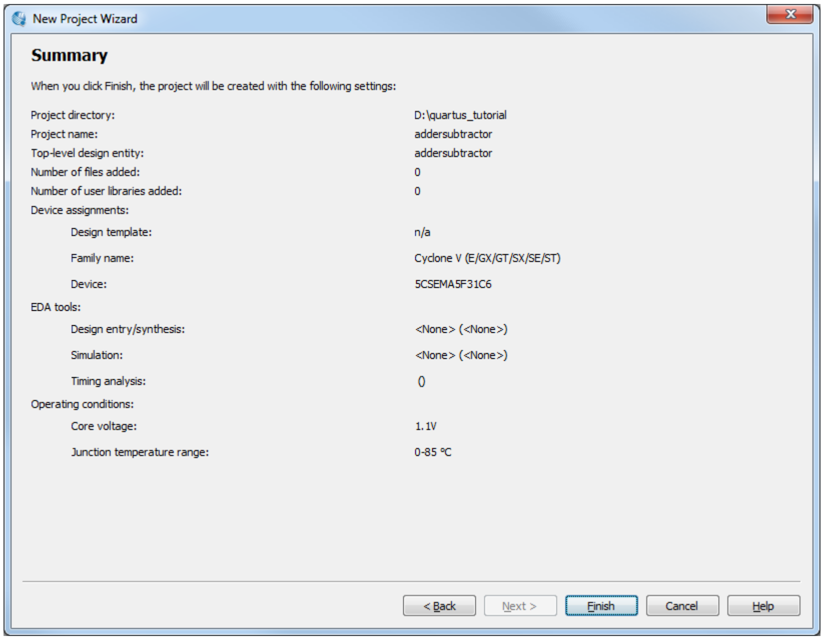


Figure 10: Summary of the project settings.

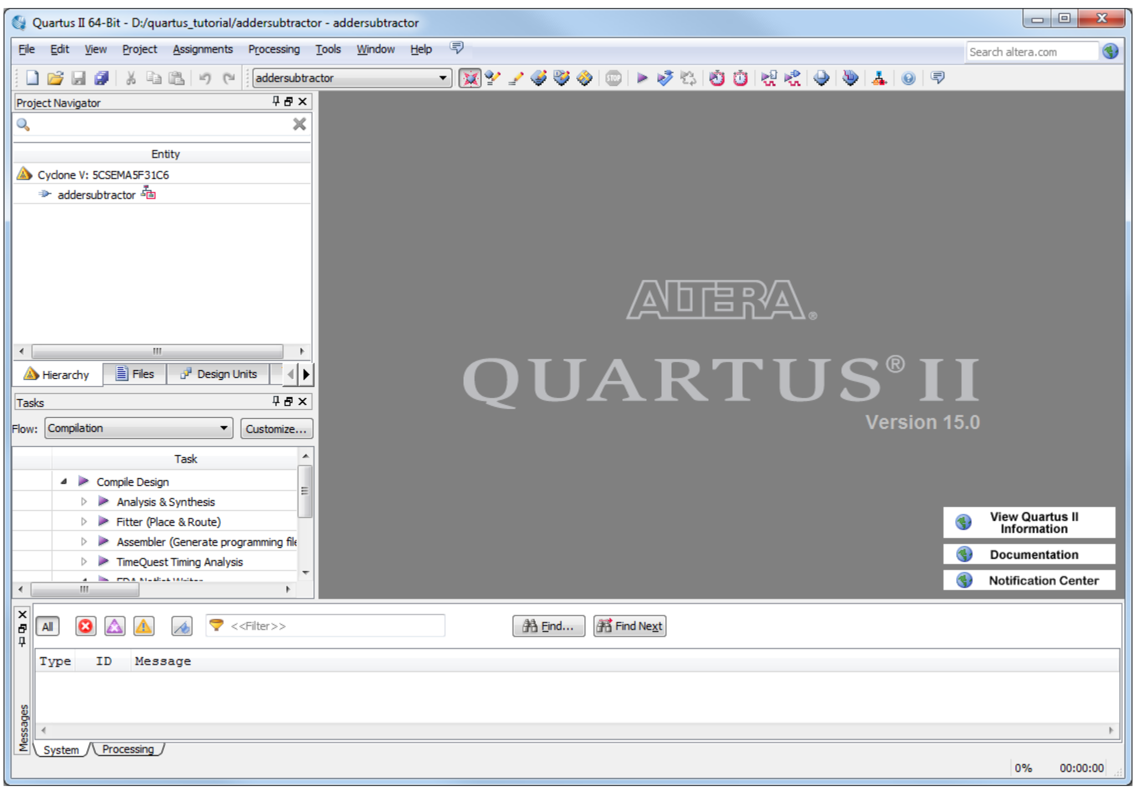


Figure 11: The Quartus® II display for the created project.

3 Design Entry Using VHDL Code

As a design example, we will use the adder/subtractor circuit shown in Figure 12. 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*.

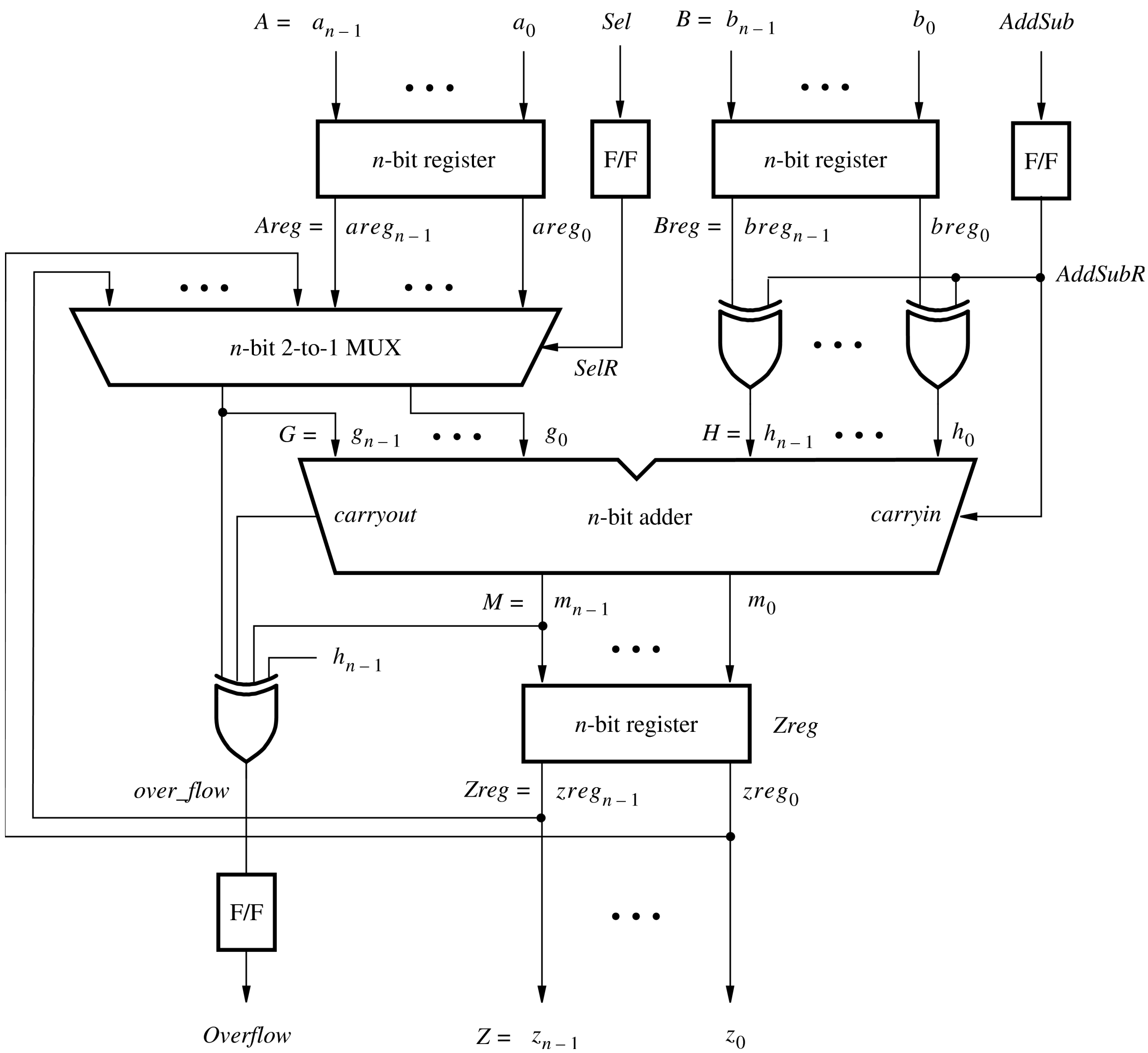


Figure 12: The adder/subtractor circuit.

The required circuit is described by the VHDL code in Figure 13. For our example, we will use a 16-bit circuit as specified by $n = 16$.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

-- Top-level entity
ENTITY addersubtractor IS
  GENERIC ( n :  INTEGER := 16 ) ;
  PORT ( A, B                   :  IN STD_LOGIC_VECTOR(n-1 DOWNTO 0) ;
         Clock, Reset, Sel, AddSub :  IN STD_LOGIC ;
         Z                      :  BUFFER STD_LOGIC_VECTOR(n-1 DOWNTO 0) ;
         Overflow               :  OUT STD_LOGIC ) ;
END addersubtractor ;

ARCHITECTURE Behavior OF addersubtractor IS
  SIGNAL G, H, M, Areg, Breg, Zreg, AddSubR_n :  STD_LOGIC_VECTOR(n-1 DOWNTO 0) ;
  SIGNAL SelR, AddSubR, carryout, over_flow :  STD_LOGIC ;
  COMPONENT mux2to1
    GENERIC ( k :  INTEGER := 8 ) ;
    PORT ( V, W :  IN STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
           Sel  :  IN STD_LOGIC ;
           F    :  OUT STD_LOGIC_VECTOR(k-1 DOWNTO 0) ) ;
  END COMPONENT ;
  COMPONENT adderk
    GENERIC ( k :  INTEGER := 8 ) ;
    PORT ( carryin  :  IN STD_LOGIC ;
           X, Y     :  IN STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
           S        :  OUT STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
           carryout :  OUT STD_LOGIC ) ;
  END COMPONENT ;
BEGIN
  PROCESS ( Reset, Clock )
  BEGIN
    IF Reset = '1' THEN
      Areg <= (OTHERS => '0'); Breg <= (OTHERS => '0');
      Zreg <= (OTHERS => '0'); SelR <= '0'; AddSubR <='0'; Overflow <= '0';
    ELSIF Clock'EVENT AND Clock = '1' THEN
      Areg <= A; Breg <= B; Zreg <= M;
      SelR <= Sel; AddSubR <= AddSub; Overflow <= over_flow;
    END IF ;
  END PROCESS ;
  nbit_adder:  adderk
    GENERIC MAP ( k => n )
    PORT MAP ( AddSubR, G, H, M, carryout ) ;
  multiplexer:  mux2to1
    GENERIC MAP ( k => n )
    PORT MAP ( Areg, Z, SelR, G ) ;
  AddSubR_n <= (OTHERS => AddSubR) ;
  H <= Breg XOR AddSubR_n ;
  Z <= Zreg ;
  over_flow <= carryout XOR G(n-1) XOR H(n-1) XOR M(n-1) ;
END Behavior;
```

... continued in Part *b*

Figure 13: VHDL code for the circuit in Figure 12 (Part *a*)

```
-- k-bit 2-to-1 multiplexer
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
  GENERIC ( k :  INTEGER := 8 ) ;
  PORT ( V, W :  IN STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
         Sel  :  IN STD_LOGIC ;
         F    :  OUT STD_LOGIC_VECTOR(k-1 DOWNTO 0) ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
  PROCESS ( V, W, Sel )
  BEGIN
    IF Sel = '0' THEN
      F <= V ;
    ELSE
      F <= W ;
    END IF ;
  END PROCESS ;
END Behavior ;

-- k-bit adder
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adderk IS
  GENERIC ( k :  INTEGER := 8 ) ;
  PORT ( carryin :  IN STD_LOGIC ;
         X, Y    :  IN STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
         S       :  OUT STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
         carryout:  OUT STD_LOGIC ) ;
END adderk ;

ARCHITECTURE Behavior OF adderk IS
  SIGNAL Sum :  STD_LOGIC_VECTOR(k DOWNTO 0) ;
BEGIN
  Sum <= ( '0' & X) + ( '0' & Y) + carryin ;
  S <= Sum(k-1 DOWNTO 0) ;
  carryout <= Sum(k) ;
END Behavior ;
```

Figure 13. VHDL code for the circuit in Figure 12 (Part *b*).

Note that the top VHDL entity 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 VHDL entity. The file name must include the extension *vhd*, which indicates a VHDL file. So, we will use the name *addersubtractor.vhd*.

3.1 Using the Quartus® II Text Editor

This section demonstrates how to use the Quartus® II Text Editor. You can skip this section if you prefer to use another text editor to create the *addersubtractor.vhd* file.

1. Select File > New to get the window in Figure 14, choose VHDL File, and click OK. This opens the Text Editor window.

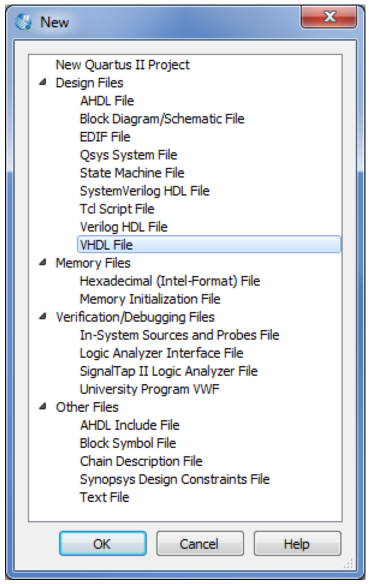


Figure 14: Choose to prepare a VHDL file.

2. 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 shown in Figure 15. In the field labeled Save as type choose VHDL File. In the field 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 16.

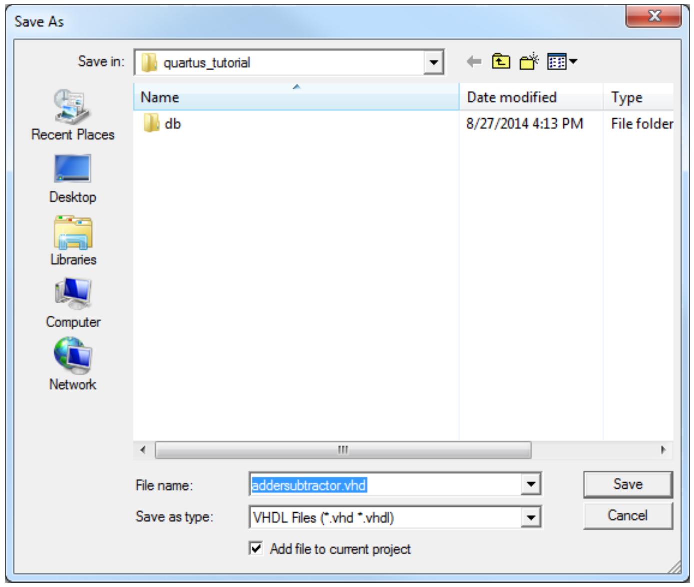


Figure 15: Name the file.

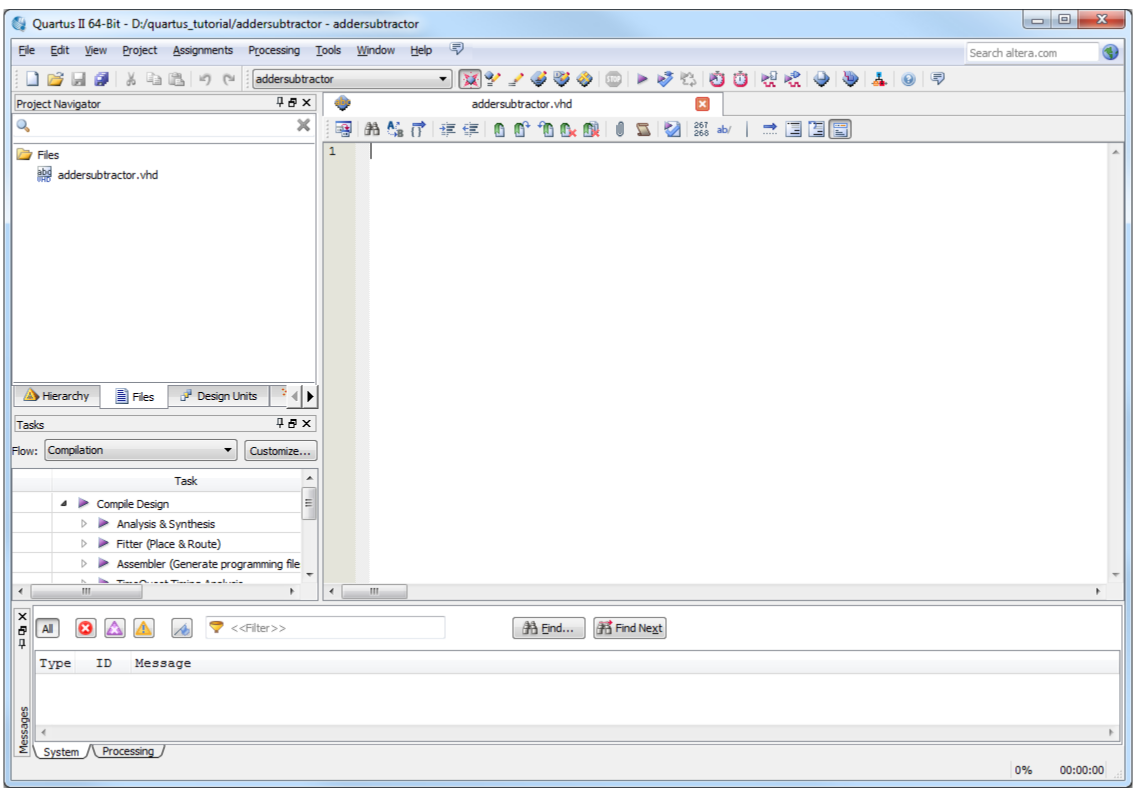


Figure 16: Text Editor window.

3. Enter the VHDL code in Figure 13 into the Text Editor Window, which is located on the right side of the screen. Save the file by going to 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 VHDL code. First, the editor can display different types of VHDL 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, as shown in Figure 17.

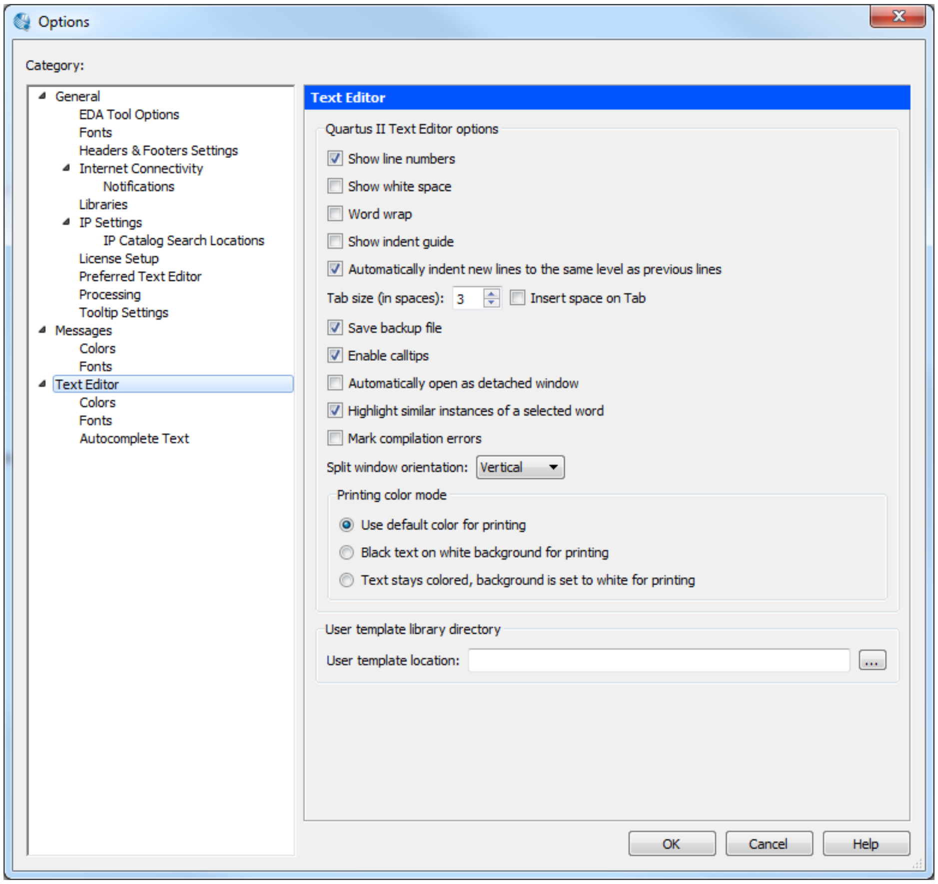


Figure 17: Text Editor Options.

3.1.1 Using VHDL Templates

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

3.2 Adding Design Files to a Project

As we indicated when discussing Figure 7, you can tell the 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... > Files, which leads to a window similar to the window in Figure 18. An alternative way of making this selection is to go to 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.vhd* file is already a part of the project and will be listed in the window in Figure 18. Otherwise, the file must be added to the project.

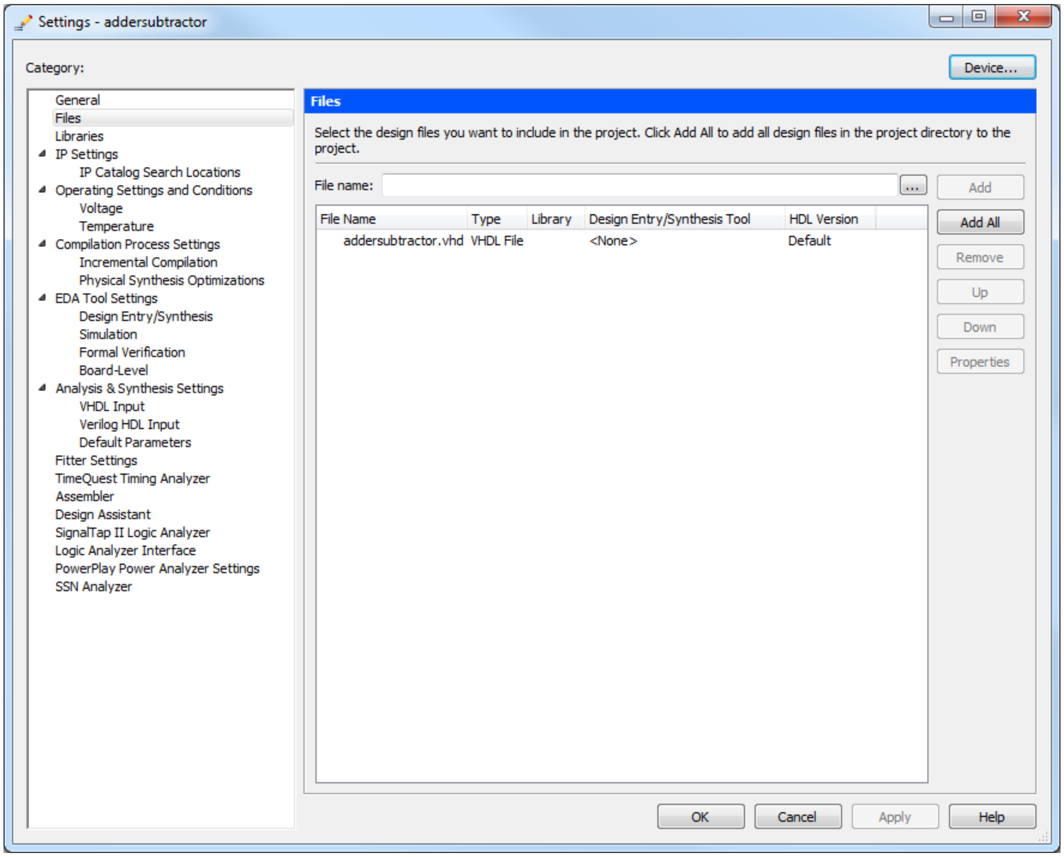


Figure 18: Settings window.

1. If not already done, place a copy of the file *addersubtractor.vhd* into the directory *quartus_tutorial*.

2. To add this file to the project, click on the  button beside the File name field in Figure 18 to get the pop-up window in Figure 19.

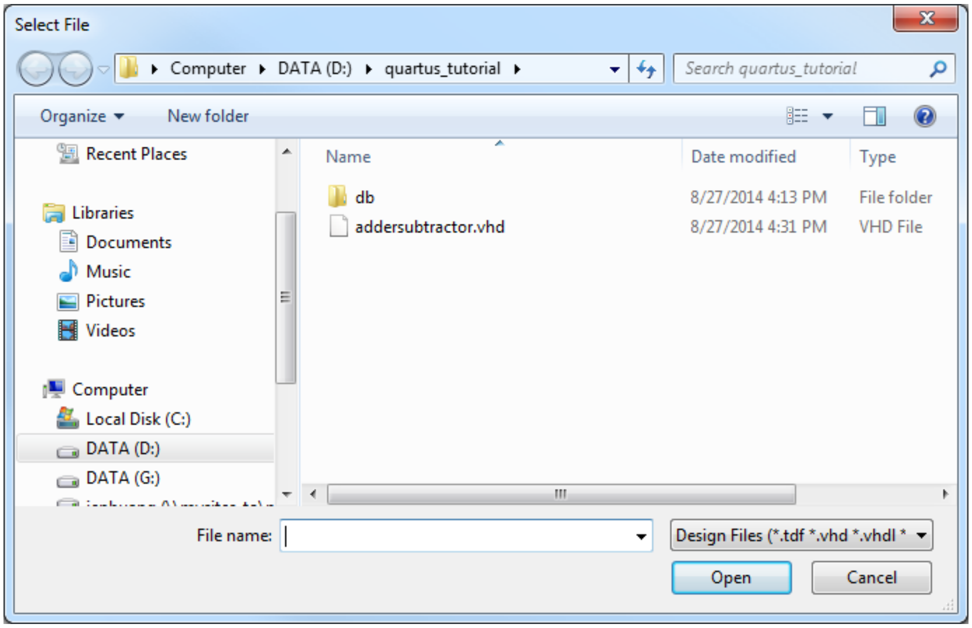


Figure 19: Select the file.

3. Select the *addersubtractor.vhd* file and click Open. The selected file is now indicated in the File name field of Figure 18. Click Add and then OK to include the *addersubtractor.vhd* 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 VHDL 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.

4 Compiling the VHDL Code

The VHDL 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*.

1. 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 Tasks window on the left side. This window also provides a comprehensive interface to edit, start, and monitor different stages of the compilation. Successful (or unsuccessful) compilation is indicated in a pop-up box. Acknowledge it by clicking OK. This leads to the Quartus® II display in Figure 20, in which we have expanded the Entity hierarchy in the top left corner to show all entities in the *addersubtractor* design. In the message window, located at the bottom of the display, various messages are shown. In case of errors, there will be appropriate messages given.

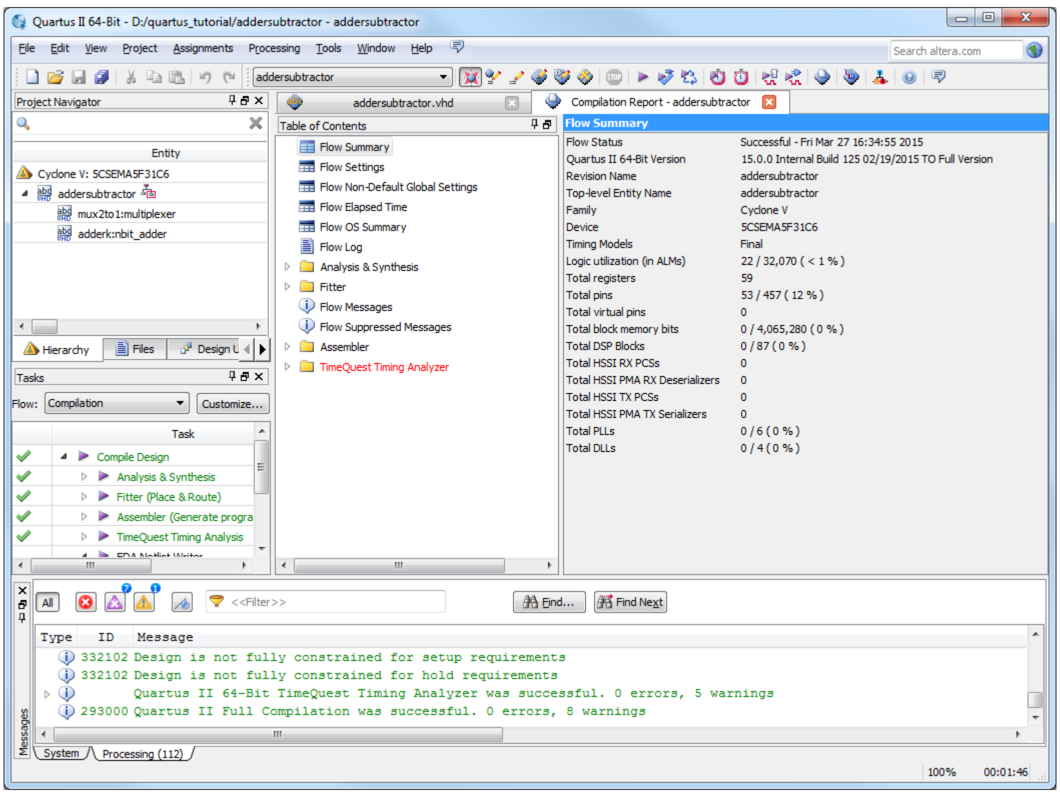


Figure 20: Display after a successful compilation.

2. When the compilation is finished, a compilation report is produced. A window showing this report, displayed in Figure 21, 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  in the toolbar. The report includes a number of sections listed on the left side of its window. Figure 21 shows 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.

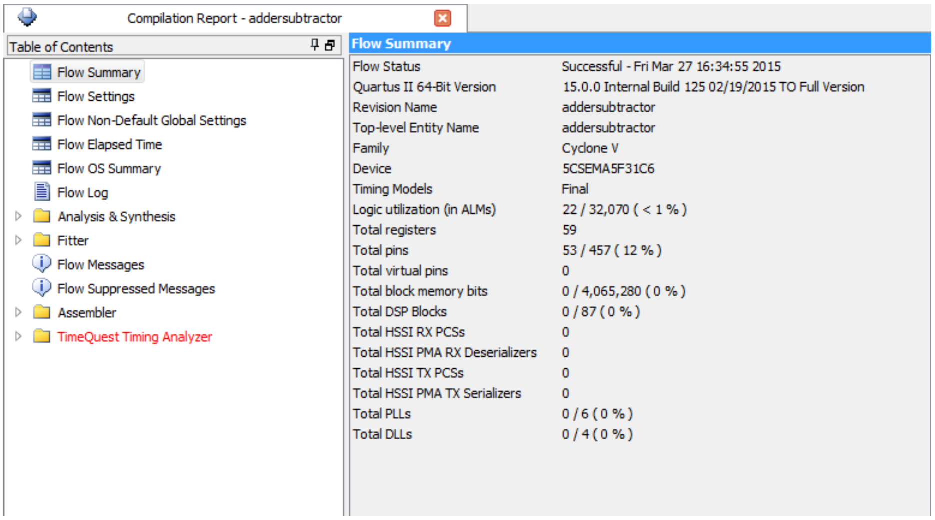


Figure 21: Compilation report.

The Compilation Report provides a lot of information that may be of interest to the designer, such as 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 *fmax*. This measure depends on the longest delay along any path between two registers clocked by the same clock. The Quartus® II software performs a timing analysis to determine the expected performance of the circuit. It evaluates several parameters, which are listed in the TimeQuest Timing Analyzer section of the Compilation Report.

3. Expand the TimeQuest Timing Analyzer section of the report, as shown in Figure 22. Notice there are multiple models included, which describe the performance of the circuit under different operating conditions. Expand the report for Slow 1100mV 85C Model and click on the item Fmax Summary to display the table in Figure 22. The table shows that the maximum frequency for our circuit implemented on the specified chip is 308.07 MHz. You may get a different value of *fmax*, dependent on the specific version of the Quartus® II software installed on your computer.

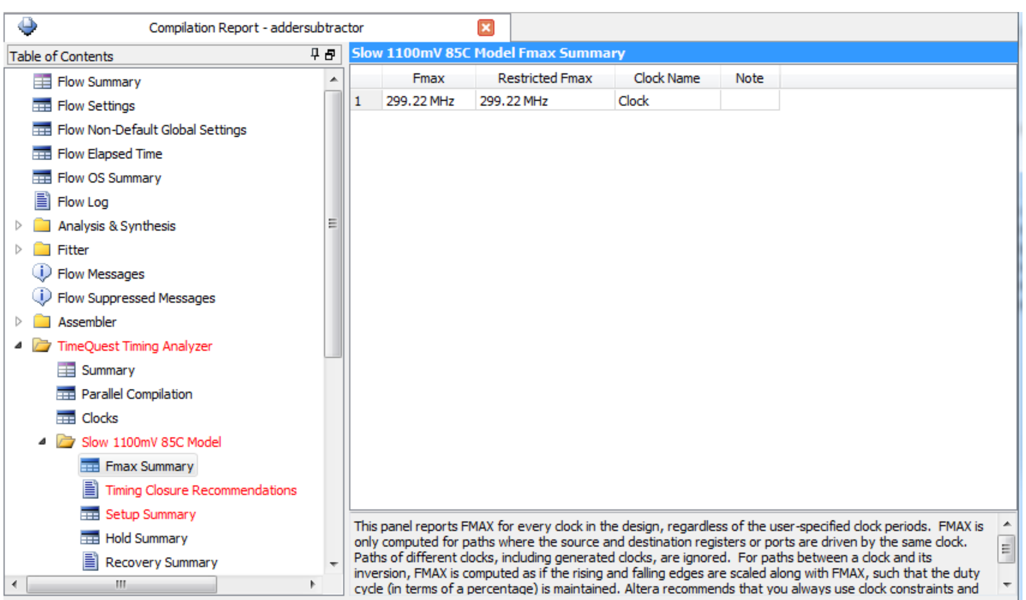
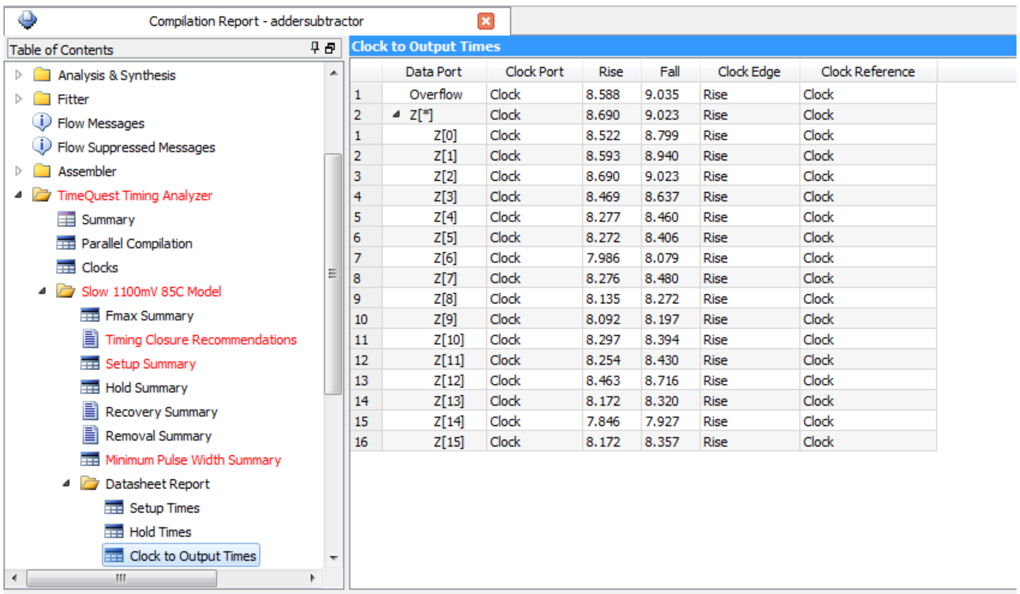


Figure 22: Fmax Summary of TimeQuest Timing Analysis.

4. While *fmax* 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. 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 *Clock to Output Time* at that pin. To see this parameter, expand Datasheet Report under the Slow 1100mV 85C Model heading and select Clock to Output Times to obtain the display in Figure 23. For each output signal, the delays for rise edge and fall edge are listed. The clock signal and its active edge are also shown in the table. Two other parameters listed in the Datasheet Report are *Setup Times* and *Hold Times*. The *Setup Time* measures the length of time for which data that feeds a register must be present at an input pin before the clock signal is asserted at the clock pin. The *Hold Time* measures the minimum length of time for which data that feeds a register must be retained at an input pin after the clock signal is asserted at the clock pin.



	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	Overflow	Clock	8.588	9.035	Rise	Clock
2	Z[*]	Clock	8.690	9.023	Rise	Clock
1	Z[0]	Clock	8.522	8.799	Rise	Clock
2	Z[1]	Clock	8.593	8.940	Rise	Clock
3	Z[2]	Clock	8.690	9.023	Rise	Clock
4	Z[3]	Clock	8.469	8.637	Rise	Clock
5	Z[4]	Clock	8.277	8.460	Rise	Clock
6	Z[5]	Clock	8.272	8.406	Rise	Clock
7	Z[6]	Clock	7.986	8.079	Rise	Clock
8	Z[7]	Clock	8.276	8.480	Rise	Clock
9	Z[8]	Clock	8.135	8.272	Rise	Clock
10	Z[9]	Clock	8.092	8.197	Rise	Clock
11	Z[10]	Clock	8.297	8.394	Rise	Clock
12	Z[11]	Clock	8.254	8.430	Rise	Clock
13	Z[12]	Clock	8.463	8.716	Rise	Clock
14	Z[13]	Clock	8.172	8.320	Rise	Clock
15	Z[14]	Clock	7.846	7.927	Rise	Clock
16	Z[15]	Clock	8.172	8.357	Rise	Clock

Figure 23: The *Clock to Output Time* delays.

5. An indication of where the circuit is implemented on the chip is available by selecting Tools > Chip Planner(Floorplan and Chip Editor), or by clicking on the icon . This opens the Chip Planner display, as shown in Figure 24. This display highlights the location of the logic elements used to implement the circuit. To make the image appear as shown in Figure 24 you may have to select View > Fit in Window (shortcut Ctrl-Alt-w).

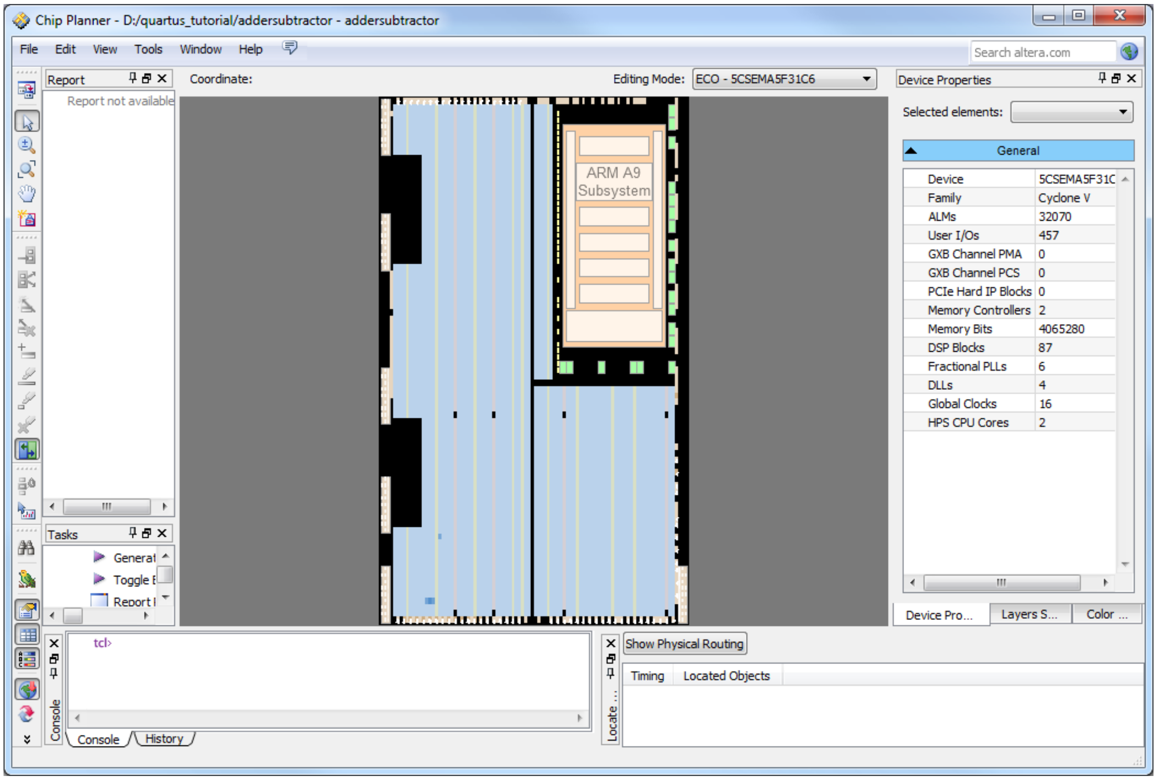


Figure 24: View of the floorplan.

6. A Zoom Tool, activated by the icon  in the left hand toolbar, can be used to enlarge parts of the image even more. You can click and drag a box over an area of the chip to quickly zoom into that part of the chip. Figure 25 shows a zoomed-in view of the floorplan that highlights the implemented circuit. By positioning the cursor on any logic element, the designer can see what part of the circuit is implemented in this resource. The chip planner tool has several icons that can be used to view aspects such as fan-in and fan-out of nodes, connecting paths between nodes, and so on. For more information on using this tool, refer to Help by selecting Help > Search > Contents > Achieving Timing Closure > Working With Assignments in the Chip Planner from the main Quartus® II display.

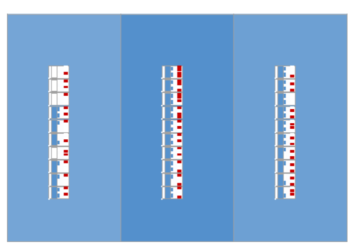


Figure 25: A portion of the expanded view.

4.1 Errors

The Quartus® II software displays messages produced during compilation in the Messages window. If the VHDL 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 VHDL 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 VHDL 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.

1. To see the effect of an error, open the file *addersubtractor.vhd*. Line 47 has the statement

   H <= Breg XOR AddSubR_n ;

   Replace H with J in this statement, illustrating a typographical error that is easily made because H and J are adjacent on the keyboard. Compile the erroneous design file. Quartus® II software will display a pop-up box indicating that the compilation was not successful. Acknowledge it by clicking OK. The compilation report summary, given in Figure 26, now confirms the failed result.

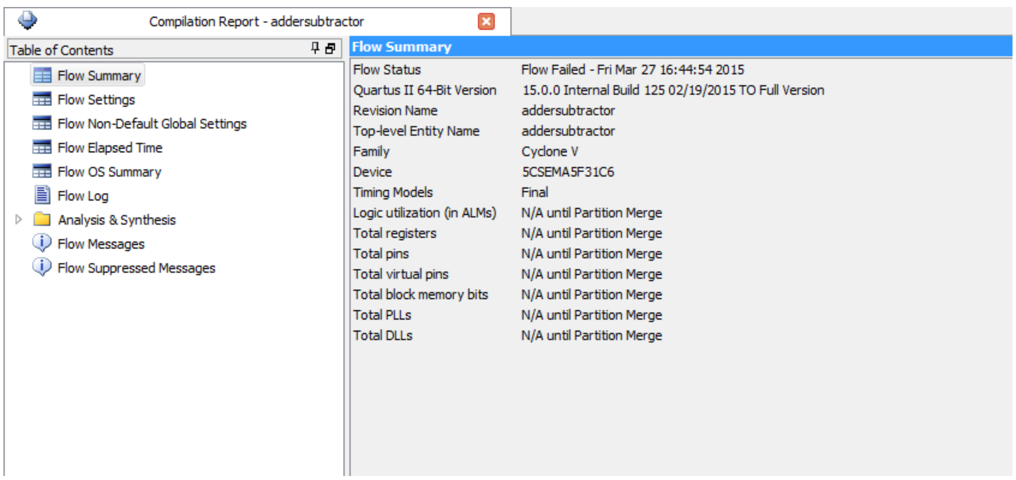


Figure 26: Compilation report for the failed design.

2. In this window, Click on Analysis & Synthesis > Messages to have all messages displayed as shown in Figure 27.

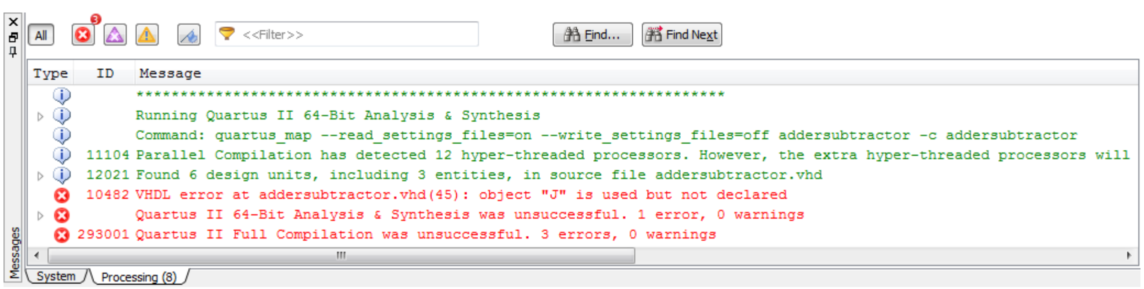


Figure 27: Error messages.

3. Double-click on the first error message, which states that variable J is not declared. The Quartus® II software responds by opening the *addersubtractor.vhd* file and highlighting the erroneous statement as shown in Figure 28. Correct the error and recompile the design.

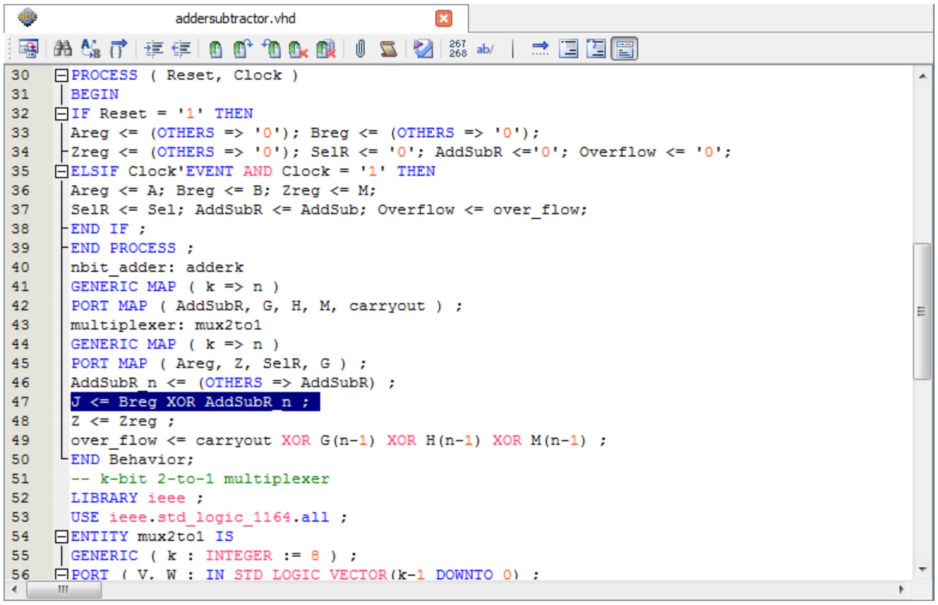


Figure 28: Identifying the location of the error.

5 Using the RTL Viewer

The 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*.

1. Click Tools > Netlist Viewers > RTL Viewer, to reach the window shown in Figure 29.

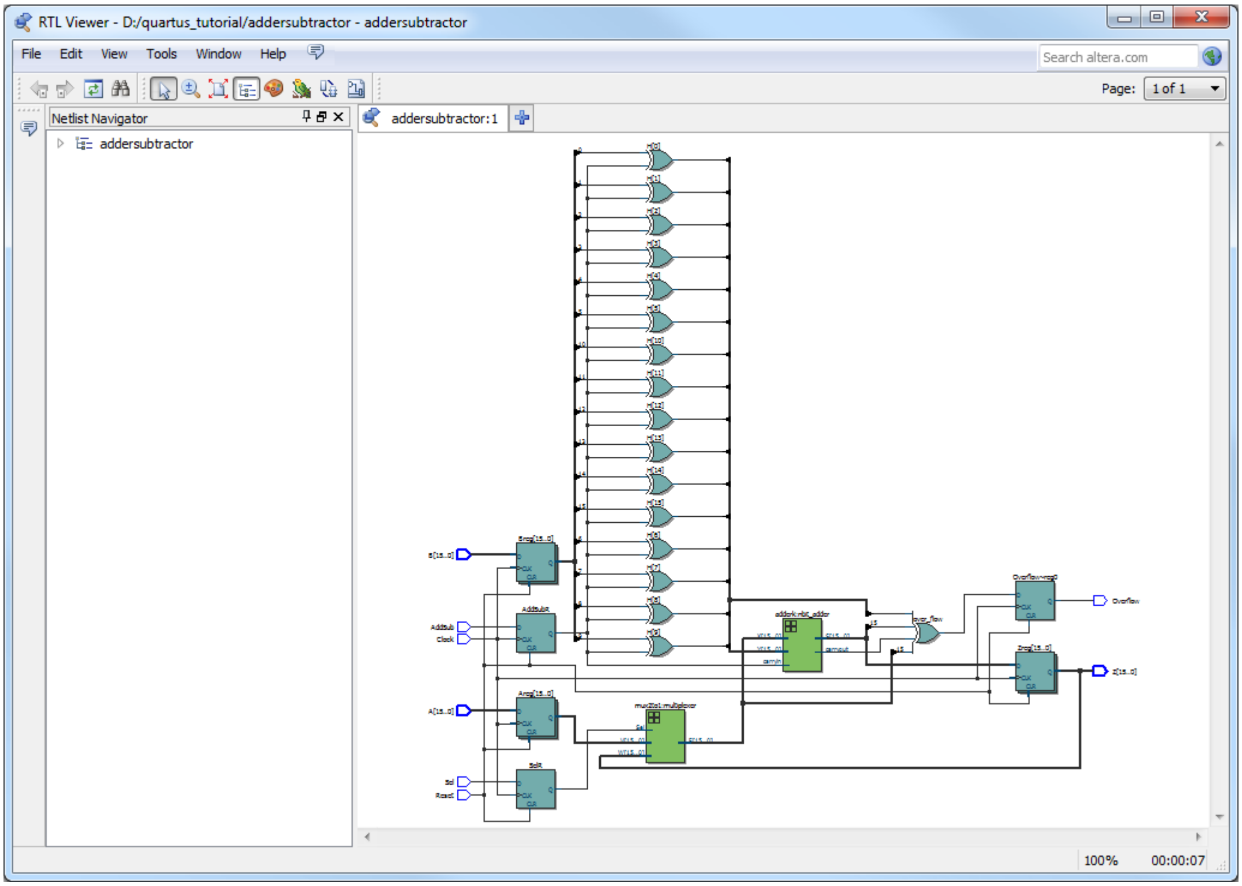


Figure 29: The *addersubtractor* circuit displayed by the RTL Viewer.

The displayed image shows 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* entities, are drawn as shaded boxes and their respective names appear above the 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.

2. Use the Zoom Tool, located in the toolbar, to enlarge the image and view the left portion of the circuit, as illustrated in Figure 30. 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 as *Areg* and *Breg*.

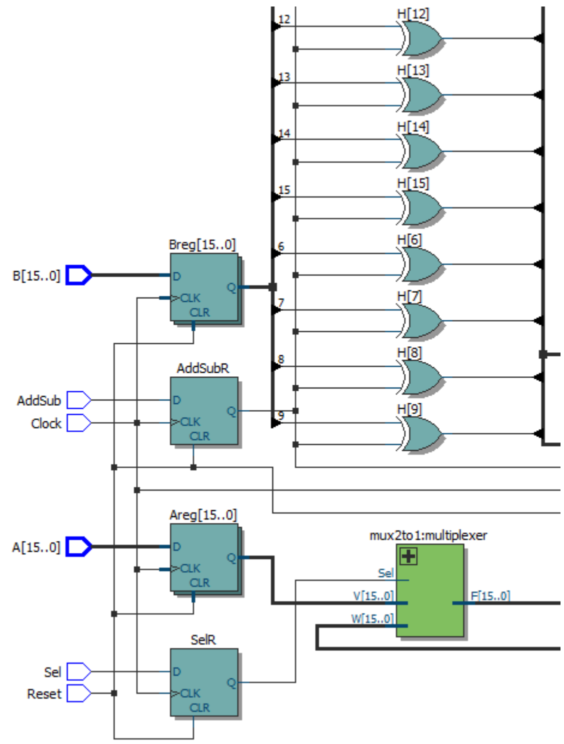


Figure 30: An enlarged view of the circuit.

3. Details of subcircuits can be seen by clicking on the box that represents a subcircuit. Select the Selection Tool from the toolbar (near the Zoom Tool), and double-click on the *mux2to1* box to obtain the image in Figure 31. 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 *Sel* signal. Observe that the multiplexer data inputs are labeled as specified in the VHDL code for the *mux2to1* entity in part b of Figure 13, namely as V and W rather than *Areg* and Z.

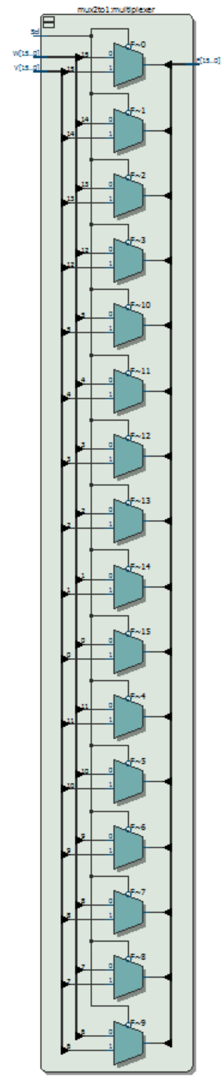


Figure 31: The multiplexer subcircuit.

The RTL viewer is a useful tool. It can be used effectively to facilitate the development of VHDL 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 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 20, which is reproduced in Figure 32, there are five windows.

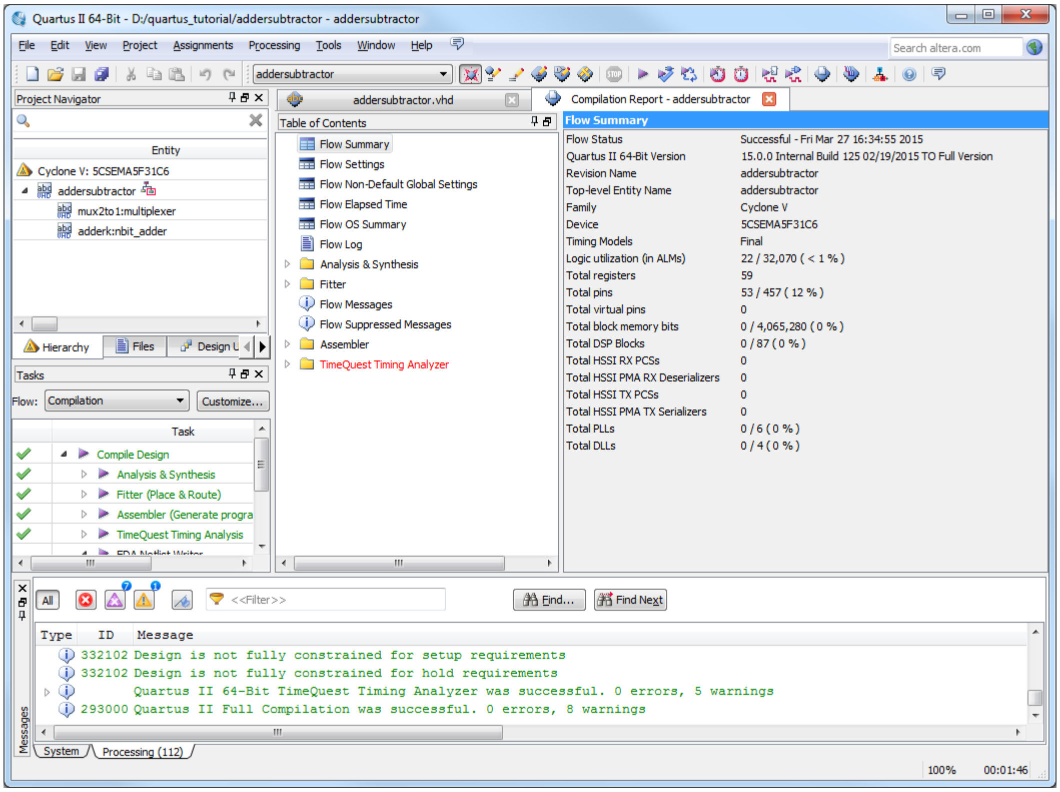


Figure 32: The main Quartus® II display.

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 entities in the VHDL code of Figure 13.

1. To see the usefulness of this window, open the previously compiled project *quartus_tutorial\addersubtractor.qpf* to get a window similar to Figure 32.
2. Double-click on the name *adderk* in the hierarchy under the Entity heading. The Quartus® II software will open the file *addersubtractor.vhd* and highlight the VHDL entity that specifies the adder subcircuit.
3. Right-click on the same name and choose Locate > Locate in Chip Planner(Floorplan & Chip Editor) from the pop-up menu that appears. This causes the Quartus® II software to display the floorplan, as in Figure 24, and highlight the part that implements the adder subcircuit.

The Tasks window is located below the Project Navigator window in the Quartus® II main window. As you have already observed, this window displays the compilation progress. It can also be used to edit and start different stages of the compilation. Double-clicking on a compilation stage from the Tasks window causes that stage of the compilation to be re-run.

At the bottom of the Quartus® II main window is the Message window, which displays user messages produced during the compilation process.

The large area in the middle-right of the Quartus® II window is used for various purposes. As we have seen, it is used by Report Viewers and the Text Editor.

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 of that window. A particular utility window can be opened by selecting it from the View > Utility Windows menu.