Timing Considerations with VHDL-Based Designs

This tutorial describes how Altera's Quartus® II software deals with the timing issues in designs based on the VHDL hardware description language. It discusses the various timing parameters and explains how specific timing constraints may be set by the user.

Contents:

Quartus II software includes a Timing Analyzer module which performs a detailed analysis of all timing delays for a circuit that is compiled for implementation in an FPGA chip. This tutorial discusses the types of analyses performed and shows how particular timing requirements may be specified by the user. The discussion assumes that the reader is familiar with the basic operation of Quartus II software, as may be learned from an introductory tutorial.

Doing this tutorial, the reader will learn about:

- Parameters evaluated by the Timing Analyzer
- Specifying the desired values of timing parameters
- Using timing simulation

The timing results shown in the examples in this tutorial were obtained using Quartus II version 5.0, but other versions of the software can also be used.

1 Example Circuit

Timing issues are most important in circuits that involve long paths through combinational logic elements with registers at inputs and outputs of these paths. As an example, we will use the adder/subtractor circuit shown in Figure 1. It 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, they are loaded 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*.

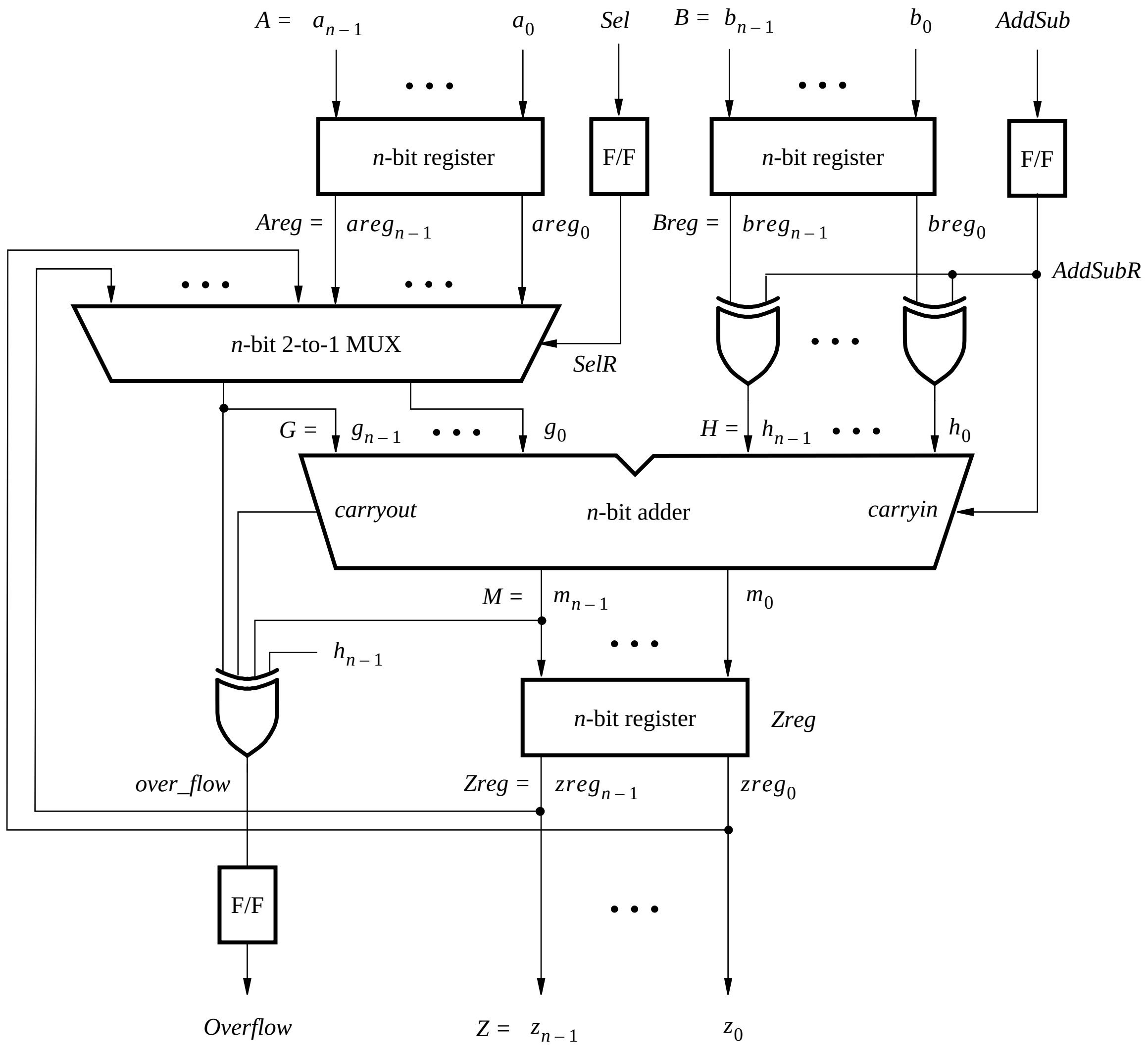


Figure 1. The adder/subtractor circuit.

The required circuit is described by the VHDL code in Figure 2. For our example, we use a 16-bit circuit as specified by $n = 16$. Implement this circuit as follows:

- Create a project *addersubtractor*.
- Include a file *addersubtractor.vhd*, which corresponds to Figure 2, in the project. For convenience, this file is provided in the directory *DE2_tutorials*\design_files, which is included on the CD-ROM that accompanies the DE2 board and can also be found on Altera's DE2 web pages.
- Choose the Cyclone II EP2C35F672C6 device, which is the FPGA chip on Altera's DE2 board.
- Compile the design.

```
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) ;
                        Selm   :  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 ;
      over_flow <= carryout XOR G(n−1) XOR H(n−1) XOR M(n−1) ;
      Z <= Zreg ;
END  Behavior;
```

. . . continued in Part *b*

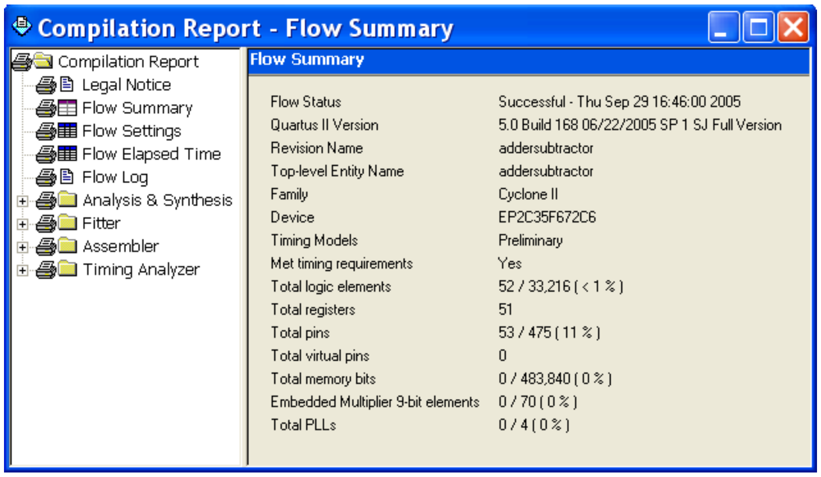
Figure 2. VHDL code for the circuit in Figure 1 (Part *a*).

```
−− 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) ;
            Selm   : IN  STD_LOGIC ;
            F      : OUT  STD_LOGIC_VECTOR(k−1 DOWNTO 0) ) ;
END mux2to1 ;

ARCHITECTURE  Behavior  OF  mux2to1  IS
BEGIN
    PROCESS  ( V, W, Selm )
    BEGIN
        IF  Selm = '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 2. VHDL code for the circuit in Figure 1 (Part *b*).

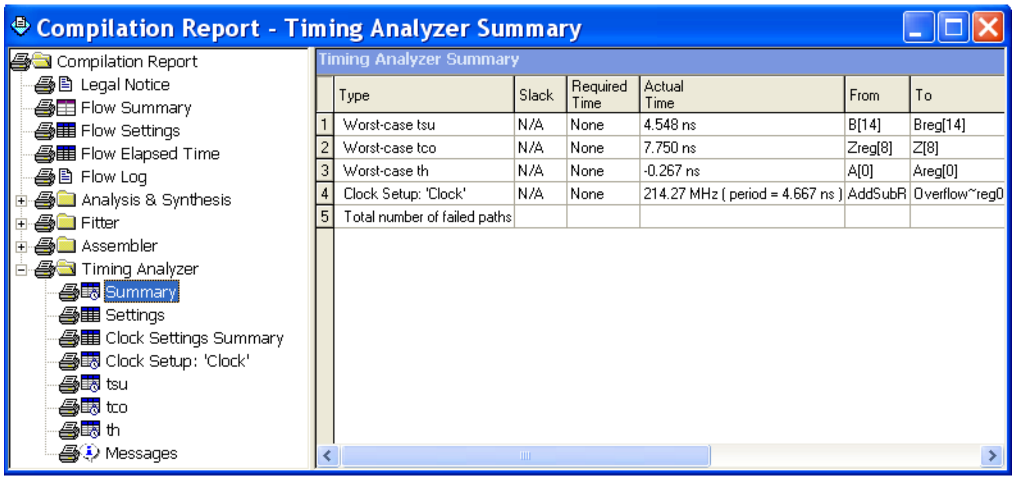
2 Timing Analyzer Report

Successful compilation of our circuit generates the Compilation Report in Figure 3. This report provides a lot of useful information. It shows the number of logic elements, flip-flops (called registers), and pins needed to implement the circuit. It gives detailed information produced by the Synthesis and Fitter modules. It also 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 *fmax*. This measure depends on the longest delay along any path, called the *critical 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, and then click on the Timing Analyzer item Summary which displays the table in Figure 4. The last entry in the table shows that the maximum frequency for our circuit implemented on the specified chip is 214.27 MHz. You may get a different value of *fmax*, dependent on the specific version of Quartus II software that you are using. To see the paths in the circuit that limit the *fmax*, click on the Timing Analyzer item Clock Setup: 'Clock' in Figure 4 to obtain the display in Figure 5. This table shows that the critical path begins at the flip-flop *AddSubR* and ends at the flip-flop *Overflow*.



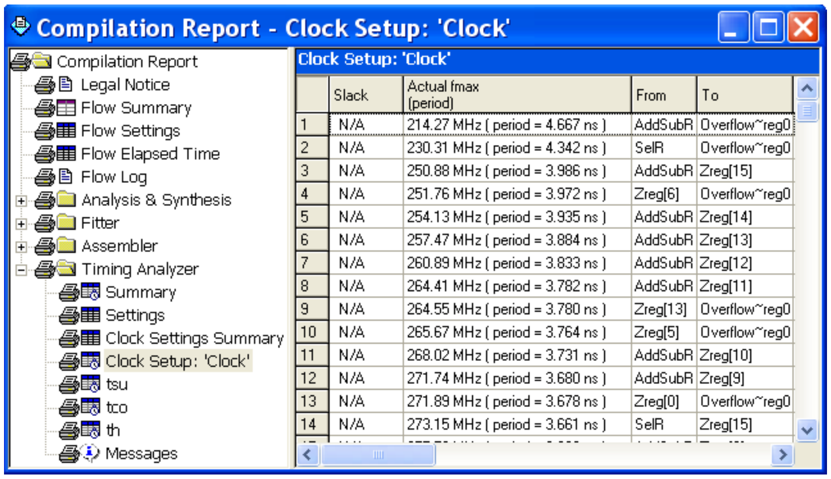
Flow Status	Successful - Thu Sep 29 16:46:00 2005
Quartus II Version	5.0 Build 168 06/22/2005 SP 1 SJ Full Version
Revision Name	addersubtractor
Top-level Entity Name	addersubtractor
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Preliminary
Met timing requirements	Yes
Total logic elements	52 / 33,216 (< 1 %)
Total registers	51
Total pins	53 / 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 3. The Compilation Report.



	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	4.548 ns	B[14]	Breg[14]
2	Worst-case tco	N/A	None	7.750 ns	Zreg[8]	Z[8]
3	Worst-case th	N/A	None	-0.267 ns	A[0]	Areg[0]
4	Clock Setup: 'Clock'	N/A	None	214.27 MHz (period = 4.667 ns)	AddSubR	Overflow~reg0
5	Total number of failed paths					

Figure 4. The Timing Analyzer Summary.

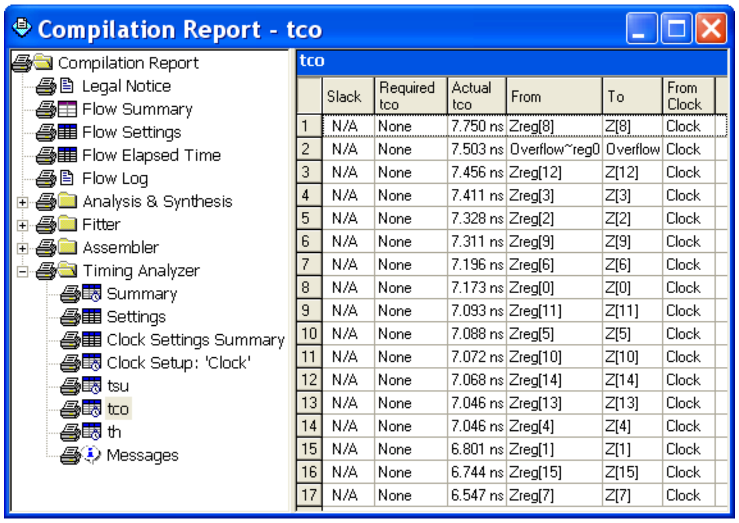


Clock Setup: 'Clock'

	Slack	Actual fmax (period)	From	To
1	N/A	214.27 MHz (period = 4.667 ns)	AddSubR	Overflow~reg0
2	N/A	230.31 MHz (period = 4.342 ns)	SelR	Overflow~reg0
3	N/A	250.88 MHz (period = 3.986 ns)	AddSubR	Zreg[15]
4	N/A	251.76 MHz (period = 3.972 ns)	Zreg[6]	Overflow~reg0
5	N/A	254.13 MHz (period = 3.935 ns)	AddSubR	Zreg[14]
6	N/A	257.47 MHz (period = 3.884 ns)	AddSubR	Zreg[13]
7	N/A	260.89 MHz (period = 3.833 ns)	AddSubR	Zreg[12]
8	N/A	264.41 MHz (period = 3.782 ns)	AddSubR	Zreg[11]
9	N/A	264.55 MHz (period = 3.780 ns)	Zreg[13]	Overflow~reg0
10	N/A	265.67 MHz (period = 3.764 ns)	Zreg[5]	Overflow~reg0
11	N/A	268.02 MHz (period = 3.731 ns)	AddSubR	Zreg[10]
12	N/A	271.74 MHz (period = 3.680 ns)	AddSubR	Zreg[9]
13	N/A	271.89 MHz (period = 3.678 ns)	Zreg[0]	Overflow~reg0
14	N/A	273.15 MHz (period = 3.661 ns)	SelR	Zreg[15]

Figure 5. Critical paths.

The table in Figure 4 also shows other timing results. 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. 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 *tco* delay at that pin. In the worst case, the *tco* in our circuit is 7.750 ns. Click on tco in the Timing Analyzer section to view the table given in Figure 6. The first entry in the table shows that it takes 7.750 ns from when an active clock edge occurs until a signal propagates from bit 8 in register *Zreg* to the output pin z_8. The other two parameters given in Figure 4 are setup time, *tsu*, and hold time, *th*.



tco

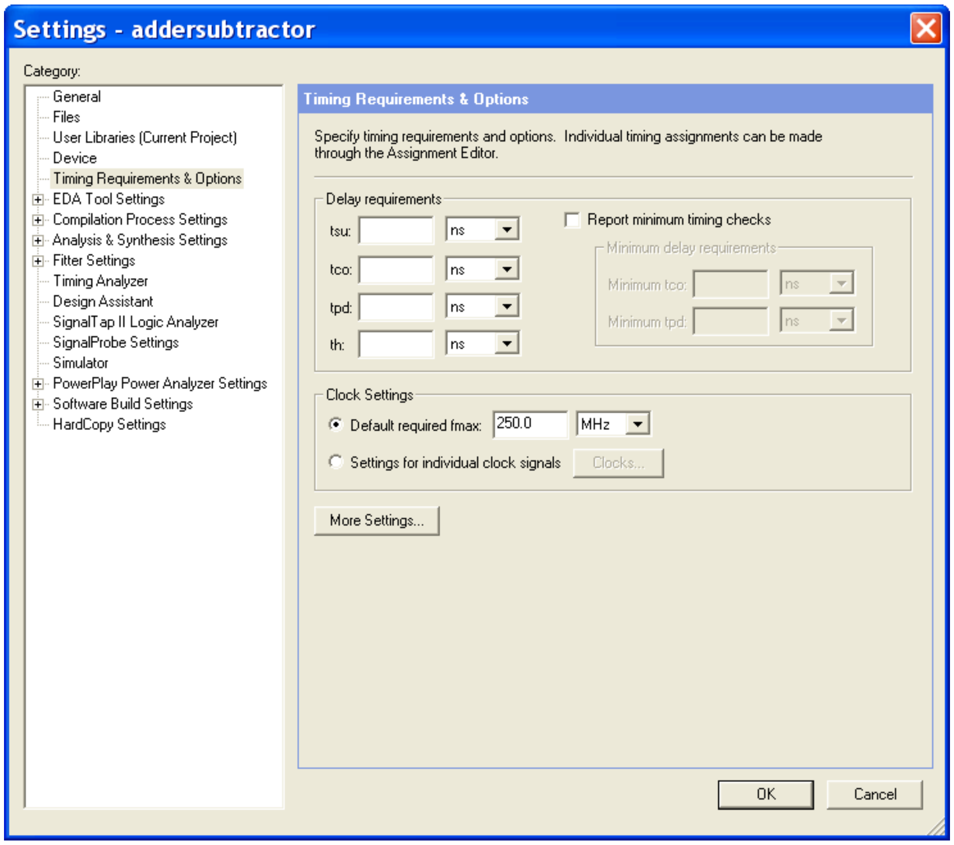
	Slack	Required tco	Actual tco	From	To	From Clock
1	N/A	None	7.750 ns	Zreg[8]	Z[8]	Clock
2	N/A	None	7.503 ns	Overflow~reg0	Overflow	Clock
3	N/A	None	7.456 ns	Zreg[12]	Z[12]	Clock
4	N/A	None	7.411 ns	Zreg[3]	Z[3]	Clock
5	N/A	None	7.328 ns	Zreg[2]	Z[2]	Clock
6	N/A	None	7.311 ns	Zreg[9]	Z[9]	Clock
7	N/A	None	7.196 ns	Zreg[6]	Z[6]	Clock
8	N/A	None	7.173 ns	Zreg[0]	Z[0]	Clock
9	N/A	None	7.093 ns	Zreg[11]	Z[11]	Clock
10	N/A	None	7.088 ns	Zreg[5]	Z[5]	Clock
11	N/A	None	7.072 ns	Zreg[10]	Z[10]	Clock
12	N/A	None	7.068 ns	Zreg[14]	Z[14]	Clock
13	N/A	None	7.046 ns	Zreg[13]	Z[13]	Clock
14	N/A	None	7.046 ns	Zreg[4]	Z[4]	Clock
15	N/A	None	6.801 ns	Zreg[1]	Z[1]	Clock
16	N/A	None	6.744 ns	Zreg[15]	Z[15]	Clock
17	N/A	None	6.547 ns	Zreg[7]	Z[7]	Clock

Figure 6. The *tco* delays.

3 Specifying Timing Constraints

So far we have compiled our VHDL code without indicating to the Quartus II software the required speed performance of the circuit. In the absence of such timing constraints the Quartus II software implements a designed circuit in a good but not necessarily the best way in order to keep the compilation time short. If the result does not meet the user's expectations, it is possible to specify certain timing constraints that should be met. For example, suppose that we want our example circuit to operate at a clock frequency of at least 250 MHz, rather than the 214.27 MHz as indicated by the value of *fmax* in Figure 4. To see if this can be achieved we can set the *fmax* constraint as follows:

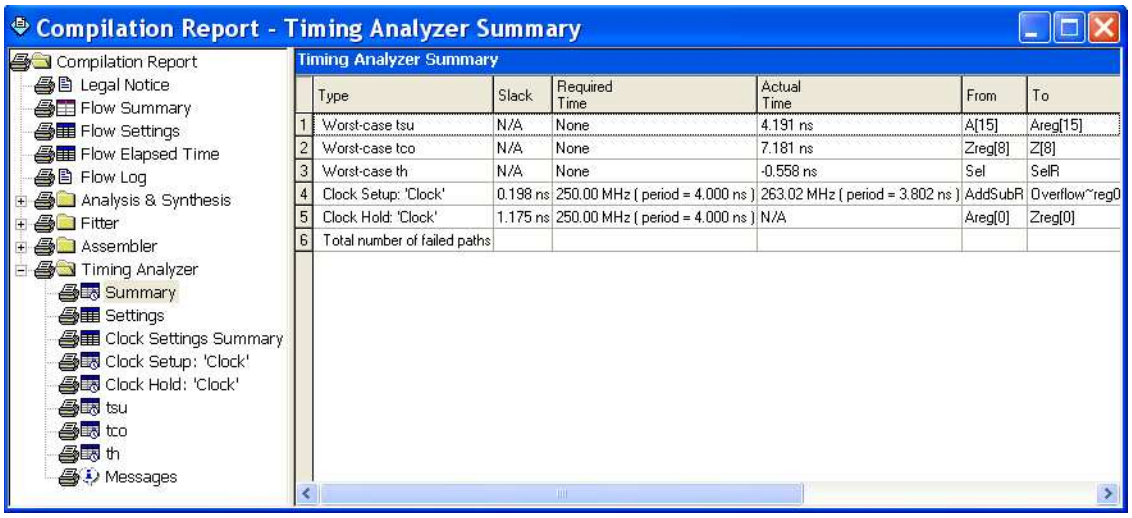
1. Select Assignments > Timing Settings to reach the Timing Requirements & Options window in Figure 7. In this window it is possible to specify the requirements for a number of different parameters.
2. In the box Clock Settings specify that the required value of *fmax* is 250 MHz. Click OK.
3. Recompile the circuit.
4. Open the Timing Analyzer Summary to see that the new *fmax* is 263.02 MHz, as indicated in Figure 8. You may get a slghtly different result depending on the version of the Quartus II software used.


Settings - addersubtractor
Category:
General
Files
User Libraries (Current Project)
Device
Timing Requirements & Options
EDA Tool Settings
Compilation Process Settings
Analysis & Synthesis Settings
Fitter Settings
Timing Analyzer
Design Assistant
SignalTap II Logic Analyzer
SignalProbe Settings
Simulator
PowerPlay Power Analyzer Settings
Software Build Settings
HardCopy Settings
Timing Requirements & Options
Delay requirements
tsu: ns
tco: ns
tpd: ns
th: ns
Report minimum timing checks
Minimum delay requirements
Minimum tco: ns
Minimum tpd: ns
Clock Settings
Settings for individual clock signals
Clocks...
More Settings...
OK
Cancel

Specify timing requirements and options. Individual timing assignments can be made through the Assignment Editor.

Default required fmax: 250.0 MHz

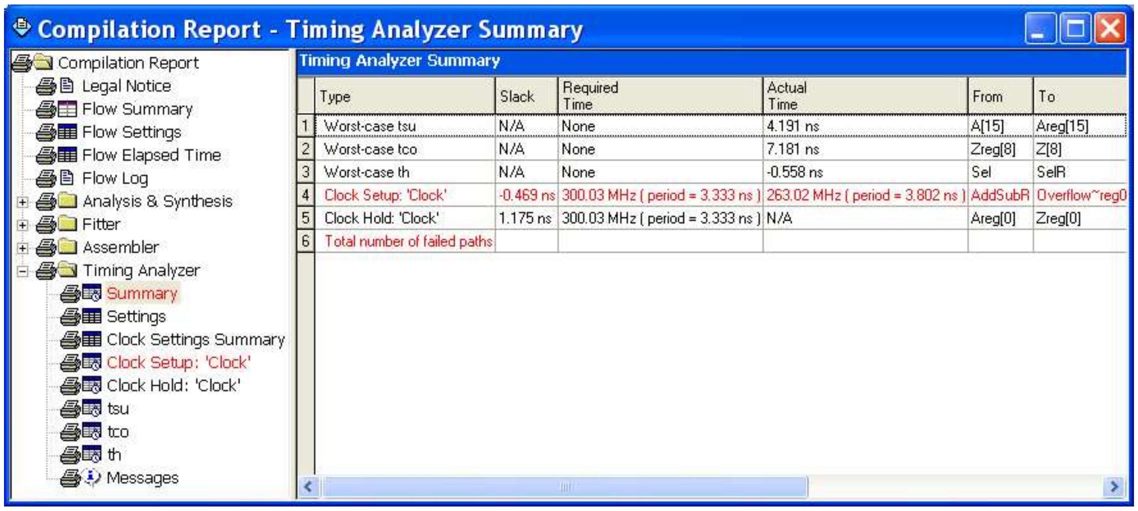
Figure 7. Specify the timing constraints in the Settings window.



	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	4.191 ns	A[15]	Areg[15]
2	Worst-case tco	N/A	None	7.181 ns	Zreg[8]	Z[8]
3	Worst-case th	N/A	None	-0.558 ns	Sel	SelR
4	Clock Setup: 'Clock'	0.198 ns	250.00 MHz (period = 4.000 ns)	263.02 MHz (period = 3.802 ns)	AddSubR	Overflow~reg0
5	Clock Hold: 'Clock'	1.175 ns	250.00 MHz (period = 4.000 ns)	N/A	Areg[0]	Zreg[0]
6	Total number of failed paths					

Figure 8. New timing results.

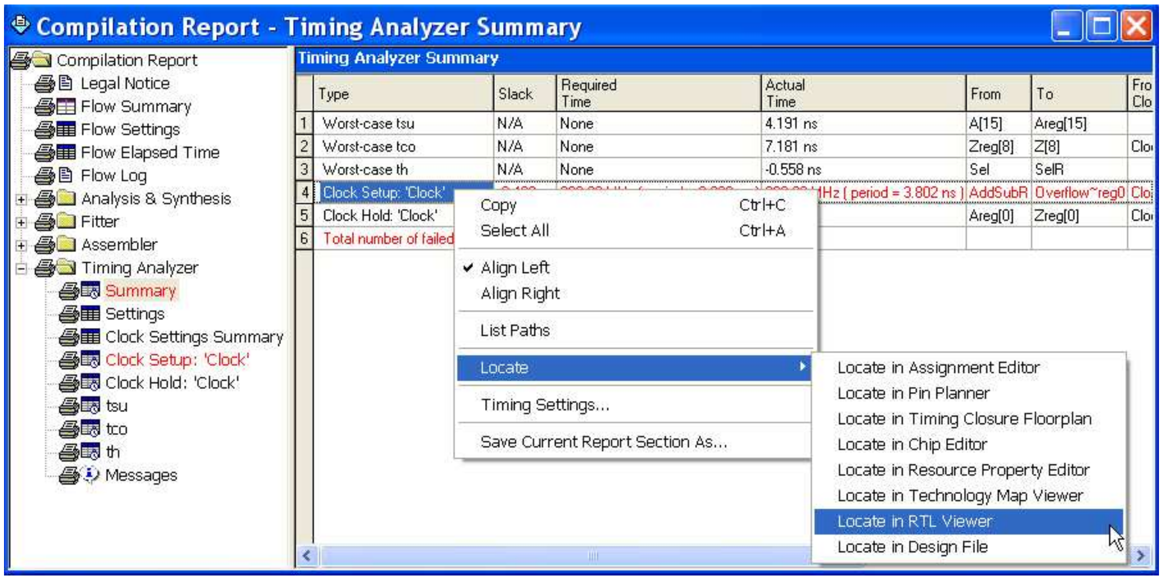
If the specified constraint is too high, the Quartus II compiler will not be able to satisfy it. For example, set the *fmax* constraint to 300 MHz and recompile the circuit. Now, the Timing Analyzer Summary will show that this constraint cannot be met, as seen in Figure 9.



	Type	Slack	Required Time	Actual Time	From	To
1	Worst-case tsu	N/A	None	4.191 ns	A[15]	Areg[15]
2	Worst-case tco	N/A	None	7.181 ns	Zreg[8]	Z[8]
3	Worst-case th	N/A	None	-0.558 ns	Sel	SelR
4	Clock Setup: 'Clock'	-0.469 ns	300.03 MHz (period = 3.333 ns)	263.02 MHz (period = 3.802 ns)	AddSubR	Overflow~reg0
5	Clock Hold: 'Clock'	1.175 ns	300.03 MHz (period = 3.333 ns)	N/A	Areg[0]	Zreg[0]
6	Total number of failed paths					

Figure 9. The timing constraint cannot be met.

The specified *fmax* of 300 MHz cannot be achieved because one or more paths in the circuit have long propagation delays. To locate the most critical path highlight the Clock Setup entry in the table by clicking on it. Then, right-click to get the pop-up menu shown in Figure 10. Select Locate > Locate in RTL Viewer which will cause the RTL Viewer to display the critical path as presented in Figure 11. Note that this path begins at flip-flop *AddSubR* and ends at the *Overflow* flip-flop.



Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	Fro Clo
1	Worst-case tsu	N/A	None	4.191 ns	A[15]	Areg[15]	
2	Worst-case tco	N/A	None	7.181 ns	Zreg[8]	Z[8]	Clo
3	Worst-case th	N/A	None	-0.558 ns	Sel	SelR	
4	Clock Setup: 'Clock'			Hz (period = 3.802 ns)	AddSubR	Overflow~reg0	Clo
5	Clock Hold: 'Clock'				Areg[0]	Zreg[0]	Clo
6	Total number of failed						

Figure 10. Locate the critical path.

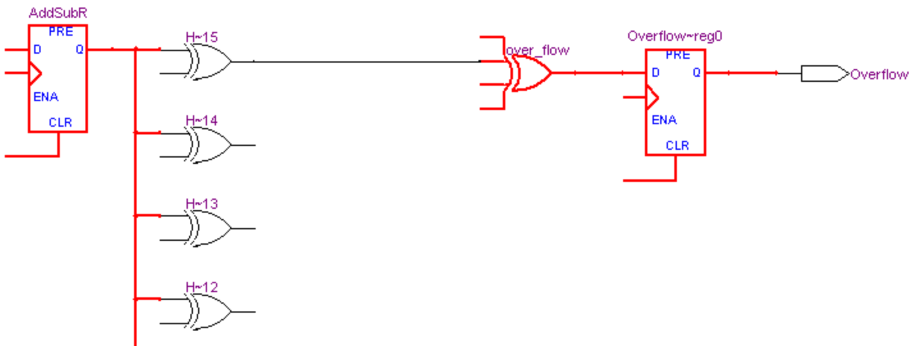
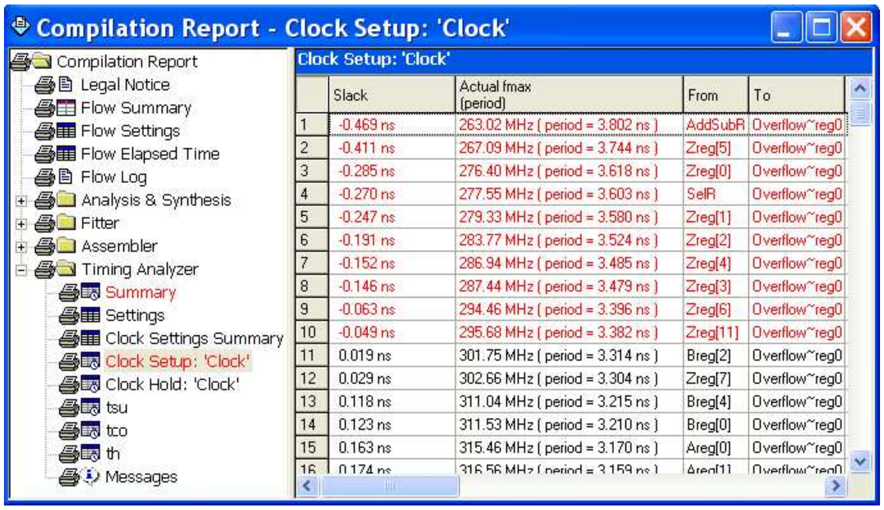


Figure 11. Path for which the timing constraint cannot be met.

It is likely that there are other paths that make it impossible to meet the specified constraint. To identify these paths choose Clock Setup: 'Clock' on the left side of the Compilation Report in Figure 9. As seen in Figure 12, there are 10 paths with propagation delays that are too long. Observe a column labeled Slack. The term *slack* is used to indicate the margin by which a timing requirement is met or not met. In the top row in Figure 12 we see that the timing delays along the path from the *AddSubR* flip-flop to the *Overflow* flip-flop are 0.469 ns longer than the maximum of 4 ns that is the period of the 250-MHz clock specified as the *fmax* constraint.



Clock Setup: 'Clock'

	Slack	Actual fmax (period)	From	To
1	-0.469 ns	263.02 MHz (period = 3.802 ns)	AddSubR	Overflow~reg0
2	-0.411 ns	267.09 MHz (period = 3.744 ns)	Zreg[5]	Overflow~reg0
3	-0.285 ns	276.40 MHz (period = 3.618 ns)	Zreg[0]	Overflow~reg0
4	-0.270 ns	277.55 MHz (period = 3.603 ns)	SelR	Overflow~reg0
5	-0.247 ns	279.33 MHz (period = 3.580 ns)	Zreg[1]	Overflow~reg0
6	-0.191 ns	283.77 MHz (period = 3.524 ns)	Zreg[2]	Overflow~reg0
7	-0.152 ns	286.94 MHz (period = 3.485 ns)	Zreg[4]	Overflow~reg0
8	-0.146 ns	287.44 MHz (period = 3.479 ns)	Zreg[3]	Overflow~reg0
9	-0.063 ns	294.46 MHz (period = 3.396 ns)	Zreg[6]	Overflow~reg0
10	-0.049 ns	295.68 MHz (period = 3.382 ns)	Zreg[11]	Overflow~reg0
11	0.019 ns	301.75 MHz (period = 3.314 ns)	Breg[2]	Overflow~reg0
12	0.029 ns	302.66 MHz (period = 3.304 ns)	Zreg[7]	Overflow~reg0
13	0.118 ns	311.04 MHz (period = 3.215 ns)	Breg[4]	Overflow~reg0
14	0.123 ns	311.53 MHz (period = 3.210 ns)	Breg[0]	Overflow~reg0
15	0.163 ns	315.46 MHz (period = 3.170 ns)	Areg[0]	Overflow~reg0
16	0.174 ns	316.56 MHz (period = 3.159 ns)	Areg[1]	Overflow~reg0

Figure 12. The longest delay paths.

We have shown how to set the *fmax* constraint. The other constraints depicted in the window in Figure 7 can be set in the same way.

4 Timing Simulation

Timing simulation provides a graphical indication of the delays in the implemented circuit, as can be observed from the displayed waveforms. For a discussion of simulation see the tutorial *Quartus II Simulation with VHDL Designs*, which uses the same *addersubtractor* circuit as an example.