# 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 9.0, 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*.

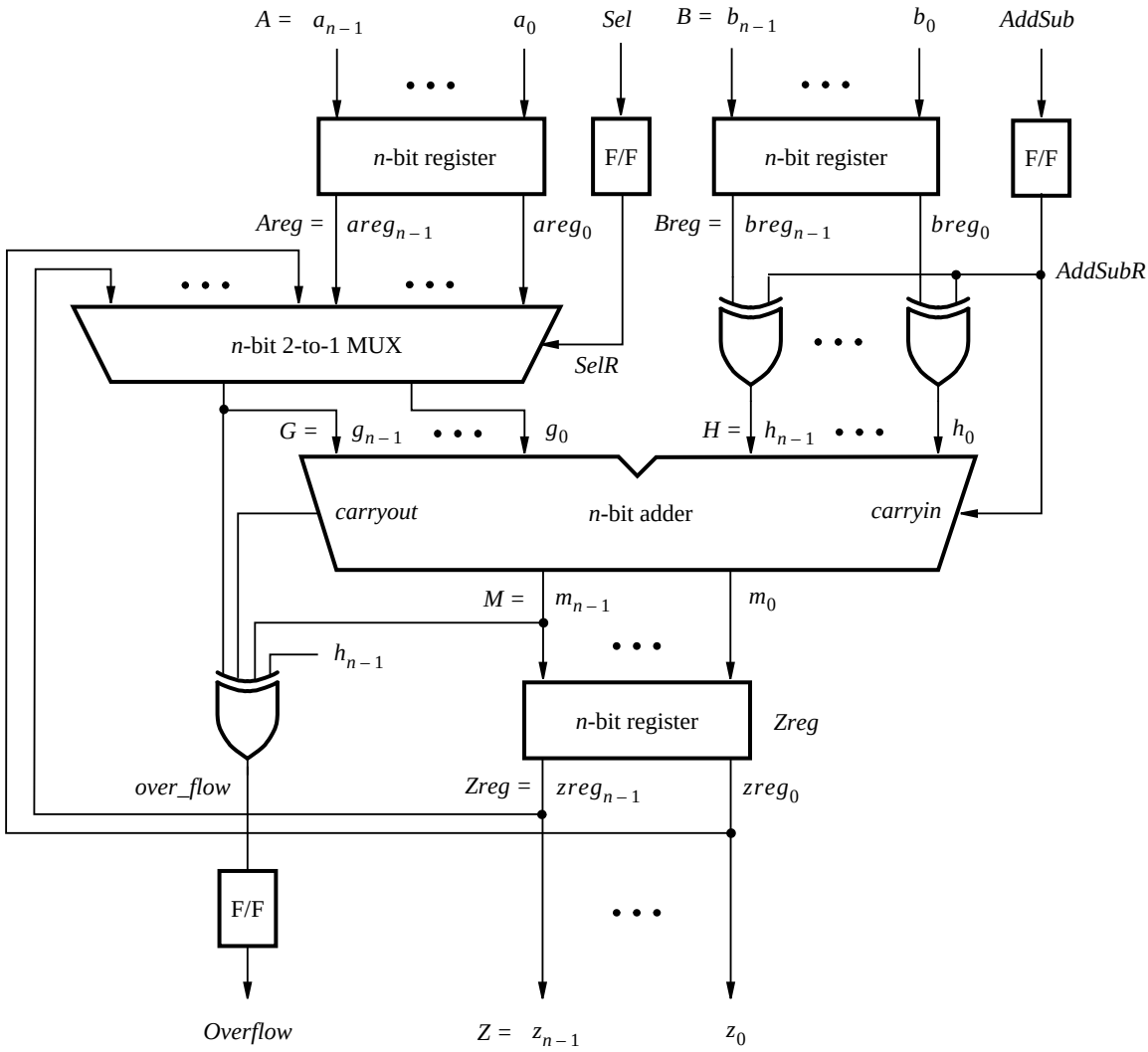



Figure 1. The adder/subtractor circuit.

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 *DE1_tutorials\design_files*, which is included on the CD-ROM that accompanies the DE1 board and can also be found on Altera's DE1 web pages.
- Choose the Cyclone II EP2C20F484C7 device, which is the FPGA chip on Altera's DE1 board.
- Compile the design.

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

-- Top-level module
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 216.59 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 *SelR* and ends at the flip-flop *Overflow*.

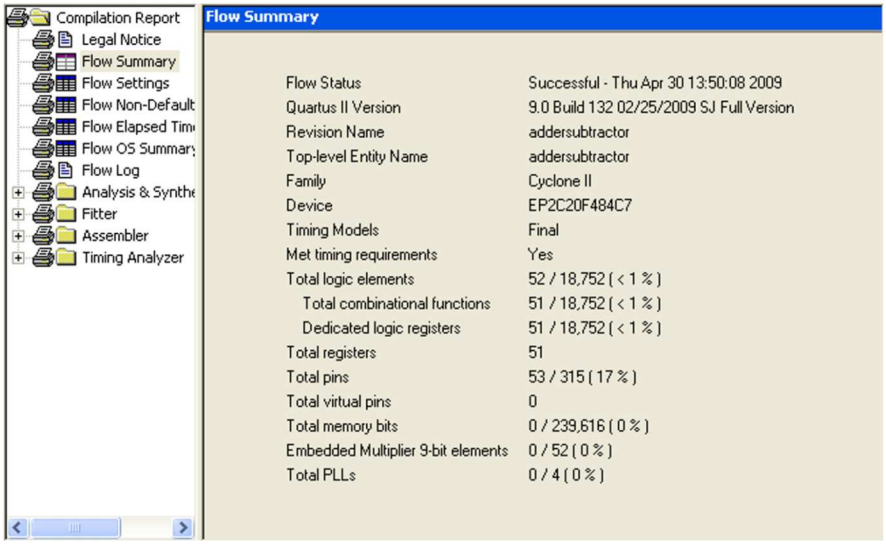



Figure 3. The Compilation Report.

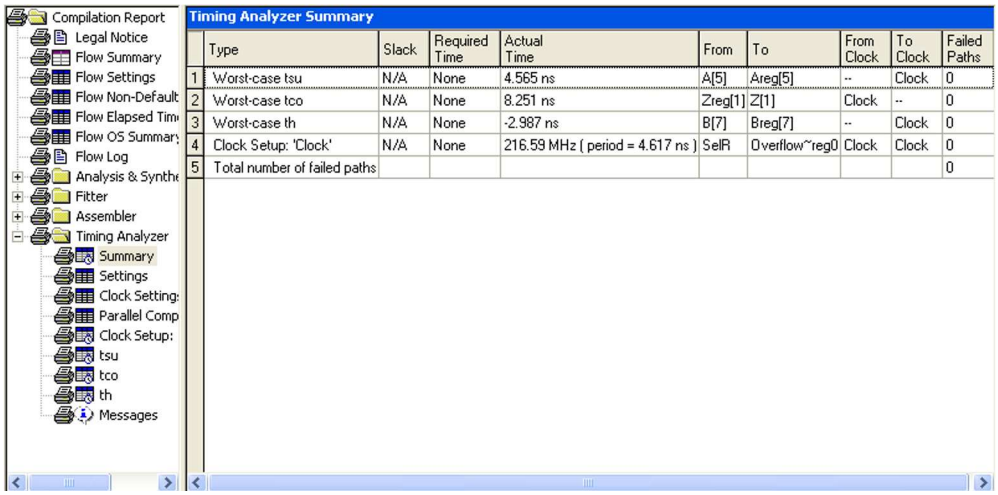



Figure 4. The Timing Analyzer Summary.

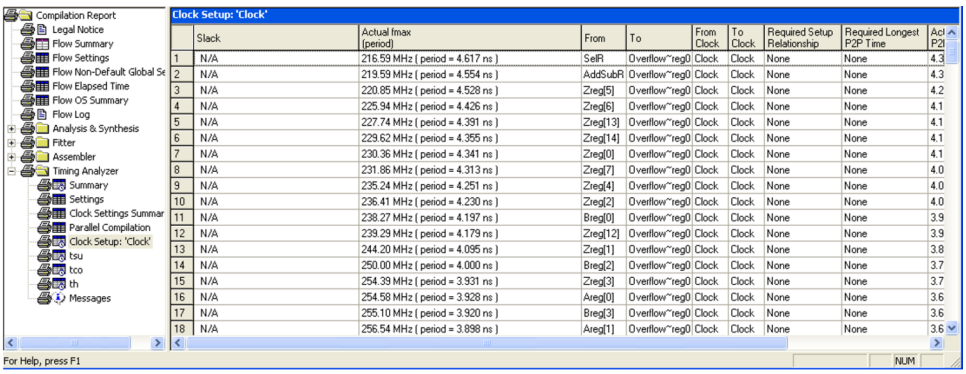



Figure 5. Critical paths.

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 8.251 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 8.251 ns from when an active clock edge occurs until a signal propagates from bit 1 in register *Zreg* to the output pin $z_1$. The other two parameters given in Figure 4 are setup time, *tsu*, and hold time, *th*.

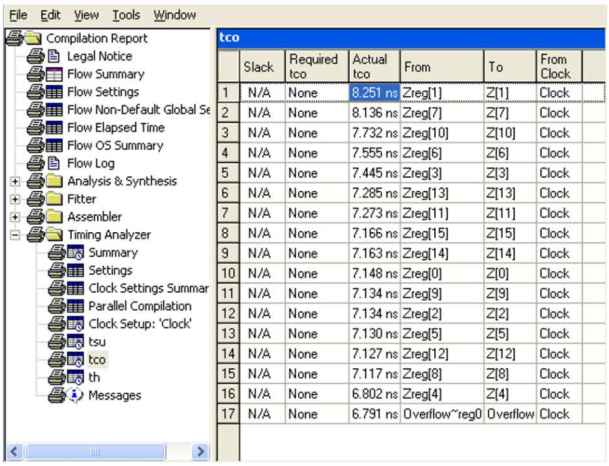



Figure 6. The *tco* delays.

# 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. 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 220 MHz. To see if this can be achieved we can set the *fmax* constraint as follows:

1. Select Assignments > Timing Analysis Settings > Classic Timing Analyzer Settings to reach the

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 220 MHz. Click OK.
3. Recompile the circuit.
4. Open the Timing Analyzer Summary to see that the new *fmax* is 220.60 MHz, as indicated in Figure 8. You may get a slightly different result depending on the version of the Quartus II software used.

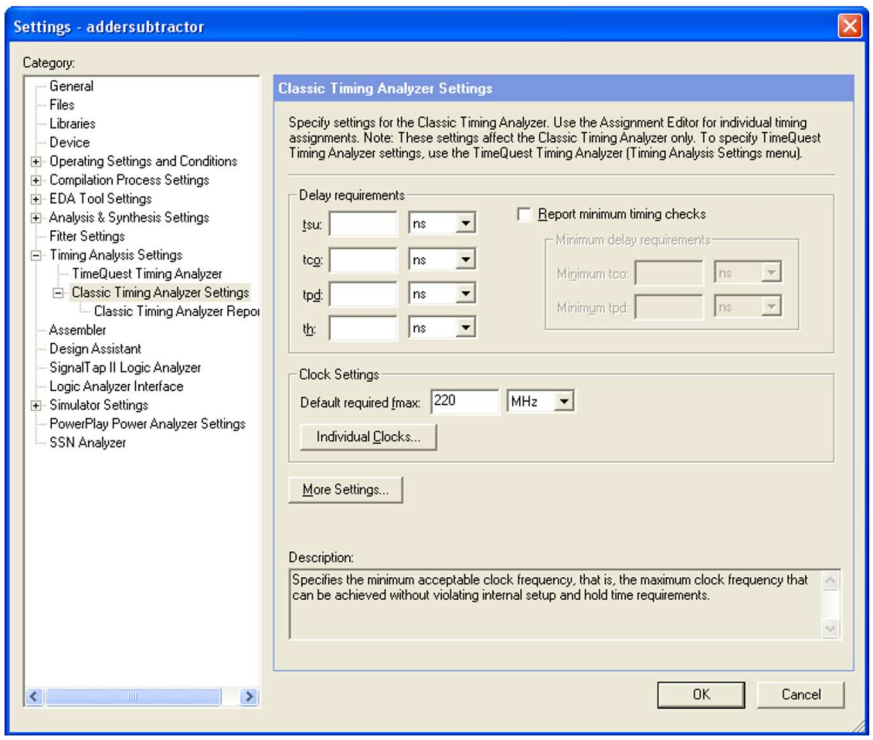



Figure 7. Specify the timing constraints in the Settings window.

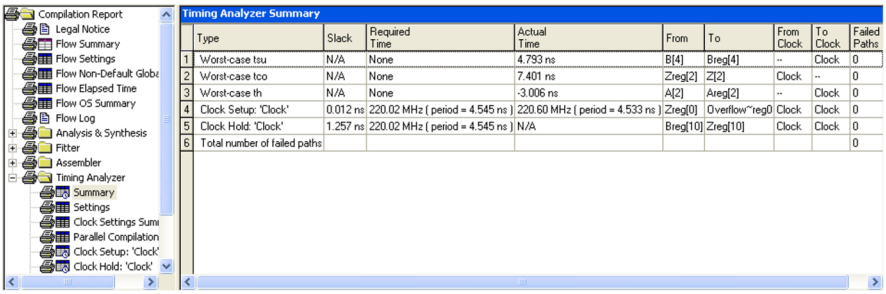
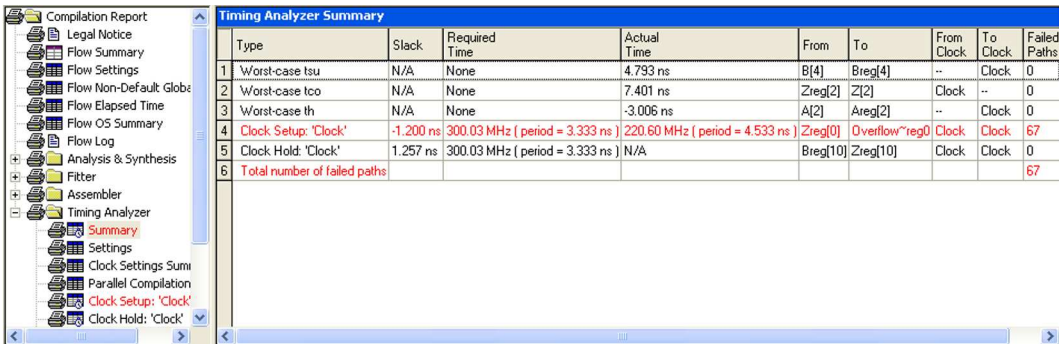



Figure 8. New timing results.

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 | From Clock | To Clock | Failed Paths |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Worst-case tsu | N/A | None | 4.793 ns | B[4] | Breg[4] | -- | Clock | 0 |
| 2 | Worst-case tco | N/A | None | 7.401 ns | Zreg[2] | Z[2] | Clock | -- | 0 |
| 3 | Worst-case th | N/A | None | -3.006 ns | A[2] | Areg[2] | -- | Clock | 0 |
| 4 | Clock Setup: 'Clock' | -1.200 ns | 300.03 MHz ( period = 3.333 ns ) | 220.60 MHz ( period = 4.533 ns ) | Zreg[0] | Overflow~reg0 | Clock | Clock | 67 |
| 5 | Clock Hold: 'Clock' | 1.257 ns | 300.03 MHz ( period = 3.333 ns ) | N/A | Breg[10] | Zreg[10] | Clock | Clock | 0 |
| 6 | Total number of failed paths | | | | | | | | 67 |

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 bit 0 in register *Zreg* and ends at the *Overflow* flip-flop.

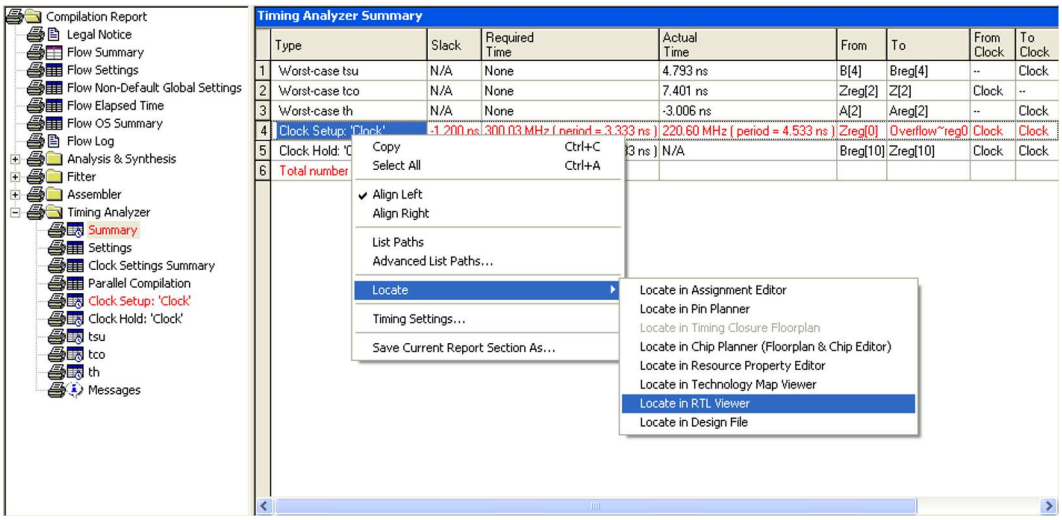



Figure 10. Locate the critical path.

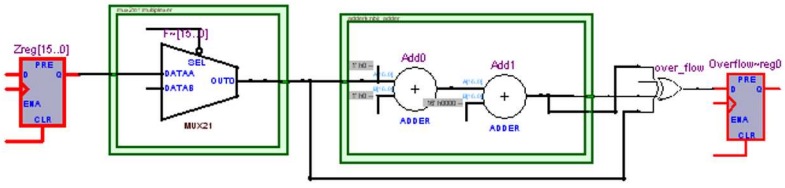
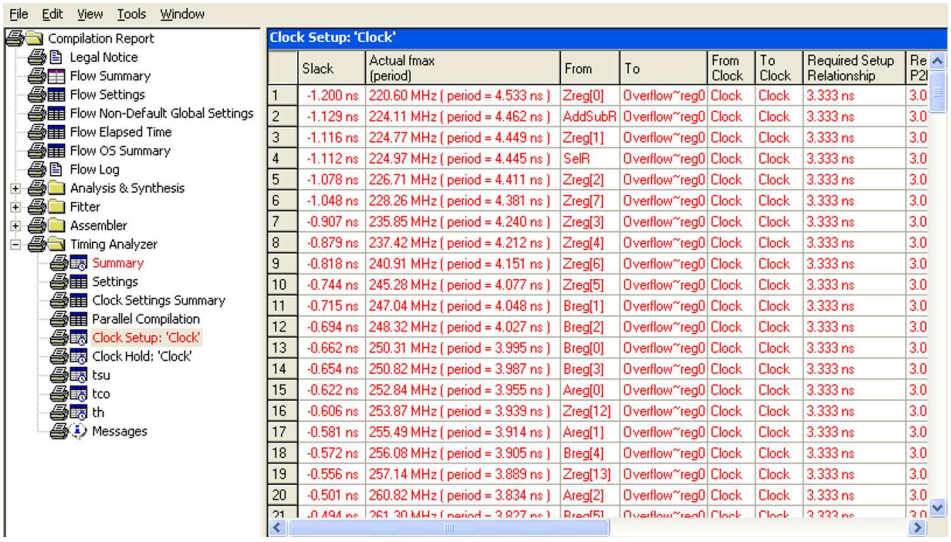



Figure 11. Path for which the timing constraint cannot be met.

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 67 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 delay along the path from bit 0 in register *Zreg* flip-flop to the *Overflow* flip-flop is 1.200 ns longer than the maximum of 3.33 ns that is the period of the 300-MHz clock specified as the *fmax* constraint.




| | Slack | Actual fmax (period) | From | To | From Clock | To Clock | Required Setup Relationship | Re P2I |
|---|---|---|---|---|---|---|---|---|
| 1 | -1.200 ns | 220.60 MHz ( period = 4.533 ns ) | Zreg[0] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 2 | -1.129 ns | 224.11 MHz ( period = 4.462 ns ) | AddSubR | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 3 | -1.116 ns | 224.77 MHz ( period = 4.449 ns ) | Zreg[1] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 4 | -1.112 ns | 224.97 MHz ( period = 4.445 ns ) | SelR | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 5 | -1.078 ns | 226.71 MHz ( period = 4.411 ns ) | Zreg[2] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 6 | -1.048 ns | 228.26 MHz ( period = 4.381 ns ) | Zreg[7] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 7 | -0.907 ns | 235.85 MHz ( period = 4.240 ns ) | Zreg[3] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 8 | -0.879 ns | 237.42 MHz ( period = 4.212 ns ) | Zreg[4] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 9 | -0.818 ns | 240.91 MHz ( period = 4.151 ns ) | Zreg[6] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 10 | -0.744 ns | 245.28 MHz ( period = 4.077 ns ) | Zreg[5] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 11 | -0.715 ns | 247.04 MHz ( period = 4.048 ns ) | Breg[1] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 12 | -0.694 ns | 248.32 MHz ( period = 4.027 ns ) | Breg[2] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 13 | -0.662 ns | 250.31 MHz ( period = 3.995 ns ) | Breg[0] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 14 | -0.654 ns | 250.82 MHz ( period = 3.987 ns ) | Breg[3] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 15 | -0.622 ns | 252.84 MHz ( period = 3.955 ns ) | Areg[0] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 16 | -0.606 ns | 253.87 MHz ( period = 3.939 ns ) | Zreg[12] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 17 | -0.581 ns | 255.49 MHz ( period = 3.914 ns ) | Areg[1] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 18 | -0.572 ns | 256.08 MHz ( period = 3.905 ns ) | Breg[4] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 19 | -0.556 ns | 257.14 MHz ( period = 3.889 ns ) | Zreg[13] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 20 | -0.501 ns | 260.82 MHz ( period = 3.834 ns ) | Areg[2] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |
| 21 | -0.494 ns | 261.30 MHz ( period = 3.827 ns ) | Breg[5] | Overflow~reg0 | Clock | Clock | 3.333 ns | 3.0 |

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.