Digital Engineering
Lab 2

Y3890959
Y3878784

31st January 2023

Task A: Timing Simulation

1.1.1 Self-Checking Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity algorithm_tb is
    generic (data_size : integer := 16);
end algorithm_tb;

architecture Behavioral of algorithm_tb is

    -- Inputs
    signal InputA : STD_LOGIC_VECTOR(data_size-1 downto 0);
    signal InputB : STD_LOGIC_VECTOR(data_size-1 downto 0);
    signal InputC : STD_LOGIC_VECTOR(data_size-1 downto 0);
    signal InputD : STD_LOGIC_VECTOR(data_size-1 downto 0);
    signal clk, rst : STD_LOGIC;

    -- Output
    signal Output : STD_LOGIC_VECTOR(data_size*2-1 downto 0);

    constant clk_period : time := 70ns;  -- Clock period
    constant latency : natural := 2;     -- Circuit latency

    -- Defining a record of test patterns to verify the circuit
    type test_vector is record
        InputA : STD_LOGIC_VECTOR(data_size-1 downto 0);
        InputB : STD_LOGIC_VECTOR(data_size-1 downto 0);
        InputC : STD_LOGIC_VECTOR(data_size-1 downto 0);
        InputD : STD_LOGIC_VECTOR(data_size-1 downto 0);
        Output : STD_LOGIC_VECTOR(data_size*2-1 downto 0);
    end record;

    type test_vector_array is array
        (natural range <>) of test_vector;
```

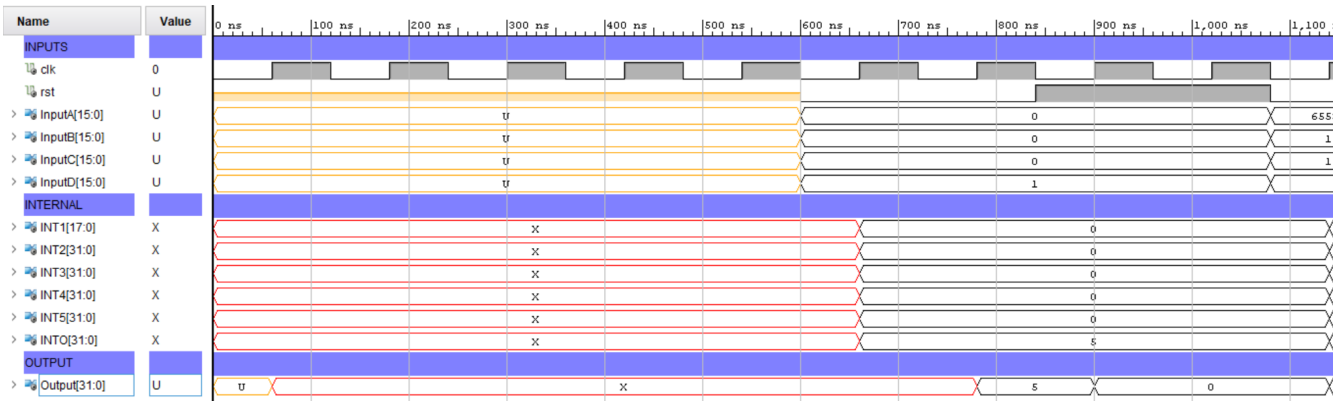
```
-- Test strategy:
--
--  Initialisation:
--      Outside of this vector, the set_inputs process will
--      initialise all inputs to x'0000', and D to x'0001'.
--      After that, the circuit is reset with a 2-clock rst
--      pulse.
--
--  TEST 1 ('large' vector for INT1, D):
--      The max input for A we can provide without an INT1
--      overflow is x'FFFF' as INT1 is 18-bit but A in only
--      16-bit in width. We should see a signal of x'2FFFD'
--      in INT1. We set D to its highest possible value of
--      x'FFFF' to isolate all other vectors.
--
--  TEST 2 ('large' vector for INT2, INT3, B, C, D):
--      We can induce a large INT2 vector by inputting the
--      max value of x'FFFF' to B and C. This will result in
--      a x'FFFE0001' INT2 vector. As a result this, INT3 will
--      also be large with x'FFFE0004', we can isolate other
--      signals by setting D to x'FFFF'
--
--  TEST 3 ('large' vector for INT3, INT2, INT1, A, B, C, D):
--      By inputting a large value for A, B, C and D, we can
--      induce a large vector for INT3. We need to set either
--      B or C to x'FFFE' to mitigate INT3 overflow, I've chosen
--      C as this slightly reduces (isolates) the INT5 vector.
--      INT3 vector should be x'FFFFFFFF'.
--
--  TEST 4 ('large' vector for INT4, INT3, INT2, INT1, B, C, A):
--      We can induce a large INT4 vector by doing the same as
--      TEST 3, by setting A and C to max and C to x'FFFE'. We
--      need to set D to the lowest possible value, this will
--      result in a INT4 vector of x'7FFFFFFF'
--
--  TEST 5 ('large' vector for INT5, Output, INT4, INT3, INT2,
--              INT1, A, B, C):
--      By doing the same as TEST 4, a large input to A, B and C,
--      results in a large INT1, INT2 and INT3. Input D of x'0001
--      results in a large INT4, and adding that to the already
--      large C gives us a large INT5. By setting C to x'FFFA'
--      we can prevent an INT3,4,5,O overflow and thus can expect
--      as INT5 vector of x'FFFCFFFD'.
--
--  TEST 6 ('large' vector for A) [ERRONEOUS TEST]:
--      We can test a large vector for A, B and C with x'FFFF',
--      this test will result in an overflow in INT3 and thus
--      will fail.

constant test_vectors : test_vector_array := (
    -- IN A     IN B     IN C     IN D     Output
    (X"FFFF", X"0001", X"0001", X"FFFF", X"00000009"),  -- TEST 1
    (X"0001", X"FFFF", X"FFFF", X"FFFF", X"00020003"),  -- TEST 2
    (X"FFFF", X"FFFF", X"FFFE", X"FFFF", X"00020004"),  -- TEST 3
    (X"5555", X"FFFF", X"FFFF", X"0002", X"80008004"),  -- TEST 4
    (X"FFFF", X"FFFF", X"FFFA", X"0001", X"FFFD0002"),  -- TEST 5
    (X"FFFF", X"FFFF", X"FFFF", X"0002", X"80018003")   -- TEST 6
);
```

```
begin

UUT : entity work.algorithm
    PORT MAP (
        clk => clk,
        rst => rst,
        A => InputA,
        B => InputB,
        C => InputC,
        D => InputD,
        O => Output
    );


clk_process : process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;

set_inputs: process
begin
    wait for 500ns;
    wait until falling_edge(clk);

    -- Initialise inputs
    rst <= '0';
    InputA <= X"0000";
    InputB <= X"0000";
    InputC <= X"0000";
    InputD <= X"0001";

    -- Initial global reset
    wait for clk_period*2;
    rst <= '1';
    wait for clk_period*2;
    rst <= '0';

    -- Test pattern input loop, inputting at every clock cycle
    for i in test_vectors'range loop
        InputA <= test_vectors(i).InputA;
        InputB <= test_vectors(i).InputB;
        InputC <= test_vectors(i).InputC;
        InputD <= test_vectors(i).InputD;
        wait for clk_period;
    end loop;
    wait;
end process;

check_outputs: process
begin
    wait for 500ns;
    wait until falling_edge(clk);

    wait for clk_period*latency;

    wait for clk_period*2;
    wait for clk_period*2;
```

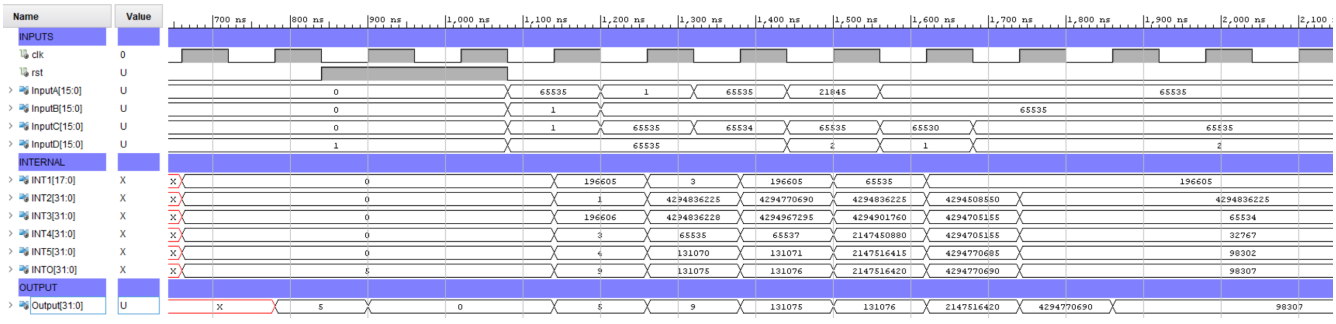
```
    -- Test pattern check loop, checking at every clock cycle
    for i in test_vectors'range loop
        assert (( Output = test_vectors(i).Output ))
        report "TEST VECTOR " & integer'image(i) &
               " A = " &
               integer'image(to_integer(unsigned(InputA))) &
               " B = " &
               integer'image(to_integer(unsigned(InputB))) &
               " C = " &
               integer'image(to_integer(unsigned(InputC))) &
               " D = " &
               integer'image(to_integer(unsigned(InputD))) &
               " O_observed = " &
               integer'image(to_integer(unsigned(Output))) &
               " O_expected = " &
               integer'image(to_integer(unsigned(test_vectors(i).Output)))
        severity failure;

        report "TEST VECTOR " & integer'image(i) & " PASS."
        severity note;
        wait for clk_period;
    end loop;
    wait;
end process;

end Behavioral;
```

1.1.2 Behavioural Simulation

Waveform 1: Global Initialisation/Reset



Waveform 2: Test Sequence



Console Output

Tcl Console | Messages | Log

```
restart
INFO: [Simtcl 6-17] Simulation restarted
run 6 us
Note: TEST VECTOR 0 PASS.
Time: 800 ns  Iteration: 0  Process: /algorithm_tb/check_outputs  File: C:/Users/[redacted]/Documents/GitHub/DigitalEngineering2023/Lab2/Lab2.srcs/sim_1/new/algorithm_tb.vhd
Note: TEST VECTOR 1 PASS.
Time: 850 ns  Iteration: 0  Process: /algorithm_tb/check_outputs  File: C:/Users/[redacted]/Documents/GitHub/DigitalEngineering2023/Lab2/Lab2.srcs/sim_1/new/algorithm_tb.vhd
Note: TEST VECTOR 2 PASS.
Time: 900 ns  Iteration: 0  Process: /algorithm_tb/check_outputs  File: C:/Users/[redacted]/Documents/GitHub/DigitalEngineering2023/Lab2/Lab2.srcs/sim_1/new/algorithm_tb.vhd
Note: TEST VECTOR 3 PASS.
Time: 950 ns  Iteration: 0  Process: /algorithm_tb/check_outputs  File: C:/Users/[redacted]/Documents/GitHub/DigitalEngineering2023/Lab2/Lab2.srcs/sim_1/new/algorithm_tb.vhd
Note: TEST VECTOR 4 PASS.
Time: 1 us  Iteration: 0  Process: /algorithm_tb/check_outputs  File: C:/Users/[redacted]/Documents/GitHub/DigitalEngineering2023/Lab2/Lab2.srcs/sim_1/new/algorithm_tb.vhd
Failure: TEST VECTOR 5 A = 65535 B = 65535 C = 65535 D = 2 O_observed = 98307 O_expected = -2147385341
Time: 1050 ns  Iteration: 0  Process: /algorithm_tb/check_outputs  File: C:/Users/[redacted]/Documents/GitHub/DigitalEngineering2023/Lab2/Lab2.srcs/sim_1/new/algorithm_tb.vhd
$finish called at time : 1050 ns : File "C:/Users/[redacted]/Documents/GitHub/DigitalEngineering2023/Lab2/Lab2.srcs/sim_1/new/algorithm_tb.vhd" Line 173
```

1.1.3 Design Runs - WNS

Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								897	96	0.00	0	0
impl_1	constrs_1	route_design Complete!	29.183	0.000	0.610	0.000	0.000	0.108	0	886	96	0.00	0	0

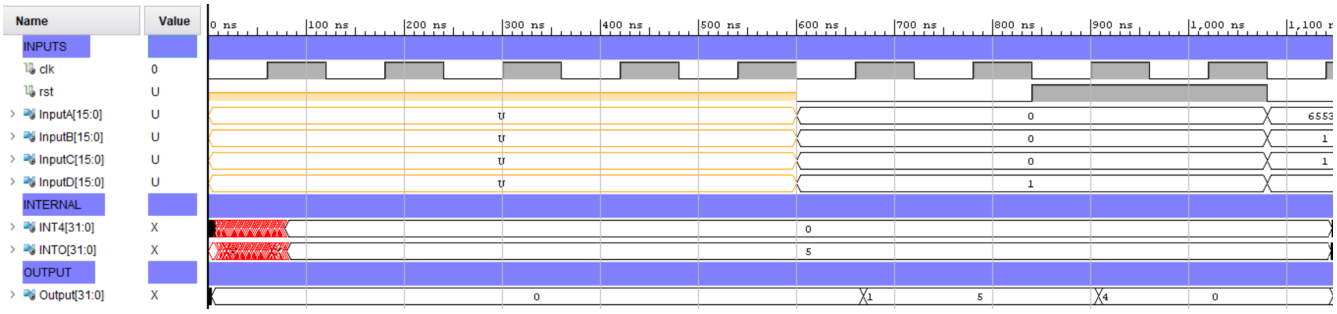
The implementation process consists of 3 steps: 'Translate', where the netlist and constraints are merged into an Xilinx design file. 'Map', where the design is implemented on the target device. And finally, 'place and route', where the components are placed and routes are designed according to the timing constraints. This process incorporates proprietary and classified algorithms that rely on random factors, therefore, the same design will never be implemented twice.

After the 'place and route' step is complete, the tools will be able to provide an estimate of all the propagation delays for all signals within the entire device after the 'static timing analysis' step. With the timing information, the tools can calculate the 'worst negative slack' (WNS).

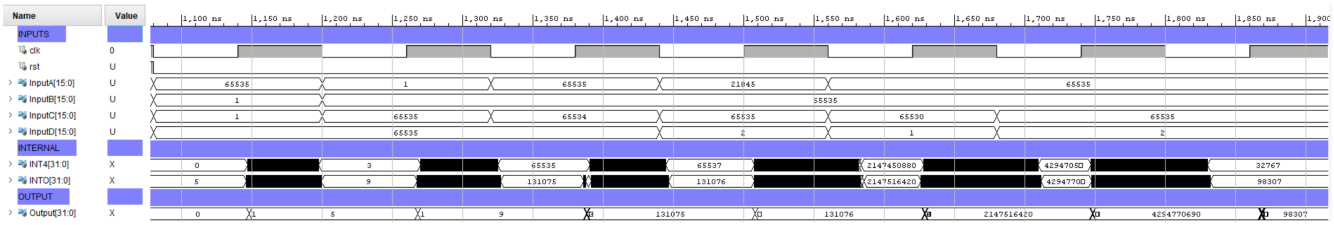
From the WNS value we got from our implementation, we can run our circuit with a clock signal as fast as 90.817ns, or around 11.01MHz.

1.1.4 Timing Simulation (120ns Clock)

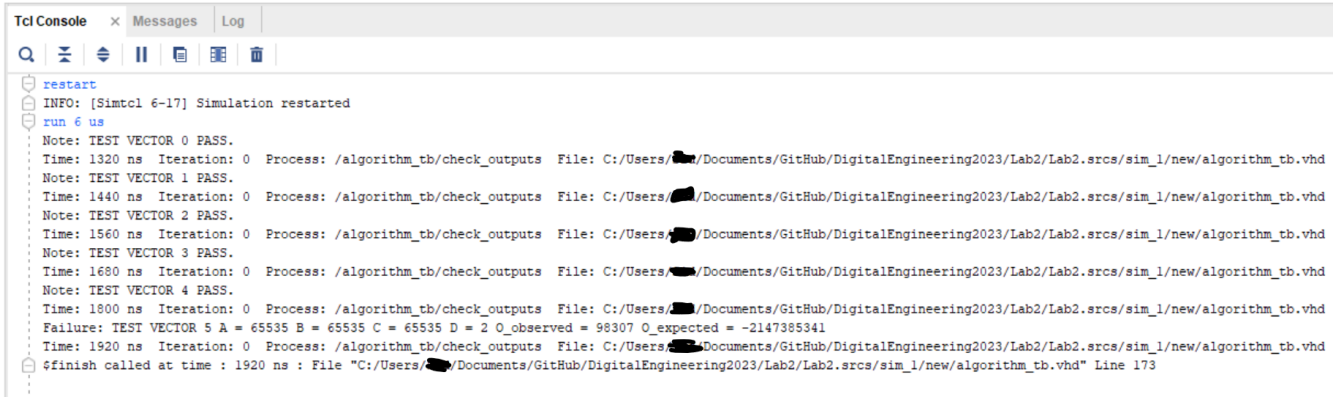
Waveform 1: Global Initialisation/Reset



Waveform 2: Test Sequence

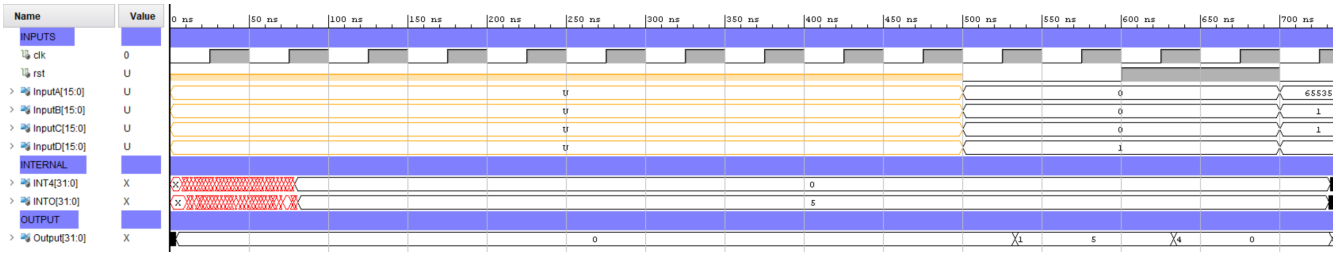


Console Output

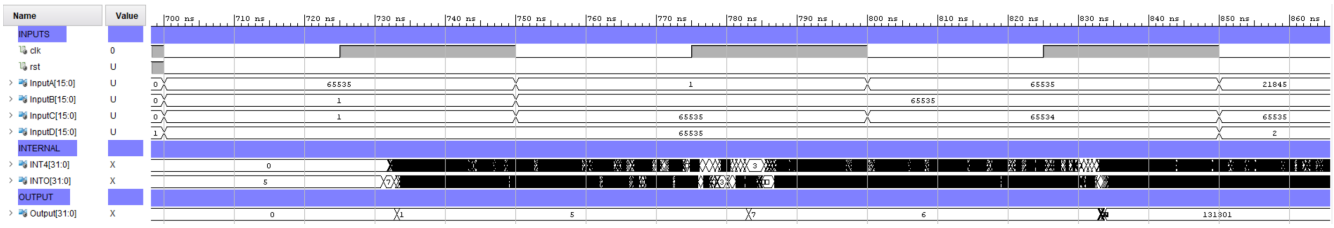


1.1.5 Timing Simulation (50ns Clock)

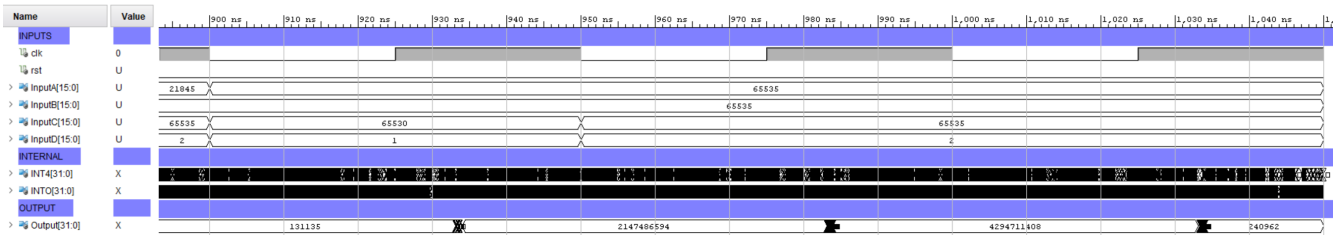
Waveform 1: Global Initialisation/Reset



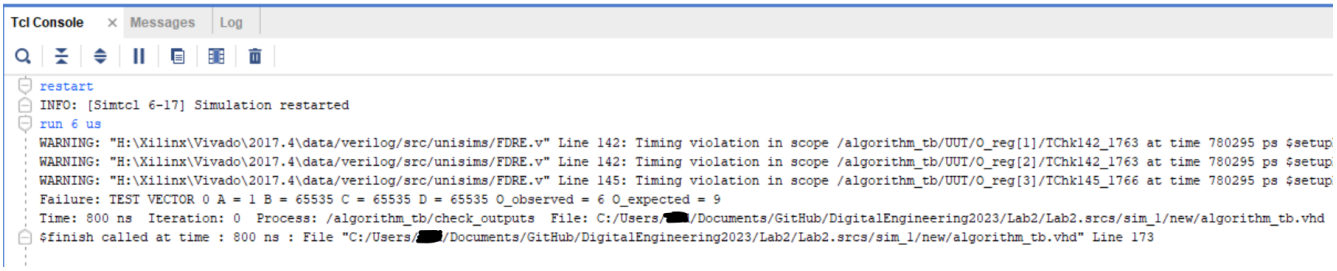
Waveform 2: Test Sequence (Vector #1-3)



Waveform 3: Test Sequence (Vector #3-6)



Console Output



1.1.6 Analysis of Results

Question 1

During the synthesis stage to generate the netlist, the tools will process the project VHDL files to check syntax and optimise designs. Due to this optimisation step, the tools will optimise across all hierarchy levels (unless it's explicitly told not to do so), therefore, hierarchy is often destroyed.

As the implementation stage needs to be run for the timing simulations, these simulations are based on the netlist. Therefore, we can only see the INT4 and INTO internal signals, all others have been destroyed by the tools for optimisation purposes.

Question 2

As propagation delays aren't taken into account with a simple behavioural simulation, we can notice everything changes at the exact edge of the clock. The post-implementation timing simulation on the other hand takes into account of these delays, therefore, changes only happen once the signal has propagated and before that, intermediate transient values are produced due to unequal propagation.

Comparing the 'INTO' signals, we see that the value changes to the correct one only after half a clock cycle of metastability, this is due to unequal propagation delays. The 'Output' signal is slightly delays relative to the rising clock edge, and there seems to be an intermediate transient value, likely caused by the metastability of 'INTO'.

Question 3

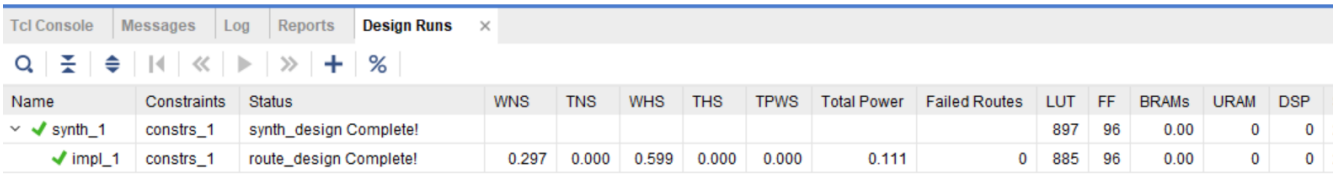
In the simulation results from 1.1.4, we say 'INTO' metastability lasting only around half a clock cycle, in the waveform from 1.1.5, we see metastability throughout INTO. We can also see the increase in metastability as the simulation progresses; in Waveform 2 (Vector #1-3), there are a few moments where the value stays fixed for a nanosecond or two, but in Waveform 3 (Vector #3-6), the output for INTO turns into a dark black rectangle.

The behaviour for the Output signal is also quite consistent with the one from 1.1.4, the only difference being the Output is incorrect (due to the metastable INTO). But we can notice as INTO metastability increases in Waveform 2, there seems to be an increase in metastable transient values between register outputs.

Task B: Tool Optimisations

1.2.1 Design Runs With Varying Timing Contraints

80ns Clock Period



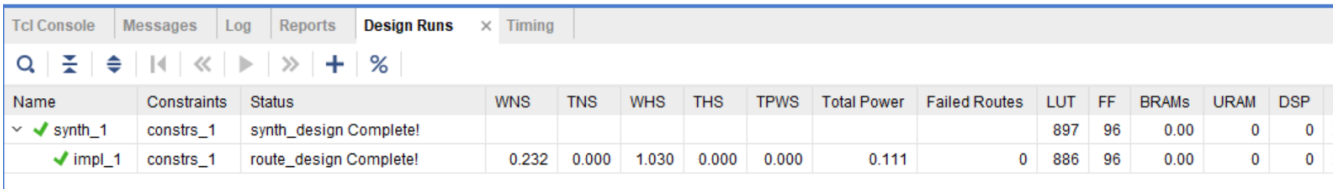
Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								897	96	0.00	0	0
impl_1	constrs_1	route_design Complete!	0.297	0.000	0.599	0.000	0.000	0.111	0	885	96	0.00	0	0

From the WNS value we got from our implementation with an 80ns clock, we can run our circuit with a clock signal as fast as 79.703ns, or around 12.55MHz. With an 80ns clock signal in the testbench, we can indeed verify that the implementation works as the self-checking testbench passes.

When the clock period is changed in the constraints file (.XDC), Xilinx has to recalculate the WNS. If the clock period is reduced, the logic will have less time to execute before the next clock edge arrives, this can reduce the WNS value.

75ns Clock Period

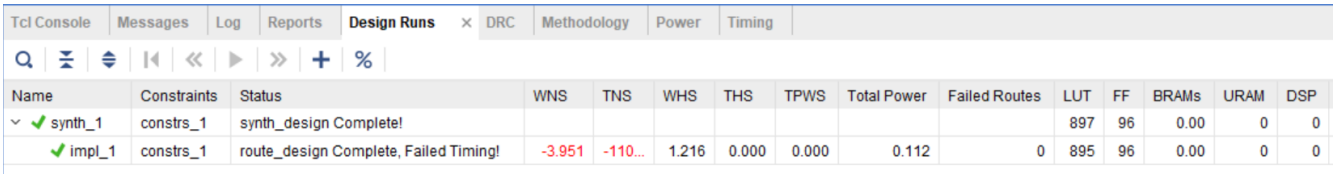


Tcl Console | Messages | Log | Reports | Design Runs | Timing

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								897	96	0.00	0	0
impl_1	constrs_1	route_design Complete!	0.232	0.000	1.030	0.000	0.000	0.111	0	886	96	0.00	0	0

From the WNS value we got from our implementation with a 75ns clock, we can run our circuit with a clock signal as fast as 74.268ns, or around 13.37MHz. Running the testbench with a 75ns simulated clock period, we can notice a fail in test vector #4, where the inputs, A, B and C are 65535, and D is 2. Test vectors #0-3 all pass. This can be explained as the WNS is only 0.232ns, this is such a small margin to work with and due to propagation delays and other limitations of a physical (real-life) device, this margin cannot be maintained.

70ns Clock Period



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								897	96	0.00	0	0
impl_1	constrs_1	route_design Complete, Failed Timing!	-3.951	-110...	1.216	0.000	0.000	0.112	0	895	96	0.00	0	0

This time, Xilinx couldn't meet the timing requirements that we set. With the displayed value, the maximum clock period has to be 73.951ns, or around 13.52MHz. It is however surprising that, just like for the 75ns clock period above, test vectors #0-3 are passing, and only vector #4 is failing instead of all of them.

Explanation

As there are an infinite number of device arrangements for the logic, the process of implementation is an iterative one. The Xilinx tools will iterate until an implementation that complies with the defined timing constraints is found. After it has tried hard enough (iterated a maximum defined number of times), the tools may not be able to find a configuration that complies with the constraints, in this case, it will display a negative value in red, meaning that a valid clock period where the circuit will work is off by that value. Running a timing simulation when the timing is not met results in a lot of metastabilities within the internal signals, which leads to inaccurate and extremely unstable outputs and failing testbenches.

1.2.2 Setup Violation and Critical Path

Setup time relates to the short duration required for a signal to be present before the clock edge for a sequential device. When this condition is not met, a setup violation will arise. To mitigate this violation, the clock period can be decreased, allowing for more time for the clock signal to propagate, or the delay of the incoming data path logic can be reduced. This will result in an increase in the critical path.

1.2.3 Post-Route Timing Report: Max Delay Path

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (VIOLATED) :        -3.951ns  (required time - arrival time)
  Source:                 INTC_reg[2]/C
                            (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns fall@35.000ns period=70.000ns})
  Destination:            O_reg[29]/D
                            (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns fall@35.000ns period=70.000ns})
  Path Group:             clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            70.000ns  (clk rise@70.000ns - clk rise@0.000ns)
  Data Path Delay:        74.024ns  (logic 48.542ns (65.576%)  route 25.482ns (34.424%))
```

Task C: Logic Optimisations

1.3.1 Modified Algorithm Circuit

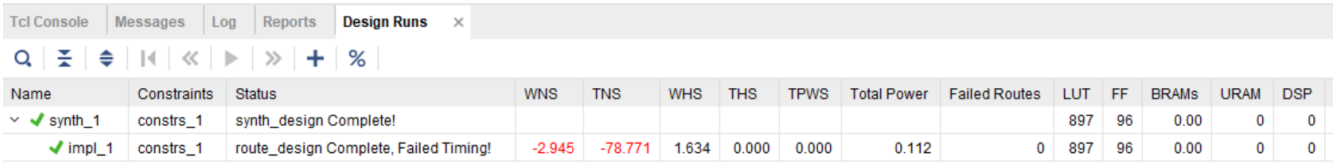
Modified VHDL Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Entity description:
-- The entity implements, with no optimization, a sequence of operations:
--         O <= (A*3 + B*C)/D + C + 5
-- where A,B,C, and D are UNSIGNED vectors of parameterizable size

-- Note 1: There is no particular "meaning" to the equation - it is designed for
--  experimentation with logic optimization for performance
-- Note 2: There is no provision for overflow. Some input vectors can cause
--  overflow and the result will be incorrect.
-- Note 3: Inputs and outputs are registered (rising edge, synchronous reset).
--  This introduces a latency of 2 clock cycles between inputs and outputs.
-- Note 4: D is the divisor in one of the operations, so can never have value 0

entity algorithm is
    generic (data_size : integer := 16); -- defines the size of the data
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           -- The four (parameterizable) data inputs
           A : in  STD_LOGIC_VECTOR (data_size-1 downto 0);
           B : in  STD_LOGIC_VECTOR (data_size-1 downto 0);
           C : in  STD_LOGIC_VECTOR (data_size-1 downto 0);
           D : in  STD_LOGIC_VECTOR (data_size-1 downto 0);
           -- Output = (A*3 + B*C)/D + C +5
           O : out  STD_LOGIC_VECTOR (data_size*2-1 downto 0)
           );
end algorithm;

architecture Behavioral of algorithm is

-- Registered inputs
signal INTA, INTB, INTC, INTD : UNSIGNED (data_size-1 downto 0);
-- Internal signals for intermediate operations (note the sizes)
signal INT1 : UNSIGNED (data_size+1 downto 0);   -- INTA + 3
signal INT2 : UNSIGNED (data_size*2-1 downto 0); -- INTB * INTC
signal INT3 : UNSIGNED (data_size*2-1 downto 0); -- INT1 + INT2
signal INT4 : UNSIGNED (data_size*2-1 downto 0); -- INT3 / INTD
-- INT5 needs to be resized to match INTC+1, as this vector size
-- is larger than the required vector size for a decimal '5'
signal INT5 : UNSIGNED (data_size downto 0);   -- [UPDATED] INTC + 5
```

```
-- INTO needs to be resized to match INT4, as this is the larger
-- vector size compared to INT5 as we updated that to be smaller
signal INTO : UNSIGNED (data_size*2-1 downto 0); -- [UPDATED] INT4 + INT5

begin

-- Input registers (D-type, rising edge, synchronous reset)
input_regs: process (clk) is
begin
  if rising_edge(clk) then
    if rst = '1' then
      INTA <= (others => '0');
      INTB <= (others => '0');
      INTC <= (others => '0');
--    INTD <= (0 => '1', others => '0');  -- aggregate notation
      INTD <= to_unsigned(1,INTD'length); -- type conversion notation
    else
      INTA <= unsigned(A);
      INTB <= unsigned(B);
      INTC <= unsigned(C);
      INTD <= unsigned(D);
    end if;
  end if;
end process input_regs;

-- Mathematical operations on the data (combinational)
INT1 <= INTA * to_unsigned(3, 2);
INT2 <= INTB * INTC;
INT3 <= INT1 + INT2;
INT4 <= INT3 / INTD;
-- This logic has been moved forward along side the multiplication
-- so that the adders aren't exactly one after the other operations
-- in INT1 and INT2. In the previous design, this logic would've
-- been carried out by INTO, we need to make sure that the to_unsigned
-- function has the correct size parameter (width of INTC + 1)
INT5 <= INTC + to_unsigned(5, INTC'length+1);
-- Bringing the INT5 logic forward needs to be implemented, therefore
-- INTO has been adapted to take into account this change. Critical
-- path is reduced as the INT5 adder is along side the rest of the
-- logic, INTO doesn't have to wait for INT5 like before in the
-- pre-modifications design.
INTO <= INT4 + INT5;

-- Input registers (D-type, rising edge, synchronous reset)
output_regs: process (clk) is
begin
  if rising_edge(clk) then
    if rst = '1' then
      O <= (others => '0');
    else
      O <= std_logic_vector(INTO);
    end if;
  end if;
end process output_regs;


end Behavioral;
```

Design Runs

70ns Clock Period

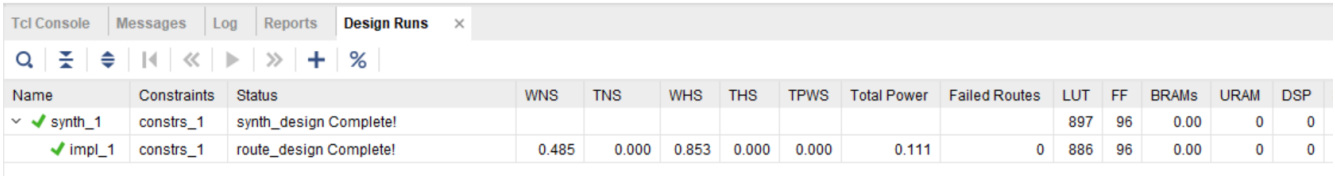


Tcl Console | Messages | Log | Reports | **Design Runs**

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								897	96	0.00	0	0
impl_1	constrs_1	route_design Complete, Failed Timing!	-2.945	-78.771	1.634	0.000	0.000	0.112	0	897	96	0.00	0	0

Although this timing constraint still fails, we can see an improvement of almost 1ns when compared to the pre-modification case.

80ns Clock Period



Tcl Console | Messages | Log | Reports | **Design Runs**

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								897	96	0.00	0	0
impl_1	constrs_1	route_design Complete!	0.485	0.000	0.853	0.000	0.000	0.111	0	886	96	0.00	0	0

In this case, the timing constraint passes. When compared to the pre-modification case, we can notice a drastic improvement of 189ns.