Verification

Last updated 5/14/20

Verification

These slides outline the verification processes used with out FPGA designs

Upon completion: You should be able to detail the types of verification that can implemented and write simple brute force testbenches

Verification

- Verification Strategy
  - Register Transfer Level (RTL) Simulation
    - Verifies functionality
    - Pre synthesis
    - Based on code only
  - Gate Level Simulation
    - Verifies functionality
    - Post synthesis
    - No timing information
  - Timing Simulation
    - Verifies performance
    - Post place-and-route
    - Includes technology information
      - Process corners, temperature, voltage

Verification

- Verification Strategy
  - Component Test
    - Bench
      - Limited ability to exercise all possible situations
    - ATE
      - SCAN Chains
      - Test Vectors
  - System Test
    - Integrate component into larger system
    - Verify interfaces
    - Verify overall operation to specification

Simulation Testbenches

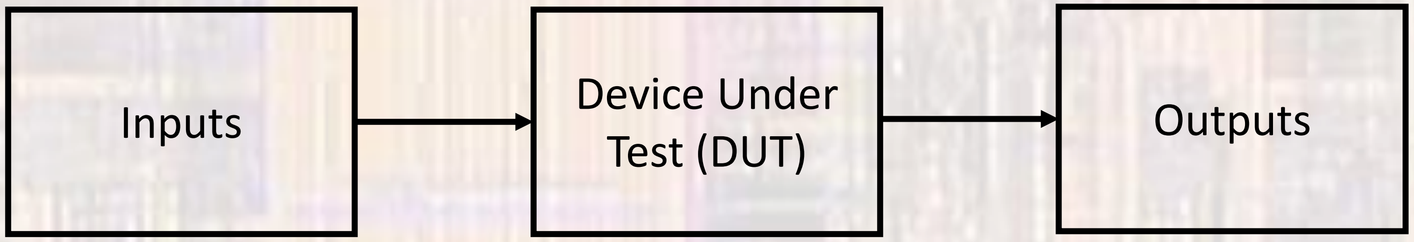
- Testbench strategy
  - Verify blocks in manageable chunks
  - Verify hierarchically

  - Re-verify after all block changes
    - Test bench does not change after it has been validated

  - Ideally – created separately from the module
    - Based on design specification

Simulation Testbenches

- General Testbench Structure



Graphical	RTL	Graphical
HDL	Gate Level	
Text File	Post Fit	Text File

Simulation Testbenches

- Manual Testbench
  - Inputs
    - HDL generated
  - Expected Outputs
    - User generated
  - Results
    - Compare waveforms to expectations
  - Limited to
    - Very small systems
    - Systems with very few inputs and outputs

Simulation Testbenches

- Automated Testbench
  - Inputs
    - HDL generated
    - Read from a file
  - Expected Outputs
    - HDL generated
      - Be careful not to use the same code
    - Read from a file
      - Be careful not to use the same code
  - Results
    - Identify errors and document
      - Test #, Time, Expected Value, Actual Value, …

Simulation Testbenches

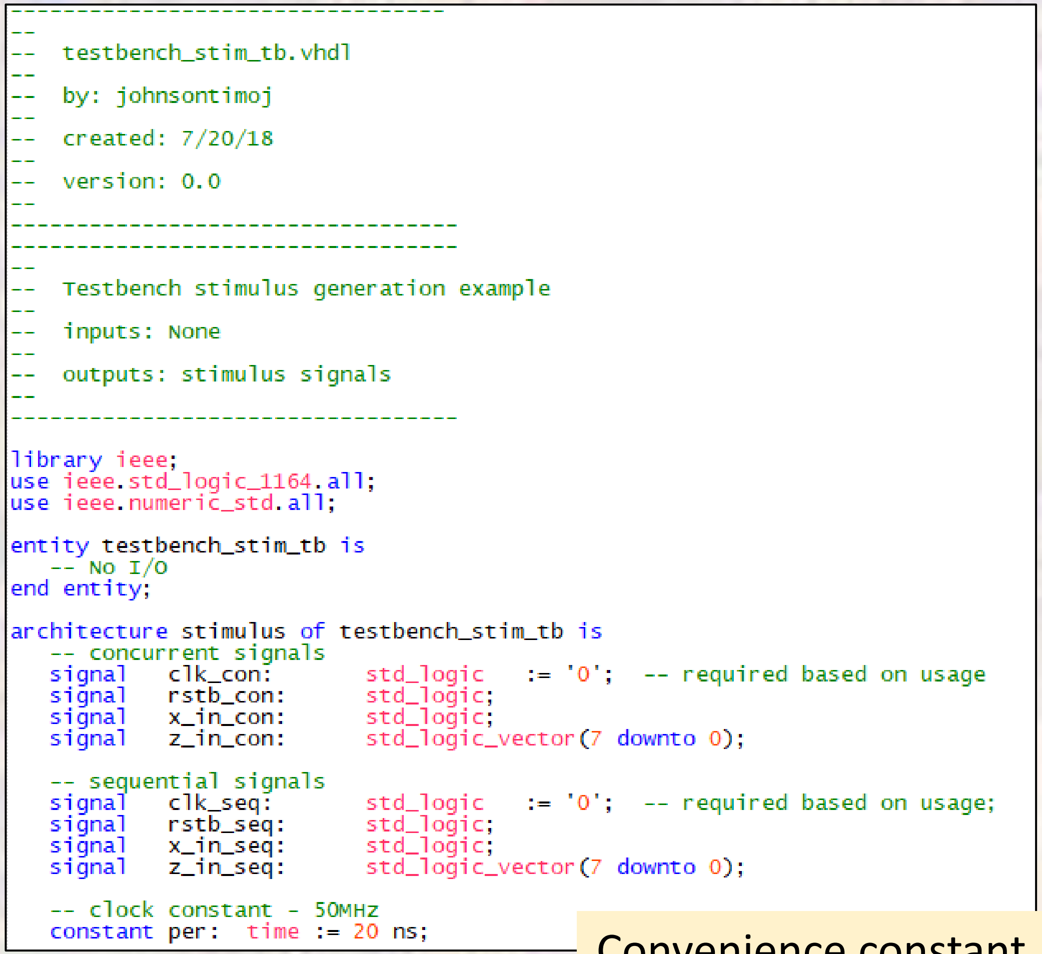
- Quartus Interaction and ModelSim
  - Test benches can contain un-synthesizable code

  - Quartus will generate errors on compilation if you set the testbench to the top level entity
    - Use the block under test as the top level entity

Simulation Testbenches

- Testbench
  - Input Signal Generation – concurrent statements

No I/O



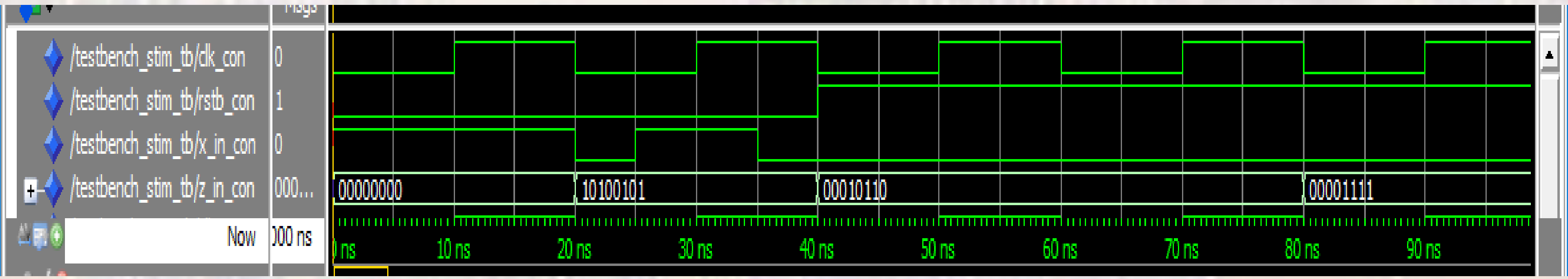
```
--------------------------------
--
--   testbench_stim_tb.vhdl
--
--   by: johnsontimoj
--
--   created: 7/20/18
--
--   version: 0.0
--
---------------------------------
---------------------------------
--
--   Testbench stimulus generation example
--
--   inputs: None
--
--   outputs: stimulus signals
--
---------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testbench_stim_tb is
   -- No I/O
end entity;

architecture stimulus of testbench_stim_tb is
   -- concurrent signals
   signal   clk_con:      std_logic   := '0';   -- required based on usage
   signal   rstb_con:     std_logic;
   signal   x_in_con:     std_logic;
   signal   z_in_con:     std_logic_vector(7 downto 0);

   -- sequential signals
   signal   clk_seq:      std_logic   := '0';   -- required based on usage;
   signal   rstb_seq:     std_logic;
   signal   x_in_seq:     std_logic;
   signal   z_in_seq:     std_logic_vector(7 downto 0);

   -- clock constant - 50MHz
   constant per:  time := 20 ns;
```

Convenience constant

Simulation Testbenches

- Testbench
  - Input Signal Generation – concurrent statements

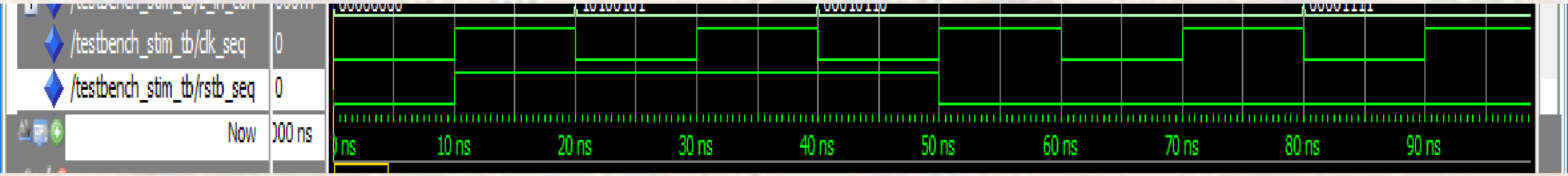
```vhdl
begin
   --
   -- concurrent assignments
   --
   -- note "after" uses continuous timing within a single statement
   --
   clk_con <= not clk_con after per/2;

   rstb_con <= '0',
               '1' after per*2;

   x_in_con <= '1',
               '0' after 20 ns,
               '1' after 25 ns,
               '0' after 35 ns;

   z_in_con <= (others => '0'),
               "10100101" after 20 ns,
               std_logic_vector(to_unsigned(22, z_in_con'length)) after 40 ns,
               std_logic_vector(to_unsigned(15, z_in_con'length)) after 80 ns;

   --
```


/testbench_stim_tb/clk_con
0
/testbench_stim_tb/rstb_con
1
/testbench_stim_tb/x_in_con
0
/testbench_stim_tb/z_in_con
000...
00000000
10100101
00010110
00001111
Now
)00 ns
) ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns

Simulation Testbenches

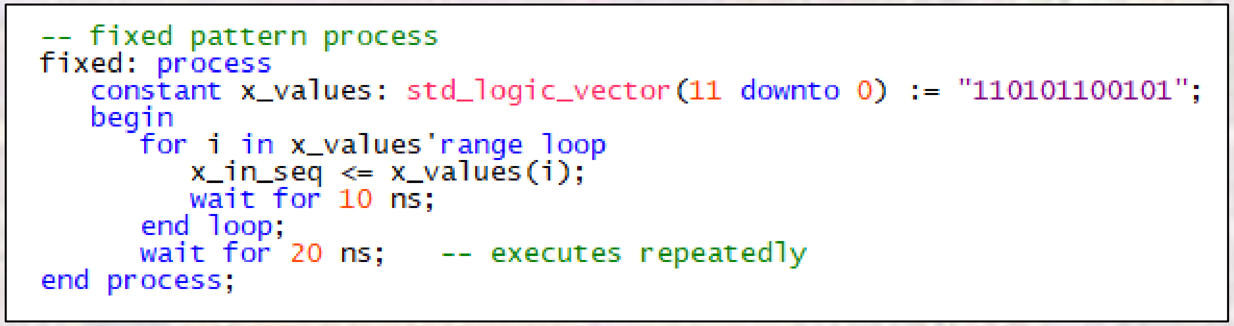
- Testbench
  - Input Signal Generation – sequential statements

```
--
-- sequential assignments
--

-- Clock Process
clk: process
   begin
      wait for per/2;
      clk_seq <= not clk_seq;
end process;

-- Reset process (active low)
rstb: process
   begin
      rstb_seq <= '0';
      wait for per/2;
      rstb_seq <= '1';
      wait for per*2;
      rstb_seq <= '0';
      wait;          -- only executes once
end process;
```

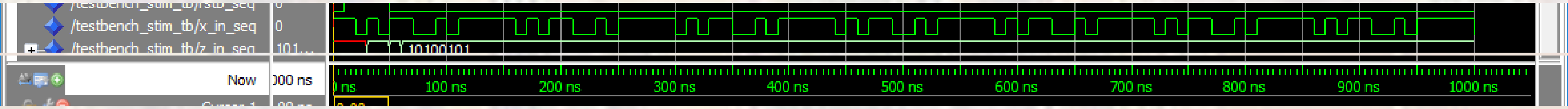


Simulation Testbenches

- Testbench
  - Input Signal Generation – sequential statements

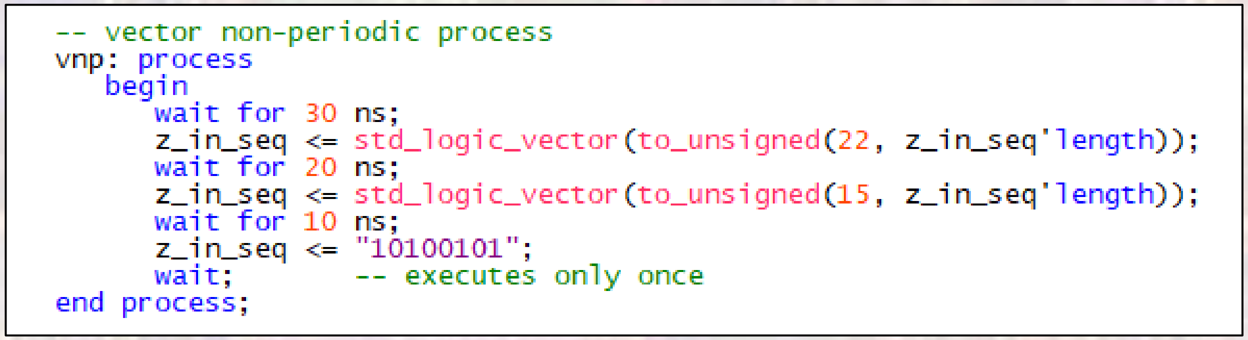


```vhdl
-- fixed pattern process
fixed: process
   constant x_values: std_logic_vector(11 downto 0) := "110101100101";
   begin
      for i in x_values'range loop
         x_in_seq <= x_values(i);
         wait for 10 ns;
      end loop;
      wait for 20 ns;    -- executes repeatedly
end process;
```

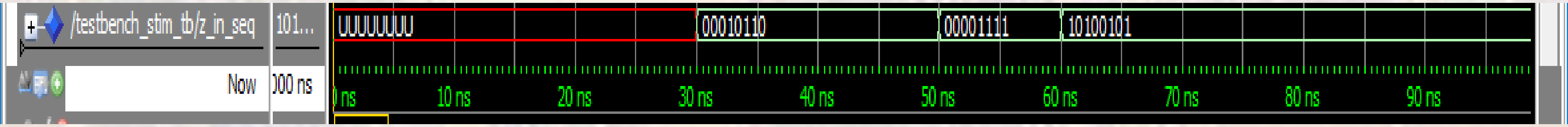


Simulation Testbenches

- Testbench
  - Input Signal Generation – sequential statements



```
-- vector non-periodic process
vnp: process
   begin
      wait for 30 ns;
      z_in_seq <= std_logic_vector(to_unsigned(22, z_in_seq'length));
      wait for 20 ns;
      z_in_seq <= std_logic_vector(to_unsigned(15, z_in_seq'length));
      wait for 10 ns;
      z_in_seq <= "10100101";
      wait;          -- executes only once
end process;
```


/testbench_stim_tb/z_in_seq
101...
UUUUUUUU
00010110
00001111
10100101
Now
)00 ns
) ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns