

Programmable Electronics using FPGAs

This week

- Recap: Procedural statements – wait and loops
- Test bench design
- Generate clock and reset
- Test bench reporting
- Test strategies
- Assessment 1: A programmable 4-bit Counter

Recap: Procedural statements – wait and loops

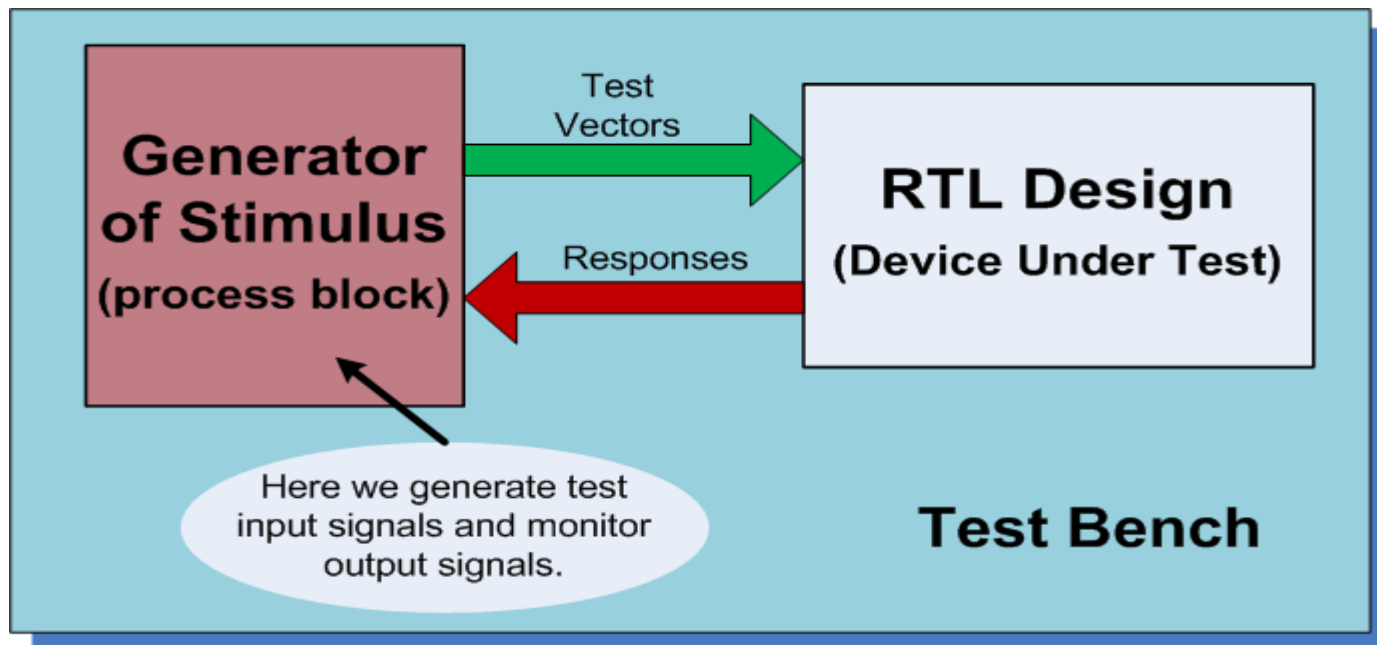
- wait is used to suspend the execution of a process and to specify a condition when a process will be resumed.
 - wait for 5 ns;
 - wait until rising_edge(Clock);
- While and for loops are used to repeat a block of code a number of times.

```
while i <= 3 loop
    wait for 20 ns;
    i <= i + 1;
end loop;
```

```
for i in 0 to 3 loop
    wait until rising_edge(Clock);
end loop;
```

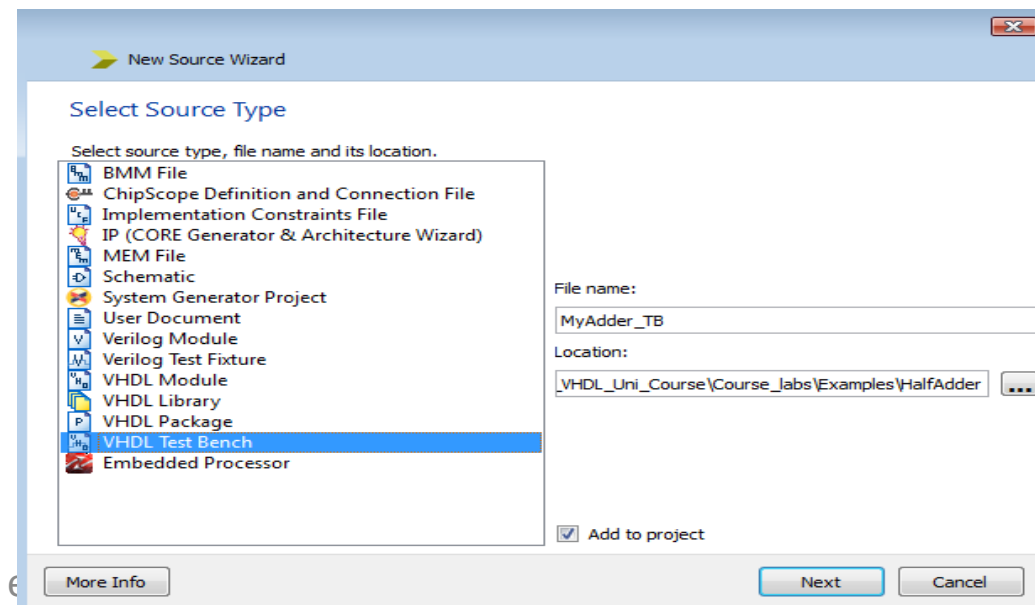
Test bench design

- Once we have written our RTL, we are not even half done!
- Testing is the most important part of development.
- We build a test bench to test and simulate our RTL design.
- Test benches are for testing, not for synthesis.
- Sometimes we start by designing our test bench!



Test bench design

- A test bench consists of:
 - An entity declaration without any ports.
 - An architecture body that includes an instance of the design under test.
 - Apply sequences of test values to input signals.
 - Monitors values of output signals – in a simulator or within a process.
- We can generate a template
 - Create a *New Source* as a VHDL Test Bench.



Test bench design - structure

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MyRTL_TB IS
END MyRTL_TB;

ARCHITECTURE TB of MyRTL_TB IS
COMPONENT MyRTL IS          -- The Design Under Test
    PORT (
        input, clock : IN STD_LOGIC;
        output : OUT STD_LOGIC);
END COMPONENT;

signal x, clk : STD_LOGIC := '0';
Begin
    DUT: MyRTL port map (input => x, output => y, clock => clk);

    clk <= not clk after 10 ns;  -- Generates 50 MHz clock signal

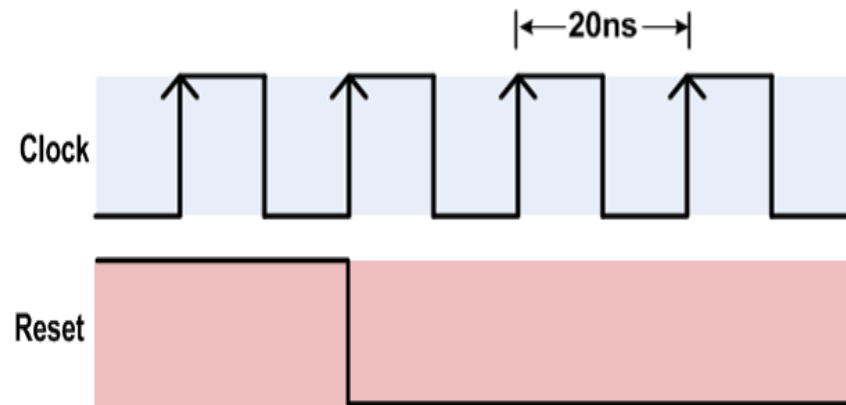
    Stimulus: process
        begin
            -- Apply sequences of test values to input signals x, clk.
            -- Monitors values of output signal y.
            wait;
        end process;

end TB;
```

Generated clock and reset

- clock and reset signals are generated in a test bench for testing a sequential circuit design.
- clock is a periodic signal which alternates high and low at regular time intervals.

- clock `<= NOT clock` after 10 ns;
- Clock signal must be initialized normally to 0, sometimes to 1;



- reset signal can be either
 - Asynchronous reset is not dependent on a clock
 - Synchronous reset time must be equal or greater than a clock cycle.

```
Stimulus: process
begin
  Reset <= '1';
  wait for 30 ns;
  Reset <= '0';

  -- Generate test patterns
  wait;
end process;
```

Test bench reporting

- There are reporting features that allow us to monitor the output of a design.
- Report keyword
 - The keyword report will always print a string.
 - Used for outputting the process of a test.

```
Stimulus: process begin
    A <= '1';
    B <= '1';
    wait for 10 ns;
    if Sum = '0' AND Cout = '1' then
        report "OK";
    else
        report "ERROR";
    end if;
end process;
```


Test bench reporting

➤ Assert keyword

- The keyword assert will evaluate a Boolean expression.
- If the expression is false, it will print the report.
- Severity levels may also reported with the predefined values ERROR, WARNING, NOTE, or FAILURE.
- FAILURE causes a break point

```
Stimulus: process
begin
    A <= '1' ;
    B <= '1' ;
    wait for 10 ns;
    assert(Sum = '0' AND Cout = '1') report "Test Failed!" severity FAILURE;
    assert(Sum /= '0' OR Cout /= '1') report "Test Passed!";

    wait;
end process;
```

Test strategies: Functional testing

- Two test strategies are commonly used:
 - Functional testing
 - Regression testing
- Functional testing is where the input/output combinations are checked against known results.
 - We generate sequences of test values to input signals (test vectors) and then monitor the expected results of output signals in a simulator or within a process.
 - Manual way of testing – you have to ensure you have checked every combination of inputs.

Example: Functional testing

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

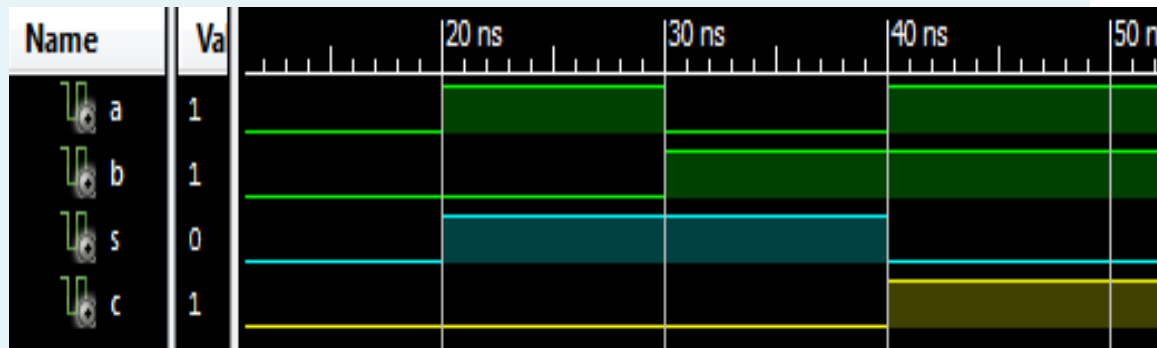
ENTITY MyAdder_TB IS
END MyAdder_TB;

ARCHITECTURE TB of MyAdder_TB IS
COMPONENT MyAdder IS      -- This is Design Under Test
    PORT ( A, B          : IN STD_LOGIC;
           Sum, Cout     : OUT STD_LOGIC);
END COMPONENT;

SIGNAL A, B : STD_LOGIC := '0'; SIGNAL index : STD_LOGIC_VECTOR(2 downto 0);
Begin
    DUT: MyAdder port map (A => A, B => B, Sum => Sum, Cout => Cout);

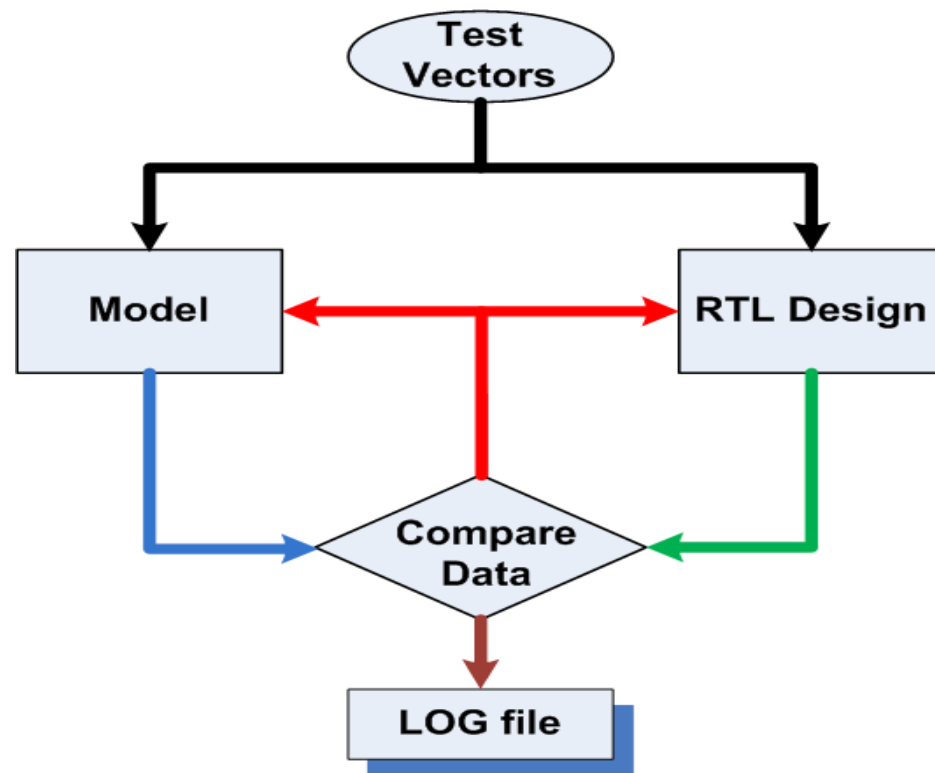
    Stimulus: process begin
        while index <= 3 loop
            A <= index(0);
            B <= index(1);
            wait for 10 ns;
            index <= index + 1;
        end loop;
        wait;
    end process;
end TB;

```



Test strategies: Regression testing

- Regression testing is about comparing the model (which is assumed to be correct) with the RTL design outputs.
- Comparing the specification (model) with the implementation (RTL).
- Sometimes the model may be implemented in VHDL, so we use 2 instances in test bench – one for RTL design and one for Model.



Example: Regression testing

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL; USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY MyAdder_TB IS
END MyAdder_TB;

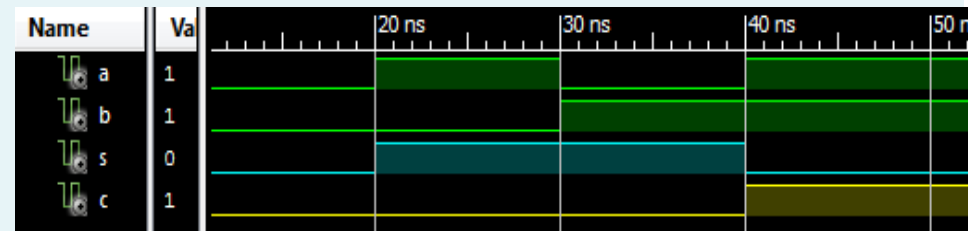
ARCHITECTURE TB of MyAdder_TB IS
COMPONENT MyAdder IS          -- This is Design Under Test
    PORT ( A, B      : IN STD_LOGIC;
           Sum, Cout : OUT STD_LOGIC);
END COMPONENT;

SIGNAL A,B : STD_LOGIC := '0'; SIGNAL index,result: STD_LOGIC_VECTOR(2 downto 0);
Begin

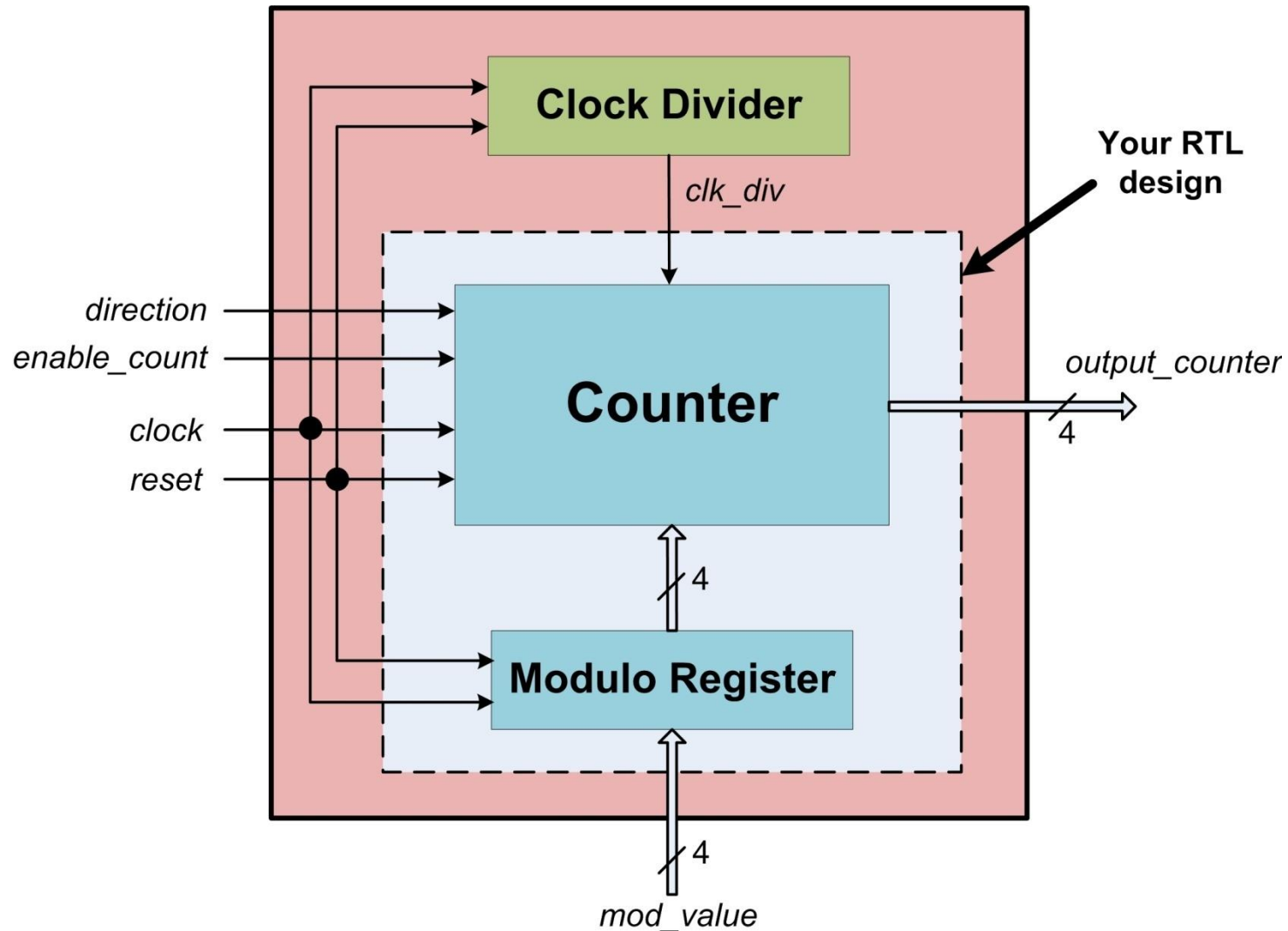
    DUT: MyAdder port map (A => A, B => B, Sum => Sum, Cout => Cout);

    Stimulus: process begin
        while index <= 3 loop
            A <= index(0); B <= index(1);
            wait for 10 ns;
            result <= ('0' & A) + ('0' & B);
            wait for 1 ns;
            if result(0) = Sum AND result(1) = Cout then
                report "Test Passed!";
            else
                report "Test Failed!" severity ERROR;
            end if;
            index <= index + 1;
        end loop;
    end process;
end TB;




```



Assessment 1: A programmable 4-bit Counter Design



Assessment 1: A programmable 4-bit Counter Design

clock	reset	enable_count	direction	mod_value	output_counter	Description
-	1	-	-	<i>Modulo Value</i>	0	Output will be zero and counter's upper limit sets to <i>modulo value</i>
	0	1	0	-	<i>Counter++</i>	Incremented by 1 where upper limit is <i>modulo value</i>
	0	1	1	-	<i>Counter--</i>	Decrement by 1 where upper limit is <i>modulo value</i>
	0	0	-	-	<i>Counter</i>	No change

Assessment 1: points to note

- Details and instructions available on Blackboard next week
- Clock Divider is supplied
- You implement Counter and Modulo Register
 - Test bench
 - Report
- Individual exercise!
- Submitted via Blackboard.