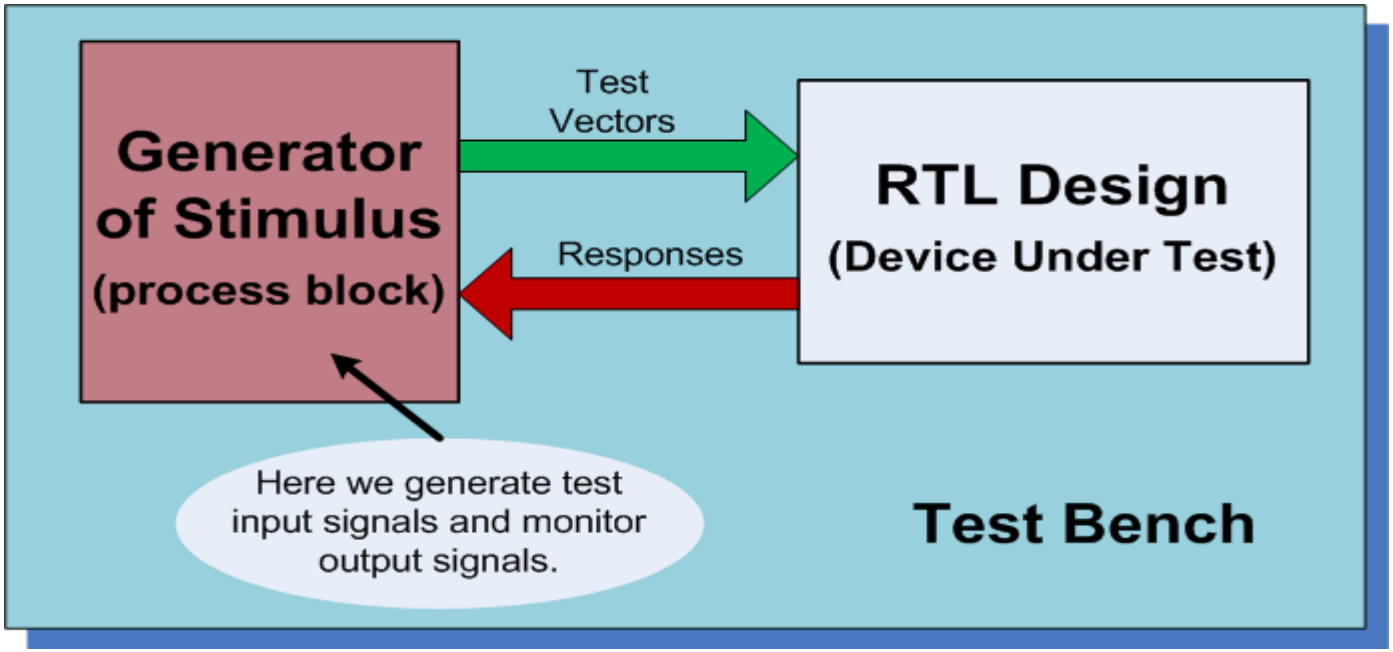
Programmable Electronics using FPGAs

This week

- Recap: Test bench design
- Data type conversions
- DŜyŜǋʎŏ ŏƶΎΎl·yŘ
- Customized design approach
- Clock divider
- Assessment 1: A programmable 4-bit Counter

Recap: Test bench design

- Once we have written our RTL, we are not even half done.
- Testing is the most important part of the design.
- 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!



Data types and data type conversions

- A data type defines:
  - the range of values that an object can have and
  - the set of operations that may be performed on it.
- VHDL is a strongly typed language – we cannot assign a value of one data type to a signal of a different data type
- Unless we explicitly convert values from one data type into values for another data type.
- This conversion needs to be well-defined.

Data type conversions: examples

- signal A : integer := 0;
- signal B1 : real := 3.14; signal B2 : real := 3.50; signal B3 : real := 3.85;
- signal C : signed (3 downto 0) := “1011”; -- value is -5
- signal D : unsigned (3 downto 0) := “0000”;

Wrong!

```
A <= B1;
```

A = ?

Correct!

```
A <= integer (B1);
```
A = 3

```
A <= integer (B2);
```
A = 4

```
A <= integer (B3);
```
A = 4

Wrong!

```
D <= C;
```

D = ?

Correct!

```
D <= unsigned (C);
```

D = 1011

Data type conversions: examples

- signal E : integer := 1024;
- signal F : unsigned (7 downto 0) := “11110000”;
- signal G : signed (3 downto 0) := “1111”;
- signal H : std_logic_vector (7 downto 0) := “11111111”;

Wrong!

```
G <= F;
```

G = ?

Correct!

```
G <= signed (F(3 downto 0));
```

G = 0000

Wrong!

```
H <= E;
```

H = ?

Correct!

```
H <= std_logic_vector (to_unsigned(E,8));
```

H = 00000000

Customising designs

- Sometimes we need to instantiate a component several times in our design with different specifications.
  - For example a RAM component can be used in different sizes, such as 2K x 8 or 3K x 16
- Generic keyword allows us to parameterise entities
  - With optional default values
- Parameters may be, for instance:
  - Customize timing,
  - Alter range of subtypes,
  - Change size of arrays.

A customized clock divider

- The clock divider you have been given for assignment 1 is an example of this.
- This component generates a pulse signal (*Clk_Div*) for one clock cycle when its counter reaches the LIMIT argument.

```
Entity Clock_Divider is
      GENERIC ( LIMIT : integer := 2 );
      PORT ( Clock, Reset  : IN    STD_LOGIC;
                    Clk_Div : OUT STD_LOGIC );
end Clock_Divider;
architecture Behavioral of Clock_Divider is
signal cnt : STD_LOGIC_VECTOR  (31 downto 0);
Begin
PC: process (Clock, Reset)  begin
      if rising_edge(Clock) then
          if Reset = '1' then
                cnt <= (others => '0');
                Clk_Div <= '0';
          elsif cnt = LIMIT-1 then
                cnt <= (others => '0');
                Clk_Div <= '1';
          else
                cnt <= cnt+1;
                Clk_Div <= '0';
          end if;
      end if;
end process;  end Behavioral;
```

A customized clock divider

- This shows how we can use the customized component.
- Remember that you have to use the supplied customized component in assessment 1!

```
Entity TOP is
    PORT ( Clock, Reset, Inputs  : IN  STD_LOGIC;
                         Outputs : OUT STD_LOGIC );
end TOP;
architecture Behavioral of TOP is
    component Clock_Divider is
        GENERIC ( LIMIT : integer := 2 );
        PORT ( Clock    : IN  STD_LOGIC;
               Reset    : IN  STD_LOGIC;
               Clk_Div  : OUT STD_LOGIC );
    end component;
signal Clk_Div  : STD_LOGIC;
Begin
    Clock_Divider GENERIC MAP (LIMIT => 50000000)
                  PORT MAP ( Clock   => Clock,
                             Reset   => Reset,
                             Clk_Div => Clk_Div );

    -- Your Logic will be here

end Behavioral;
```

Assessment 1: A programmable 4-bit Counter Design