Traineeships in Advanced Computing for High Energy Physics (TAC-HEP)

GPU & FPGA module training: Part-2

Week-7: Introduction to VHDL

Lecture-14: May 3rd 2023

Varun Sharma

University of Wisconsin – Madison, USA

So Far…

- ✓ **FPGA and its architecture**
  - Registor/Flip-Flops, LUTs/Logic Cells, DSP, BRAMs
  - Clock Frequency, Latency
  - Extracting control logic & Implementing I/O ports
- ✓ **Parallelism in FPGA**
  - Scheduling, Pipelining, DataFlow
- ✓ **Vivado HLS**
  - Introduction, Setup, Hands-on for GUI/CLI, Introduction to Pragmas
  - Different Pragmas and their effects on performance
  - Practices to follow while writing HLS code – do's & don'ts
- ✓ **LHC and CMS Experiment: Level-1 Trigger System**
- ✓ **Project: Clustering algorithm for Regional Calorimeter Trigger**

Today:
- Questions related to Project
- Introduction to VHDL

Project – Question/Concerns?

Write an algorithm to cluster ECAL and HCAL energies for Regional Calorimeter Trigger using HLS and synthesis the results

1. Input per tower (ECAL + HCAL)
2. Cluster ECAL energies for each tower
   - Divide the RCT card further to make life simple

❖ Provide Input for algo

3. Stitch together the clusters for neighbouring towers
4. Sort the final list
5. Send just 12 towers per RCT region

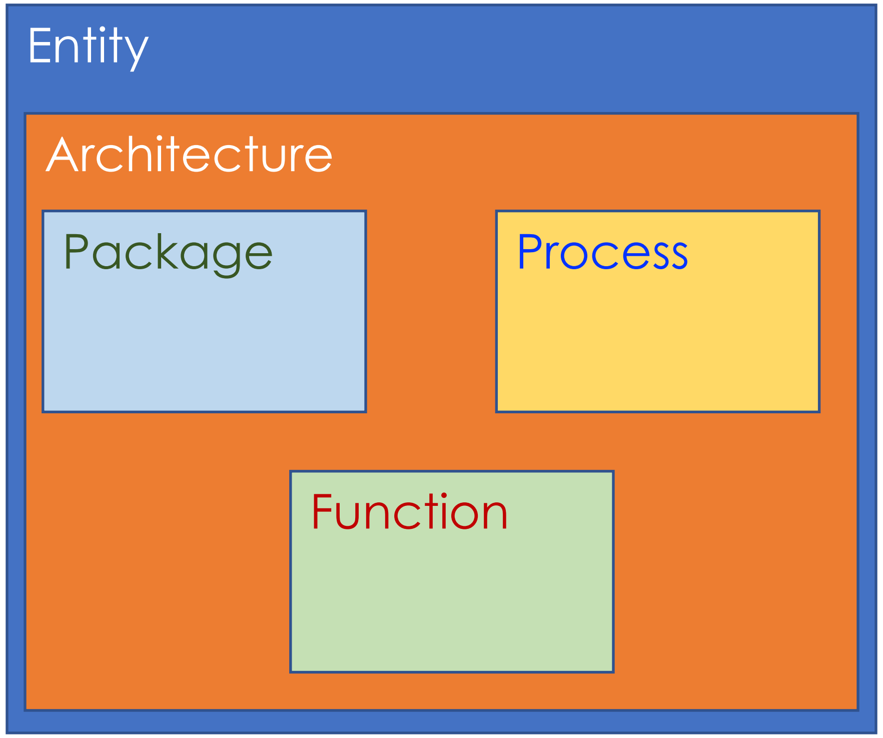
TAC-HEP 2023

VHDL

VHSIC **H**ardware **D**escription **L**anguage

- VHSIC: **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit

VHDL Basic Syntax



- Architecture Body
  - Architecture declaration
    - Component declaration
    - Type declarations
    - Constant declarations
    - Signal declarations
    - Functions, procedure definitions
  - Architecture statements

- Case – INSENSITIVE language
- Double dash (--) indicates a comment
- (;) End of declaration or a statement

Number Representations in VHDL

- **Integers:** represented with base-10 (decimal numbers) by default
  - 5, 3E2(=300), 2#0111#(=7), 5#341#(=96), 16#7DF#(=2015),
- **Binary Values:** Written either in single quotes (single bit) or double quotes (multi-bits)
  - '0' (=0), "0101"(=5), b"0101"(=5), O"54"(=44), X"C2F"(=3119)
- **Unsigned values:** all non-negative numbers
  - Range for N-bit word – 0 to 2^N -1
- **Signed Values:**
  - N bit words: -2^{N-1} to 2^{N-1} -1
- **Characters:** Extended ASCII table
  - 'A', '$', "VHDL", "mp4"

VHDL Objects

- **CONSTANT:** Whose value can't be changed
  - Can be declared in the declarative part of ENTITY, ARCHITECTURE, PACKAGE, BLOCK, PROCESS, FUNCTION, …

<Syntax> CONSTANT constant_name: constant_type := constant_value

```
CONSTANT       bits:  INTEGER                := 16;
CONSTANT     words:   INTEGER                := 2**bits;
CONSTANT       flag:   BIT                   := '1';
CONSTANT      mask:   BIT_VECTOR(1 TO 8) := "00001111";
```

VHDL Objects

- **OTHERS:** Useful keyword for making assignments
  - Represents all index values that were left unspecified

- The constant below is **a** = "000000".

```
CONSTANT a: BIT_VECTOR(5 DOWNTO 0) := (OTHERS=>'0');
```

- The next constant is **b** = "01111111" (index 7 gets '0', the others, '1’).

```
CONSTANT b: BIT_VECTOR(7 DOWNTO 0) := (7=>'0', OTHERS=>'1');
```

- The signal below is **c** = "01100000" ("|" means "or").

```
SIGNAL c: STD_LOGIC_VECTOR(1 TO 8) := (2|3=>'1', OTHERS=>'0');
```

- The variable below is **d** = "1111111100000000".

```
VARIABLE d: BIT_VECTOR(1 TO 16) := (1 TO 8=>'1', OTHERS=>'0');
```

Signal vs Variable

SIGNAL

- Serves to pass values in/out of the circuit, as well as b/w its internal units
- Represents circuit interconnects (wires)
- All ports of an entity are signal by default
- Inside sequential code, its update is not immediate, instead, new value is expected after the conclusion of current process or sub-program
- In concurrent code: multiple assignments will lead to compilation errors

VARIABLE

- Represent only local information as it can be seen/modified inside the sequential unit
- Update is immediate
- Mutiple assignments to same variables are fine

VHDL Operators

Logical Operators	
and	Logical And
or	Logical Or
nand	Logical Nand
nor	Logical Nor
xor	Logical Xor
xnor	Logical Xnor

Relational Operators	
=	Equal
/=	Not Equal
<	Less Than
<=	Less Than or Equal To
>	Greater Than
>=	Greater Than or Equal To

Concatenation Operators	
&	Concatenate

Arithmetic Operators

Arthimetic Operators	
+, -, *, /	
**	Exponentiation
mod	Modulo division
rem	Modulo remainder
abs	Absolute Value

- These operators are defined for "integer" and "real" datatypes
- For "std_logic" data type, these operators are overloaded in "ieee.std_logic_unsigned" package

Shift Operator

- Shift operators are used for shifting data vectors
  - BIT_VECTOR

Shift left logic	**SLL**	Positions on the right are filled with '0's
Shift right logic	**SRL**	Positions on the left are filled with '0's
Shift left arithmetic	**SLA**	Rightmost bit is replicated on the right
Shift right arithmetic	**SRA**	Leftmost bit is replicated on the left
Rotate Left	**ROL**	Circular shift to the left
Rotate Right	**ROR**	Circular shift to the right

Shift Operator - Example

x = "01001"

y <= x SLL 2;	--y <= "00100"	(y <= x(2 DOWNTO 0) & "00";
y <= x SLA 2;	--y <= "00111"	(y <= x(2 DOWNTO 0) & x(0) & x(0);
y <= x SRL 3;	--y <= "00001"	(y <= "000" & x(4 DOWNTO 3);
y <= x SRA 3;	--y <= "00001"	(y <= x(4) & x(4) & x(4) & x(4 DOWNTO 3);
y <= x ROL 2;	--y <= "00101"	(y <= x(2 DOWNTO 0) & x(4 DOWNTO 3) ;
y <= x SRL -2;	-- same as "x SLL 2"	

Data-Types

- BIT: 0 or 1:
  - Values assigned in single quotes: '0' or '1'

- BIT_VECTOR:
  - Vector version of BIT types consisting of two or more bits
  - Each bit in BIT_VECTOR can only have value 0 or 1
  - Values assigned in double quotes: "1011"

- STD_LOGIC:

scalar	bit
	boolean
	integer
	character
	std_ulogic
	std_logic
composite	bit vector
	string
	std_ulogic_vector
	std_logic_vector

Standard Logic Data Types

- STD_(U)LOGIC and STD_LOGIC_VECTOR
  - "U" stand for unresolved sub-type

```
TYPE STD_ULOGIC IS ('U','X','0','1','Z','W','L','H','-');
TYPE STD_LOGIC IS resolved STD_ULOGIC;
```

'U'	Uninitialized
'X'	Forcing unknown
'0'	Forcing Low
'1'	Forcing High
'Z'	High impedance
'W'	Weak unknown
'L'	Weak Low
'H'	Weak High
'-'	Don't care

Entity

- Names entity and defines interfaces between entity and its environment
- The I/O ports of the circuit are declared in the entity

```
ENTITY entity-name IS
    PORT (
        port-name-A: port_mode signal_type;
        port-name-B: port_mode signal_type;
        …);
END [ENTITY][entity-name];
```

```
library ieee;
use ieee.std_logic_1164.all;

entity EXAMPLE is
    port (
            A,B,C : in   std_logic;
              D,E : out std_logic
         );
end EXAMPLE;
```

"entity-name" can be any word except VHDL and other VHDL keywords

Entity – more options

```
ENTITY entity-name IS
  [GENERIC (
        const_name: const_type const_value;
        ...);]
  [PORT (
        port-name-A: port_mode signal_type;
        port-name-B: port_mode signal_type;
        ...); ]
  [entity_statement_part]
  [BEGIN
        entity_statement_part]
END [ENTITY][entity-name];
```

Optional GENERIC section (before PORT):

- Declaring constants that are globally visible to the design, including to PORT
- Parameterize a design, conferring code more flexibity and reusability
- Only declaration allowed before PORT

Declarative (optional) part, and the statements (code) part (from BEGIN down:

- Rarely used
- Contains, sub-program declarations, type declarations, constant declaration, attribute declarations, etc.

Entity – more options

```
ENTITY entity-name IS
  [GENERIC (
        const_name: const_type const_value;
        ...);]
   [PORT (
        port-name-A: port_mode signal_type;
        port-name-B: port_mode signal_type;
        ...); ]
   [entity_statement_part]
   [BEGIN
        entity_statement_part]
END [ENTITY][entity-name];
```

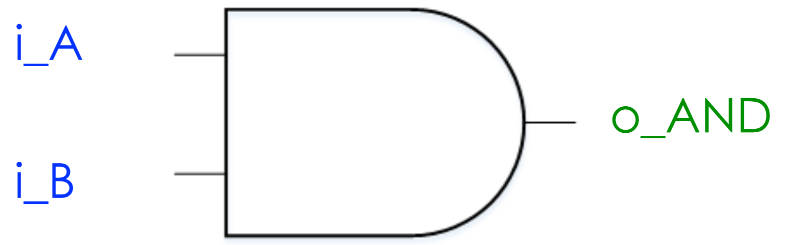
Optional GENERIC section (before PORT): declaring constants that are globally visible to the design, including to PORT

- Parameterize a design, conferring code more flexibity and reusability
- Only declaration allowed before PORT

```
ENTITY entity_name IS
      GENERIC (
                  m: INTEGER                        := 8;
                  n: BIT_VECTOR(3 DOWNTO 0) := "0101"
                 );
      PORT (...);
END entity_name;
```

Example

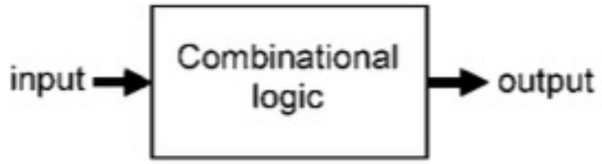
```
library ieee;
use ieee.std_logic_1164.all;

ENTITY example IS
   PORT (
      i_A           : in  std_logic;
      i_B           : in   std_logic;
      o_AND      : out std_logic
   );
END example;

ARCHITECTURE behav OF example IS
BEGIN
     p_PROCESS: PROCESS (i_A, i_B)
     BEGIN
          o_AND <= i_A and i_B;
     END PROCESS p_PROCESS;
END behav;
```



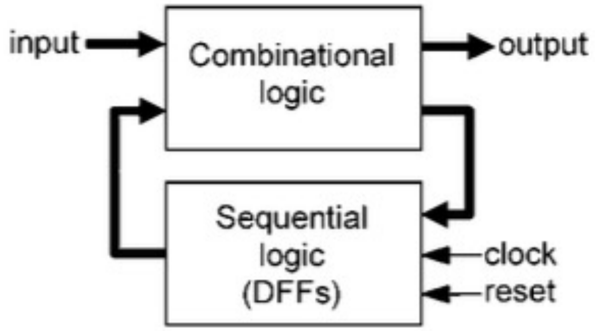
An AND Gate

Concurrent Code

- **Combinations Logic:** Output depend solely on the current inputs
  - No memory needed



- **Sequential logic:** Output depend on previous system state(s)
  - Storage needed
  - Clock needed to control system evolution
  - Reset signal



Concurrent statements

- WHEN-ELSE
- WITH-SELECT-WHEN

WHEN statement

WHEN: Simplest conditional statement

- Approximately equivalent to IF statement

```
assignment_expression WHEN conditions ELSE
        assignment_value WHEN conditions ELSE
          ...;
```

```
x <=   '0'   WHEN rst='0'   ELSE
       '1'   WHEN a='0'   OR   b='1' ELSE
       '-' ;   --don't care

y <=  "00"  WHEN (a AND b)="01"  ELSE
      "11"  WHEN (a AND b)="10" ELSE
      "ZZ";    --high impedance
```

SELECT statement

SELECT: Another Concurrent statement

- Approximately equivalent to CASE statement
- All values of signal must be listed
- Values must be mutually exclusive

```
WITH identifier SELECT
   assignment_expression WHEN values
        assignment_value WHEN values
         ...;
```

```
WITH control SELECT
   y <= "000" WHEN 0 | 1,
        "100" WHEN 2 TO 5,
        "Z--" WHEN OTHERS;
```

```
WITH (a AND b) SELECT
      y <= "00" WHEN "001",
           "11" WHEN "100",
           UNAFFECTED WHEN OTHERS;
```

TAC-HEP 2023

Sequential statements

- IF-THEN-ELSE
- CASE-WHEN
- LOOP
- WAIT

IF Statement

IF, WAIT, LOOP, CASE statements are intended for sequential code, can only be used inside PROCESS or sub-program

```
[label:] IF conditions THEN
      assignments;

ELSIF conditions THEN
      assignments;
 ...

ELSE
      assignments;
END IF [label];
```

```
IF (x<y) THEN
      temp:= "00001111";

ELSIF (x=y AND w='0') THEN
      temp:= "11110000";

ELSE
      temp := (OTHERS => '0');

END IF;
```

CASE Statement

- CASE and SELECT are very similar
- CASE is for concurrent code, SELECT is for sequential code

```
WITH sel & ena SELECT
    x <=  a WHEN "00" | "11",
          b WHEN "01",
          c WHEN OTHERS;

WITH sel & ena SELECT
     y <= "0000" WHEN "11",
          "1--1"   WHEN OTHERS;
```

```
CASE sel & ena IS
     WHEN "00”       => x <= a; y <= "1--1";
     WHEN "01"       => x <= b; y <= "1--1";
     WHEN "11"       => x <= a; y <= "0000";
     WHEN OTHERS => x <= c; y <= "1--1";
END CASE;
```

LOOP Statements

- LOOP is used when a piece of code must be instantiated several times

Unconditional LOOP

```
[label:] LOOP
    sequential_statements
END LOOP [label];
```

LOOP with FOR

```
[label:] FOR identifier IN range LOOP
      sequential_statements
END LOOP [label];
```

LOOP with WHILE

```
[label:] WHILE condition LOOP
      sequential_statements
END LOOP [label];
```

LOOP with EXIT

```
[loop_label:] [FOR identifier IN range] LOOP
   ...
  [exit_label:] EXIT [loop_label] [WHEN condition];
  ...
END LOOP [loop_label];
```

LOOP Statements – Examples

Unconditional LOOP

```
LOOP
     WAIT UNTIL clk='1';
     count := count + 1;
END LOOP;
```

LOOP with FOR

```
FOR i IN 0 TO 5 LOOP
       x(i) <= a(i) AND b(5-i);
       y(0, i) <= c(i);
END LOOP;
```

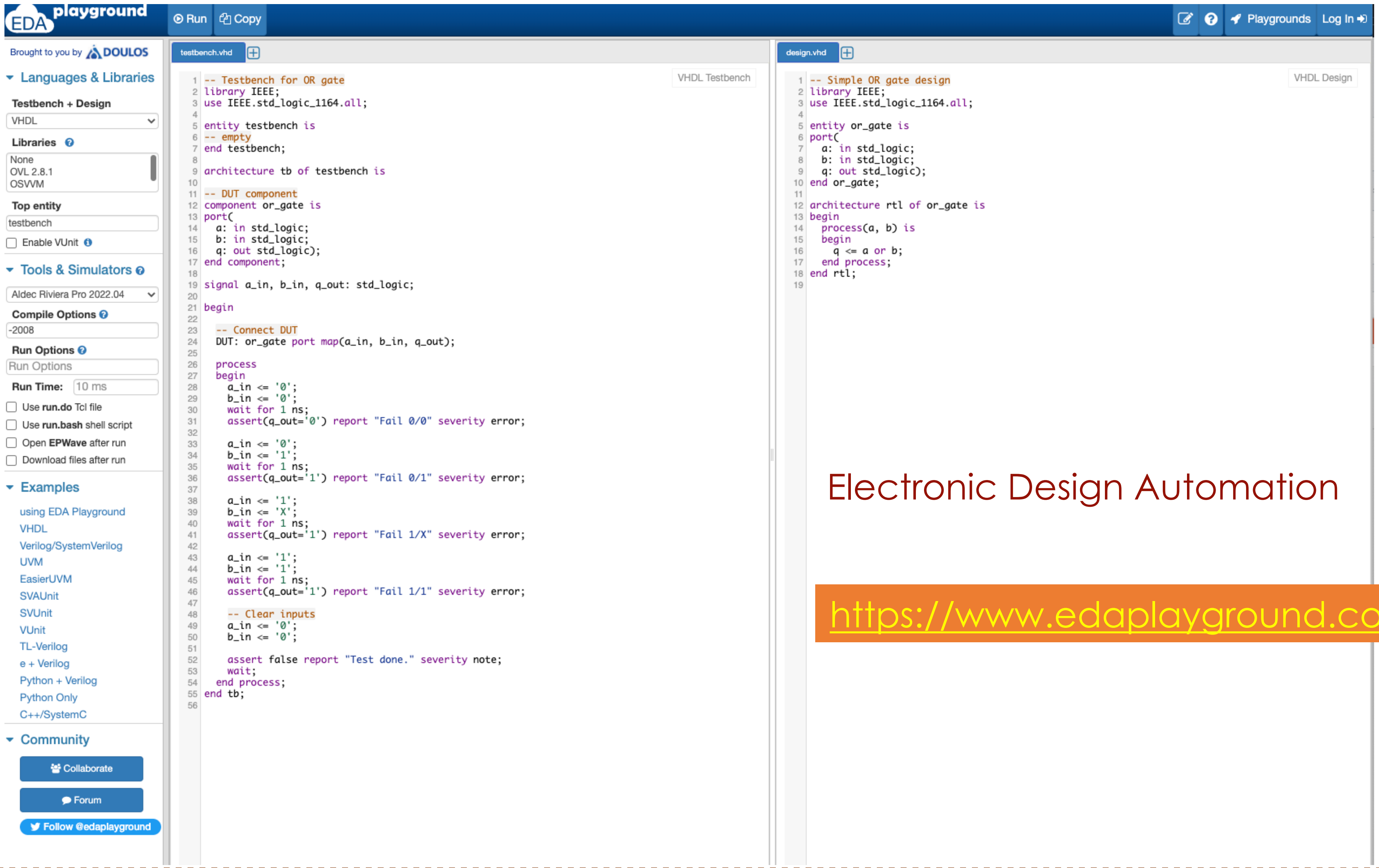
LOOP with WHILE

```
WHILE (i<10) LOOP
   WAIT UNTIL clk'EVENT AND clk='1';
   ...
END LOOP;
```

LOOP with EXIT

```
FOR i IN data'RANGE LOOP
     CASE data(i) IS
          WHEN '0' => count:=count+1;
          WHEN OTHERS => EXIT;
      END CASE;
END LOOP;
```

VHDL EDA Playground



Brought to you by DOULOS

Languages & Libraries

Testbench + Design: VHDL

Libraries: None, OVL 2.8.1, OSVVM

Top entity: testbench

- [ ] Enable VUnit

Tools & Simulators

Aldec Riviera Pro 2022.04

Compile Options: -2008

Run Options: Run Options

Run Time: 10 ms

- [ ] Use run.do Tcl file
- [ ] Use run.bash shell script
- [ ] Open EPWave after run
- [ ] Download files after run

Examples

using EDA Playground, VHDL, Verilog/SystemVerilog, UVM, EasierUVM, SVAUnit, SVUnit, VUnit, TL-Verilog, e + Verilog, Python + Verilog, Python Only, C++/SystemC

Community

Collaborate, Forum, Follow @edaplayground

testbench.vhd

```vhdl
-- Testbench for OR gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end component;

signal a_in, b_in, q_out: std_logic;

begin

  -- Connect DUT
  DUT: or_gate port map(a_in, b_in, q_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    wait for 1 ns;
    assert(q_out='0') report "Fail 0/0" severity error;

    a_in <= '0';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 0/1" severity error;

    a_in <= '1';
    b_in <= 'X';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/X" severity error;

    a_in <= '1';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/1" severity error;

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';

    assert false report "Test done." severity note;
    wait;
  end process;
end tb;

```

design.vhd

```vhdl
-- Simple OR gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end or_gate;

architecture rtl of or_gate is
begin
  process(a, b) is
  begin
    q <= a or b;
  end process;
end rtl;

```

Electronic Design Automation

https://www.edaplayground.com/x/A4

Summary

✓Introduction to FPGA and its basic architecture

✓A guide to begin with HLS programs

✓Importance & effect of different pragma's

✓Brief introduction to VHDL programming

TAC-HEP 2023

Thank you!

Hope you enjoyed and learnt something new!

TAC-HEP 2023

Questions?

TAC-HEP 2023

Additional material

Assignment submission

- Where to submit:
  - https://pages.hep.wisc.edu/~varuns/assignments/TAC-HEP/
- Use your login machine credentials
- Submit one file per week
- Try to submit by following week's Tuesday

Assignment Week-3

- Use target device: **xc7k160tfbg484-2**
- Clock period of 10ns

1. Execute the code (lec5Ex2.tcl) using CLI (slide-25) and compare the results with GUI results for C-Simulation, C-Synthesis

2. Vary following parameters for two cases: high and very high values and compare with 1 for both CLI and GUI
    - Variable: "samples"
    - Variable: "N"

3. Run example lec3Ex2a

Assignment Week-4

1. Do a matrix multiplication of two 1-dimensional arrays – A[N]*B[N], where N > 5
   a) Report synthesis results without any pragma directives
   b) Add as many pragma directives possible
      i. Report any conflicts (if reported in logs) between two pragmas

2. Compare the analysis perspective (Performance) for different case shared today
3. For Array_partitioning, instead of using complete, use block and cyclic with different factors

Assignment Week-5

1. Do exercise mention on slide-24
2. A matrix multiplication using two for loops and compare results for pragma loop_flatten & unroll
3. Write a simple program doing arithmetic operations(+, -, *, /, %) between two variable use of arbitrary precision to compare results between stand c/c++ data types and using ap_(u)int<N>
4. Write a program using an array with N(=10/15/20) elements and then restructure the code with a struct having N-data member. Compare the results of two programs

Project

Write an algorithm to cluster ECAL and HCAL energies for Regional Calorimeter Trigger using HLS and synthesis the results

1. Input per tower (ECAL + HCAL)
2. Cluster ECAL energies for each tower
   - Divide the RCT card further to make life simple
3. Stitch together the clusters for neighbouring towers
4. Sort the final list
5. Send just 12 towers per RCT region

Jargons

- **ICs - Integrated chip:** assembly of hundreds of millions of transistors on a minor chip
- **PCB:** Printed Circuit Board
- **LUT - Look Up Table aka 'logic'** - generic functions on small bitwidth inputs. Combine many to build the algorithm
- **FF - Flip Flops** - control the flow of data with the clock pulse. Used to build the pipeline and achieve high throughput
- **DSP - Digital Signal Processor** - performs multiplication and other arithmetic in the FPGA
- **BRAM - Block RAM** - hardened RAM resource. More efficient memories than using LUTs for more than a few elements
- **PCIe or PCI-E - Peripheral Component Interconnect Express**: is a serial expansion bus standard for connecting a computer to one or more peripheral devices
- **InfiniBand** is a computer networking communications standard used in high-performance computing that features very high throughput and very low latency
- **HLS -** High Level Synthesis - compiler for C, C++, SystemC into FPGA IP cores
- **DRCs -** Design Rule Checks
- **HDL** - Hardware Description Language - low level language for describing circuits
- **RTL** - Register Transfer Level - the very low level description of the function and connection of logic gates
- **FIFO** – First In First Out memory
- **Latency** - time between starting processing and receiving the result
  - Measured in clock cycles or seconds
- **II - Initiation Interval** - time from accepting first input to accepting next input