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

## GPU & FPGA module training: Part-2

**Week-7**: Introduction to VHDL

*Lecture-13: May 2nd 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

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

**V**HSIC **H**ardware **D**escription **L**anguage

## VHDL

**V**HSIC **H**ardware **D**escription **L**anguage

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

# VHDL

- Is an industry standard language used to describe hardware from the abstract to the concrete level
  - Standardized as IEEE standards 1076—1987, 1076-1993 & 1076-1164 (standard logic data type)
  - Specify the behaviour and structure of a digital circuit
  - Concurrent and sequential statements
- Powerful language with numerous language constructs capable of describing very complex behaviour
- One of the two languages used to design FPGAs and ASICs
- Verilog is another, equally popular, hardware description language (HDL)

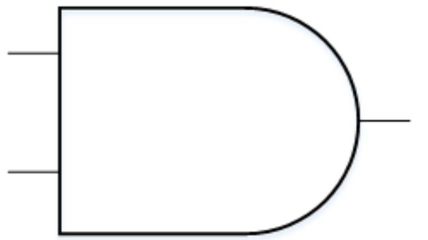
## New to VHDL

- a <= b
  - a gets the value of b

- a <= b after 10 ns
  - a get the value of b when 10ns of time have elapsed

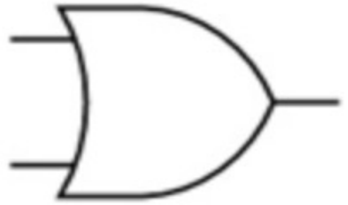
# New to VHDL

Lets Create a VHDL file that *describes* an And Gate



An AND Gate

```
signal and_gate : std_logic;
and_gate <= input_1 and input_2;
```



An OR Gate

```
signal and_gate : std_logic;
and_gate <= input_1 or input_2;
```

# VHDL design unit

A VHDL design unit consist of:

- Entity declaration
  - Names entity and defines interfaces between entity and its environment

```
ENTITY entity_name IS
PORT (name_list : mode type);
END entity_name;
```

- Architecture
  - Establishes relationship between inputs and outputs of design

```
ARCHITECTURE body_name OF entity _name IS
-- declarative_statements
BEGIN
-- activity_statements
END body_name;
```

- Entities and Architectures are used together to define a piece of functionality
- Only one entity and architecture for each file

## Entity Declaration

- 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: mode type;
        port-name-B:  mode type;
        port-name-C: mode 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;
```

Large FPGA design is broken into many entity combinations

- The entity contains port map, which is used to define all input and output signals for a particular entity

## Port

- Each I/O signal in the entity statement is referred to as **port**
- A port is analogous to a pin on a schematic
- A port is a data object
- Can be assigned values
- Can be used in expressions

## Mode

- The mode describes the direction in which data is transferred through a port
- There are 4 different modes:

| Mode | Description |
|---|---|
| in | Data only flows into the entity (input) |
| out | Data only flows out of the entity (output) |
| Inout | Data flows into or out of the entity (bidirectional) |
| buffer | Used for internal feedback |

## Type

- VHDL is a strongly typed language
  - Data objects of different types cannot be assigned to one another without the use of a type-conversion function

- Two broad categories of data types:
  - **Scalar –** stores single value
  - **Composite –** stores multiple values

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

# Type

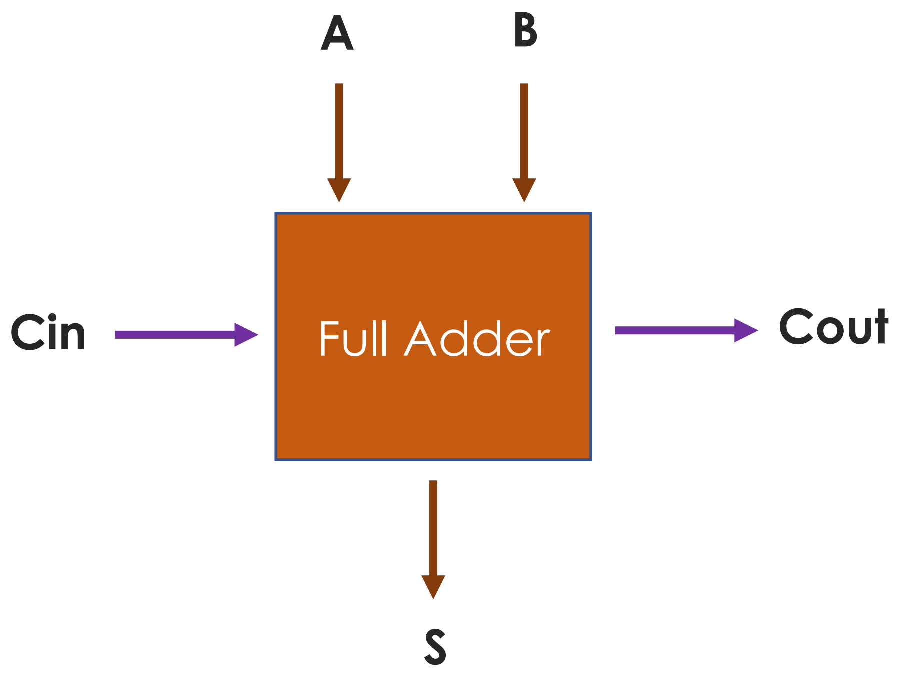
- VHDL is a strongly typed language
  - Data objects of different types cannot be assigned to one another without the use of a type-conversion function

- Two broad categories of data types:
  - **Scalar –** stores single value
  - **Composite –** stores multiple values

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

## Entity Declaration - Example

```
entity FULL_ADDER is
 port (
       A, B, Cin:   in   std_logic;
              S:  out std_logic;
           Cout: out std_logic;
     );
 end FULL_ADDER;
```



## Architecture Declaration

- Establishes relationship between inputs and outputs of design

```
ARCHITECTURE architecture_name OF entity _name IS
-- declarative_statements
BEGIN
-- architecture body
END [architecture][architecture_name];
```

- Several different models or styles may be used in the architecture body including:
  - Behavioral: set of statements to model the function or behavior
    - Dataflow: concurrent statements, order is unimportant
    - Algorithmic: sequential statements, ordering importamt
  - Structural

- These models allow to describe the design at different levels of abstraction

## Architecture Statement

- One or more architecture statements may be associated with an entity statement
  - Only one may be referenced at a time
- Declarations
  - Signals and components
- Architecture body
  - Statements that describe the functionality of the design (i.e., the circuit)

# Some Coding Guidelines

High readability of the code

Less error prone

| Prefix | Description | Details |
|---|---|---|
| **i_** | Input signal | • Most important style<br>• Difficult & annoying to look through the code to determine the direction of a signal<br>• *i_address, o_data_valid* / *address, data_valid* |
| **o_** | Output signal | |
| **r_** | Register signal (has registered logic) | • Second most important style<br>• Distinguishes signal as *register* or *wire*<br>• *Register:* have initial conditions<br>• *Wire:* Should never appear on left hand side of an assignment operator in a sequential process |
| **w_** | Wire signal (has no registered logic) | |
| **c_** | Constant | • Helpful indicators |
| **g_** | Generic | |
| **t_** | User-defined Type | |

## Entity/Architecture Example

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

entity example_entity_architecture is
   port (
      i_bit_1  : in  std_logic;
      i_bit_2  : in   std_logic;
      o_bit     : out std_logic
   );
end example_entity_architecture;

architecture behave of
example_entity_architecture is
begin
    p_PROCESS: process (i_bit_1, i_bit_2)
    begin
         o_bit <= i_bit_1 and i_bit_2;
    end process p_PROCESS;
end behave;
```

# VHDL Reserved Words

| | | | | |
|---|---|---|---|---|
| abs | disconnect | label | package | sla |
| access | downto | library | port | sll |
| after | else | linkage | postponed | sra |
| alias | elsif | literal | procedure | srl |
| all | end | loop | process | subtype |
| and | entity | map | protected | then |
| architecture | exit | mod | pure | to |
| array | file | nand | range | transport |
| assert | for | new | record | type |
| attribute | function | next | register | unaffected |
| begin | generate | nor | reject | units |
| block | generic | not | rem | until |
| body | group | null | report | use |
| buffer | guarded | of | return | variable |
| bus | if | on | rol | wait |
| case | impure | open | ror | when |
| component | in | or | select | while |
| configuration | inertial | others | severity | with |
| constant | inout | out | shared | xnor |
| | is | | signal | xor |

# 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 |

# Signal/Variable

- **Signal:** Represents interconnection wires that connect component instantiation ports together
  - Sometimes referred as fundamental unit of VHDL
  - Can be used inside or outside processes
  - Can be used in multiple processes but assigned only in one
  - Defined in architecture before begin statement
  - Assignment operator (**<=**)

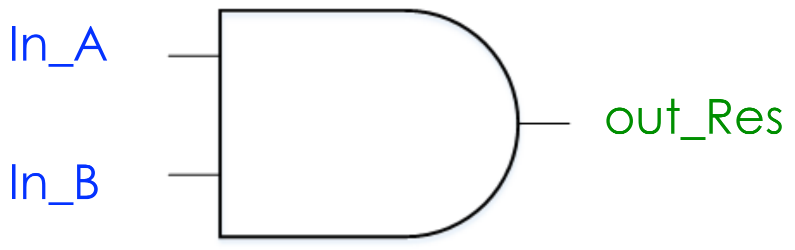
- **Variable:** Used for local storage of temporary data
  - Can be used only inside a process
  - Created in one process, can't be used in another (like a local variable in C/C++)
  - Need to be defined aftere keyword process before keyword begin
  - Assignment operator (**:=**)

# Signal/Variable

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

entity variable_vs_signal is
 port (
     i_clk          : in    std_logic;
     o_var_done     : out std_logic;
     o_sig_done     : out std_logic
   );
end variable_vs_signal;

architecture rtl of variable_vs_signal is

    signal  r_Var_Done  : std_logic                  := '0';
    signal  r_Count     : natural range 0 to 6 := 0;
    signal  r_Sig_Done  : std_logic                  := '0';

begin

  VAR_VS_SIG : process (i_clk)
      variable v_Count : natural range 0 to 5 := 0;
  begin
    if rising_edge(i_clk) then
       v_Count  :=  v_Count + 1;         -- Variable
       r_Count  <=  r_Count + 1;         -- Signal

      -- Variable Checking
      if v_Count = 5 then
          r_Var_Done <= '1';
          v_Count := 0;
      else
          r_Var_Done <= '0';
      end if;

      -- Signal Checking
      if r_Count = 5 then
          r_Sig_Done <= '1';
          r_Count    <= 0;
      else
          r_Sig_Done <= '0';
       end if;

    end if;
  end process VAR_VS_SIG;

  o_var_done  <=  r_Var_Done;
  o_sig_done  <=  r_Sig_Done;

end rtl;
```

## Example: AND gate

Lets Create a VHDL file that *describes* an And Gate



An AND Gate

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

entity example_and is
  port (
    input_1       : in  std_logic;
    input_2       : in  std_logic;
    and_result  : out std_logic
    );
end example_and;

architecture rtl of example_and is
  signal and_gate : std_logic;
begin
  and_gate   <= input_1 and input_2;
  and_result <= and_gate;
end rtl;
```

Code defines an architecture called rtl of entity example_and

# For Loop

- For loop perform differently in a software language than in VHDL
- For loop in synthesizable code are used to expand replicated logic

```
// Example software Code:
For (int i=0; i<10; i++)
    data[i] = data[i] + 1;
```

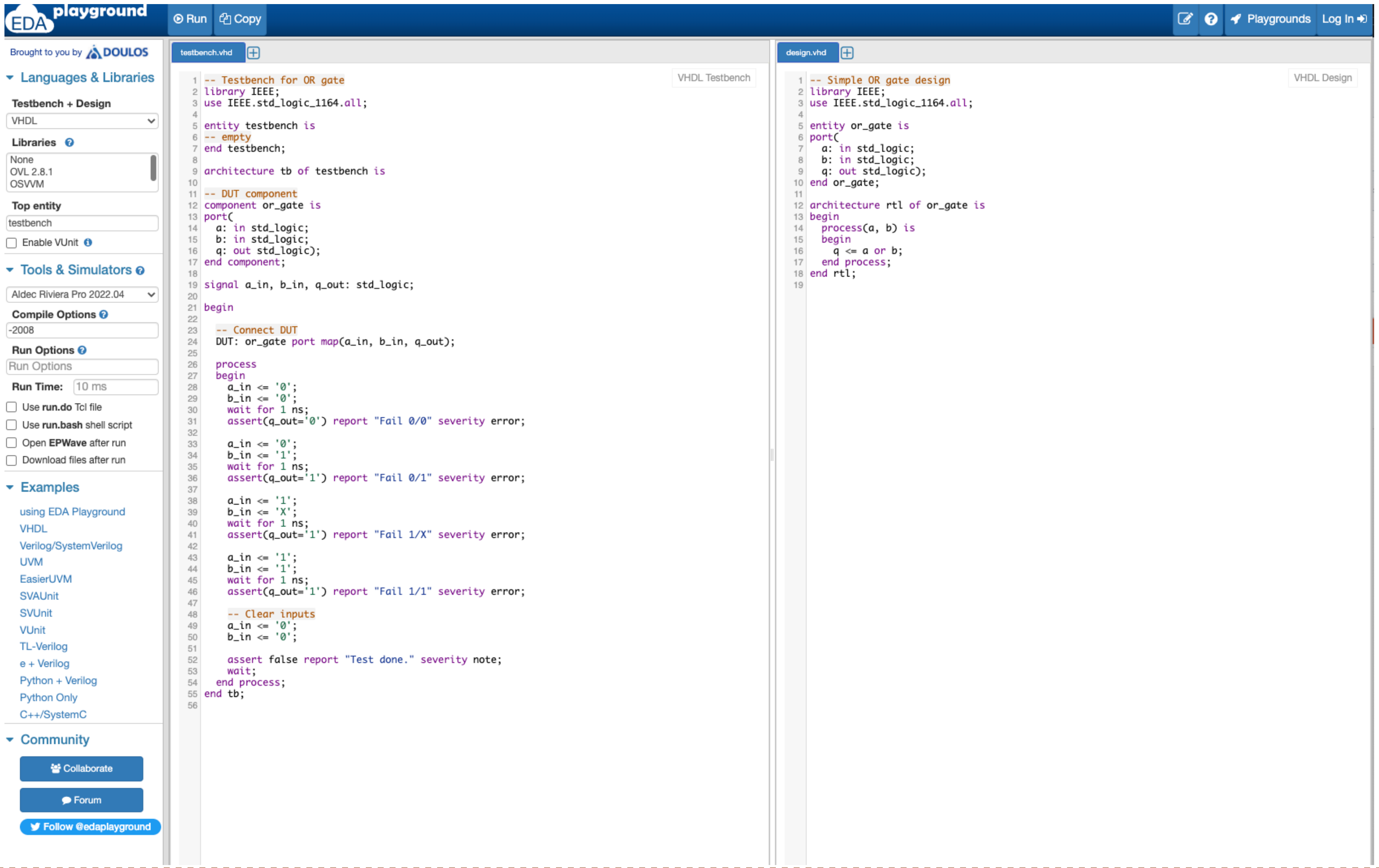
```
P_INCREMENT : process (clock)
begin
    if rising_edge(clock) then
        if index < 10 then
            data(index)   <= data(index) + 1;
            index         <= index + 1;
        end if;
    end if;
end process P_INCREMENT;
```

# Some Common Terms

- **Entity:** Most basic building block in a design
- **Architecture:** Describes behavior of the entity
- **Configuration:** Used to bind a component instance to an entity-architecture pair
  - Like a parts list for a design, which part to use for each part in the design
- **Package:** Collection of commonly used data types and sub-programs used in a design
- **Driver:** Source on a signal
  - If a signal is driven by two sources, then when both sources are active, the signal will have two drives
- **Bus:** A group of signals or a particular method of communicattion
- **Attrbute:** Data that are attached to VHDL objects or predefined data about VHDL objects
- **Generic:** VHDL's term of a parameter that passes information to an entity
- **Process:** Basic unit of execution in VHDL
  - All operations that are performed, broken into single or multiple processes

# VHDL Playground

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



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

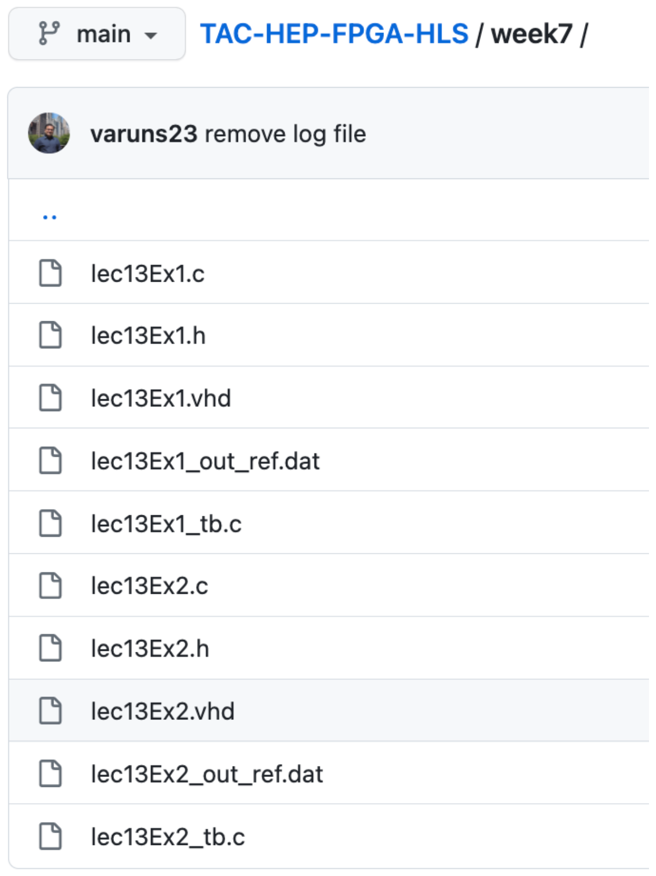
testbench.vhd

```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;
```

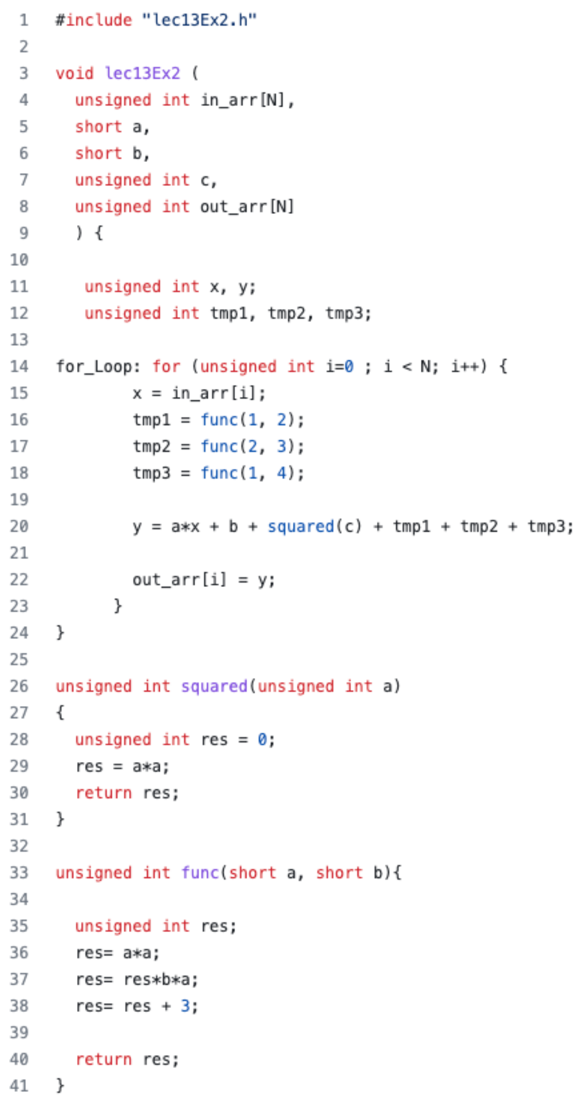
# Example



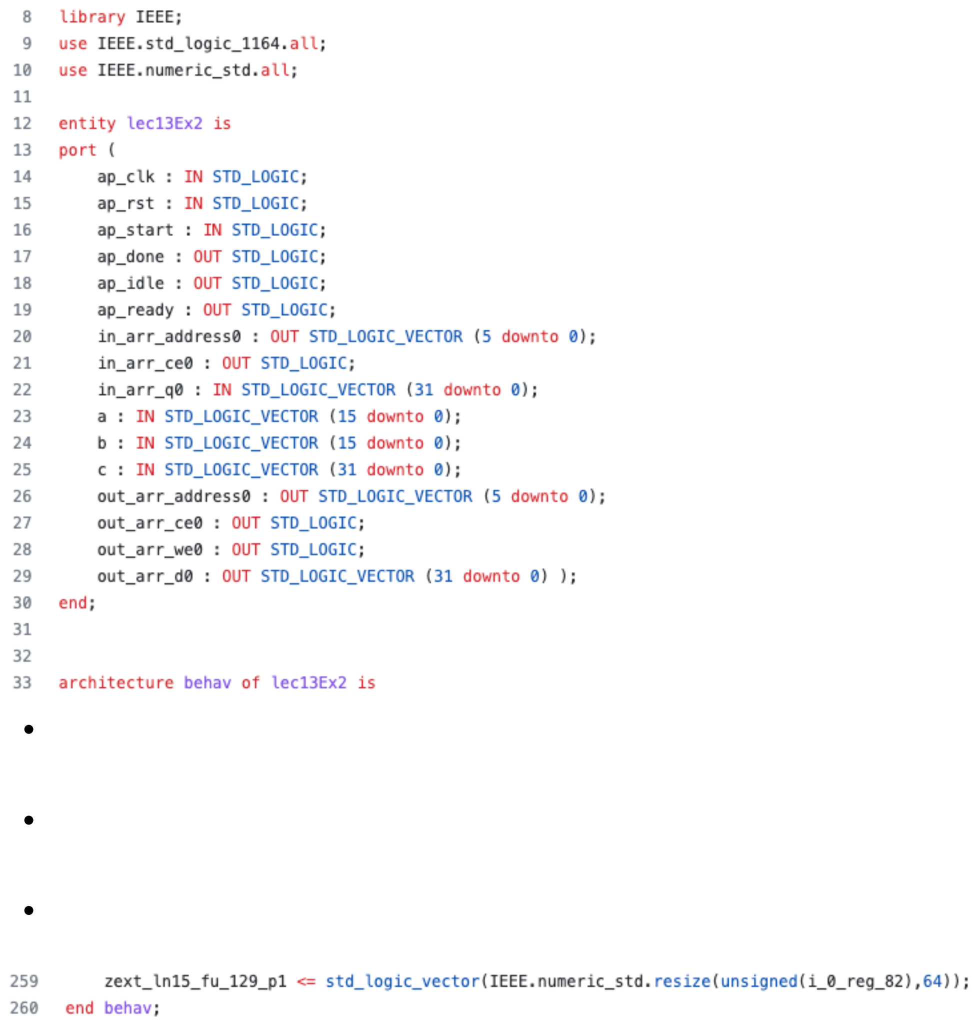
main TAC-HEP-FPGA-HLS / week7 /

varuns23 remove log file

..
lec13Ex1.c
lec13Ex1.h
lec13Ex1.vhd
lec13Ex1_out_ref.dat
lec13Ex1_tb.c
lec13Ex2.c
lec13Ex2.h
lec13Ex2.vhd
lec13Ex2_out_ref.dat
lec13Ex2_tb.c



```c
#include "lec13Ex2.h"

void lec13Ex2 (
  unsigned int in_arr[N],
  short a,
  short b,
  unsigned int c,
  unsigned int out_arr[N]
  ) {

   unsigned int x, y;
   unsigned int tmp1, tmp2, tmp3;

for_Loop: for (unsigned int i=0 ; i < N; i++) {
        x = in_arr[i];
        tmp1 = func(1, 2);
        tmp2 = func(2, 3);
        tmp3 = func(1, 4);

        y = a*x + b + squared(c) + tmp1 + tmp2 + tmp3;

        out_arr[i] = y;
      }
}

unsigned int squared(unsigned int a)
{
  unsigned int res = 0;
  res = a*a;
  return res;
}

unsigned int func(short a, short b){

  unsigned int res;
  res= a*a;
  res= res*b*a;
  res= res + 3;

  return res;
}
```



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity lec13Ex2 is
port (
    ap_clk : IN STD_LOGIC;
    ap_rst : IN STD_LOGIC;
    ap_start : IN STD_LOGIC;
    ap_done : OUT STD_LOGIC;
    ap_idle : OUT STD_LOGIC;
    ap_ready : OUT STD_LOGIC;
    in_arr_address0 : OUT STD_LOGIC_VECTOR (5 downto 0);
    in_arr_ce0 : OUT STD_LOGIC;
    in_arr_q0 : IN STD_LOGIC_VECTOR (31 downto 0);
    a : IN STD_LOGIC_VECTOR (15 downto 0);
    b : IN STD_LOGIC_VECTOR (15 downto 0);
    c : IN STD_LOGIC_VECTOR (31 downto 0);
    out_arr_address0 : OUT STD_LOGIC_VECTOR (5 downto 0);
    out_arr_ce0 : OUT STD_LOGIC;
    out_arr_we0 : OUT STD_LOGIC;
    out_arr_d0 : OUT STD_LOGIC_VECTOR (31 downto 0) );
end;


architecture behav of lec13Ex2 is
•
•
•
    zext_ln15_fu_129_p1 <= std_logic_vector(IEEE.numeric_std.resize(unsigned(i_0_reg_82),64));
 end behav;
```

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