ENGR 2323 Digital Design Lab
Introduction to VHDL

VHDL

Hardware description languages (HDL) allow for the design, documentation, synthesis, and verification of complex digital designs. Hardware description languages are high-level languages resembling programming languages that allow one to specify a circuit design in a form that resembles a computer program. Hardware description languages allow the designer to describe a design at various levels of abstraction from high-level behavioral descriptions to low-level using primitive components. HDLs are geared to describing hardware structures and the hardware structure's behavior and they differ from conventional high-level programming languages in that they support parallel (concurrent) operation.

VHDL stands for VHSIC HDL, where VHSIC stands for very high-speed integrated circuit and HDL stands for hardware description language. VHDL is non-proprietary and an IEEE standard (IEEE 1076 – 2019). Verilog is another commonly used digital circuit HDL, but we will focus on VHDL in this course.

VHDL Designs

HDLs are typically used with CAD tools such as Quartus that take the HDL design and synthesize the hardware (convert description to primitive components and necessary connections) similar to how a compiler takes a high-level program and generates a machine language program. Designs can be specified using waveforms, schematics, or HDLs. We have used the Quartus schematic (block diagram) editor for both combinational and sequential circuit design. HDLs can be a better design choice for more complex circuits.

A VHDL design consists of two parts: the interface (entity) and the implementation (architecture). The interface specifies the inputs and outputs, in other words the interface specifies what the circuit looks like to user from outside. The implementation specifies the operation of the circuit and is primarily important to the designer (not the user). The implementation for a circuit is not unique, for example a designer may use one architecture to test their logic but use another for the actual implementation and verify timing.

Consider the digital circuit schematic of Figure 1 that realizes the Boolean expression $F = \overline{U} \cdot V + \overline{U} \cdot \overline{V} \cdot W$.

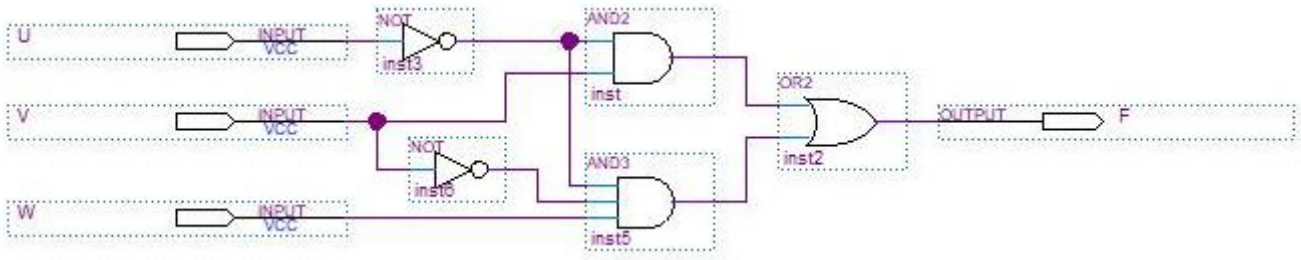

Figure 1. Schematic Design

The circuit inputs are U, V, and W (all single bit) and the circuit output is F (single bit). The VHDL entity of Figure 2a describes the interface and the architecture of Figure 2b shows a dataflow implementation.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ckt IS
    PORT (U, V, W : IN STD_LOGIC;
      F : OUT STD_LOGIC);
END ckt;
```

Figure 2a. VHDL Entity

```
ARCHITECTURE dataflow OF ckt IS
BEGIN
    F <= (not U and V) or  (not U and not V and W);
END dataflow;
```

Figure 2b. VHDL Dataflow Architecture

In Figures 2a and 2b, the words `entity`, `is`, `port`, `in`, `out`, `end`, `architecture`, `begin`, `and`, `or`, `not` are all VHDL reserved words. The symbols :, ;, and <= are VHDL symbols. The <= symbol is for signal assignment. VHDL is case-insensitive and is a strongly typed language (no type conversion). Semicolons (;) indicate the end of a VHDL statement. Two dashes (--) indicate a comment.

In Quartus projects, design modules can be schematic designs, HDL designs, or a mixture of both. The top-level design module must have the same name as the project.

Entity

The library statements in the VHDL code define the compiled design units that will be used in the design. These statements will almost always be included at the top of the VHDL code and give access to previously designed units known as packages.

The use statement specifies which entities or packages to use out of the library. In the entity of Figure 2a, `USE ieee.std_logic_1164.all;` imports all of the std_logic_1164 package from the IEEE library. The std_logic_1164 package defines a 9-valued logic system, of which only 4 values can be used for synthesis of designs (1, 0, - (don't care), Z (high impedance).

The entity defines the interface of the module and the port list defines the signals accessible outside the module. The signals can either be input (in), output (out), or bidirectional (inout) and the signal type can either be std_logic (single bit signal) or std_logic_vector (bus). The

width of the bus can be defined in either ascending or descending order (0 upto 3) or (3 downto 0).

Architecture
The architecture defines the operation of the module. Multiple architectures can be defined for an entity and a configuration statement can be used to specify which architecture to use (we will not do this in this course).

Statements in the architecture block are concurrent (not sequential as in most programming languages). The order in which they appear does not affect the outcome and all the operations are performed at the same time. VHDL does allow sequential statements to be used as part of a process.

Designs cane be described using four styles of modeling: structural, dataflow, behavioral, and mixed.

Structural Architectures
Structural architectures describe the operation in terms of components and the interconnections between them. Structural architectures are similar to the circuit schematic that they model.

Dataflow Architectures
Dataflow architectures describe the operation using concurrent assignment statements. The assignment statements may be logic expressions or register transfer language (RTL) expressions describing how signals flow through the design

Behavioral Architectures
Behavioral architectures describe the operation in terms of the algorithm performed by the module. These architectures look similar to high level program implementations of algorithms. Behavioral architectures often use sequential assignment statements inside process statements.

Mixed Architectures
Mixed architectures use any combination of the above three styles to describe the operation.

Full Adder Design
Consider the design of a full adder. A full adder takes two one-bit numbers and a carry in and generates a one-bit sum and a carry out.

The sum and carry out can be realized using the expressions

$$S = X \oplus Y \oplus C_{in}$$

$$Co = X \cdot Y + X \cdot C_{in} + Y \cdot C_{in}$$

X Y Cin	S	Co
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0
1 0 1	0	1
1 1 0	0	1
1 1 1	1	1

Figure 3. Function Table for Full Adder

Figure 4a shows the entity for the full adder design. There are three inputs X, U, and Cin. The inputs are each 1-bit of type standard logic. There are two outputs S and Cout, each 1-bit and of type standard logic.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fulladder IS
PORT(X, Y, Cin : IN STD_LOGIC;
     S, Cout : OUT STD_LOGIC);
END fulladder;
```

Figure 4a. fulladder Entity

Figure 4b shows a dataflow architecture for the full adder. The operation is described using concurrent assignment statements corresponding to the Boolean expressions describing the design.

```
ARCHITECTURE dataflow of fulladder IS
BEGIN
     S <= X XOR Y XOR Cin;
     Cout <= (X AND Y) OR (X AND Cin) OR (Y AND Cin);
END dataflow;
```

Figure 4b. fulladder Dataflow Architecture

Figure 5 shows a structural architecture for the full adder.

```
ARCHITECTURE structural of fulladder IS

COMPONENT AND_gate
PORT(A, B : IN STD_LOGIC;
     Y : OUT STD_LOGIC);
END COMPONENT;

COMPONENT OR_gate
PORT(A, B : IN STD_LOGIC;
     Y : OUT STD_LOGIC);
END COMPONENT;

COMPONENT XOR_gate
PORT(A, B : IN STD_LOGIC;
     Y : OUT STD_LOGIC);
END COMPONENT;

SIGNAL V1 : STD_LOGIC;
SIGNAL V2, V3, V4, V5 : STD_LOGIC;

BEGIN
-- S <= X XOR Y XOR Cin;
gate1: XOR_gate PORT MAP(X, Y, V1);
gate2: XOR_gate PORT MAP(V1, Cin, S);
-- Cout <= (X AND Y) OR (X AND Cin) OR (Y AND Cin);
gate3: AND_gate PORT MAP(X, Y, V2);
gate4: AND_gate PORT MAP(X, Cin, V3);
gate5: AND_gate PORT MAP(Y, Cin, V4);
gate6: OR_gate PORT MAP(V2, V3, V5);
gate7: OR_gate PORT MAP(V4, V5, Cout);

END structural;
```

Figure 5. fulladder Structural Architecture

The structural design uses the AND_gate, OR_gate, and XOR_gate components. These were previously designed and compiled. The AND_gate component is shown in Figure 6. Signals are used for internal signals to make connections between components. These are like having a wire with the signal name.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.All;

ENTITY AND_gate IS
PORT(A, B : IN STD_LOGIC;
     Y : OUT STD_LOGIC);
END AND_gate;

ARCHITECTURE dataflow of AND_gate IS
BEGIN
     Y <= A AND B ;
END dataflow;
```

Figure 6. AND_gate Component

The components are declared using the component statement and then instantiated each time the component is used. As part of instantiating the component, the component connections are specified using a port map. The signals are matched to the component entity by the order in the port map. Gate3 is an AND_gate component. The inputs of the gate are X (mapped to A of AND_gate) and Y (mapped to B of AND_gate) and the output is signal V1 (mapped to Y of the AND_gate).

Behavioral Architecture and Process Statements

Behavioral architectures often look nothing like how the design ends up being implemented and one is relying on the compiler to translate the design to a hardware implementation. Behavioral descriptions are often used to model complex designs that would be difficult to model using one of the other architectures such as a commercial microprocessor. Behavioral descriptions can also be used to evaluate a design and determine if functionally it is correct and later convert the design or portions of the design to a structural architecture.

Behavioral architectures often use process statements. Statements in processes are executed sequentially but the signal assignments are all made at the end of the process. A process is a concurrent statement and can be thought of as a single operation. Processes are executed concurrently with other concurrent statements (signal assignment statements, other processes, etc.) in the architecture.

Figures 7 and 8 show two behavioral architectures for the full adder design. The Figure 7 description does not use a process and is describing the operation conditionally, similar to the way a function table specifies operation.

```
ARCHITECTURE behavior of fulladder IS
BEGIN
     S <= '1' WHEN X='0' AND Y='0' AND Cin='1' ELSE
          '1' WHEN X='0' AND Y='1' AND Cin='0' ELSE
          '1' WHEN X='1' AND Y='0' AND Cin='0' ELSE
          '1' WHEN X='1' AND Y='1' AND Cin='1' ELSE
          '0';

     Cout <= '1' WHEN X='0' AND Y='1' AND Cin='1' ELSE
             '1' WHEN X='1' AND Y='0' AND Cin='1' ELSE
             '1' WHEN X='1' AND Y='1' AND Cin='0' ELSE
             '1' WHEN X='1' AND Y='1' AND Cin='1' ELSE
             '0';
END behavior;
```

Figure 7. fulladder Behavorial Architecture

The Figure 8 description also describes the full adder operation conditionally but using a process statement.

```
ARCHITECTURE behavior of fulladder IS
BEGIN
     PROCESS (X, Y, Cin)
     BEGIN
          IF X='0' AND Y='0' AND Cin='0' THEN
               S <= '0';
               Co <= '0';
          ELSIF X='0' AND Y='0' AND Cin='1' THEN
               S <= '1';
               Co <= '0';
          ELSIF X='0' AND Y='1' AND Cin='0' THEN
               S <= '1';
               Co <= '0';
          ELSIF X='0' AND Y='1' AND Cin='1' THEN
               S <= '0';
               Co <= '1';
          ELSIF X='1' AND Y='0' AND Cin='0' THEN
               S <= '1';
               Co <= '0';
          ELSIF X='1' AND Y='0' AND Cin='1' THEN
               S <= '0';
               Co <= '1';
          ELSIF X='1' AND Y='1' AND Cin='0' THEN
               S <= '0';
               Co <= '1';
          ELSIF X='1' AND Y='1' AND Cin='1' THEN
               S <= '1';
```

```
            Co <= '1';
        END IF;
    END PROCESS;
END behavior;
```

Figure 8. fulladder Behavioral Architecture using Process

The process starts execution when an event (signal change) for one of the signals in the sensitivity list occurs (in Figure 8, the sensitivity list is X, Y, Cin). The process statements are executed sequentially in order. When the last statement is executed the process is finished and any signal assignment is performed. The process is not executed again until another event occurs for a signal in the sensitivity list.

Process statements can be used to describe combinational and sequential logic, but one must be careful using processes for combinational logic as the compiler often interprets latches between the inputs and outputs. The design of Figure 8 compiled using Quartus generates warnings that latches have been inferred for the outputs. For the full adder to be properly synthesized, the process would need to be reworked to not infer latches for the outputs.

Warning (10631): VHDL Process Statement warning at fulladder.vhd(35): inferring latch(es) for signal or variable "S", which holds its previous value in one or more paths through the process
Warning (10631): VHDL Process Statement warning at fulladder.vhd(35): inferring latch(es) for signal or variable "Co", which holds its previous value in one or more paths through the process

Info (10041): Inferred latch for "Co" at fulladder.vhd(35)
Info (10041): Inferred latch for "S" at fulladder.vhd(35)

Overview of VHDL Language
VHDL Data Types
VHDL Standard:

- bit values: '0', '1'
- boolean values: TRUE, FALSE
- integer values: -(231) to +(231 - 1) {SUN Limit}
- natural values: 0 to integer'high (subtype of integer)
- positive values: 1 to integer'high (subtype of integer)
- character values: ASCII characters (eg. 'A')
- time values include units (eg. 10ns, 20us)
- bit_vector array (natural range <>) of bit

IEEE Standard 1164 (package ieee.std_logic_1164.all)

- std_ulogic values: 'U','X','1','0','Z','W','H','L','-'
- 'U' = uninitialized
- 'X' = unknown
- 'W' = weak 'X'

- 'Z' = floating
- 'H'/'L' = weak '1'/'0'
- '-' = don't care
- std_logic resolved "std_ulogic" values
- std_logic_vector array (natural range <>) of std_logic

VHDL Statements and Operators

VHDL concurrent statements include:

- Signal assignments (direct assignment, when else, with select)
- Process statement
- Component instantiation
- Generate statement
- Block statement

VHDL Sequential Statements include:

- Signal assignment
- Variable assignment
- Conditional statements (if elsif else, case)
- Loop statements (along with next and exit)

VHDL Arithmetic and Logical Operators include:

- Logical: and, or, nand, nor, xor, not (for boolean or bit ops)
- Relational: =, /=, <, <=, >, >=
- Arithmetic: +, -, *, /, mod, rem, **, abs
- Concatenate: &

References

None

Last modified Wednesday, August 24, 2022