**VHDL Introduction**

A language for describing the structural, physical and behavioral characteristics of digital systems.

Execution of a VHDL program results in a simulation of the digital system.

  Allows us to validate the design prior to fabrication.

  The definition of the VHDL language provides a range of features that support simulation of digital systems.

VHDL supports both *structural* and *behavioral* descriptions of a system at multiple levels of abstraction.

Structure and behavior are complementary ways of describing systems.

  A description of the *behavior* of a system says nothing about the *structure* or the components that make up the system.

  There are many ways in which you can build a system to provide the same behavior.

Events, Propagation Delay and Concurrency

VHDL allows you to specify:

- The *components* of a circuit.
- Their interconnection.
- The behavior of the components in terms of their input and output signals.

What are its behavioral properties of the half-adder circuit?

The *event* on \( a \), from 1 to 0, changes the outputs after a 5ns *propagation* delay. Both gates (and wires) have inertia or a natural resistance to change.
Events, Propagation Delay and Concurrency

A third property of this circuit is concurrency.

Both the xor and and gate compute new output values concurrently when an input changes state.

These new events may go on to initiate the computation of other events in other parts of the circuit, e.g. s1 and s3.

Data driven system: Events on signals lead to computations that may generate events on other signals.
Discrete Event Simulation
We can view VHDL as a programming language for describing the generation of events in digital systems supported by a *discrete event simulator*.

A *discrete event simulator* executes VHDL code, modeling the passage of time and the occurrence of events at various points in time.

It maintains an event list data structure to keep track of the order of all future events in the circuit.

Simulator clock

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>5ns</td>
<td>1 -&gt; 0</td>
</tr>
<tr>
<td>5ns</td>
<td>a@5ns</td>
</tr>
<tr>
<td>10ns</td>
<td>0 -&gt; 1</td>
</tr>
<tr>
<td>10ns</td>
<td>b@10ns</td>
</tr>
</tbody>
</table>

An **event**: A change in the value of a signal

**Timestamp**: Time at which an event is to occur.

Advance simulation clock to time of next event, update signals receiving values. Evaluate all components affected by signal updates and schedule new events.
Basic Language Concepts

*Signals*: Like variables in a programming language such as C, signals can be assigned values, e.g., 0, 1, Z.

However, signals also have an associated *time value*.

A signal receives a value at a specific point in time and retains that value until it receives a new value at a future point in time.

The sequence of values assigned to a signal over time is the *waveform* of the signal.

A variable always has one current value.

At any instant in time, a signal may be associated with several *time-value* pairs.
**Entity-Architecture**

Design entity: A component of a system whose behavior is to be described and simulated.

Two components to the description:
- The interface to the design: `entity` declaration.
- The internal behavior of the design: `architecture` construct.

Entity example for half adder:

```vhdl
entity half_adder is port(  -- Note: VHDL is CaSe insensitive.
    a, b: in bit;
    sum, carry: out bit);
end half_adder;
```

`half_adder` is the name given to the design entity.

The input and outputs signals; `a`, `b`, `sum` and `carry`, are referred to as _ports_.

---

*Note: V H D L  is C a S e in s e n t i v e.*
**Entity-Architecture**

Each port has a type, *bit* and *bit_vector* can assume values of 0 and 1.

Each port has a mode; *in*, *out* or *inout* (bidirectional signals).

Bit vectors are specified as:

```
entity ALU32 is port(
  A, B: in bit_vector(31 downto 0);
  C: out bit_vector(31 downto 0);
  Op: in bit_vector(5 downto 0);
  N, Z: out bit);
end half_adder;
```

**A** and **B** are 32 bits long with the most significant bit as 31.

A more general definition of *bit* and *bit_vector* are *std_logic* and *std_logic_vector*, which can assume more than just 0 and 1.
Entity-Architecture

Architecture construct:

architecture arch_name of entity_name is
    -- place declarations here
begin
    -- place description of behavior here
end half_adder_arch;

Concurrent statements:
Signal assignment statements specify the new value and the time at which the
signal is to acquire this value.
The textual order of the concurrent signal assignment statements (CSAs) do
NOT effect the results.

architecture half_adder_arch of half_adder is
begin
    sum <= (a xor b) after 5 ns;
carry <= (a and b) after 5 ns;
end half_adder_arch;
Entity-Architecture

We can also use (local) signals internal to the architecture, e.g., \( s_1 \), \( s_2 \) and \( s_3 \) in the full adder circuit.

```
entity full_adder is port(
    a, b, c_in: in bit;
    sum, carry: out bit);
end half_adder;
```
**Entity-Architecture**

```vhdl
architecture full_adder_arch of full_adder is

  signal s1, s2, s3: bit;
  constant gate_delay: Time:= 5 ns;

begin
  L1: s1 <= (a xor b) after gate_delay;
  L2: s2 <= (c_in and s1) after gate_delay;
  L3: s3 <= (a and b) after gate_delay;
  L4: sum <= ( s1 xor c_in) after gate_delay;
  L5: carry <= (s2 or s3) after gate_delay;

end full_adder_arch;
```

[Diagram of signal timing with labels for a, b, c_in, sum, and carry with time markers 15 to 45.]
Entitly-Architecture

The following statements are also legal:

\[
\begin{align*}
s1 & \le (a \texttt{ xor } b) \texttt{ after } 5 \text{ ns}, (a \texttt{ or } b) \texttt{ after } 10 \text{ ns}, (\texttt{not } a) \texttt{ after } 15 \text{ ns}; \\
\text{wave} & \le '0', '1' \texttt{ after } 10 \text{ ns}, '0' \texttt{ after } 15 \text{ ns}, '1' \texttt{ after } 25 \text{ ns};
\end{align*}
\]

A \textit{driver list} that specifies a waveform.

This statement generates a set of \textit{transactions} (time-value pairs) to be carried out at distinct times in the future.
...}

with addr1 select

The Selected Signal Assignment Statement behaves similarly. The first conditional found to be true determines the value transferred to the output.

end behavioral;

end

architecture behavioral of mux4 is

end mux4;

Z : out bit_vector(7 downto 0);

SO, SI : in bit;
in0, in1, in2, in3 : in bit;

entity mux4 is port(

Conditional Signal Assignment Statement

Other VHDL Constructs

VHDL

Principles of VLSI Design

CmSc 491B/711

UMB
Modeling Behavior, Processes

Processes are used:

- For describing component behavior when they cannot be simply modeled as delay elements.
- To model systems at high levels of abstraction.

Process incorporate conventional programming language constructs.

A process is a **sequentially** executed block of code, which contains:

- arrays and queues.
- Variable assignments, e.g., \( x := y \), which, unlike signals, take effect immediately.
- *if-then-else* and loop statements to control flow.
- Signal assignments to external signals.

Processes contain *sensitivity lists* in which signals are listed, which determine when the process executes.

In reality, CSAs are also processes without the *process, begin* and *end* keywords.
Modeling Behavior, Processes

entity half_adder is port(
    a, b: in bit;
    sum, carry: out bit;
end mux4;

architecture behavior of half_adder is
begin
    sum_proc: process (a, b) begin
        if (a = b) then
            sum <= '0' after 5 ns;
        else
            sum <= (a or b) after 5 ns;
        end if;
    end process;

carry_proc: process (a, b) begin
    case a is
        when '0' => carry <= a after 5 ns;
        when '1' => carry <= b after 5 ns;
        when others => carry <= 'X' after 5 ns;
    end case;
end process;
end behavior;
Modeling Behavior, Looping

Looping constructs include the *for* and *while* statements, e.g.,

```vhdl
for index in 1 to 32 loop
  if product_register(0) = '1' then
    product_register(63 downto 32) := produce_register(63 downto 32) +
    multiplicand_register(31 downto 0);
  end if;
  product_register(63 downto 0) := '0' product_register(63 downto 1);
end loop;
```

The loop *index* is implicitly declared, local to the loop and cannot be changed.

Alternatively;

```vhdl
while (j < 32) loop
  ...
  j := j + 1;
end loop;
```
Modeling Behavior, The Wait Statement

Processes are executed once upon initialization.

Thereafter, they are executed in a data-driven manner by either:
- an event on one or more signals in the sensitivity list.
- waiting for the occurrence of specific event using a wait statement.

The wait statement specifies the conditions under which a process may resume execution after being suspended, e.g.,

\[
\text{wait for } \text{time expression}; \quad \text{-- wait for a specified time interval.}
\]
\[
\text{wait on } \text{signal}; \quad \text{-- wait on a signal(s).}
\]
\[
\text{wait until } \text{condition}; \quad \text{-- wait until condition becomes true;}
\]

The first and third form allow processes to model components that are not necessarily data driven.

wait statements also allow processes to suspend at multiple points, and not just at the beginning.
A D flip-flop with asynchronous reset (R) and set (S) inputs given in reference.

On signal Clk:

Note attribute Clk'event which is true when an event (rising or falling edge) occurs on signal Clk.

end diff-arch;
end process output;
output: process begin
begin
architecture diff-arch of dff is
end dff;
entity dff is port
  Clk: in bit;
  D, Q, Qbar: out bit;
end entity dff;
end architecture dff_arch;

wait until ( Clk'event and Clk = '1' );

For example, a positive edge-triggered flip-flop:

Modeling Behavior, The Wait Statement
Modelling State Machines

A state machine (Mealy machine):

- **Combinational Logic**

  - **Inputs**
  - **Outputs**
  - **State**
  - **Clock**

Combinational part implemented in one process, sensitive to events on the input signals or state variables.

Sequential part implemented in a second process, sensitive to the rising edge of the clock.
Modeling State Machines

entity state_machine is port(
  reset, clk, x: in bit;
  z: out bit;
end state_machine;

architecture behavioral of state_machine is

type statetype is (state0, state1);
signal state, next_state: statetype := state0;
begin

  comb_process: process (state, x) begin
    case state is
      when state0 => if x = '0' then next_state <= state1; z <= '1';
                     else next_state <= state0; z <= '0'; end if;
      when state1 => if x = '1' then next_state <= state0; z <= '0';
                    else next_state <= state1; z <= '1'; end if;
    end case; end process comb_process;

  clk_process: process begin
    wait until (clk'event and clk = '1')
    if reset = '1' then state <= statetype'left;
    else state <= next_state; end if;
  end process clk_process;
end behavioral;
Modeling Structure

Structural model: A description of a system in terms of the interconnection of its components, rather than a description of what each component does.

A structural model does NOT describe how output events are computed in response to input events.

How do we simulate the circuit?

Behavioral models of each component are assumed to be provided.

A VHDL structural description must possess:

• The ability to define the list of components.
• The definition of a set of signals to be used to interconnect them.
• The ability to uniquely label (distinguish between) multiple copies of the same component.
Modeling Structure

A structural description of a full adder:

```vhdl
entity full_adder is port(
  in1, in2, c_in: in bit;
  sum, c_out: out bit;
end full_adder;

architecture structural of full_adder is
  component half_adder
    port ( a, b: in bit; sum, carry: out bit );
  end component;
  component or_2
    port ( a, b: in bit; c: out bit );
  end component;
  signal s1, s2, s3: bit;
begin
  H1: half_adder port map( a => in1, b => in2, sum => s1, carry => s3 );
  H2: half_adder port map( a => s1, b => c_in, sum => sum, carry => s2 );
  O1: or_2 port map( a => s2, b => s3, c => c_out );
end structural;
```

Component Declarations

Signal Declarations

Component Interconnection
**Modeling Structure**

A state machine of a bit-serial adder:

![Diagram of a bit-serial adder with state transitions and logic gates]

- **Inputs:** a, b
- **Outputs:** z, carry
- **States:** 0, 1
- **Transitions:**
  - 00/0 → 01/1
  - 01/1 → 10/0
  - 10/0 → 11/1
  - 11/1 → 00/1

---

**Combination Logic**

- **Inputs:** a, b
- **Output:** z
- **Carry:**
  - 00/0 → 1
  - 01/1 → 0
  - 10/1 → 0
  - 11/1 → 1
- **Clock (Clk):**
  - 00/0 → Q
  - 01/1 → Q
  - 10/1 → Q
  - 11/1 → Q
- **Reset (R):**
  - 00/0 → Q
  - 01/1 → Q
  - 10/1 → Q
  - 11/1 → Q
Modeling Structure

A structural description of a bit-serial adder:

```vhdl
entity serial_adder is port(
    a, b, clk, reset: in bit;
    z: out bit;
end serial_adder;

architecture structural of serial_adder is
    component comb
        port ( a, b, c_in: in bit; z, carry: out bit );
    end component;
    component dff
        port ( clk, reset, d: in bit; q, qbar: out bit );
    end component;
    signal s1, s2: bit;
begin
    C1: comb port map( a => a, b => b, c_in => s1, z => z, carry => s2 );
    D1: dff port map( clk => clk, reset => reset, d => s2, q => s1,
                      qbar => open );
end structural;
```