A Half-adder in VHDL

-- A half-adder circuit
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
  port(a, b : in std_logic;
       sum, carry : out std_logic);
end half_adder;

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

Entities and Architectures

* Every VHDL design has at least one entity / architecture pair (or design entity).
* An entity declaration defines a circuit's input and output interfaces (like schematic block symbol).
* Every referenced entity in a VHDL design must be bound to an architecture.
* The architecture describes the actual function or contents (like a lower-level schematic).

Entity Declaration

* Provides complete interface for a circuit.
* The entity declaration includes:
  * A name (i.e., half_adder).
  * A port list defining the inputs and outputs.
* The port list includes:
  * Name of each port (i.e., a, b, sum, and carry).
  * Direction of each port (i.e., in, out, and inout).
  * Type of each port (i.e., std_logic).
Architecture Declaration and Body

- Describes underlying function and/or structure of a circuit.
- Multiple architectures allowed for each entity.
- The architecture declaration includes:
  - A unique name (i.e., conur_beih).
  - Name of the entity bound to (i.e., half_adder).
  - Optional local declarations (signals, components, etc.).
  - Concurrent statements between begin and end to describe the function or contents of the block.

Example: A Comparator

Comparator

-- Eight-bit comparator
library ieee;
use ieee.std_logic_1164.all;
entity compare is
  port(A,B:in std_logic_vector(7 downto 0);
       EQ:out std_logic);
end compare;

architecture compare1 of compare is
begin
  EQ <= '1' when (A=B) else '0';
end compare1;

Concurrent Signal Assignments

- Describe logic that is inherently parallel.
- No relevance to order of assignments.
- Used to describe combinational logic or connections between lower-level components.

architecture arch3 of simple_circuit is
signal A,B:std_logic;
signal Y1,Y2:std_logic;
begin
  Y1<=not (A and B) after 5 ns;
  Y2<=not (A or B) after 5 ns;
end arch3;
### Std_uLogic

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U'</td>
<td>Uninitialized</td>
</tr>
<tr>
<td>'X'</td>
<td>Forcing Unknown</td>
</tr>
<tr>
<td>'0'</td>
<td>Forcing 0</td>
</tr>
<tr>
<td>'1'</td>
<td>Forcing 1</td>
</tr>
<tr>
<td>'Z'</td>
<td>High impedance</td>
</tr>
<tr>
<td>'W'</td>
<td>Weak unknown</td>
</tr>
<tr>
<td>'L'</td>
<td>Weak 0</td>
</tr>
<tr>
<td>'H'</td>
<td>Weak 1</td>
</tr>
<tr>
<td>'-'</td>
<td>Don't care</td>
</tr>
</tbody>
</table>

### Resolved and Unresolved Types

- A signal requires *resolution* whenever it is simultaneously driven with more than one value.
- Default is unresolved, resulting in errors when multiple values driven onto a signal.
- If multiple values are intended to drive a signal (e.g., a bus interface), you need a *resolved* type.
- To resolve a type, a *resolution* function is provided specifying all possible combinations and result.

### Std_logic

```vhdl
subtype std_logic is resolved std_uLogic;
```

<table>
<thead>
<tr>
<th>'U'</th>
<th>'X'</th>
<th>'0'</th>
<th>'1'</th>
<th>'Z'</th>
<th>'W'</th>
<th>'L'</th>
<th>'H'</th>
<th>'-'</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
</tr>
<tr>
<td>'X'</td>
<td>'U'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
</tr>
<tr>
<td>'0'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'X'</td>
<td>'0'</td>
<td>'0'</td>
<td>'0'</td>
<td>'X'</td>
</tr>
<tr>
<td>'1'</td>
<td>'U'</td>
<td>'X'</td>
<td>'1'</td>
<td>'1'</td>
<td>'1'</td>
<td>'1'</td>
<td>'1'</td>
<td>'X'</td>
</tr>
<tr>
<td>'Z'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'Z'</td>
<td>'W'</td>
<td>'L'</td>
<td>'H'</td>
</tr>
<tr>
<td>'W'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'W'</td>
<td>'W'</td>
<td>'W'</td>
<td>'W'</td>
</tr>
<tr>
<td>'L'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'L'</td>
<td>'W'</td>
<td>'L'</td>
<td>'W'</td>
</tr>
<tr>
<td>'H'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'H'</td>
<td>'W'</td>
<td>'W'</td>
<td>'H'</td>
</tr>
<tr>
<td>'-'</td>
<td>'U'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
</tr>
</tbody>
</table>
Conditional Signal Assignment

-- 4-input multiplexer

architecture mux1 of mux is
begin
Y <= A after 5 ns when (Sel="00") else
  B after 5 ns when (Sel="01") else
  C after 5 ns when (Sel="10") else
  D after 5 ns;
end mux1;

Conditional vs. Selected Assignment

Q1<="01" when A='1' else
  "10" when B='1' else
  "11" when C='1' else
  "00"

* is identical to the selected assignment:

with std_logic_vector'(A,B,C) select
q2<="01" when "100",
  "01" when "101",
  "01" when "110",
  "01" when "111",
  "10" when "010",
  "10" when "011",
  "11" when "001",
  "00" when others;

Selected Signal Assignment

architecture mux2 of mux is
begin
  with Sel select
    Y <= A after 5 ns when "00",
        B after 5 ns when "01",
        C after 5 ns when "10",
        D after 5 ns when others;
end mux2;

Conditional vs. Selected Assignment

* If A, B, and C never active at the same time:

with std_logic_vector'(A,B,C) select
q2<="01" when "100",
  "10" when "010",
  "11" when "001",
  "00" when others;
Conditional vs. Selected Assignment

- You must include all possible conditions in a selected assignment (can use `others` clause).
- The selection may include ranges:

```vhdl
with Address select
  CS <= SRAM when 0x"0000" to 0x"7FFF",
        PORT when 0x"8000" to 0x"81FF",
        UART when 0x"8200" to 0x"83FF",
        PROM when others;
```

Conditional vs. Selected Assignment

- VHDL'93 adds keyword `unaffected`:

```vhdl
with Sel select
  Y <= A when Sel="00",
       B when Sel="01",
       C when Sel="10",
       unaffected when others;
```
- Above may result in latch from synthesis.

Delay Specifications

- VHDL allows signal assignments to include wire and gate delays in the form of an `after` clause.
- While not typically used for synthesis, it can improve the accuracy of your simulation.
- There are two types of delay specifications:
  - Inertial - models delay through a gate, in which a minimum pulse length is needed for event to occur.
  - Transport - models delay on a wire, in which pulses of any width are propagated.

Inertial Delay

```vhdl
Y1 <= \text{not}(A \text{ and } B) \text{ after } 7 \text{ ns};
```

![Inertial Delay Diagram]
Transport Delay

Y2 <= not(A and B) transport after 7 ns;

Event is propagated

Reject Time

※ VHDL’93 adds feature called reject time.
※ For inertial delay, a minimum pulse can be given:
Y1 <= reject 3 ns not(A and B) after 7 ns;
※ Any event greater than 3 ns in width propagates.
※ Note: without a reject time, it would be 7 ns.

Delta Delays

What We’ve Learned So Far

※ Entity and architecture constructs.
※ Difference between std_logic and std_ulogic.
※ Concurrent signal assignment:
  ※ simple,
  ※ conditional, and
  ※ selected.
※ Inertial, transport, and delta delays.
※ NEXT: Modeling behavior in VHDL.