LECTURE 3: The VHDL N-bit Adder

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University
Full Adder: Truth Table

- A Full-Adder is a Combinational circuit that forms the arithmetic sum of three input bits.
- It consists of three inputs \((z, x, y)\) and two outputs \((\text{Carry}, \text{Sum})\) as shown.

<table>
<thead>
<tr>
<th>z</th>
<th>x</th>
<th>y</th>
<th>c</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[s = x \oplus y \oplus z\]

\[c = xy + xz + yz = xy + z (x \oplus y)\]

Karnaugh maps
Combinatorial Logic Operators

NOT
\[ z \leq \text{NOT}(x); \quad z \leq \text{NOT} \ x; \]

AND
\[ z \leq x \text{ AND} \ y; \]

NAND
\[ z \leq \text{NOT}(x \text{ AND} \ y); \]

OR
\[ z \leq x \text{ OR} \ y; \]

NOR
\[ z \leq \text{NOT}(x \text{ OR} \ y); \]

XOR
\[ z \leq (x \text{ and NOT} \ y) \text{ OR} (\text{NOT} x \text{ AND} \ y); \]

XNOR
\[ z \leq (x \text{ and} \ y) \text{ OR} (\text{NOT} x \text{ AND} \ \text{NOT} y); \]
Full Adder: Architecture

ENTITY full_adder IS
  PORT (x, y, z: IN std_logic;
        Sum, Carry: OUT std_logic);
END full_adder;

ARCHITECTURE full_adder_arch_1 OF full_adder IS
BEGIN
  Sum <= ((x XOR y) XOR z);
  Carry <= ((x AND y) OR (z AND (x AND y)));
END full_adder_arch_1;
SIGNAL: Scheduled Event

- **SIGNAL**
  Like variables in a programming language such as C, signals can be assigned values, e.g. 0, 1
- 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 (i.e. scheduled event)
- The waveform of the signal is
  a sequence of values assigned to a signal over time
- For example
  wave <= ‘0’, ‘1’ after 10 ns, ‘0’ after 15 ns, ‘1’ after 25 ns;
Full Adder: Architecture with Delay

ARCHITECTURE full_adder_arch_2 OF full_adder IS
SIGNAL S1, S2, S3: std_logic;
BEGIN
s1 <= ( a XOR b ) after 15 ns;
s2 <= ( c_in AND s1 ) after 5 ns;
s3 <= ( a AND b ) after 5 ns;
Sum <= ( s1 XOR c_in ) after 15 ns;
Carry <= ( s2 OR s3 ) after 5 ns;
END;
**Signal order: Does it matter? No**

ARCHITECTURE `full_adder_arch_2` OF `full_adder` IS
SIGNAL S1, S2, S3: std_logic;
BEGIN
  s1  <= ( a XOR b ) after 15 ns;
  s2  <= ( c_in AND s1 ) after 5 ns;
  s3  <= ( a AND b ) after 5 ns;
  Sum <= ( s1 XOR c_in ) after 15 ns;
  Carry <= ( s2 OR s3 ) after 5 ns;
END;

ARCHITECTURE `full_adder_arch_3` OF `full_adder` IS
SIGNAL S1, S2, S3: std_logic;
BEGIN
  Carry <= ( s2 OR s3 ) after 5 ns;
  Sum  <= ( s1 XOR c_in ) after 15 ns;
  s3   <= ( a AND b ) after 5 ns;
  s2   <= ( c_in AND s1 ) after 5 ns;
  s1   <= ( a XOR b ) after 15 ns;
END;
The Ripple-Carry n-Bit Binary Parallel Adder
Hierarchical design: 2-bit adder

- The design interface to a two bit adder is

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY adder_bits_2 IS
  PORT (Cin: IN std_logic;
        a0, b0, a1, b1: IN std_logic;
        S0, S1: OUT std_logic;
        Cout: OUT std_logic);
END;

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY adder_bits_2 IS
  PORT (Cin: IN std_logic;
        a0, b0, a1, b1: IN std_logic;
        S0, S1: OUT std_logic;
        Cout: OUT std_logic);
END;
```

- Note: that the ports are **positional dependant**
  
  (Cin, a0, b0, a1, b1, S0, S1, Cout)
Hierarchical design: Component Instance

ARCHITECTURE ripple_2_arch OF adder_bits_2 IS
  COMPONENT full_adder
    PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);
  END COMPONENT;
  SIGNAL t1: std_logic;
BEGIN
  FA1: full_adder PORT MAP (Cin, a0, b0, S0, t1);
  FA2: full_adder PORT MAP (t1, a1, b1, s1, Cout);
END;
Positional versus Named Association

- Positional Association (must match the port order)

  FA1: full_adder PORT MAP (Cin, a0, b0, S0, t1);

- Named Association: signal => port_name

  FA1: full_adder PORT
     MAP (Cin=>x, a0=>y, b0=>z, S0=>Sum, t1=>Carry);

  FA1: full_adder PORT
     MAP (Cin=>x, a0=>y, b0=>z, t1=>Carry, S0=>Sum);

  FA1: full_adder PORT
     MAP (t1=>Carry, S0=>Sum, a0=>y, b0=>z, Cin=>x);
ARCHITECTURE ripple_2_arch OF adder_bits_2 IS
  COMPONENT full_adder
    PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);
  END COMPONENT;
  SIGNAL t1: std_logic; -- Temporary carry signal
BEGIN
  -- Named association
  FA1: full_adder PORT MAP (Cin=>x, a0=>y, b0=>z, S0=>Sum, t1=>Carry);

  -- Positional association
  FA2: full_adder PORT MAP (t1, a1, b1, s1, Cout);
END; -- Comments start with a double dash
**Using vectors: std_logic_vector**

By using **vectors**, there is less typing of variables, a₀, a₁, ...

```vhdl
ENTITY adder_bits_2 IS
  PORT (Cin: IN std_logic;
        a0, b0, a1, b1: IN std_logic;
        S0, S1: OUT std_logic;
        Cout: OUT std_logic)
); END;
```

```vhdl
ENTITY adder_bits_2 IS
  PORT (Cin: IN std_logic;
        a, b: IN std_logic_vector(1 downto 0);
        S: OUT std_logic_vector(1 downto 0);
        Cout: OUT std_logic)
); END;
```
2-bit Ripple adder using std_logic_vector

- Note, the signal variable usage is now different:
  a0 becomes a(0)

ARCHITECTURE ripple_2_arch OF adder_bits_2 IS
  COMPONENT full_adder
    PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);
  END COMPONENT;
  SIGNAL t1: std_logic; -- Temporary carry signal
BEGIN
  FA1: full_adder PORT MAP (Cin, a(0), b(0), S(0), t1);

  FA2: full_adder PORT MAP (t1, a(1), b(1), s(1), Cout);
END;
ARCHITECTURE ripple_4_arch OF adder_bits_4 IS

COMPONENT full_adder

PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);

END COMPONENT;

SIGNAL t: std_logic_vector(3 downto 1);

BEGIN

FA1: full_adder PORT MAP (Cin, a(0), b(0), S(0), t(1));
FA2: full_adder PORT MAP (t(1), a(1), b(1), S(1), t(2));
FA3: full_adder PORT MAP (t(2), a(2), b(2), S(2), t(3));
FA4: full_adder PORT MAP (t(3), a(3), b(3), S(3), Cout);

END;

• std_vectors make it easier to replicate structures
ARCHITECTURE ripple_4_arch OF adder_bits_4 IS

  COMPONENT full_adder
      PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);
  END COMPONENT;

  SIGNAL t: std_logic_vector(3 downto 1);
  CONSTANT n: INTEGER := 4;

BEGIN

  FA1: full_adder PORT MAP (Cin, a(0), b(0), S(0), t(1));
  FA2: full_adder PORT MAP (t(1), a(1), b(1), S(1), t(2));
  FA3: full_adder PORT MAP (t(2), a(2), b(2), S(2), t(3));
  FA4: full_adder PORT MAP (t(n), a(n), b(n), S(n), Cout);

  FA_f: for i in 1 to n-2 generate
      FA_i: full_adder PORT MAP (t(i), a(i), b(i), S(i), t(i+1));
  end generate;

END;

LABEL: before the for is not optional

Constants never change value
ARCHITECTURE ripple_4_arch OF adder_bits_4 IS

COMPONENT full_adder
  PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);
END COMPONENT;

SIGNAL t: std_logic_vector(4 downto 0);
CONSTANT n: INTEGER := 4;

BEGIN
  t(0) <= Cin; Cout <= t(n);
  FA_f: for i in 0 to n-1 generate
    FA_i: full_adder PORT MAP (t(i), a(i), b(i), S(i), t(i+1));
  end generate;
END;
N-bit adder using **generic**

- **ENTITY** adder_bits_4 IS
  
  PORT (Cin: IN std_logic;
        a, b: IN std_logic_vector(3 downto 0);
        S:    OUT std_logic_vector(3 downto 0);
        Cout: OUT std_logic);
  
  END;  

- **ENTITY** adder_bits_n IS
  
  GENERIC(n: INTEGER := 2);
  
  PORT (Cin: IN std_logic;
        a, b: IN std_logic_vector(n-1 downto 0);
        S:    OUT std_logic_vector(n-1 downto 0);
        Cout: OUT std_logic);
  
  END;

- **By using generics, the design can be generalized**
For-Generate statement: third improvement

ARCHITECTURE ripple_n_arch OF adder_bits_n IS
  COMPONENT full_adder
    PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);
  END COMPONENT;
  SIGNAL t: std_logic_vector(n downto 0);
BEGIN
  t(0) <= Cin; Cout <= t(n);
  FA: for i in 0 to n-1 generate
    FA_i: full_adder PORT MAP (t(i), a(i), b(i), S(i), t(i+1));
  end generate;
END;
ARCHITECTURE tb OF tb_adder_4 IS

COMPONENT adder_bits_n
    GENERIC(n: INTEGER := 2);
    PORT ( Cin: IN     std_logic;
          a, b: IN     std_logic_vector(n-1 downto 0);
          S: OUT std_logic_vector(n-1 downto 0);
          Cout: OUT std_logic
    END COMPONENT;

SIGNAL        x, y, Sum:      std_logic_vector(n downto 0);
SIGNAL        c, Cout:          std_logic;

BEGIN
    x <= "0000", "0001" after 50 ns, "0101", after 100 ns;
    y <= "0010", "0011" after 50 ns, "1010", after 100 ns;
    c <= ‘1’, ‘0’ after 50 ns;
    UUT_ADDER_4: adder_bits_n GENERIC MAP(4)
    PORT MAP (c, x, y, Sum, Cout);

END;
ENTITY tb_adder_4 IS
  PORT (Sum: std_logic_vector(3 downto 0);
        Cout: std_logic
  ); END;

The output of the testbench will be observed by the digital waveform of the simulator.