Basic Language Concepts: Synthesis

vhdl model

entity my_ckt is
port(x, y :in bit;
     z : out bit)
end entity my_ckt;
architecture behavioral of my_ckt is
begin
    --
    -- some code here
    --
end architecture behavioral;

- Synthesis process depends on
  - hardware primitives
  - synthesis compiler
- Relationship between implementations and concurrent signal assignment statements
Inference From Declarations

- How are signals implemented?
  - as wires
  - as latches of flip flops
  - choice depends on how a signal is used

- Types
  - for simulation provides a range of values
  - for synthesis determines bit widths
    ```
    signal result: std_logic_vector (12 downto 0);
    signal count: integer;
    signal index: integer range 0 to 18;
    
    type state_type is (state0, state1, state2, state3);
    signal next_state: statetype;
    ```
Inference From Declarations (cont.)

• Bit width required for a signal can be determined explicitly or implicitly
• Hints go a long way towards minimizing hardware
• Think of hardware description!
Inference from Simple Concurrent Signal Assignment Statements

- Behavior corresponds to that of a combinational circuit
  - model produces one signal assignment statement for every signal in the circuit
  - when inputs (RHS) change the output is recomputed
- Operator inferencing
  - operations, gate types
  - interconnect
- Delay information removed
Inference from Simple CSAs

• Operator precedence controls the depth of the circuit

library IEEE;
use IEEE.std_logic_1164.all;

entity concurrent is
port (s, t, u, w: in std_logic;
     v: out std_logic);
end entity concurrent;

architecture dataflow of concurrent is
signal s1, s2 : std_logic;
begins
L1: s1 <= s and t and u and w;
L2: s2 <= (s and t) and (u and w);
L3: v <= s1 or s2;
end architecture dataflow;
Inference from Simple CSAs - Example

architecture dataflow of full_adder is
signal s1, s2, s3: std_ulogic;
begin
sum <= in1 xor in2 xor c_in;
c_out <= (in1 and c_in) or (in2 and c_in) or (in1 and in2);
end architecture dataflow;
Inference from Conditional CSAs

- Multiplexor logic

```
architecture behavioral of mux4 is
begin
z <= in0 when s0 = '0' and s1 = '0' else
    in1 when s0 = '0' and s1 = '1' else
    in2 when s0 = '1' and s1 = '0' else
    in3 when s0 = '1' and s1 = '1' else
    '0';
end architecture behavioral;
```

- Any change on input signals causes recomputation
- Implied priority order!
  - priority logic may be optimized for mutually exclusive branches
- Six variable boolean equation and 4 input LUTs
  - effectively produces a gate level implementation of a mux
Inference from Conditional CSAs

Synthesized
Gate Level
Implementation

FPGA
Implementation
Example: Priority Encoder

architecture behavior of priority is
begin
  z <= "00" when datain (0) = '1' and valid = '1' else
  "10" when datain (2) = '1' and valid = '1' else
  "01" when datain (1) = '1' and valid = '1' else
  "11" when datain (3) = '1' and valid = '1' else
  "00";
end architecture behavior;

• note generation of priority logic
• must be mapped to LUTs
Synthesis of Comparison Logic

- Comparisons that make sense in a simulation may not be meaningful for synthesis
  - for example, comparisons with don’t care symbols are assumed to return false!
- Consider implementations when writing synthesizeable VHDL code
- Equality tests for other than 0/1 return false
Example: Synthesis and Comparison Logic

```
architecture behavior of priority is
begin
  z <= "00" when datain = "---1" and valid = '1' else
  "10" when datain = "-100" and valid = '1' else
  "01" when datain = "--10" and valid = '1' else
  "11" when datain = "1000" and valid = '1' else
  "00";
end architecture behavior;
```
Inference from Selected CSAs

- All choices are evaluated and only one must be true
- No priority logic implied
- Example

```vhdl
architecture behavior of priority is
begin
    z <= "00" when datain = "---1" and valid = '1' else
         "10" when datain = "-100" and valid = '1' else
         "01" when datain = "--10" and valid = '1' else
         "11" when datain = "1000" and valid = '1' else
         "00";
end architecture behavior;
```

- note how output bits are set
- two gate (LUT) implementation
Inference from Selected CSAs

• The “unaffected” keyword (VHDL’93 only)

```vhdl
with datain select
result <= "00" when "0001",
"01" when "0010",
"10" when "0100",
"11" when "1000",
unaffected when others;
```

• for this value of the select expression the output is unaffected
• the output retains its previous value

→ a latch is inferred!
Synthesis Hints/Issues Using CSAs

- Simulation - synthesis mismatches
  - delay statements
  - comparison logic
- No initialization in declarations: use explicit resets
- Provide hints in declarations
- Use of the “unaffected” keyword may cause latches to be inferred
- Optimize the “when others” clause: use of don’t care logic
- Parentheses to control circuit depth and therefore speed
- Selected signal assignment vs. conditional signal assignment statements
  - former causes less logic to be inferred