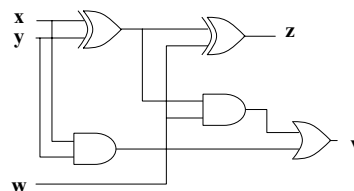


Basic Language Concepts: Synthesis

Synthesis from VHDL Descriptions

```
entity my_ckt is
  port(x, y, w : in bit;
        v, z : out bit);
end entity my_ckt;

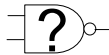
architecture behavioral of my_ckt is
begin
  --
  -- some code here
  --
end architecture behavioral;
```



- Produce (infer) a correct hardware implementation
 - Note that multiple alternative implementations are feasible
- Synthesis process depends on
 - Hardware primitives
 - Synthesis goals, e.g., area vs. speed
- What are the challenges/issues?

```
library IEEE;
use IEEE.std_logic_1164.all;
entity synth is
  port (A, B, C, D: in integer;
        Sel : in std_logic_vector(1 downto 0);
        Z : out integer);
end entity synth;
```

```
architecture behavioral of synth is
begin
  with Sel select
  Z <=   A+B when "00",
        C + D   when "10",
        x"00000000" when others;
end architecture behavioral;
```



- We need to infer
 - Bit widths
 - Number of functional units
- What about control?
 - Is there an implied priority order among options?
- What is the baseline against which reconcile options?
 - Language defines the semantics

- Inference from declarations
- Inference from concurrent signal assignment statements
- In all cases, inference is grounded in language semantics
 - For example, the definition of an integer as comprising 32-bits



Inference from Declarations

```
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;
```

- How are signals implemented?
 - As wires
 - As latches of flip flops
 - Choice depends on how a signal is used
- Inference from data types
 - For simulation, type definition determines a range of values
 - For synthesis, type definition determines bit widths

ECE 4170 (5)



Inference from Declarations (cont.)

- Bit width required for a signal can be determined explicitly or implicitly
 - Programmer defined
 - Via program analysis
- Hints go a long way towards minimizing hardware
- Enumerated types implicitly define the bit width of a signal
- What can you do?

ECE 4170 (6)

Inference from Simple Concurrent Signal Assignment Statements

- Take a *signal-centric* view
 - Consider signals with a single driver
- Behavior corresponds to that of a combinational circuit
 - When inputs (RHS) change the output is recomputed
 - Model produces one signal assignment statement for every signal in the circuit
- Operator inferencing
 - Operations and corresponding gate types
 - Interconnection between operators
- Delay information is ignored
 - Determined from the library primitives

ECE 4170 (7)

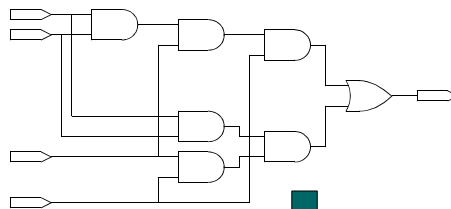
Inference from Simple CSAs

```
library IEEE;
use IEEE.std_logic_1164.all;

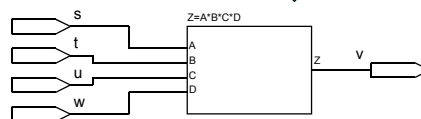
entity concurrent is
port (s, t, u, w: in std_logic;
      v: out std_logic);
end entity concurrent;

architecture dataflow of concurrent is
signal s1, s2 : std_logic;
begin
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;
```

Synthesized gate level implementation



FPGA implementation

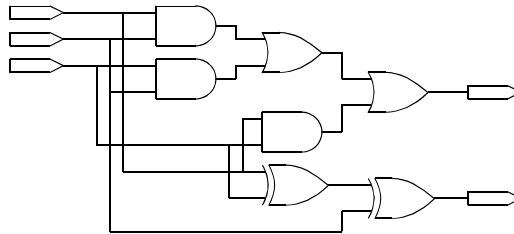
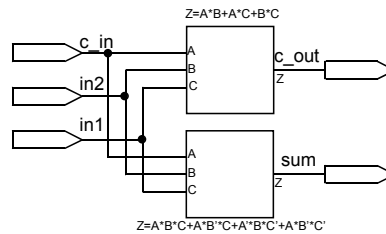


- Operator precedence controls the depth of the circuit

ECE 4170 (8)

Inference from Simple CSAs (cont.)

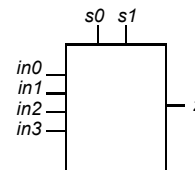
```
architecture dataflow of full_adder is
  signal s1, s2, s3: std_logic;
begin
  sum <= in1 xor in2 xor c_in;
  c_out <= (in1 and c_in) or (in2 and c_in) or (in1 and in2);
end architecture dataflow;
```



ECE 4170 (9)

Inference from Conditional CSAs

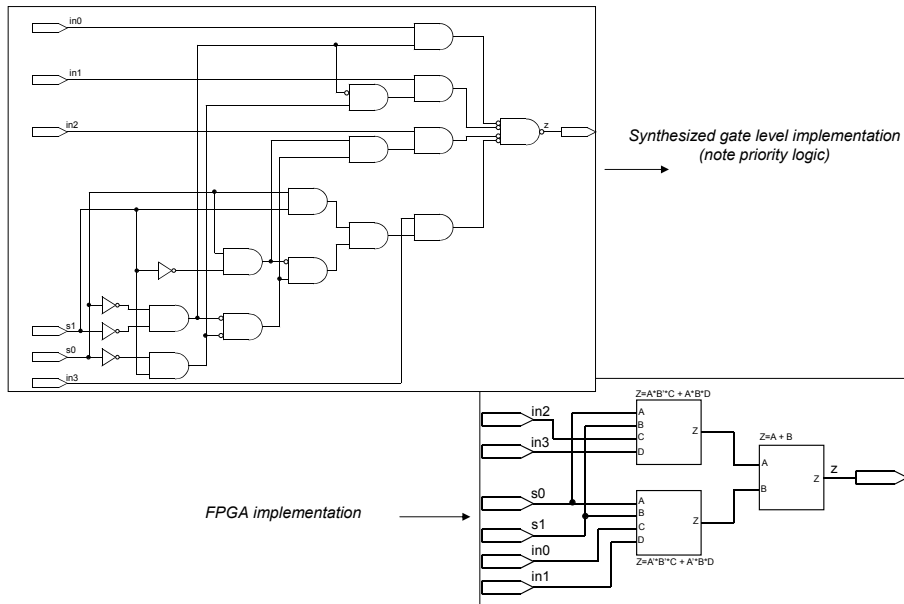
```
architecture behavioral of mux4 is
begin
  z <=
    in0 when s0 = '0' and s1 = '0' else
    in1 when s0 = '0' and s1 = '1' else
    in2 when s0 = '1' and s1 = '0' else
    in3 when s0 = '1' and s1 = '1' else
    '0';
end architecture behavioral;
```



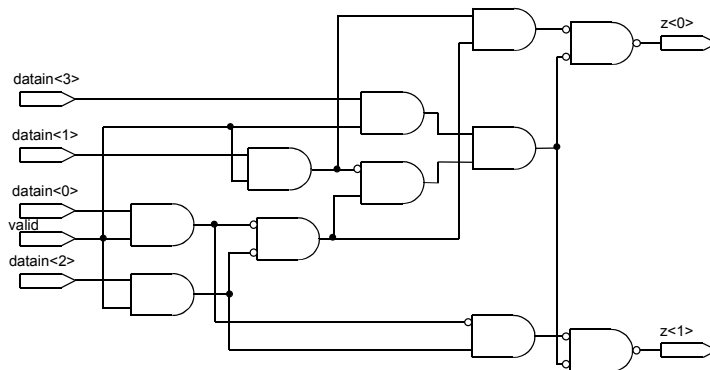
- Any change on input signals causes re-computation of the output signal → combinational logic
- Implied priority order!
 - Priority logic may be optimized for mutually exclusive branches
- Six variable boolean equation and 4 input LUTs
 - Effectively produces a gate level implementation of a multiplexor

ECE 4170 (10)

Inference from Conditional CSAs (cont.)



Example: Priority Encoder



architecture behavior of priority is
begin

$z \leq "00"$ 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!
- Equality tests for other than 0/1 return false
- Note: consider these implementations when writing synthesizable VHDL code

ECE 4170 (13)

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

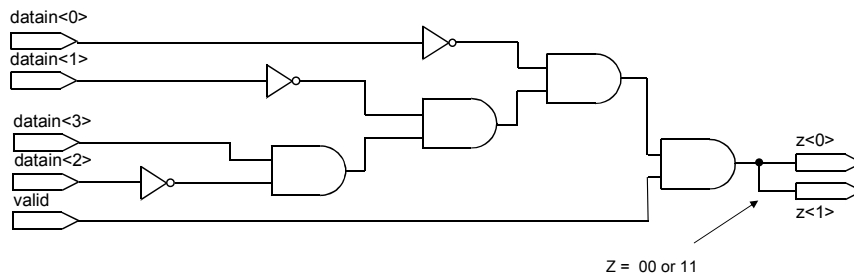
"01" when datain = "--10" and valid = '1' else

"11" when datain = "1000" and valid = '1' else

“00”;

- These tests (with don't care) always return false

end architecture behavior;



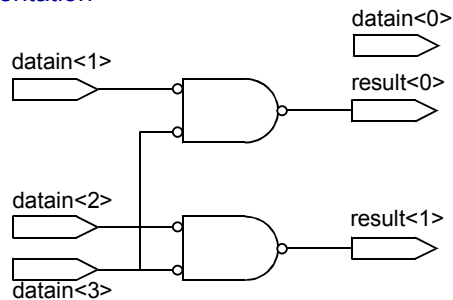
$Z = 00$ or 11

ECE 4170 (14)

Inference from Selected CSAs

- All choices are evaluated and only one must be true
- No priority logic implied
- Example
 - Note how output bits are set
 - Two gate (LUT) implementation

```
with datain select
result <= "00" when "0001",
         "01" when "0010",
         "10" when "0100",
         "11" when "1000",
         "XX" when others;
end architecture behavioral;
```



ECE 4170 (15)

Inference from Selected CSAs

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

```
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!

ECE 4170 (16)



Synthesis Hints/Issues Using CSAs

- Simulation - synthesis mismatches
 - Delay statements are ignored for synthesis
 - Simulated timing may not match timing of synthesized result
 - Comparison logic
 - Results may differ
- Do not utilize initialization in declarations: use explicit resets
 - Mimic normal hardware design process
- 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

ECE 4170 (17)



Synthesis Hints/Issues Using CSAs (cont.)

- Use parentheses to control circuit depth and therefore speed
- Selected signal assignment vs. conditional signal assignment statements
 - Former causes less logic to be inferred
 - Consider whether order of evaluation matters or not

ECE 4170 (18)

- Declarations can be rich source of information for synthesis
- Language semantics define how one can infer logic for the implementation of each signal assignment statement
 - The inference of priority logic
 - Use of non-synthesizable concepts, e.g., don't care values
- Programming style has a major impact on quality of results
 - Synthesis compilers seek powerful analysis techniques to infer beyond the obvious