Outline

• What we know
  – Combinational Networks
  – Sequential Networks:
    • Basic Building Blocks, Mealy & Moore Machines,
      Max Frequency, Setup & Hold Times, Synchronous Design

• What we do not know
  – Equivalent states and reduction of state tables
  – Hardware Description Languages

---

Review: Mealy Sequential Networks

General model of Mealy Sequential Network

(1) X inputs are changed to a new value
(2) After a delay, the Z outputs and next state appear at the output of CM
(3) The next state is clocked into the state register and the state changes

Review: General Model of Moore Sequential Machine

Outputs depend only on present state!

Intro to VHDL

• Technology trends
  – 1 billion transistor chip running at 20 GHz in 2007

• Need for Hardware Description Languages
  – Systems become more complex
  – Design at the gate and flip-flop level becomes very tedious and time consuming

• HDLs allow
  – Design and debugging at a higher level before conversion to the gate and flip-flop level
  – Tools for synthesis do the conversion

• VHDL, Verilog
• VHDL – VHSIC Hardware Description Language

Intro to VHDL

• Developed originally by DARPA
  – for specifying digital systems

• International IEEE standard (IEEE 1076-1993)
• Hardware Description, Simulation, Synthesis
  – Provides a mechanism for digital design and reusable design documentation
  – Support different description levels
    – Structural (specifying interconnections of the gates),
    – Dataflow (specifying logic equations), and
    – Behavioral (specifying behavior)

• Top-down, Technology Dependent
**VHDL Description of Combinational Networks**

Concurrent Statements:
- \( C := A \) and \( B \) after 5 ns;
- \( E := C \) or \( D \) after 5 ns;

If delay is not specified, "default" delay is assumed:
- \( E := C \) or \( D \);

Order of concurrent statements is not important:
- \( E := C \) or \( D \);
- \( C := A \) and \( B \);

This statement executes repeatedly:
- \( C := C \) after 10 ns;

This statement causes a simulation error:
- \( C := C \);

---

**Entity-Architecture Pair**

**Full Adder Example**

```
entity FullAdder is
  port (X, Y, Cin: in bit;
        Cout, Sum: out bit);
end FullAdder;

architecture Equations of FullAdder is
begin
  Sum <= X xor Y xor Cin after 10 ns;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
```

---

**Entity Architecture**

```
entity entity-name is
  [port(interface-signal-declaration)];
end entity;

architecture architecture-name of entity-name is
  [declarations]
begin
  architecture body
  end architecture [architecture-name];
```

---

**VHDL Program Structure**

```
entity entity-name is
  [port(interface-signal-declaration)];
end entity;

architecture architecture-name of entity-name is
  [declarations]
begin
  architecture body
  end architecture [architecture-name];
```

---

**4-bit Adder**

```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Cin: in bit;
        Sout: out bit_vector(3 downto 0); Co: out bit);
end Adder4;

architecture Structure of Adder4 is
  component FullAdder
    port (x, y, Cin: in bit;
          Cout, Sum: out bit);
    begin
      -- Inputs
      Cout <= Sum after 10 ns;
    end FullAdder;
  end component;
  signal C: bit_vector(3 downto 0);
begin
  -- instantiate four copies of the FullAdder
  FA0: FullAdder port map (A(0), B(0), Cin, S(0));
  FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
  FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
  FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

---

**4-bit Adder - Simulation**

```
list A B C S C1 S2 S3 S4:
  -- put these signals on the output list
  force A 1111: set the A inputs to 1111
  force B 0001: set the B inputs to 0001
  force Cin 1:
  run 50:
  -- run the simulation for 50 ns
  force C1 0:
  force A 001:
  force B 1110:
  run 50;
```

---

**4-bit Adder (cont’d)**
Modeling Flip-Flops Using VHDL Processes

General form of process

```
process(sensitivity-list)
begin
sequential-statements
end process;
```

- Whenever one of the signals in the sensitivity list changes, the sequential statements are executed in sequence one time.

Concurrent Statements vs. Process

A, B, C, D are integers
A = 1, B = 2, C = 3, D = 0
D changes to 4 at time 10

<table>
<thead>
<tr>
<th>time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Simulation Results

D Flip-flop Model

Bit values are enclosed in single quotes.

```
entity DIFF is
port (D, CLK, Q): in bit;
-- initialize Q to '0' since bit signals are initialized to '0' by default.
end DIFF;
architecture DFD of DIFF is
begin
-- process is executed when CLK changes
if CLK = '0' then
-- rising edge of clock
Q <= '1' after 10 ns;
end if;
end process;
end DFD;
```

JK Flip-Flop Model

```
entity JFF is
port (D, R, N, J, K, CLK, Q): in bit;
-- same as above, see Note 1
end JFF;
architecture JFF of JFF is
begin
-- process is executed when CLK changes
if CLK = '0' then
-- rising edge of clock
if R = '0' and N = '1' after 10 ns then
Q <= '0' after 10 ns;
else if K = '1' and N = '0' then
Q <= '1' after 10 ns;
else
Q <= Q after 10 ns;
end if;
end if;
end process;
end JFF;
```

Using Nested IFs and ELSEIFs

```
if (C1) then S1; S2;
else if (C2) then S3; S4;
else if (C3) then S5; S6;
else S7; S8;
end if;
```

```
VHDL Models for a MUX

Sel represents the integer equivalent of a 2-bit binary number with bits A and B.

If a MUX model is used inside a process, the MUX can be modeled using a CASE statement (cannot use a concurrent statement):

case Sel is
when 0 => F <= A(0);
when 1 => F <= A(1);
when 2 => F <= A(2);
when others => F <= A(3);
end case;

Compilation and Simulation of VHDL Code

- Compiler (Analyzer) – checks the VHDL source code
  - does it conform to VHDL syntax and semantic rules
  - are references to libraries correct
- Intermediate form used by a simulator or by a synthesizer
- Elaboration
  - create ports, allocate memory storage, create interconnections, ...
  - establish mechanism for executing of VHDL processes
Timing Model

- VHDL uses the following simulation cycle to model the stimulus and response nature of digital hardware:

  - Start Simulation
  - Delay
  - Update Signals
  - Execute Processes
  - End Simulation

Delay Types

- All VHDL signal assignment statements prescribe an amount of time that must transpire before the signal assumes its new value.
- This prescribed delay can be in one of three forms:
  - **Transport** — prescribes propagation delay only
  - **Inertial** — prescribes propagation delay and minimum input pulse width
  - **Delta** — the default if no delay time is explicitly specified

Delay Types (cont.)

- Example of gate with ‘inertia’ smaller than propagation delay:
  - E.g. Inverter with propagation delay of 10ns which suppresses pulses shorter than 5ns

Transport Delay

- Transport delay must be explicitly specified:
  - I.e. keyword “TRANSPORT” must be used
- Signal will assume its new value after specified delay

- TRANSPORT delay example

  ```vhdl
  Input <= 0; Output <= TRANSPORT NOT Input AFTER 10 ns;
  ```

Inertial Delay

- Provides for specification propagation delay and input pulse width, i.e. ‘inertia’ of output:

  ```vhdl
  Output <= 0; Target <= REJECT time_expression INERTIAL waveform;
  ```

- Inertial delay is default and REJECT is optional:

Inertial Delay (cont.)

- Example of gate with ‘inertia’ smaller than propagation delay:
  - E.g. Inverter with propagation delay of 10ns which suppresses pulses shorter than 5ns

Delta Delay

- Default signal assignment propagation delay if no delay is explicitly prescribed:
  - VHDL signal assignments do not take place immediately
  - Delta is an infinitesimal VHDL time unit so that all signal assignments can result in signals assuming their values at a future time

- E.g.:

  ```vhdl
  Output <= NOT Input;
  ```

- Supports a model of concurrent VHDL process execution:
  - Order in which processes are executed by simulator does not affect simulation output

- Note: the REJECT feature is new to VHDL 1076-1993
### Problem #1

- Using the labels, list the order in which the following signal assignments are evaluated if \( \text{in2} \) changes from a '0' to a '1'. Assume \( \text{in1} \) has been a '1' and \( \text{in2} \) has been a '0' for a long time, and then at time \( t \) \( \text{in2} \) changes from a '0' to a '1'.

```vhdl
entity not_another_prob is
    port (in1, in2: in bit; a: out bit);
end not_another_prob;
architecture oh_behave of not_another_prob is
    signal b, c, d, e, f: bit;
begin
    L1: d <= not(in1);
    L2: c<= not(in2);
    L3: f <= (d and in2);
    L4: e <= (c and in1);
    L5: a <= not b;
    L6: b <= e or f;
end oh_behave;
```

### Problem #2

- Under what conditions do the two assignments below result in the same behavior? Different behavior? Draw waveforms to support your answers.

```vhdl
out <= reject 5 ns inertial (not a) after 20 ns;
out <= transport (not a) after 20 ns;
```

### Modeling a Sequential Machine

#### Mealy Machine for 8421 BCD to 8421 BCD + 3 bit serial converter

How to model this in VHDL?

### Behavioral VHDL Model

Two processes:
- the first represents the combinational network;
- the second represents the state register

### Simulation of the VHDL Model

Simulation command file:
```
wave CLK X State NextState Z
force CLK 0.0, 1.00, 0.0
force X 0.0, 1.350, 0.535, 1.750, 0.950, 1.1350
run 1600
```

Waves:
Dataflow VHDL Model

— The following is a description of the sequential machine of
— Figure 1.7 in terms of its next-state equations.
— The following state assignment was used:
— S0 -> Q0, S1 -> Q1, S2 -> Q2, S3 -> Q3, S4 -> Q4

entity SPI_2 is
  port (H: in bit;  
        Q: out bit;  
        Y: out bit;  
        Z: in bit);  
end SPI_2;

architecture Equation1 of SPI_2 is
  signal Q1: bit_vector(2 downto 0);  
  signal (Q1, Q2, Q3, Q4): bit;
begin
  process(Q)
  begin
    if (Q<1) then  
      Q1 <= 0;
    elsif (Q<2) then  
      Q1 <= 1;
    end if;
  end process;
end Equation1;

Simulation of the Structural Model

Simulation command file:

```vhdl
wave CLK X Q1 Q2 Q3 Z  
force CLK 0 0 1 100 >>= invariant 200  
force X 0 0 1 350 0 550 1 750 0 951 1 1350  
run 1600
```

Waveforms:

Forms of Wait Statements

- Wait on
  - until one of the signals in the sensitivity list changes
- Wait for
  - until the time specified by the time expression has elapsed
  - What is this: wait for 0 ns;

- Wait until
  - the boolean expression is evaluated whenever one of the signals in the expression changes, and the process continues execution when the expression evaluates to TRUE

Wait Statements

- ... an alternative to a sensitivity list
  - Note: a process cannot have both wait statement(s) and a sensitivity list

Generic form of a process with wait statement(s)

```vhdl
begin
  sequential-statements
  wait statement
  sequential-statements
  wait-statement
  ...
end process;
```

How wait statements work?

- Execute seq. statement until
  - a wait statement is encountered.
- Wait until the specified condition is satisfied.
- Then execute the next set of sequential statements until
  - the next wait statement is encountered.
- When the end of the process is reached start over again at the beginning.

Using Wait Statements (1)
Using Wait Statements (2)

To Do

- Read
  - Textbook chapters 2.1, 2.2