Outline

- **What we know**
  - Combinational Networks
    - Analysis, Synthesis, Simplification, Hazards, Building Blocks, PALs, PLAs, ROMs
  - Sequential Networks: Basic Building Blocks
  - Design: Mealy
    - Setup and hold times, Max clock frequency

- **What we do not know**
  - Design: Moore
  - Equivalent States
  - State Table Reduction
  - Intro to VHDL

**Review: Mealy Sequential Networks**

General model of Mealy Sequential Network

1. Inputs (X) 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: 8421 BCD to Excess3 BCD Code Converter**

<table>
<thead>
<tr>
<th>X</th>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
<th>1000</th>
<th>1001</th>
<th>1010</th>
<th>1011</th>
<th>1100</th>
<th>1101</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
</tr>
<tr>
<td>Q</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
</tr>
</tbody>
</table>

**Sequential Network Timing (cont’d)**

Timing diagram assuming a propagation delay of 10 ns for each flip-flop and gate
(State has been replaced with the state of three flip-flops)

**Setup and Hold Times**

- For a real D-FF
  - D input must be stable for a certain amount of time before the active edge of clock cycle \(\Rightarrow\) Setup time
  - D input must be stable for a certain amount of time after the active edge of the clock \(\Rightarrow\) Hold time

- Propagation time: from the time the clock changes to the time the output changes

Manufacturers provide minimum \(t_{SU}\), \(t_{TH}\), and maximum \(t_{PLH}\), \(t_{PHL}\)
### Maximum Clock Frequency

- $t_{c_{\text{max}}}$: Max propagation delay through the combinational network
- $t_{p_{\text{max}}}$: Max propagation delay from the time the clock changes to the flip-flop output changes ($= \max\{t_{\text{plh}}, t_{\text{phl}}\}$)
- $t_{\text{ck}}$: Clock period

\[
\begin{align*}
t_{\text{ck}} & \geq t_{c_{\text{max}}} + t_{p_{\text{max}}} + t_{\text{su}} \\
t_{\text{ck}} & \geq \frac{1}{f_{\text{max}}} \\
\end{align*}
\]

Example:

\[f_{\text{max}} = \frac{1}{50 \text{ ns}} = 20 \text{ MHz}\]

### Hold Time Violation

- Occur if the change in $Q$ fed back through the combinational network and cause $D$ to change too soon after the clock edge

\[
h_{\text{min}} + t_{\text{min}} \geq t_{\text{cy}}
\]

Hold time is satisfied if:

\[
t_{\text{cy}} \geq t_{\text{ck}_{\text{max}}} + t_{\text{su}}
\]

Make sure that input changes propagate to the flip-flops inputs such that setup time is satisfied.

\[
t_{\text{cy}} \geq t_{\text{ck}_{\text{min}}} - t_{\text{su}}
\]

Make sure that $X$ does not change too soon after the clock.

### Moore Sequential Networks

Outputs depend only on present state!

- $X = x_1 x_2 ... x_n$
- $Q = Q_1 Q_2 ... Q_k$
- $Z = z_1 z_2 ... z_m$

\[
\begin{align*}
Z(t) & = F(Q(t)) \\
Q(t^+) & = G(X(t), Q(t))
\end{align*}
\]

### General Model of Moore Sequential Machine

Outputs depend only on present state!

- $X = x_1 x_2 ... x_n$
- $Q = Q_1 Q_2 ... Q_k$
- $Z = z_1 z_2 ... z_m$

\[
\begin{align*}
X(t) & = F(Q(t)) \\
Q(t^+) & = G(X(t), Q(t)) \\
Z(t) & = F(Q(t))
\end{align*}
\]

### Code Converter: Moore Machine

Do we need state So? How many states does Moore machine have? How many states does Mealy machine have?
Moore Machine: State Table

<table>
<thead>
<tr>
<th>S0</th>
<th>N5</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>X0</td>
<td>X1</td>
</tr>
<tr>
<td>S1</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>S4</td>
</tr>
<tr>
<td>S3</td>
<td>S5</td>
<td>S6</td>
</tr>
<tr>
<td>S4</td>
<td>S7</td>
<td>S8</td>
</tr>
<tr>
<td>S5</td>
<td>S5</td>
<td>S6</td>
</tr>
<tr>
<td>S6</td>
<td>S9</td>
<td>S10</td>
</tr>
<tr>
<td>S7</td>
<td>S7</td>
<td>S8</td>
</tr>
<tr>
<td>S8</td>
<td>S8</td>
<td>S9</td>
</tr>
<tr>
<td>S9</td>
<td>S9</td>
<td>S10</td>
</tr>
<tr>
<td>S10</td>
<td>S10</td>
<td>S11</td>
</tr>
</tbody>
</table>

Note: state S0 could be eliminated (S0 == S9), if S9 was start state!

Moore Machine Timing

- X = 0010_1001 → Z = 1110_0011

State Assignments

- Guidelines to reduce the amount of combinational logic
- Rule I: (S0, S9, S10), (S4, S5), (S6, S7)
- Rule II: (S1, S2), (S3, S4), (S4, S5), (S6, S7), (S7, S8), (S9, S10)
- Rule III: (S0, S2, S4, S6, S8, S9)
  (S1, S3, S5, S7, S10)

Moore Machine: Another Example

- Coding schemes for serial data transmission
  - NRZ: nonreturn-to-zero
  - NRZI: nonreturn-to-zero-inverted
  - RZ: return-to-zero
- Manchester

Moore Network for NRZ-to-Manchester
**Synchronous Design**

- Use a clock to synchronize the operation of all flip-flops, registers, and counters in the system.
  - All changes occur immediately following the active clock edge.
  - Clock period must be long enough so that all changes to flip-flops, registers, counters will have time to stabilize before the next active clock edge.

- Typical design: Control section + Data Section
  - Control Inputs
  - Control Section
  - Condition Signals
  - Data In
  - Data registers
  - Arithmetic Units
  - Counters
  - Buses, Muxes, ...

**An Example**

- Data section \( s = n(T_n + a) \)
  - \( R_1 = n, R_2 = a \) \( \Rightarrow R_1 = s \)

- Design flowchart for SMUL operation

**Principles of Synchronous Design**

- Method
  - All clock inputs to flip-flops, registers, counters, etc., are driven directly from the system clock or from the clock ANDed with a control signal.

- Result
  - All state changes occur immediately following the active edge of the clock signal.

- Advantage
  - All switching transients, switching noise, etc., occur between the clock pulses and have no effect on system performance.

**Asynchronous Design**

- Disadvantage - More difficult
  - Problems
    - Race conditions: final state depends on the order in which variables change.
    - Hazards
  - Special design techniques are needed to cope with races and hazards.

- Advantages = Disadvantages of Synchronous Design
  - In high-speed synchronous design, propagation delay in wiring is significant => clock signal must be carefully routed so that it reaches all devices at essentially same time.
  - Inputs are not synchronous with the clock => need for synchronizers.
  - Clock cycle is determined by the worst-case delay.
To Do

- Read
  - Textbook chapters 1.6, 1.7, 1.8, 1.10, 1.11, 1.12

Equivalent States

State Equivalence Theorem

- Two states are equivalent $S_i = S_j$ if and only if for every single input $X$, the outputs are the same and the next states are equivalent.

State Table Reduction

1) States $a$ and $h$ have the same next states and outputs ($X=0$ and $X=1$).
2) Eliminate $h$ from the table and replace with $a$.
3) States $a$ and $b$ have the same output $\Rightarrow$ they are same if $c=d$ and $e=f$. We say $c-d$ and $e-f$ are implied pairs for $a-b$.

To keep track of the implied pairs we make an implication chart.

- Make another pass through the chart. $E-g$ cell contains $c-e$ and $b-g$; since $c-e$ cell contains $x$, $c\neq e \Rightarrow e\neq g$ put $X$.
- Repeat the step 4 until no additional squares are $X$-ed. (Put $X$ in $f-t$, $a-c$, $a-d$, $b-c$, $b-d$ squares).
- The remaining squares indicate equivalent state pairs $\Rightarrow a=e$, $b=d$, $a=f$.

- Make another pass through the chart. $E-g$ cell contains $c-e$ and $b-g$; since $c-e$ cell contains $x$, $c\neq e \Rightarrow e\neq g$ put $X$.
- Repeat the step 4 until no additional squares are $X$-ed. (Put $X$ in $f-t$, $a-c$, $a-d$, $b-c$, $b-d$ squares).
- The remaining squares indicate equivalent state pairs $\Rightarrow a=e$, $b=d$, $a=f$.
Implication Table Method

1. Construct a chart that contains a square for each pair of states.
2. Compare each pair in the state table. If the outputs associated with states i and j are different, place an X in square i-j to indicate that i≠j.
   - If outputs are the same, place the implied pairs in square i-j.
   - If outputs and next states are the same (or i-j implies only itself), i=j.
3. Go through the implication table square by square.
   - If square i-j contains the implied pair m-n, and square m-n contains X, then i=j, and place X in square i-j.
4. If any Xs were added in step 3, repeat step 3 until no more Xs are added.
5. For each square i-j that does not contain an X, i=j.

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

VHDL Description of Combinational Networks

- 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

Entity-Architecture Pair

Full Adder Example

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

VHDL Program Structure

```
entity entity-name is
(port(interface-signal-declaration);)
end entity [entity-name];
architecture architecture-name of entity-name is
(declarations);
begin
architecture body
end [architecture] [architecture-name];
```
### 4-bit Adder

#### 4-bit Adder - Simulation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>( C_{in} )</th>
<th>( S )</th>
<th>( C_{out} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<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>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Simulation Results:

- **Process (5, 2, 2)**
  - \( A = 5; \) --- statement 1
  - \( B = 2; \) --- statement 2
  - \( C = 2; \) --- statement 3

- **Process (3, 2, 1)**
  - \( A = 3; \) --- statement 1
  - \( B = 2; \) --- statement 2
  - \( C = 1; \) --- statement 3

### 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.
JK Flip-Flop Model

entity JKFF is
  port (SN, RN, J, K, CA in bit; Q, Hold, Q', out bit = ’1’); -- see Note 1.
end JKFF;

architecture JKFF of JKFF is
begin
  process (SN, RN, CL) -- see Note 2.
  begin
    if (SN = ’0’ then Q <= ’1’ after 10 ns;
      elsif (SN = ’1’ then Q <= ’0’ after 10 ns;
    end if;
    if CA = ’1’ and CL = ’0’ then Q <= Q;
    else if (J = ’1’ and K = ’0’ then Q <= Q;
    else if (J = ’0’ and K = ’1’ then Q <= Q;
    else if (J = ’1’ and K = ’1’ then Q <= Q;
    end if;
  end process;
end JKFF;

Note 1: Q is declared as input instead of output because it appears on both the left and right sides of an assignment with the architecture.
Note 2: The flip-flop can change state in response to changes in SN, RN, and CA, so these 3 signals are at the same timing level.
Note 3: The condition (SN = ’0’ and CA = ’1’) means if SN has just changed from ’1’ to ’0’. 
Note 4: Characteristic equation which describes behavior of JK Flip-Flop.

Using Nested IFs and ELSEIFs

if (C1) then S1; S2;
else if (C2) then S3; S4;
else if (C3) then S5; S6;
else if (C4) then S7; S8;
end if;

VHDL Models for a MUX

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity SELECTOR is
  port (A : in std_logic_vector(15 downto 0);
        SEL : in std_logic_vector( 3 downto 0);
        Y   : out std_logic);
end SELECTOR;

architecture RTL1 of SELECTOR is
begin
  p0 : process (A, SEL)
  begin
    if    (SEL = "0000") then       Y <= A(0);
    elsif (SEL = "0001") then      Y <= A(1);
    elsif (SEL = "0010") then      Y <= A(2);
    elsif (SEL = "0011") then      Y <= A(3);
    elsif (SEL = "0100") then      Y <= A(4);
    elsif (SEL = "0101") then      Y <= A(5);
    elsif (SEL = "0110") then      Y <= A(6);
    elsif (SEL = "0111") then      Y <= A(7);
    elsif (SEL = "1000") then      Y <= A(8);
    elsif (SEL = "1001") then      Y <= A(9);
    elsif (SEL = "1010") then      Y <= A(10);
    elsif (SEL = "1011") then      Y <= A(11);
    elsif (SEL = "1100") then      Y <= A(12);
    elsif (SEL = "1101") then      Y <= A(13);
    elsif (SEL = "1110") then      Y <= A(14);
    else      Y <= A(15);
  end if;
end process;
end RTL1;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity SELECTOR is
  port (A : in std_logic_vector(15 downto 0);
        SEL : in std_logic_vector( 3 downto 0);
        Y   : out std_logic);
end SELECTOR;

architecture RTL3 of SELECTOR is
begin
  with SEL select
  Y <= A(0) when "0000",
       A(1) when "0001",
       A(2) when "0010",
       A(3) when "0011",
       A(4) when "0100",
       A(5) when "0101",
       A(6) when "0110",
       A(7) when "0111",
       A(8) when "1000",
       A(9) when "1001",
       A(10) when "1010",
       A(11) when "1011",
       A(12) when "1100",
       A(13) when "1101",
       A(14) when "1110",
       A(15) when others;
end RTL3;

MUX Models (1)

MUX Models (2)
MUX Models (3)

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity SELECTOR is
  port(
    A : in std_logic_vector(15 downto 0);
    SEL : in std_logic_vector(3 downto 0);
    Y : out std_logic);
end SELECTOR;

architecture RTL2 of SELECTOR is
begin
  p1 : process (A, SEL)
  begin
    case SEL is
      when "0000" => Y <= A(0);
      when "0001" => Y <= A(1);
      when "0010" => Y <= A(2);
      when "0011" => Y <= A(3);
      when "0100" => Y <= A(4);
      when "0101" => Y <= A(5);
      when "0110" => Y <= A(6);
      when "0111" => Y <= A(7);
      when "1000" => Y <= A(8);
      when "1001" => Y <= A(9);
      when "1010" => Y <= A(10);
      when "1011" => Y <= A(11);
      when "1100" => Y <= A(12);
      when "1101" => Y <= A(13);
      when "1110" => Y <= A(14);
      when others => Y <= A(15);
    end case;
  end process;
end RTL2;

MUX Models (4)

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity SELECTOR is
  port(
    A : in std_logic_vector(15 downto 0);
    SEL : in std_logic_vector(3 downto 0);
    Y : out std_logic);
end SELECTOR;

architecture RTL4 of SELECTOR is
begin
  Y <= A(conv_integer(SEL));
end RTL4;