Overview of Sequential Circuits

- Storage Elements
  - Sequential circuits
  - Storage elements: Latches & Flip-flops
  - Registers and counters

- Circuit and System Timing

- Sequential circuit analysis and description: State tables & State diagrams
Counters

- Counter: a special type of register that incorporates an incrementer or decrementer, which allows it to count upward or downward.

- We shall examine VHDL code for the following counters:
  - Up counter
  - Down counter

VHDL for Up-Counter

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY upcount IS
GENERIC ( N : INTEGER := 4 );
PORT ( Clock, Resetn, Enable : IN STD_LOGIC;
Q: BUFFER STD_LOGIC_VECTOR (N-1 DOWNTO 0));
END upcount;

ARCHITECTURE cnt OF upcount IS
SIGNAL count : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
BEGIN
PROCESS (Resetn, Clock)
BEGIN
IF Resetn = '0' THEN
  count <= ( OTHERS => '0' );
ELSIF (Clock'EVENT AND Clock = '1') THEN
  IF Enable = '1' THEN
    count <= count +1;
  ELSE
    count <= count;
  END IF;
END IF;
END PROCESS;
Q <= count;
END cnt;
```
### Timing Diagram for Up-Counter

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0.5ns</th>
<th>1 ns</th>
<th>1.5 ns</th>
<th>2 ns</th>
<th>2.5 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clock</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Reset</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Enable</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Internal Logic
- Incrementer: Q+0 or Q+1
- Contraction of a ripple carry adder with one operand fixed at 000X

#### Symbol for Synchronous Counter

### Synchronous Counters

- **Internal Logic**
  - Incrementer: Q+0 or Q+1
- **Contraction of a ripple carry adder with one operand fixed at 000X**
- **Symbol for Synchronous Counter**
Synchronous Counters (Contd.)

- Contraction of carry-lookahead adder
  - Reduce path delays
  - Called *parallel gating*
  - Lookahead can be used on COs and ENs to prevent long paths in large counters

Counter with Parallel Load

<table>
<thead>
<tr>
<th>Load</th>
<th>Count</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Hold Stored Value</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Count Up Stored Value</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>Load D</td>
</tr>
</tbody>
</table>

- Add path for input data D
  - enabled for Load = 1

- Add logic to:
  - When Load = 1 disable count logic (feedback from outputs)
  - When Load = 0 and Count = 1 enable count logic
BCD Counter

architecture Behavioral of bcd_counter is
begin
  signal regcnt : std_logic_vector(3 downto 0);
  count: process(reset,clk) is
  begin
    if ( reset='1' ) then
      regcnt <= "0000";
    elsif ( clk'event and clk='1') then
      regcnt <= regcnt+1;
      if (regcnt = "1001") then
        regcnt <= "0000";
      end if;
    end if;
  end process;
end Behavioral;

Overview of Sequential Circuits

- Storage Elements
  - Sequential circuits
  - Storage elements: Latches & Flip-flops
  - Registers and counters

- Circuit and System Timing

- Sequential circuit analysis and description: State tables & State diagrams
Flip-Flop Timing Parameters

- $t_w$ - clock pulse width
- $t_s$ - setup time: Equal to a time interval that is generally much less than the width of the triggering pulse
- $t_h$ - hold time - Often equal to zero
- $t_{px}$ - propagation delay
  - Same parameters as for gates except
  - Measured from clock edge that triggers the output change to the stabilization of the output to a new value

![Flip-Flop Timing Parameters Diagram](image)

Circuit and System Level Timing

- Consider a system comprised of ranks of flip-flops connected by logic
- If the clock period is too short, some data changes will not propagate through the circuit to next flip-flop inputs before the setup time interval begins

![Circuit and System Level Timing Diagram](image)
### Circuit and System Level Timing (Contd.)

- **Timing components along a path from flip-flop to flip-flop**

![Timing diagram](image)

- **Timing Equations**
  \[ t_p = t_{\text{slack}} + (t_{pd,FF} + t_{pd,COMB} + t_s) \]
  \[ t_p \geq \text{max} (t_{pd,FF} + t_{pd,COMB} + t_s) \]
  for all paths from flip-flop output to flip-flop input

### Calculation of Allowable \( t_{pd,COMB} \)

- **Compute the allowable \( t_{pd,COMB} \) for a circuit using edge-triggered flip-flops**

- **Parameters**
  - \( t_{pd,FF}(\text{max}) = 1.0 \text{ ns} \)
  - Clock frequency = 250 MHz, \( t_p = 1/\text{clock frequency} = 4.0 \text{ ns} \)
  - \( t_s(\text{max}) = 0.3 \text{ ns} \) for edge-triggered flip-flops
  - for a gate, average \( t_{pd} = 0.3 \text{ ns} \)

- **Calculations:**
  - Edge-triggered: \( 4.0 \geq 1.0 + t_{pd,COMB} + 0.3 \)
  \[ \Rightarrow t_{pd,COMB} \leq 2.7 \text{ ns}, \text{approximately 9 gates allowed on a path} \]
Overview of Sequential Circuits

- **Storage Elements**
  - Sequential circuits
  - Storage elements: Latches & Flip-flops
  - Registers and counters

- **Circuit and System Timing**

- **Sequential circuit analysis and description:**
  State tables & State diagrams

---

### Example 1

- **Input:** \( x(t) \)
- **Output:** \( y(t) \)
- **State:** \( (A(t), B(t)) \)

**Output Function?**
- \( y(t) = x(t)(B(t) + A(t)) \)

**Next State Function?**
- \( A(t+1) = A(t)x(t) + B(t)x(t) \)
- \( B(t+1) = \overline{A(t)}x(t) \)
Example 1 (Contd.)

- Where in time are inputs, outputs and states defined?

\[ y(t) = x(t)(B(t) + A(t)) \]
\[ A(t+1) = A(t)x(t) + B(t)x(t) \]
\[ B(t+1) = \overline{A(t)}x(t) \]

State Table Characteristics

- The state table can be filled in using the next state and output equations: \( A(t+1) = A(t)x(t) + B(t)x(t) \), \( B(t+1) = \overline{A(t)}x(t) \)
\[ y(t) = x(t)(B(t) + A(t)) \]

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(t) B(t) x(t)</td>
<td>A(t+1) B(t+1) y(t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
</tbody>
</table>
State Diagram

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(t) B(t)</td>
<td>x(t)</td>
<td>A(t+1) B(t+1)</td>
<td>y(t)</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0</td>
<td>1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
</tbody>
</table>

Design Procedure for Sequential Circuits

- **Formulation**: Obtain a state diagram or state table
- **State Assignment**: Assign binary codes to the states
- **Flip-Flop Input Equation Determination**: Select flip-flop types, derive flip-flop input equations from next state entries in the table
- **Output Equation Determination**: Derive output equations from output entries in the table
- **Optimization**: Optimize the equations
- **Technology Mapping**: Find circuit from equations and map to flip-flops and gate technology
- **Verification**: Verify correctness of final design
Formulation: Finding a State Diagram

- State: an abstraction of the history of past-applied inputs to the circuit (including power-up or system reset)
  - Asynchronous reset normally resets to the initial state
- In specifying a circuit, states remember meaningful properties of past input sequences that are essential to predicting future output values, e.g.:
  - State A represents the fact that a 1 input has occurred among the past inputs.
  - State B represents the fact that a 0 followed by a 1 have occurred as the most recent past two inputs.

Example: Sequence Recognizer Procedure

- A sequence recognizer: produces an output value ‘1’ whenever a prescribed pattern of inputs occur in sequence
- Steps:
  - Begin in an initial state (typically “reset” state), when NONE of the initial portion of the sequence has occurred
  - Add states
    - that recognize each successive symbol occurring
    - the final state represents the input sequence occurrence.
  - Add state transition arcs which specify what happens when a symbol not in the proper sequence has occurred.
  - Add other arcs on non-sequence inputs which transition to states.
    - The last step is required because the circuit must recognize the input sequence regardless of where it occurs within the overall sequence applied since “reset.”.
Example: Recognize 1101

- Define states for the sequence to be recognized
- Starting in the initial state ("A"):

  \[ \begin{array}{c}
  \text{A} \\
  \text{B} \\
  \text{C} \\
  \text{D}
  \end{array} \]

  \( \begin{array}{c}
  1/0 \\
  1/0 \\
  0/0 \\
  1/1
  \end{array} \)

- Finally, output 1 on the arc from D means the sequence has been recognized,
  - To what state should the arc from state D go? Remember: 1101101?
  - The final 1 in the recognized sequence 1101 is a sub-sequence of 1101. It follows a 0 which is not a sub-sequence of 1101. Thus it should represent the same state reached from the initial state after a first 1 is observed.

Example: Recognize 1101 (Contd.)

- The other arcs are added to each state for inputs not yet listed. Which arcs are missing?
  - State transition arcs must represent the fact that an input subsequence has occurred.
  - Note that the 1 arc from state C back to C implies that State C means two or more 1's have occurred.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State x=0</th>
<th>Next State x=1</th>
<th>Output x=0</th>
<th>Output x=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>B</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Moore Model

- For Moore Model, outputs are associated with states. Arcs now show only state transitions.
- Add a new state E to produce the output 1:
  - State E produces the same behavior in the future as state B, but it gives a different output at the present time. Thus these states do represent a different abstraction of the input history.
- The Moore model for a sequence recognizer usually has more states than the Mealy model.

Example: Moore Model (Contd.)

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State x=0</th>
<th>Next State x=1</th>
<th>Output y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
VHDL Example: Sequential Recognizer

- VHDL for the sequential recognizer follows

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity seq_rec is
port (CLK, RESET, X: in std_logic;
      Z: out std_logic);
end seq_rec;

architecture process_3 of seq_rec is
  type state_type is (A, B, C, D);
  signal state, next_state: state_type;
begin
```

VHDL Example (Contd.)

```
--process 1: implements positive edge-triggered flipflop with asynchronous reset
state_register: process (CLK, RESET)
begin
if(RESET = '1') then
  state <= A;
elsif (CLK'event and CLK = '1') then
  state <= next_state;
end if;
end process;

--process 2: implement output as function of input X and state
output_function: process (X, state)
begin
  case state is
    when A => Z <= '0';
    when B => Z <= '0';
    when C => Z <= '0';
    when D => if X = '1' then Z <= '1';
              else Z <= '0'; end if;
  end case;
end process;
```
VHDL Example: (Process 3)

-- process 3: next state-function implemented as a function of input X and state
next_state_function: process (X, state)
begin
    case state is
    when A =>
        if X = '1' then
            next_state <= B;
        else
            next_state <= A;
        end if;
    when B =>
        if X = '1' then
            next_state <= C;
        else
            next_state <= A;
        end if;
    when C =>
        if X = '1' then
            next_state <= C;
        else
            next_state <= D;
        end if;
    when D =>
        if X = '1' then
            next_state <= B;
        else
            next_state <= A;
        end if;
    end case;
end process;
end architecture;