ECE 3401 Lecture 13
Sequential Circuits (III)
Datapath & Control Unit (Chpt. 5)

State Assignment
- Each of the \( m \) states must be assigned a unique code
- Minimum number of bits required is \( n \) such that \( n \geq \lceil \log_2 m \rceil \)
  where \( \lceil x \rceil \) is the smallest integer \( \geq x \)
- There are \( 2^n - m \) unused states

State Assignment
- Example

<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>B</td>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

State Assignment 1– Example (Contd.)
- Assignment 1: \( A = 0 \ 0, \ B = 0 \ 1, \ C = 1 \ 0, \ D = 1 \ 1 \)
- The resulting coded state table:

<table>
<thead>
<tr>
<th>Present State ( (Y, Y_2) )</th>
<th>Next State ( (D_1, D_2) ) x = 0 x = 1</th>
<th>Output ( (Z) ) x = 0 x = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00 01</td>
<td>0 0</td>
</tr>
<tr>
<td>01</td>
<td>00 10</td>
<td>0 0</td>
</tr>
<tr>
<td>10</td>
<td>11 10</td>
<td>0 0</td>
</tr>
<tr>
<td>11</td>
<td>00 01</td>
<td>0 1</td>
</tr>
</tbody>
</table>

State Assignment 2 – Example (continued)
- Assignment 2: \( A = 0 \ 0, \ B = 0 \ 1, \ C = 1 \ 1, \ D = 1 \ 0 \)
- The resulting coded state table:

<table>
<thead>
<tr>
<th>Present State ( (Y_1, Y_2) )</th>
<th>Next State ( (D_1, D_2) ) x = 0 x = 1</th>
<th>Output ( (Z) ) x = 0 x = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00 01</td>
<td>0 0</td>
</tr>
<tr>
<td>01</td>
<td>00 11</td>
<td>0 0</td>
</tr>
<tr>
<td>11</td>
<td>10 11</td>
<td>0 0</td>
</tr>
<tr>
<td>10</td>
<td>00 01</td>
<td>0 1</td>
</tr>
</tbody>
</table>
Find Output and Flip-Flop Input Equations: Optimization: Assignment 2

- Assume D flip-flops
  
  $D_1 = Y_1 Y_2 + X Y_2$
  
  $D_2 = X$
  
  $Z = X Y_1 Y_2$
  
- Performing two-level optimization:
  
  Gate Input Cost = 9

Map Technology

- Library:
  
  - D Flip-flops with Reset
  
  - NAND gates with up to 4 inputs and inverters

- Initial Circuit:

  - Clock
  
  - Reset

Microprogramming Overview (Chpt. 5)

- Datapath and control
  
  - Microoperations
  
  - Sequencing and control

Overview

- Datapath and control
  
  - Microoperations
    
    - Register transfer operations
    
    - Microoperations - arithmetic, logic, and shift
    
    - Register cell design
    
    - Serial transfers and microoperations
  
  - Sequencing and control

Datapath and Control

- Datapath - performs data transfer and processing operations
  
  - Control Unit - Determines the enabling and sequencing of the operations

  - The control unit receives:
    
    - External control inputs
    
    - Status signals
  
  - The control unit sends:
    
    - Control signals
    
    - Control outputs

Mapped Circuit - Final Result

- Clock
  
  - Reset

Microprogramming Overview (Chpt. 5)
Register Transfer Operations

- Register Transfer Operations – the movement and processing of data stored in registers
- Three basic components:
  - A set of registers (operands)
  - Transfer operations
  - Control of operations
- Elementary operations – called microoperations
  - load, count, shift, add, bitwise "OR", etc.

Register Notation

- Letters and numbers – register (e.g. R2, PC, IR)
- Parentheses () – range of register bits (e.g. R1(1), PC(7:0), AR(L))
- Arrow (→) – data transfer (ex. R1 → R2, PC(L) → R0)
- Brackets [] – Specifies a memory address (ex. R0 ← M[AR], R3 ← M(PC)
- Comma – separates parallel operations

Conditional Transfer

- If (K1 = 1) then (R2 ← R1) ⊖ K1: (R2 ← R1)
  where K1 is a control expression specifying a conditional execution of the microoperation.

Microoperations

- Logical groupings:
  - Transfer - move data from one set of registers to another
  - Arithmetic - perform arithmetic on data in registers
  - Logic - manipulate data or use bitwise logical operations
  - Shift - shift data in registers

Arithmetic Microoperations

<table>
<thead>
<tr>
<th>Symbolic Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 ← R1 + R2</td>
<td>Addition</td>
</tr>
<tr>
<td>R0 ← R1</td>
<td>Ones Complement</td>
</tr>
<tr>
<td>R0 ← R1 + 1</td>
<td>Two's Complement</td>
</tr>
<tr>
<td>R0 ← R2 + R1 + 1</td>
<td>R2 minus R1 (2's Comp)</td>
</tr>
<tr>
<td>R1 ← R1 + 1</td>
<td>Increment (count up)</td>
</tr>
<tr>
<td>R1 ← R1 - 1</td>
<td>Decrement (count down)</td>
</tr>
</tbody>
</table>

Example Microoperations

- R1 ← R1 + R2
  - Add the content of R1 to the content of R2 and place the result in R1.
- PC ← R1 * R6
- R1 ← R1 + 1
- R1 ← R1 - 1

(K1 + K2): R1 ← R1 ⊖ R3
- On condition K1 OR K2, the content of R1 is Logic bitwise Ored with the content of R3 and the result placed in R1.
- NOTE: "⊕" (as in K1 + K2) means "OR." In R1 ← R1 + R2, + means "plus."
### Logical Microoperations

<table>
<thead>
<tr>
<th>Symbolic Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 ← R1</td>
<td>Bitwise NOT</td>
</tr>
<tr>
<td>R0 ← R1 ∨ R2</td>
<td>Bitwise OR (sets bits)</td>
</tr>
<tr>
<td>R0 ← R1 ∧ R2</td>
<td>Bitwise AND (clears bits)</td>
</tr>
<tr>
<td>R0 ← R1 ⊕ R2</td>
<td>Bitwise EXOR (complements bits)</td>
</tr>
</tbody>
</table>

### Shift Microoperations

- Let R2 = 11001001

<table>
<thead>
<tr>
<th>Symbolic Designation</th>
<th>Description</th>
<th>R1 content</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 ← sl R2</td>
<td>Shift Left</td>
<td>10010010</td>
</tr>
<tr>
<td>R1 ← sr R2</td>
<td>Shift Right</td>
<td>01100100</td>
</tr>
</tbody>
</table>

- Note: These shifts "zero fill". Sometimes a separate flip-flop is used to provide the data shifted in, or to "catch" the data shifted out.
- Other shifts are possible (rotates, arithmetic)