CMSC 313 Lecture 27

• Last Time: FSM simplification

• DigSim2 Questions??
  ◦ Place your username on the paper submission

• Registers

• Memory Organization

• DRAM
DigSim Assignment 2: Finite State Machine Simplifications

Due: Tuesday December 9, 2003

Objective
The objective is to design and simplify a moderately complex a finite state machine.

Assignment
For this project you will design and implement in DigSim a Mealy finite state machine with one input bit x and one output bit z. The machine’s output z is 1 whenever the input sequence …0111 or …1000 has been detected. The patterns may overlap. For example, if the machine is given input 0111000, the output would be 0001001. [Adapted from Contemporary Logic Design, Randy H. Katz, Benjamin/Cummings Publishing, 1994.]

In order to complete the project, you must accomplish the following tasks. Note that for this project, you will submit both your design work (on paper) and the DigSim implementation of the resulting finite state machine (via submit).

1. Draw a transition diagram for the finite state machine described above. Use the state reduction algorithm to minimize the number of states in your machine. Although, you might start with an FSM with as many as 15 states, at the end of the reduction process you should have 7 states in your machine. Name your states A, B, C, D, E, F and G. You may take shortcuts, but you must show your work.

2. Assign bit patterns to each state of the machine. Use the heuristics for state assignment presented in class (available on the course lecture topics web page). Do this step carefully as it will have a large effect on the complexity of your final circuit. Hint: follow the loops in the FSM and try to assign bit patterns such that in each step of the loop only one state bit changes. You might not be able to achieve this for every step of the loop, but you can try to have only one state bit change for most steps of the loop. Write down your final state assignment in the table given.

3. Fill in the state transition table for the finite state machine given below. In this table, s2, s1 and s0 are the state bits, x is the input bit and z is the output bit. Then s2’, s1’ and s0’ are the next states to be stored in the flip flops.

4. Use the Karnaugh maps provided to simplify the Boolean formulas for a finite state machine to be implemented using D flip-flops. You will need Karnaugh maps for s2’, s1’, s0’ and z.

5. Using the excitation table for J-K flip-flops, fill in the columns j2, k2, j1, k1, j0 and k0 in the truth table below. The columns j2 and k2 represent the settings to the inputs of the J-K flip-flop that will cause it to store the state bit s2’. The columns j1, k1, j0 and k0 are analogous for the J-K flip-flops used to store state bits s1 and s0.

6. Use the Karnaugh maps provided to simplify the Boolean formulas for the J and K inputs to each J-K flip flop. You will need Karnaugh maps for j2, k2, j1, k1, j0 and k0. The formula for the output z will be the same as in Step 4.


8. Implement the resulting circuit in DigSim. Hint: it is possible to implement this FSM using fewer than 14 gates. If your circuit requires many more gates, redo the simplification steps.

What to submit
In class on Tuesday December 9, turn in your finite state diagram, truth table, Karnaugh maps and the resulting Boolean formulas on paper. Make copies of these, if you still need them to implement your circuit in DigSim.

Save your circuit as you did in DigSim Assignment 1. Submit the circuit file using the Unix submit command as in previous assignments. The submission name for this assignment is: digsim2.
CMSC 313 Digsim Exercise 2

Name: _________________________________

Minimized 7-State Transition Diagram (show work)

State Assignment:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>unused</th>
</tr>
</thead>
</table>


### Excitation Table for J-K Flip-Flops

<table>
<thead>
<tr>
<th>Q</th>
<th>Q'</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>d</td>
<td>0</td>
</tr>
</tbody>
</table>

### Truth Table:

<table>
<thead>
<tr>
<th>s2</th>
<th>s1</th>
<th>s0</th>
<th>x</th>
<th>s2'</th>
<th>s1'</th>
<th>s0'</th>
<th>z</th>
<th>j2</th>
<th>k2</th>
<th>j1</th>
<th>k1</th>
<th>j0</th>
<th>k0</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Question:
Should you use D flip-flops or J-K flip-flops to implement this circuit? Why?
Karnaugh Maps for D Flip-Flops and the output

\[
s0' = s2 = 10, 00, 01, 11, 10
\]

\[
s1' = s2 = 10, 00, 01, 11, 10
\]

\[
s2' = s0 = 10, 00, 01, 11, 10
\]

\[
z = s0' = s2 = 10, 00, 01, 11, 10
\]
Karnaugh Maps for J-K Flip-Flops

\[ j_3 = \]

\[ j_2 = \]

\[ j_1 = \]

\[ j_0 = \]
Four-Bit Register

- Makes use of tri-state buffers so that multiple registers can gang their outputs to common output lines.
Left-Right Shift Register with Parallel Read and Write

<table>
<thead>
<tr>
<th>Control</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$c_0$</td>
</tr>
<tr>
<td>0 0</td>
<td>No change</td>
</tr>
<tr>
<td>0 1</td>
<td>Shift left</td>
</tr>
<tr>
<td>1 0</td>
<td>Shift right</td>
</tr>
<tr>
<td>1 1</td>
<td>Parallel load</td>
</tr>
</tbody>
</table>
A Serial Multiplier

Multiplicand (M)

\[
\begin{array}{c}
m_3 \ m_2 \ m_1 \ m_0
\end{array}
\]

4-Bit Adder

\[
\begin{array}{cccc}
a_3 & a_2 & a_1 & a_0
\end{array}
\]

Shift and Add Control Logic

\[
\begin{array}{cccc}
q_3 & q_2 & q_1 & q_0
\end{array}
\]

Multiplier (Q)

\[
\begin{array}{c}
\text{A Register}
\end{array}
\]
Example of Multiplication Using Serial Multiplier

Multiplicand (M):

Initial values

<table>
<thead>
<tr>
<th>C</th>
<th>A</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>1 0 1 1</td>
</tr>
</tbody>
</table>

0 1 1 0 1 1 0 1 1 Add M to A
0 0 1 1 0 1 1 0 1 Shift
1 0 0 1 1 1 1 0 1 Add M to A
0 1 0 0 1 1 1 0 0 Shift
0 0 1 0 0 1 1 1 1 Shift (no add)
1 0 0 0 1 1 1 1 1 Add M to A
0 1 0 0 0 1 1 1 1 Shift

Product
Functional Behavior of a RAM Cell

```
D  Q
CLK

Select

Data
In/Out

Read
```
Simplified RAM Chip Pinout

\[ A_0-A_{m-1} \rightarrow \text{Memory Chip} \rightarrow D_0-D_{w-1} \]

\[ \text{WR} \]

\[ \text{CS} \]
A Four-Word Memory with Four Bits per Word in a 2D Organization
A Simplified Representation of the Four-Word by Four-Bit RAM
2-1/2D Organization of a 64-Word by One-Bit RAM

Row Decoder

Column Decoder (MUX/DEMUX)

One Stored Bit

Two bits wide:
One bit for data and one bit for select.
## Decoder

### Truth Tables

<table>
<thead>
<tr>
<th>Enable = 1</th>
<th>A</th>
<th>B</th>
<th>D₀</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enable = 0</th>
<th>A</th>
<th>B</th>
<th>D₀</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Logic Equations

\[
D₀ = \overline{A} \overline{B} \\
D₁ = \overline{A} B \\
D₂ = A \overline{B} \\
D₃ = AB
\]
Two Four-Word by Four-Bit RAMs are Used in Creating a Four-Word by Eight-Bit RAM
Two Four-Word by Four-Bit RAMs Make up an Eight-Word by Four-Bit RAM
Single-In-Line Memory Module

- Adapted from (Texas Instruments, MOS Memory: Commercial and Military Specifications Data Book, Texas Instruments, Literature Response Center, P.O. Box 172228, Denver, Colorado, 1991.)

<table>
<thead>
<tr>
<th>PIN NOMENCLATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0-A9</td>
</tr>
<tr>
<td>CAS</td>
</tr>
<tr>
<td>DQ1-DQ8</td>
</tr>
<tr>
<td>NC</td>
</tr>
<tr>
<td>RAS</td>
</tr>
<tr>
<td>Vcc</td>
</tr>
<tr>
<td>Vss</td>
</tr>
<tr>
<td>W</td>
</tr>
</tbody>
</table>
Types of Random Access Memory

• **Static RAM (SRAM)**
  - Each bit is stored in a type of flip-flop
  - Typically takes four or six transistors per bit
  - Faster, but takes up more space in a chip
  - Retains information as long as power is supplied
  - Not to be confused with flash memory in digital cameras (EEPROMs)

• **Dynamic RAM (DRAM)**
  - Each bit is stored in a capacitor
  - Uses one capacitor and one transistor per bit
  - Slower, but takes up less space in a chip
  - Must be refreshed periodically (milliseconds), since the capacitor leaks
• A DRAM memory cell

Word Line

Capacitor

GND

Bit Line

• Word line selects cell for reading or writing
• To write, the bit line is charged with logic 1 or 0
• To read, sensitive amplifier circuits detect small changes in bit line.
• Reading discharges the capacitor.
DRAM Read Cycle

1. Row address placed on the address bus.
2. Row Address Strobe (RAS) is asserted, allowing the row address to latch.
3. Row address decoder selects proper row.
4. Write Enable (WE) disabled.
5. Column address placed on the address bus.
6. Column Address Strobe (CAS) is activated, allowing the column address to latch.
7. Once the CAS signal has stabilized, sensing amplifiers places data from the selected row & column on data bus.
8. RAS and CAS deactivated. Cycle begins again.
DRAM Read

- **RAS**
  - Precharge
  - RAS Active
  - Precharge
  - RAS Active

- **CAS**
  - Precharge
  - CAS Active
  - Precharge
  - CAS Active

- **Address Bus**
  - Row
  - Column
  - Row
  - Column

- **WE**

- **Data Bus**
  - tRAC
  - tCAC
  - Data
  - tRAC
  - tCAC
  - Data
DRAM

- DRAM is asynchronous, ignores system bus clock.
  - $t_{RAC} = \text{Row Access Time} = \text{delay from RAS assertion until data is ready}$
  - $t_{CAC} = \text{Column Access Time} = \text{delay from CAS assertion until data is ready}$

- DRAM access is sloooow

- Each memory access must wait for time it takes to activate and deactivate RAS.

- Fast Page Mode (FPM) DRAM allows successive reads from the same row without deactivating RAS.

- Extended Data Out (EDO) DRAM overlaps CAS assertion and data reads.
Fast Page Mode Read

RAS: RAS Active

CAS: Precharge, CAS Active, Precharge

Address Bus: Row, Col. 1, Col. 2, Col. 3, Col. 4

WE

Data Bus: tRAC, Data 1, tCAC, D2, tCAC, D3, tCAC, D4
Synchronous DRAM (SDRAM)

• Uses system bus clock.

• Current models run at 433MHz (still much slower than CPU).

• Burst mode allows fast successive reads from the same row. (Good way to read in a cache line!)

• Double Data Rate (DDR) SDRAM provides data on the positive and negative edges of the clock.
SDRAM Read

Clock:

1 2 3 4 5 6 7 8 9 10

RAS:

CAS:

Address Bus:
Row Col.

BA0:

WE:

Data Bus:
Data Data Data Data Data Data Data

DQM:
Next Time

• Review of semester
• Final exam review