Unit 9
Implementing Combinational Functions with Karnaugh Maps or Memories

Outcomes

• I can use Karnaugh maps to synthesize combinational functions with several outputs
• I can determine the appropriate size and contents of a memory to implement any logic function (i.e. truth table)

KARNAUGH MAPS
A new way to synthesize your logic functions

Logic Function Synthesis

• Given a function description as a T.T. or canonical form, how can we arrive at a circuit implementation or equation (i.e. perform logic synthesis)?
• First method
  – Minterms / maxterms
    • Can simplify to find minimal 2-level implementation
    • Use a decoder + 1 gate per output
• New, second method
  – Karnaugh Maps
    • Minimal 2-level implementation (though not necessarily minimal 3-, 4-, ... level implementation)
Gray Code

- Different than normal binary ordering
- Reflective code
  - When you add the \((n+1)\)th bit, reflect all the previous \(n\)-bit combinations
- Consecutive code words differ by only 1-bit

Karnaugh Maps

- If used correctly, will always yield a minimal, _________ implementation
  - There may be a more minimal 3-level, 4-level, 5-level... implementation but K-maps produce the minimal two-level (SOP or POS) implementation
- Represent the truth table graphically as a series of adjacent _______ that allows a human to see where variables will cancel

Karnaugh Map Construction

- Every square represents 1 input combination
- Must label axes in Gray code order
- Fill in squares with given function values

\[
F = \Sigma_{XYZ}(1,4,5,6)
\]

\[
G = \Sigma_{WYZ}(1,2,3,5,6,7,9,10,11,14,15)
\]
### Karnaugh Maps

- Squares with a '1' represent minterms
- Squares with a '0' represent maxterms

**Maxterm:**
\[ w' + x' + y + z \]

**Minterm:**
\[ w'x'yz' + w'x'yz \]

**Maxterm:**
\[ w' + x + y + z \]

**Minterm:**
\[ w'x'yz' + w'x'yz' \]

### Karnaugh Maps

- Groups of adjacent 1’s will always simplify to smaller product term than just individual minterms

**Maxterm:**
\[ F = \sum_{X\neq Y,Z}(0,2,4,5,6) \]

**Minterm:**
\[ F = m0 + m2 + m6 + m4 = x'y'z' + x'yz' + xyz' + xy'z' = z'(x'y' + x'y + xy + xy') = z'(x'y + x'y + x(y + y')) = z'(x' + x) = z' \]

**Minterm:**
\[ F = m4 + m5 = xy'z' + xy'z = xy'(z' + z) = xy' \]

### Karnaugh Maps

- Groups of adjacent 1’s will always simplify to smaller product term than just individual minterms

**Maxterm:**
\[ F = \sum_{X\neq Y,Z}(0,2,4,5,6) \]

**Minterm:**
\[ F = m0 + m2 + m6 + m4 = x'y'z' + x'yz' + xyz' + xy'z' = z'(x'y' + x'y + xy + xy') = z'(x'y + x'y + x(y + y')) = z'(x' + x) = z' \]

### Karnaugh Maps

- Adjacent squares differ by 1-variable
  - This will allow us to use \( T_{10} = AB + AB' = A \) or \( T_{10'} = (A+B')(A+B) = A \)

**Maxterm:**
\[ F = \sum_{X\neq Y,Z}(0,2,4,5,6) \]

**Minterm:**
\[ F = m0 + m2 + m6 + m4 = x'y'z' + x'yz' + xyz' + xy'z' = z'(x'y' + x'y + xy + xy') = z'(x'y + x'y + x(y + y')) = z'(x' + x) = z' \]

**Minterm:**
\[ F = m4 + m5 = xy'z' + xy'z = xy'(z' + z) = xy' \]
Karnaugh Maps

- 2 adjacent 1's (or 0's) differ by only one variable
- 4 adjacent 1's (or 0's) differ by two variables
- 8, 16, ... adjacent 1's (or 0's) differ by 3, 4, ... variables
- By grouping adjacent squares with 1's (or 0's) in them, we can come up with a simplified expression using T10 (or T10' for 0's)

\[ w' \cdot x' \cdot y' \cdot z + w' \cdot x' \cdot y \cdot z + w' \cdot x \cdot y' \cdot z + w' \cdot x \cdot y \cdot z = w' \cdot z \]

\[ w\cdot x \cdot y \cdot z = w' \cdot x' \cdot y' \cdot z + w' \cdot x \cdot y' \cdot z + w' \cdot x \cdot y \cdot z + w' \cdot x' \cdot y \cdot z + w' \cdot x' \cdot y' \cdot z + w' \cdot x' \cdot y \cdot z + w' \cdot x \cdot y' \cdot z + w \cdot x \cdot y \cdot z \]

w'z are constant while all combos of x and y are present (x'y', x'y, xy', xy)

K-Map Grouping Rules

- Cover the 1's [=on-set] or 0's [=off-set] with ______ groups as possible, but make those groups ________ as possible
  - Make them as large as possible even if it means "covering" a 1 (or 0) that's already a member of another group
- Make groups of ____________, ... and they must be rectangular or square in shape.
- Wrapping is legal

Group These K-Maps

Karnaugh Maps

- Cover the remaining ‘1’ with the largest group possible even if it “reuses” already covered 1’s
Karnaugh Maps

- Groups can wrap around from:
  - Right to left
  - Top to bottom
  - Corners

\[
F = X'Z' + WXZ'
\]

Group This

K-Map Translation Rules

- When translating a group of 1’s, find the variable values that are constant for each square in the group and translate only those variables values to a product term
- Grouping 1’s yields SOP
- When translating a group of 0’s, again find the variable values that are constant for each square in the group and translate only those variable values to a sum term
- Grouping 0’s yields POS

Karnaugh Maps (SOP)
Karnaugh Maps (SOP)

\[ F = Y \]

Karnaugh Maps (SOP)

\[ F = Y + W'Z + \ldots \]

Karnaugh Maps (SOP)

\[ F = Y + W'Z + X'Z \]

Karnaugh Maps (POS)
Karnaugh Maps (POS)

\[ W \times X \times Y \times Z \]

\[ \begin{array}{cccc|c}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 \\
\end{array} \]

\[ F = (Y+Z)(W'+X'+Y) \]

Karnaugh Maps

- Groups can wrap around from:
  - Right to left
  - Top to bottom
  - Corners

\[ F = X'Z' \]

\[ F = X'Z + WXZ' \]

Exercises

\[ F_{SOP} = \sum_{xyz}(2,3,5,7) \]

\[ F_{POS} = \]

\[ P = \sum_{xyz}(2,3,5,7) \]

\[ P = \]
No Redundant Groups

Multiple Minimal Expressions

• For some functions, ____________ groupings exist which will lead to alternate minimal ____________...Pick one

Terminology

• Implicant: A product term (grouping of 1’s) that covers a subset of cases where \( F=1 \)
  – the product term is said to “imply” \( F \) because if the product term evaluates to ‘1’ then \( F=1 \)

• Prime Implicant: The largest grouping of 1’s (smallest product term) that can be made

• Essential Prime Implicant: A prime implicant (product term) that is needed to cover all the 1’s of \( F \)
Implicant Examples

An implicant

Not PRIME because not as large as possible

An essential prime implicant

(largest grouping possible, that must be included to cover all 1's)
### Implicant Examples

An essential prime implicant (largest grouping possible, that must be included to cover all 1’s)

An implicant, but not an essential implicant because it is not needed to cover all 1’s in the function

A prime implicant, but not an essential implicant because it is not as large as possible

### K-Map Grouping Rules

- Make groups (implicants) of 1, 2, 4, 8, ... and they must be rectangular or square in shape.
- Include the minimum number of essential prime implicants
  - Use only essential prime implicants (i.e. as few groups as possible to cover all 1’s)
  - Ensure that you are using prime implicants (i.e. Always make groups as large as possible reusing squares if necessary)

### 5- & 6-VARIABLE KMAPS
5-Variable K-Map

- If we have a 5-variable function we need a 32-square K-Map.
- Will an 8x4 matrix work?
  - Recall K-maps work because adjacent squares differ by 1-bit
- How many adjacencies should we have for a given square?
- But drawn in 2 dimensions we can’t have _adjacencies._

<table>
<thead>
<tr>
<th>VWX</th>
<th>000</th>
<th>001</th>
<th>011</th>
<th>010</th>
<th>110</th>
<th>111</th>
<th>101</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5-Variable Karnaugh Maps

- To represent the 5 adjacencies of a 5-variable function [e.g. \(f(v,w,x,y,z)\)], imagine two 4x4 K-Maps stacked on top of each other
  - Adjacency across the two maps

F = \(v'xy' + w'xy'\)

6-Variable Karnaugh Maps

- 6 adjacencies for 6-variables (Stack of four 4x4 maps)

DON'T CARE OUTPUTS
**Don’t-Cares**

- Sometimes there are certain input combinations that are illegal (i.e. in BCD, 1010 – 1111 can never occur)
- The outputs for the illegal inputs are “don’t-cares”
  - The output can either be 0 or 1 since the inputs can never occur
  - Don’t-cares can be included in groups of 1 or groups of 0 when grouping in K-Maps
  - Use them to make as big of groups as possible

*Use 'Don't care' outputs as wildcards (e.g. the blank tile in Scrabble™). They can be either 0 or 1 whatever helps make bigger groups to cover the ACTUAL 1's*

**Combining Functions**

- Given intermediate functions F1 and F2, how could you use AND, OR, NOT to make G
- Notice certain F1,F2 combinations never occur in G(x,y,z)...what should we make their output in the T.T.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F1</th>
<th>F2</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Don’t Care Example**

<table>
<thead>
<tr>
<th>D8</th>
<th>D4</th>
<th>D2</th>
<th>D1</th>
<th>GT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**GT6_{SOP}**

<table>
<thead>
<tr>
<th>D8</th>
<th>D4</th>
<th>D2</th>
<th>D1</th>
<th>GT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**GT6_{POS}**

**Don’t Cares**

- Reuse “d’s” to make as large a group as possible to cover 1,5, & 9

**Don’t Cares**

- Use these 4 “d’s” to make a group of 8

**F = Z + Y**
Don’t Cares

<table>
<thead>
<tr>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>d</td>
</tr>
</tbody>
</table>

You can use “d’s” when grouping 0’s and converting to POS

Designing Circuits w/ K-Maps

- Given a description...
  - Block Diagram
  - Truth Table
  - K-Map for each output bit (each output bit is a separate function of the inputs)

- 3-bit unsigned decremter (Z = X-1)
  - If X[2:0] = 000 then Z[2:0] = 111, etc.

3-bit Number Decrementer

<table>
<thead>
<tr>
<th>X_2</th>
<th>X_1</th>
<th>X_0</th>
<th>Z_2</th>
<th>Z_1</th>
<th>Z_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

3-bit Number Decrementer

\[ Z_2 = X_2X_0 + X_2X_1 + X_2'X_0'X_1' \]

\[ Z_1 = X_1'X_0' + X_1X_0 \]

\[ Z_0 = X_0' \]

Squaring Circuit

- Design a combinational circuit that accepts a 3-bit number and generates an output binary number equal to the square of the input number. (B = A^2)
- Using 3 bits we can represent the numbers from ______ to ______.
- The possible squared values range from ______ to ______.
- Thus to represent the possible outputs we need how many bits? ______
3-bit Squaring Circuit

<table>
<thead>
<tr>
<th>A</th>
<th>A_2</th>
<th>A_1</th>
<th>A_0</th>
<th>B_5</th>
<th>B_4</th>
<th>B_3</th>
<th>B_2</th>
<th>B_1</th>
<th>B_0</th>
<th>B \equiv A^2</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

B_5 =

<table>
<thead>
<tr>
<th>A</th>
<th>A_2</th>
<th>A_1</th>
<th>A_0</th>
<th>B_5</th>
<th>B_4</th>
<th>B_3</th>
<th>B_2</th>
<th>B_1</th>
<th>B_0</th>
<th>B \equiv A^2</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

B_4 =

<table>
<thead>
<tr>
<th>A</th>
<th>A_2</th>
<th>A_1</th>
<th>A_0</th>
<th>B_5</th>
<th>B_4</th>
<th>B_3</th>
<th>B_2</th>
<th>B_1</th>
<th>B_0</th>
<th>B \equiv A^2</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

B_0 =

Dimensions and Operations

USING MEMORIES TO BUILD COMBINATIONAL CIRCUITS

MEMORY BASICS
Memories

- Memories store (write) and retrieve (read) data
  - Read-Only Memories (ROM’s): Can only retrieve data (contents are initialized and then cannot be changed)
  - Read-Write Memories (RWM’s): Can retrieve data and change the contents to store new data

ROM’s

- Memories are just _____ of data with rows and columns
- When data is read, one entire _____ of data is read out
- The row to be read is selected by putting a binary number on the ________ inputs

Example

- Address = 4 dec. = 100 bin. is provided as input
- ROM outputs data in that row (1101 bin.)

Memory Dimensions

- Memories are named by their dimensions:
  - Rows x Columns
- \( n \) rows and \( m \) columns \( \Rightarrow \) ___ x ____ ROM
- \( 2^n \) rows \( \Rightarrow \) \( n \) address bits
  ...or...
- \( k \) rows \( \Rightarrow \log_2k \) address bits
- \( m \) cols \( \Rightarrow \) \( m \) data outputs

\[
\begin{array}{c|c|c|c|c}
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]
RWM’s

• Writable memories provide a set of data inputs for write data (as opposed to the data outputs for read data)
• A control signal R/W (1=READ / 0 = WRITE) is provided to tell the memory what operation the user wants to perform

Write example
– Address = 3 dec. = 011 bin.
– DI = 12 dec. = 1100 bin.
– R/W = 0 => Write op.
• Data in row 3 is overwritten with the new value of 1100 bin.

RWM’s

Memories as Look-Up Tables

• One major application of memories in digital design is to use them as LUT’s (Look-Up Tables) to implement logic functions
  – This is the core technology used by FPGAs (Field-Programmable Gate Arrays) which are used pervasively in robotics, space applications, networking, and even in search engines
• Idea: Use a memory to hold the __________ of a function and feed the inputs of the function to the __________ inputs to "__________" the answer

USING MEMORIES TO BUILD COMBINATIONAL FUNCTIONS
Implementing Functions w/ Memories

8x1 Memory

X Y Z F
0 0 0 1
0 0 1 0
0 1 0 1
0 1 1 1
1 0 0 0
1 0 1 0
1 1 0 0
1 1 1 1

Use a memory with the same dimensions as 'output' side of the truth table. It's almost TOO easy.

Implementing Functions w/ Memories

8x2 Memory

X Y Z C S
0 0 0 0 0
0 0 1 0 1
0 1 0 0 1
0 1 1 0 1
1 0 0 0 1
1 0 1 1 0
1 1 0 1 0
1 1 1 1 1

Multi-bit function (One's count)

Use a memory with the same dimensions as 'output' side of the truth table. It's almost TOO easy.

3-bit Squaring Circuit

- Q: What size memory would you use to build our 3-bit squaring circuit?
  - A: ______ memory

- Q: What would you connect to the address inputs of the memory?
  - A: ______

- Q: What bits would you program into row 5 of the memory?
  - A: ____________

4x4 Multiplier Example

Determine the dimensions of the memory that would be necessary to implement a 4x4-bit unsigned multiplier with inputs $X[3:0]$ and $Y[3:0]$ and outputs $P[2:0]$

Question: How many bits are needed for $P$?

Question: What are the contents of the numbered rows?

Example:

$$X_3 X_2 X_1 X_0 = 0010$$
$$Y_3 Y_2 Y_1 Y_0 = 0001$$

$$P = X \times Y = 2 \times 1 = 2$$

$$= 00010$$
Implementing Functions w/ Memories

• To implement a function with \( n \)-variables and \( m \) outputs
• Just place the output truth table values in the memory
• Memory will have dimensions: \( 2^n \) rows and \( m \) columns
  – Still does not scale terribly well (i.e. \( n \)-inputs requires memory \( 2^n \) outputs)
  – But it is easy and since we can change the contents of memories it allows us to create "reconfigurable" logic
  – This idea is at the heart of FPGAs