Module 11: Address Decoding

- Address decoder design methodology
- Full address decoder
- Partial address decoder
To transfer data correctly and prevent system damage, only one device at a time should be allowed to drive the data bus.

The address decoder selects and enables **one and only one** data transfer device at a time.

The decoder decodes (detects) the value of the address bus based on the memory map.
When implementing a system the designer creates a memory map. Map would include where RAM, ROM and I/O are.

Useful Constants:
- $400 = 1$ kilo = 10 lines
- $1000 = 4$ kilo
- $4000 = 16$ kilo
- $100000 = 1$ Mega
Address Decoding Strategy

- Address bus (M+N bits)
  - M bits to memory \((A_1 - A_M)\)
  - N bits to decoder \((A_{M+1} - A_{23})\)

- Address decoder
- EPROM or SRAM

- CS
- SEL
- AS

Must activate SEL only when AS = 0 because address bus is invalid at other times
How Many Bits to Decode?

In general,
- memories have many internal locations → many lines connect direct to address bus.
- I/O devices have very few location → few connections to addr bus

Examples memory chips:
- 6164 8KB SRAM or 2764 8KB EPROM:
  - 13 address lines to chip
  - # address lines to decoder → 24 – 13 = 9
- 27C040 512KB EPROM
  - 19 address lines to chip
  - #address lines to decoder → 24 – 19 = 5

Examples I/O chips:
- 68230 PI/T
  - 32 registers → 5 address lines (RS1-RS5) but must connect to A1-A5
  - List of address lines to decoder → A23 .. A6 = 17
- 74HC574 latch or 74HC541 buffer
  - 8 bits → 1 address exactly → no addr line to device
  - All address lines go to decoder
Address Decoding Methods

- **Full address decoding**
  - All the address lines are used to specify a memory location.
  - Each physical memory location is identified by a unique address.

- **Partial address decoding**
  - Since not all the address space is implemented, only a subset of the address lines are needed to point to the physical memory locations.
  - Each physical memory location is identified by several possible addresses (using all combinations of the address lines that were not used).
Implementing Address Decoders

- **Combinational logic**
  - AND, NAND, OR, NOR, NOT
  - High speed (propagation signals)
  - High chip-count
  - Lacks flexibility

- **Decoders**
  - 2-to-4, 3-to-8, etc
  - More appropriate than random logic
  - The selection of devices is determined by the physical wiring
  - All the memory blocks must have the same size

- **Other methods (beyond the scope of the lecture) are**
  - Programmable Array Logic (PAL)
  - Programmable Address Decoders
  - Programmable Read Only Memory (PROM)
  - Field Programmable Gate Arrays (FPGA) – Most complex & elegant. A single chip can provide all necessary glue logic (memory control, DTACK generator & address decoder).
Memory Decoder Design Steps

1. Determine range of address for each device:
   - Find starting address (base address)
   - Find memory system size (#bytes provided per device group)
   - Find ending address

2. Determine which address lines go to decoder
   - Find #address lines on device
   - Low address go direct from µP to memory
   - Remaining address lines to go memory decoder -> find required pattern to decode

3. Design decoder to detect the required address bus pattern
   - Write starting address in binary
   - Write ending address in binary
   - High order address bits that go to decoder must match in both starting & ending addresses
   - Draw circuit to detect these high order bits
Example 1

- Design the decoding circuit for interfacing 2764 EPROMs with a basic 68000 system. Assume that the 2764 chips is the only memory device used.

1. Find address range
   - Find starting address
     - All ROMs for basic system must start at $000000
   - Find memory system size
     - Each 2764 chip has 8KB in 8k x 8 organization -> covers only one half of data bus
     - To provide data for upper/lower data bus, must use 2 chips
     - 2 chips will provide 8 KB x 2 = 16 KB = 16 x 1024 = 16384 bytes
     - Write down 16384 in hex -> $4000
   - Find ending address
     - Ending address = starting address + memory system size – 1
     - 0 + $4000 – 1 = $3FF
Example 1 cont

2. Determine which address lines to go decoder

- Find # address lines go to memory system
  
  \[ \text{#lines} = \log_2(\text{memory system size}) \]
  
  \[ = \log_2(16384) = \log_2(16 \times 1024) \]
  
  \[ = \log_2 (2^4 \times 2^{10}) \]
  
  \[ = 14 \text{ lines} \]

- Low address go to memory system
  
  - lowest 14 lines of µP internal address bus -> A0 to A13
  
  - A0 is internal to µP -> just ignore
  
  - A1 – A13 of µP go to A0 - A12 of every chip

- Remaining address to decoder
  
  - A14 – A23

```
To decoder

<table>
<thead>
<tr>
<th>23 22 21 20</th>
<th>19 18 17 16</th>
<th>15 14 13 12</th>
<th>11 10 9 8</th>
<th>7 6 5 4</th>
<th>3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 X X</td>
<td>X X X X</td>
<td>X X X X</td>
<td>X X X X</td>
</tr>
</tbody>
</table>

To memory
```

© 2006-2008 mzabidi@ieee.org
3. Design decoder to detect the required address bus pattern
   - Starting Address = $000000:
     - $0000 0000 0000 0000 0000 0000
   - Ending address = $003FFF:
     - $0000 0000 0011 1111 1111 1111
   - Write this range in short form
     - $0000 0000 00xx xxxx xxxx xxxx
   - Bits that are not x are the bits that must be detected:
     - A23-A14 = 0000 0000 00
     - AS* = 0

SEL will be low when µP is accessing any address in the range $000000 - $003FFF
Example 2

- A circuit containing 64K words of RAM is to be interfaced to a 68000-based system
- The first address of RAM (the base address) is at $480000
  - What is the entire range of RAM addresses?
  - Design a FULL address decoder
- Solution
  - The address range for the RAM is from $480000 to $480000 + (128K = $20000) - 1 = $4A0000 - 1 = $49FFFF

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

These 7 address lines set the base address of the memory

These 16 address lines will select one of the $2^{16}$ (64K) locations inside each RAM IC

This address line is implemented with UDS*/LDS*
Example 3

- Design a full address decoder for a 68000-based system that contains:
  - 2MB of EPROM at a starting address $00 0000$ using 512Kx8 chips
  - 2MB of RAM at a starting address $20 0000$ using 256Kx8 chips
  - 64KB I/O space starting at $FF0000$

**SOLUTION**

- For the EPROM we will need 4 512Kx8 chips, organized as 2 pairs of 512x8 chips (in order to use UDS*/LDS*). We will call these pairs ROM1 and ROM2.
- For the RAM we will need 8 256Kx8 chips, organized as 4 pairs of 256Kx8: RAM1 to RAM4.
### Example 3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ROM2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RAM1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RAM2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RAM3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RAM4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
| I/O  | 1   | 1   | 1   | 1   | 1   | 1   | 1   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X

![Logic Diagrams]
Partial Decoding

- The memory space covered by all memory chip is usually much less than the full 16 MB space addressable by the 68k.
- For systems with not more than 8 different devices, using the 74HC138 decoder is better solution.
- Each device occupy block of same size.
74HC138

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Select</td>
</tr>
<tr>
<td>E1</td>
<td>E2</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

When chip is not enabled: all 8 outputs high independent of A inputs

When chip enabled (E1*, E2*, E3=001) only one output goes low, rest high

- Inputs A2, A1, A0 select which of 8 outputs goes low

© 2006-2008 mzabidi@ieee.org
Example 4 – Partial Decoder for Simple System

<table>
<thead>
<tr>
<th>Address</th>
<th>ROM1 2MB</th>
<th>RAM 2MB</th>
<th>ROM2 2MB</th>
<th>Unused</th>
<th>Unused</th>
<th>Peripheral1 2MB</th>
<th>Peripheral2 2MB</th>
<th>Peripheral3 2MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 0000</td>
<td>A21</td>
<td>A22</td>
<td>A23</td>
<td>Y0</td>
<td>Y1</td>
<td>ROM1 0 0 0</td>
<td>RAM 0 0 1</td>
<td>Peripheral1 1 0</td>
</tr>
<tr>
<td>20 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ROM2 0 1 0</td>
<td>Peripheral2 1 1</td>
<td>Peripheral3 1 1</td>
</tr>
<tr>
<td>40 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Peripheral1 1 0</td>
<td>Peripheral2 1 1</td>
<td>Peripheral3 1 1</td>
</tr>
<tr>
<td>60 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Peripheral3 1 1</td>
</tr>
<tr>
<td>80 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Peripheral1 1 0</td>
<td>Peripheral2 1 1</td>
<td>Peripheral3 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>ROM1 2MB</th>
<th>RAM 2MB</th>
<th>ROM2 2MB</th>
<th>Unused</th>
<th>Unused</th>
<th>Peripheral1 2MB</th>
<th>Peripheral2 2MB</th>
<th>Peripheral3 2MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 0000</td>
<td>A21</td>
<td>A22</td>
<td>A23</td>
<td>Y0</td>
<td>Y1</td>
<td>ROM1 0 0 0</td>
<td>RAM 0 0 1</td>
<td>Peripheral1 1 0</td>
</tr>
<tr>
<td>20 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ROM2 0 1 0</td>
<td>Peripheral2 1 1</td>
<td>Peripheral3 1 1</td>
</tr>
<tr>
<td>40 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Peripheral1 1 0</td>
<td>Peripheral2 1 1</td>
<td>Peripheral3 1 1</td>
</tr>
<tr>
<td>60 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Peripheral3 1 1</td>
</tr>
<tr>
<td>80 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Peripheral1 1 0</td>
<td>Peripheral2 1 1</td>
<td>Peripheral3 1 1</td>
</tr>
</tbody>
</table>

Block size = memory size / #decoder outputs = 16 MB / 8 = 2 MB

© 2006-2008 mzabidi@ieee.org
Mirror Effect in Partial Decoding

- When device size < block size, aliasing or mirroring occurs
  - Device will respond to >1 address
- Example: using 27C010 (512k x 8) device in 2 MB block
  - ‘010 has 17 lines
  - A1-A17 go direct to ‘010 chips
  - A21-A23 go to decoder
  - A18-A20 unconnected → chip has 7 aliases
- Device will respond anytime A21-A23 = 000
  - e.g. CLR $100000 will have same effect as CLR $000000
- Same effect for rest of decoder outputs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM1</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Connection | To decoder | No Connect | To memory chip
Example 5 – Alternative Partial Decoding Scheme

- Each block is 256KB.
- Instead of A18-A20 unconnected, we can have A21-A23 unconnected (see analysis for ROM1 below)
- We can use A21 as additional enable to reduce aliasing

<table>
<thead>
<tr>
<th>ROM1</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A10</th>
<th>A11</th>
<th>A12</th>
<th>A13</th>
<th>A14</th>
<th>A15</th>
<th>A16</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>y</td>
<td>y</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Connection: No Connect To decoder To memory chip
Example 6 – System with 16 Devices

- Each block is 256KB.
- A21 is used to select top decoder (A21=0) or bottom decoder (A21=1)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROM1</td>
<td>y</td>
<td>y</td>
<td>w</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Connection
- N.C. To decoder
- To memory chip

© 2006-2008 mzabidi@ieee.org
### Example 3 Revisited

#### Use 1-to-2 decoder

- Use 3-to-8 decoder.
- Use 2-to-4 decoder. Can use ‘HC138 or simpler ‘HC139.
- Can be fully decoded or ignored altogether.

<table>
<thead>
<tr>
<th>Address</th>
<th>ROM1</th>
<th>ROM2</th>
<th>RAM1</th>
<th>RAM2</th>
<th>RAM3</th>
<th>RAM4</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 0000</td>
<td>512Kx8</td>
<td>512Kx8</td>
<td>ROM1</td>
<td>ROM2</td>
<td>RAM1</td>
<td>RAM2</td>
<td>RAM3</td>
</tr>
<tr>
<td>00 0000</td>
<td>512Kx8</td>
<td>512Kx8</td>
<td>ROM1</td>
<td>ROM2</td>
<td>RAM1</td>
<td>RAM2</td>
<td>RAM3</td>
</tr>
<tr>
<td>20 0000</td>
<td>512Kx8</td>
<td>512Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
</tr>
<tr>
<td>21 FF FFE</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
</tr>
<tr>
<td>FF 0000</td>
<td>512Kx8</td>
<td>512Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
</tr>
<tr>
<td>FF FF FFE</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
<td>256Kx8</td>
</tr>
</tbody>
</table>

© 2006-2008 mzabidi@ieee.org
2-Stage Decoders

Bypass decoder and take IOSEL* signal here if aliasing is acceptable
Example 7 – EASy68K Hardware Model

- ROM is not defined in the simulator!
  - ROM exists in all systems but a simulator is not a real system.
  - You must have ROM to implement TRAP #15 routines.
- You can write your program anywhere!
  - This does not happen in a real system
- Let’s create something close → EZ68k

<table>
<thead>
<tr>
<th>I/O Locations</th>
<th>00 0000</th>
<th>00 1000</th>
<th>FF FFFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 KB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User RAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 MB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O Space &amp; Stack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 MB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 1</td>
<td>E0 0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 2</td>
<td>E0 0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 3</td>
<td>E0 0004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 4</td>
<td>E0 0006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 5</td>
<td>E0 0008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 6</td>
<td>E0 000A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 7</td>
<td>E0 000C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Digit 8</td>
<td>E0 000E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED x 8</td>
<td>E0 0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rocker switch</td>
<td>E0 0012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pushbutton switch</td>
<td>E0 0014</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EZ68k Memory Map

- User RAM 2 MB
- ROM 2 MB
- Unused
- I/O Space 1 MB
- Stack RAM 1 MB
- LED Digit 1
- LED Digit 2
- LED Digit 3
- LED Digit 4
- LED Digit 5
- LED Digit 6
- LED Digit 7
- LED Digit 8
- LED x 8
- Rocker switch
- Pushbutton switch
## EZ68k Decoder Worksheet

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>User RAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ROM</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7SLED #1</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SLED #2</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SLED #3</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SLED #4</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SLED #5</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SLED #6</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SLED #7</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SLED #8</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED8</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pushbutton</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack RAM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Use 3-to-8 decoder (1xHC138)

Use 1-to-2 decoder

Use 4-to-16 decoder (2xHC138)
Decoder Circuit
Wrong Way to Connect ROM

- A0 = UDS/LDS
- A1-A14 micro-both ROMs A0-A13
- ROM1 = D8-D15
- ROM2 = D0-D7
- Address decoder selects ROMs for A16,A17,A18=000
- ROM: $0000 - $03FFF, other 138 outputs used for other devices, RAM etc

<table>
<thead>
<tr>
<th>ADDRESS BUS</th>
<th>DEVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>X X X X X X X X X X X X X X</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>X X X X X X X X X X X X X</td>
</tr>
</tbody>
</table>

U = UDS    L = LDS    X = Variable
Connecting RAM

- Addition of two 32K x 8 RAM to previous slide (two of ROM of last slide not shown for clarity)
- Again pair for 16 bits wide
- ROM A_{16}-A_{18}=000
- RAM A_{16}-A_{18}=001
- Now R/~W needed
- DTACK* as long as either ROM or RAM accessed.

<table>
<thead>
<tr>
<th>18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>Chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 X X X X X X X X X X X X X X X X X X U</td>
<td>ROM 1</td>
</tr>
<tr>
<td>0 0 0 0 X X X X X X X X X X X X X X X X X X L</td>
<td>ROM 2</td>
</tr>
<tr>
<td>0 0 1 X X X X X X X X X X X X X X X X X X X X U</td>
<td>RAM 1</td>
</tr>
<tr>
<td>0 0 1 X X X X X X X X X X X X X X X X X X X X L</td>
<td>RAM 2</td>
</tr>
</tbody>
</table>

ROM 0000-3FFFFH

RAM 8000-FFFFFH
16 bit wide ROM

- D_0-D_{15} ROM→uP
- All uP A_1-A_{16} → ROM A_0-A_{15}
- 64k x 16 ROM
- CE* from 138 decoder when A_{17}, A_{18}, A_{19}=000
  Other combinations for other devices
- As ROM all accesses are read so *OE=CE*
- DTACK* low while ROM selected

<table>
<thead>
<tr>
<th>A_{19}</th>
<th>A_{18}</th>
<th>A_{17}</th>
<th>A_{16}</th>
<th>A_{15}</th>
<th>A_{14}</th>
<th>A_{13}</th>
<th>A_{12}</th>
<th>A_{11}</th>
<th>A_{10}</th>
<th>A_{9}</th>
<th>A_{8}</th>
<th>A_{7}</th>
<th>A_{6}</th>
<th>A_{5}</th>
<th>A_{4}</th>
<th>A_{3}</th>
<th>A_{2}</th>
<th>A_{1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ROM 00000H-0FFFFFH
Next Device 10000H→
Another Device 20000H→
Connecting LEDs

- Need a byte-wide output port
- The LEDs cannot be connected directly to data bus
  - Difficult to select the LEDs
  - LEDs would only display value for very short period of time (about 400ns, or 2 clock cycles)
    - Only when data bus carries the correct signal
  - Microprocessor cannot sink enough current
Instead, we need to capture the values on the data bus, and hold them until changed

- The 74HC374 octal latch will do nicely

- Latch is very fast (around 20 ns), so DTACK* does not need to be delayed.

- Needs only 1 memory address.
Connecting LEDs

68000

Data bus

74HC374

Address bus

74HC138 decoder

© 2006-2008 mzabidi@ieee.org
74LS373 & 74LS374

<table>
<thead>
<tr>
<th>Output Control</th>
<th>Enable G</th>
<th>D</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>X</td>
<td>Q₀</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>X</td>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Control</th>
<th>Clock</th>
<th>D</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>↑</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>↑</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>X</td>
<td>Q₀</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>X</td>
<td>Z</td>
</tr>
</tbody>
</table>

H = High Level (Steady State), L = Low Level (Steady State), X = Don’t Care
↑ = Transition from low-to-high level, Z = High Impedance State
Q₀ = The level of the output before steady-state input conditions were established.
Elementary Output with 68000
Connecting Switches

- Need a byte-wide input port
- Switches cannot be connected directly to data bus
  - Must transfer switch values to data bus at the correct time
  - A tri-state buffer chip will be just the thing
74LS244 & 74LS541 Octal 3-State Buffer/Line Driver

Inputs | Output
---|---
G | A | Y
L | L | L
L | H | H
H | X | Z

L = LOW Logic Level
H = HIGH Logic Level
X = Either LOW or HIGH Logic Level
Z = High Impedance
Elementary Input with 68000

68000

D0-D7 or
D8-D15

A1-A23

AS

68000 Buffer

D0-D7

Address decoder

OE

DTACK

Vcc

Other DTACK sources
An Alternative Decoder for Glue Logic

- 74LS139 Decoder
- Contains 2 x 2-to-4 decoders in one chip
- Only one enable per decoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Select</td>
</tr>
<tr>
<td>E</td>
<td>A1  A0</td>
</tr>
<tr>
<td>1</td>
<td>X  X</td>
</tr>
<tr>
<td>0</td>
<td>0  0</td>
</tr>
<tr>
<td>0</td>
<td>0  1</td>
</tr>
<tr>
<td>0</td>
<td>1  0</td>
</tr>
<tr>
<td>0</td>
<td>1  1</td>
</tr>
</tbody>
</table>

© 2006-2008 mzabidi@ieee.org
Using FPGA or CPLD as Glue Logic

- The most elegant solution for glue logic!
- FPGA (Field-Programmable Gate Array) or CPLD (Complex Programmable Logic Device) is a very attractive device capable of implement ALL glue logic functions in one chip
  - Decoder
  - Memory read/write enables
  - DTACK generator including wait states
  - BERR generator
- Must program using Verilog or VHDL language
- Example chips:
  - Xilinx 9572 CPLD
    - (http://www.kmitl.ac.th/%7Ekswichit%20/68k/68ksbc.pdf)
Using PLD for Glue Logic

- Simpler than FPGA or CPLD
- Several versions:
  - PAL (programmable array logic) – programmable OR gates, fixed AND gates, may contain flip-flops, programmable once
  - GAL (general array logic) – replaces PAL, reprogrammable
  - ROM
  - PLA (programmable logic array)
- To implement DTACK & BERR, designer must know internal architecture of each device to avoid using too many chips
- To program GAL or PLA, may need to know another language such as ABEL