Memory Interfacing
Parallel Interfacing to Memory

- A typical memory module has
  - Address bus input
  - Data bus output (both input and output in case of a RAM)
  - The “chip select” (CS) input signal enables the device
  - The “write enable” (WE) input signal tells the device to write (in the case of a RAM)
Typical read cycle

- (A) Read bus cycle begins with falling edge of the E clock
- (B) MCU places the address on the address bus and sets R/W high for read
- (C) Chip select logic enables the device
- (D) Device places data on data bus
- (E) MCU latches data in at falling edge of E clock

The main control signals coming out of the MCU are the E-clock and the R/W signal.
Typical write cycle

- (A) Write bus cycle begins with falling edge of the E clock
- (B) MCU places the address on the address bus and sets R/W low for write
- (C) MCU places data on data bus
- (E) Device latches data in after falling edge of E clock (D)
Chip select logic

• The chip select logic selects the device
  – It decodes the address of the read or write
  – If the address falls within the block defined for the device, the device is activated
  – The device then decodes the rest of the address to select the specific location within the device

• Important
  – Only one device should be selected at a time
  – Otherwise two devices could try to drive the data bus at the same time, resulting in a short circuit

• Chip select logic divides the memory map into blocks
Chip select logic

- Example: an 8 KB RAM
  - 8 KB is 8192 bytes, or $2^{13}$ bytes. In hex, this is $0x2000$ bytes.
  - We can address any location in the RAM with 13 address bits
  - The remaining 3 bits are used by the chip select logic to place the RAM in the 64 KB memory map

```
A15  A14  A13  X  X  X  X  X  X  X  X  X
```

- **decoded by the chip select logic**  
- **decoded by the RAM**

- Example address decoder
  - $/CS = (A15*A14*A13)'$
  - The RAM is selected whenever $A15=A14=A13=1$
  - So any address from $1110\ 0000\ 0000\ 0000$ to $1111\ 1111\ 1111\ 1111$ is ok
  - Or, $\$E000$ to $\$FFFF$
FIGURE 11-3  32K- 16K- and 8K-byte Block Boundaries

<table>
<thead>
<tr>
<th>A15</th>
<th>A14</th>
<th>A13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$3FFF $4000</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>$7FFF $8000</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>$BFFF $C000</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>$FFFF</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>$0000</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>$1FFF $2000</td>
<td>0 0 1</td>
<td></td>
</tr>
<tr>
<td>$3FFF $4000</td>
<td>0 1 0</td>
<td></td>
</tr>
<tr>
<td>$5FFF $6000</td>
<td>0 1 1</td>
<td></td>
</tr>
<tr>
<td>$7FFF $8000</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>$9FFF $A000</td>
<td>1 0 1</td>
<td></td>
</tr>
<tr>
<td>$BFFF $C000</td>
<td>1 1 0</td>
<td></td>
</tr>
<tr>
<td>$DFFF $E000</td>
<td>1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
Examples

• 8 KB block
  – The module has 13 address bits; the remaining 3 bits are used by the chip select logic to place the module in the 64 KB memory map
    A15  A14  A13  X  X  X  X  X  X  X  X  X  X
  – Say we want to place the module in the range $C000 to $DFFF
    In binary, 1100 0000 0000 0000 to 1101 1111 1111 1111
  – The module should be selected whenever (A15,A14,A13) = (110)
  – So use /CS = (A15*A14*A13’)’

• 16 KB block
  – The module has 14 address bits; the remaining 2 bits are used by the chip select logic to place the module in the 64 KB memory map
    A15  A14  X  X  X  X  X  X  X  X  X  X
  – Say we want to place the module in the range $8000 to $BFFF
    In binary, 1000 0000 0000 0000 to 1011 1111 1111 1111
  – The module should be selected whenever (A15,A14) = (10)
  – So use /CS = (A15*A14’)’
Examples

• 4 KB block
  – The module has how many address bits?
  
  – How many bits are used by the chip select logic?
  
  – Say we want to place the module in the range $5000$ to $5FFF$. Equivalent addresses in binary?
  
  – The module should be selected using what values for the upper address bits?
  
  – /CS = ?
- Memory map of hypothetical system

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>Internal Registers</td>
</tr>
<tr>
<td>$01FF</td>
<td>Unused</td>
</tr>
<tr>
<td>$0200</td>
<td>Modem</td>
</tr>
<tr>
<td>$03FF</td>
<td>Unused</td>
</tr>
<tr>
<td>$0400</td>
<td>On-Chip RAM</td>
</tr>
<tr>
<td>$0407</td>
<td>Unused</td>
</tr>
<tr>
<td>$0408</td>
<td>Byte-Erasable EEPROM</td>
</tr>
<tr>
<td>$07FF</td>
<td>Unused</td>
</tr>
<tr>
<td>$0800</td>
<td>RAM</td>
</tr>
<tr>
<td>$0BFF</td>
<td>Unused</td>
</tr>
<tr>
<td>$0C00</td>
<td>Unused</td>
</tr>
<tr>
<td>$0CFF</td>
<td>Unused</td>
</tr>
<tr>
<td>$0F00</td>
<td>EPROM</td>
</tr>
<tr>
<td>$0FFF</td>
<td></td>
</tr>
<tr>
<td>$1000</td>
<td></td>
</tr>
<tr>
<td>$1FFF</td>
<td></td>
</tr>
<tr>
<td>$2000</td>
<td></td>
</tr>
<tr>
<td>$27FF</td>
<td></td>
</tr>
<tr>
<td>$2800</td>
<td></td>
</tr>
<tr>
<td>$DFFF</td>
<td></td>
</tr>
<tr>
<td>$E000</td>
<td></td>
</tr>
<tr>
<td>$FFFF</td>
<td></td>
</tr>
</tbody>
</table>
Chip select logic - RAM

• Example: place a 2 KB RAM starting at address $2000
  – 2 KB is 2048 bytes, or $2^{11}$ bytes
  – We can address any location in the RAM with 11 address bits
  – The remaining 5 bits are used by the chip select logic to place the RAM in the 64 KB memory map

$$A_{15} A_{14} A_{13} A_{12} \quad A_{11} \quad X \quad X \quad X \quad X \quad X \quad X \quad X \quad X \quad X \quad X \quad X \quad X \quad X$$

decoded by the chip select logic  \hspace{0.5cm} decoded by the RAM

• If we want the RAM to start at $2000
  – Address $2000$ is $0010 \ 0000 \ 0000 \ 0000$
  – So we want $/CS = (A_{15'} \ * \ A_{14'} \ * \ A_{13} \ * \ A_{12'} \ * \ A_{11'})'$
  – The RAM occupies the block $0010 \ 0000 \ 0000 \ 0000$ to $0010 \ 0111 \ 1111 \ 1111$
  – Or, $2000$ to $27FF$
Partial address decoding

- The chip select logic for small devices is complex (e.g., the modem uses only 8 address locations)

- We can simplify it by not using all the address bits

- Effectively, we make the memory block for the device larger than needed
  - We can do this if there is unused address space
  - This is called “partial decoding”

- In the example shown, there is plenty of unused address space between $2000$-$DFFF$
Chip select logic - RAM

- The RAM occupies an 8 KB address space
  - So we will use the upper 3 bits to select the space
  - RAM is only 2 KB, so it needs only the lower 11 address bits to address 2 KB
  - The remaining 2 bits are ignored

\[
\begin{align*}
A_{15} & A_{14} & A_{13} & A_{12} & A_{11} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} \\
\end{align*}
\]

(decoded by the chip select logic)  (decoded by the RAM)

- To put the RAM at $2000, let $/CS = (A_{15}’ \ast A_{14}’ \ast A_{13})’$
- We can write to say, RAM location 1, using a store to address $2001$
  - Address $2001$ is \[0010 \ 0000 \ 0000 \ 0001\]
    (decoded by the chip select logic)  (decoded by the RAM)
Virtual copies of RAM

- If we write to $3801
  - Address $3801 is **0011 1000 0000 0001**
- This also stores to location 1 in the RAM!
  - In fact, writes to $2001, $2801, $3001, $3801 all store to the same location
- Effectively, we have 4 images, or virtual copies, of the same RAM block
Summary / Questions

• A parallel interface to a memory module requires
  – Address and data bus
  – Some kind of “write enable” signal
  – Some kind of “chip select” signal

• An address decoder determines whether the address on the address bus is within the space occupied by the memory module, and drive the “chip select” signal

• In some cases, why can the address decoder get away with not decoding all of the address bits?