Memory Interface

Dr. Mohammed Morsy
Memory Interface

• This chapter will discuss:
  o Memory Devices: ROM, EEPROM, SRAM, DRAM.
  o Address Decoding.
  o 8088 Memory Interface.
  o 8086 Memory Interface.
Memory Types

• Every microprocessor-based system has a memory system

• Two basic types:
  o Read-only memory (ROM): It stores system software and permanent system data
  o Random access memory (RAM): It stores temporary data and application software (data and instructions)

• Four commonly used memories:
  o ROM
  o Flashable EEPROM
  o Static RAM (SRAM)
  o Dynamic RAM (DRAM), SDRAM, RAMBUS, DDR RAM
Memory Devices

- Generic pin configuration:

  Address connection

  $A_0 \ A_1 \ \ldots \ \ A_N$

  Output/Input-output connection

  $O_0 \ O_1 \ \ldots \ \ O_N$

  $A_0 \ A_1 \ \ldots \ \ A_N$

  $O_0 \ O_1 \ \ldots \ \ O_N$

  WE

  WE

  OE

  OE

  CS

  CS

  Write

  Read

  Select
Address Pins

- All memory devices have address inputs.
- They select a memory location within the memory device.
- The number of address pins is related to the number of memory locations.
  - Common sizes today are 1K to 256M locations.
  - Therefore, between 10 and 28 address pins are present.
- Address inputs are labeled from $A_0$ to $A_N$.
- $N$ is the total number of address pins minus 1.
- Example: The 2K memory:
  - It has 11 address lines.
  - The labels are ($A_0$ - $A_{10}$).
  - If the start address is 10000H so the end address is:
    $10000H + ((2*1024)_{10} = 800H) = 107FFH$
Data Pins

• All memory devices have a set of data outputs or input/outputs.
• They are used to enter the data for storage or extract the data for reading.
• The data pins are typically bi-directional in read-write memories.
  o The number of data pins is related to the size of the memory location.
  o For example, an 8-bit wide (byte-wide) memory device has 8 data pins.
  o Catalog listing of 1K X 8 indicate a byte addressable 8Kbit memory with 10 address pins.
• Memory devices are defined by memory locations times bit per location.
• Examples: 1K×8, 16K×1, 64K×4.
Selection Pins

- Each memory device has at least one chip select (\(\overline{CS}\)) or chip enable (\(\overline{CE}\)) or select (\(\overline{S}\)) pin that enables the memory device.
  - This enables read and/or write operations.
  - If more than one are present, then all must be 0 in order to perform a read or write.

- If they are active (logic 0), the memory device performs a read or write operation.

- If they are inactive (logic 1), the memory is disabled and do not do any operation.

- If more than one selection connection is present. All must be activated to read or write.
Control Pins

- Each memory device has at least one control pin.
- For ROMs, an Output Enable ($\overline{OE}$) or Gate ($\bar{G}$) is present.
- The $\overline{OE}$ pin enables and disables a set of tri-state buffers.
- For RAMs, a read-write ($R/\bar{W}$) or write enable ($\overline{WE}$) and read enable ($\overline{OE}$) are present.
- For dual control pin devices, it must be hold true that both are not 0 at the same time.
ROMs

- Non-volatile memory: Maintains its state when powered down.
- There are several forms:
  - ROM: Factory programmed, cannot be changed. Older style.
  - Programmable Read-Only Memory (PROM): Field programmable but only once. Older style.
  - Erasable Programmable Read-Only Memory (EPROM): Reprogramming requires up to 20 minutes of high-intensity UV light exposure.
  - Electrically Erasable Programmable ROM (EEPROM): Also called Electrically Alterable ROM (EAROM) and NOVRAM (Non-Volatile RAM).
    - Writing is much slower than a normal RAM.
    - Used to store setup information, e.g. video card, on computer systems.
    - Can be used to replace EPROM for BIOS memory.
ROMs

• The 27XXX series of the EPROM includes:
  o 2704 (512 × 8).
  o 2708 (1K × 8).
  o 2716 (2K × 8).
  o 2732 (4K × 8).
  o 2764 (8K × 8).
  o 27128 (16K × 8).
  o 27256 (32K × 8).
  o 27512 (64K × 8).
  o 271024 (128K × 8).

• Each EPROM has:
  o Address pins
  o 8 data connections
  o One or more selection inputs and one output enable pin.
Intel 2716 EPROM (2K X 8)

**Intel 2716 EPROM (2K X 8):**

- V_{PP} is used to program the device by applying 25V and pulsing PGM while holding CS high.

**Pin(s) | Function**
---|---
A_{0}-A_{10} | Address
PD/PGM | Power down/Program
CS | Chip Select
O_{0}-O_{7} | Outputs

**Address Inputs**

**Data Outputs**

- Chip Select
- PWR Down
- Prog Logic

- Output Buffers
- Y-Gating
- 16,384 Cell Matrix
Intel 2716 EPROM (2K X 8)

2716 Timing diagram:

Read Mode (PD/PGM = V\text{IL})

Sample of the data sheet for the 2716 A.C. Characteristics.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Test Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{ACC1}</td>
<td>Addr. to Output Delay</td>
<td>250</td>
<td>450</td>
<td>ns</td>
</tr>
<tr>
<td>t_{OH}</td>
<td>Addr. to Output Hold</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{DF}</td>
<td>Chip Deselect to Output Float</td>
<td>0</td>
<td>100</td>
<td>ns</td>
</tr>
</tbody>
</table>

This EPROM requires a wait state for use with the 8086 (460\text{ns} constraint).
SRAM

- SRAMs are virtually identical to the EPROM with respect to the pinout although access time is faster (250ns).
- SRAMs used for caches have access times as low as 10ns.

**TI TMS 4016 SRAM (2K X 8):**

<table>
<thead>
<tr>
<th>Pin(s)</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&lt;sub&gt;0&lt;/sub&gt;-A&lt;sub&gt;10&lt;/sub&gt;</td>
<td>Address</td>
</tr>
<tr>
<td>DQ&lt;sub&gt;0&lt;/sub&gt;-DQ&lt;sub&gt;7&lt;/sub&gt;</td>
<td>Data In/Data Out</td>
</tr>
<tr>
<td>S (CS)</td>
<td>Chip Select</td>
</tr>
<tr>
<td>G (OE)</td>
<td>Read Enable</td>
</tr>
<tr>
<td>W (WE)</td>
<td>Write Enable</td>
</tr>
</tbody>
</table>
DRAM

- SRAMs are limited in size (up to about 128K X 8).
- DRAMs are available in much larger sizes, e.g., 64M X 1.
- DRAMs MUST be refreshed (rewritten) every 2 to 4 ms. Since they store their value on an integrated capacitor that loses charge over time.
- This refresh is performed by a special circuit in the DRAM which refreshes the entire memory.
- Refresh also occurs on a normal read or write.
- The large storage capacity of DRAMs make it impractical to add the required number of address pins.
- Instead, the address pins are multiplexed.
TI TMS4464 DRAM (64K X 4)

- The TMS4464 can store a total of 256K bits of data.
- It has 64K addressable locations which means it needs 16 address inputs, but it has only 8.
- The row address (A₀ through A₇) are placed on the address pins and strobed into a set of internal latches.
- The column address (A₈ through A₁₅) is then strobed in using CAS.
TI TMS4464 DRAM (64K X 4)

Pin(s)          Function
-------------------------
A₀-A₇          Address
DQ₀-DQ₃        Data In/Data Out
RAS            Row Address Strobe
CAS            Column Address Strobe
G              Output Enable
W              Write Enable
TI TMS4464 DRAM (64K X 4)

Timing Diagram:

**RAS**

**CAS**

Row      Column      **Dont care**

CAS also performs the function of the chip select input.

Address BUS

A₀ A₁ A₂ A₉ A₁₀ A₃ A₁₁

74157 (2-to-1MUX)

1A 1B 2A 2B 3A 3B 4A 4B

Inputs to DRAM

A₀ A₁ A₂ A₃

S

1Y 2Y 3Y 4Y

A₀ A₁ A₂ A₃

S

74157 (2-to-1MUX)

A₄ A₁₂ A₅ A₁₃ A₁₄ A₆ A₇ A₁₅

0: latch A to Y
1: latch B to Y
• Larger DRAMs are available which are organized as 1M X 1, 4M X 1, 16M X 1, 64M X 1, 256M X 1.
• DRAMs are typically placed on SIMM (Single In-line Memory Modules) or DIMM (Dual In-line Memory Modules) boards.
Memory Address Decoding

• The processor can usually address a memory space that is much larger than the memory space covered by an individual memory chip.
• In order to splice a memory device into the address space of the processor, decoding is necessary.
• For example, the 8088 issues 20-bit addresses for a total of 1MB of memory address space.
• However, the BIOS on a 2716 EPROM has only 2KB of memory and 11 address pins.
• A decoder can be used to decode the additional 9 address pins and allow the EPROM to be placed in any 2KB section of the 1MB address space.
NAND Decoder Example

Address Bus:

- A_19
- A_18
- A_17
- A_16
- A_15
- A_14
- A_13
- A_12
- A_11

Data Bus:

- A_0
- A_1:
- A_10:
- O_0
- O_1
- O_7

2716 (2K X 8) EPROM

CS

RD of 8088/86 or MRDC bus signal.

Logic 0 when A_11 through A_19 are all 1.
NAND Decoder Example

- To determine the address range that a device is mapped into

- NAND gate decoders are not often used
  - Large fan-in NAND gates are not efficient
  - Multiple NAND gate IC’s might be required to perform such decoding
  - Rather the 3-to-8 Line Decoder (74LS138) is more common.
The 3-to-8 Line Decoder (74LS138)
Sample Decoder Circuit

A_{13} through A_{15} select a 2764
A_{16} through A_{19} enable the decoder

Address Bus

74LS138

2764
(8K X 8)
EPROM

Data Bus

A_0
A_{10}
A_{12}
...
A_7

Address space
F0000H-FFFFFH

RD of 8088/86

The EPROMs cover a 64KB section of memory.
Dual 2-to-4 Line Decoder

- 74LS139 is a dual 2-to-4 line decoder
Programmable Decoder

• Programmable Logic Devices (PLDs) can be used as a decoder

• PLDs come in three varieties:
  o PLA (Programmable Logic Array)
  o PAL (Programmable Array Logic)
  o GAL (Gated Array Logic)

• PLDs have been around since the mid-1970s but have only recently appeared in memory systems (PALs have replaced PROM address decoders).

• PALs and PLAs can be fuse-programmed (like the PROM) or erasable (like the EPROM).
AMD 16L8 PAL decoder

Example

Programmed to decode address lines A_{19} - A_{13} onto 8 outputs.

\[ \text{Equations:} \]
\[\begin{align*}
/O1 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot /A15 \cdot /A14 \cdot /A13 \\
/O2 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot /A15 \cdot /A14 \cdot A13 \\
/O3 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot /A15 \cdot A14 \cdot /A13 \\
/O4 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot /A15 \cdot A14 \cdot A13 \\
/O5 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot A15 \cdot /A14 \cdot /A13 \\
/O6 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot A15 \cdot /A14 \cdot A13 \\
/O7 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot A15 \cdot A14 \cdot /A13 \\
/O8 &= A19 \cdot A18 \cdot A17 \cdot A16 \cdot A15 \cdot A14 \cdot A13
\end{align*}\]
8088 and 80188 (8-bit) Memory Interface

- The memory system "sees" the 8088 as a device with:
  - 20 address connections (A\textsubscript{19} to A\textsubscript{0}).
  - 8 data bus connections (AD\textsubscript{7} to AD\textsubscript{0}).
  - 3 control signals: IO/\overline{M}, \overline{RD}, and \overline{WR}.

- We'll present examples of the 8088 interfacing with:
  - 32K of EPROM (at addresses F8000H through FFFFFFFH).
  - 512K of SRAM (at addresses 00000H through 7FFFFFFH).
8088 and 80188 (8-bit) EPROM Memory Interface Example

- The EPROM interface uses a 74LS138 (3-to-8 line decoder) plus 8 2732 (4K X 8) EPROMs.
- The EPROM will also require the generation of a wait state.
  - The EPROM has an access time of 450ns.
  - The 74LS138 requires 12ns to decode.
  - The 8088 runs at 5MHz and only allows 460ns for memory to access data.
  - A wait state adds 200ns of additional time
8088 and 80188 (8-bit) EPROM Memory Interface Example

To wait state generator

WAIT

\[ \begin{align*}
A_{12} & \\
A_{13} & \\
A_{14} & \\
\text{IO/M} & \\
A_{15} & \\
A_{16} & \\
A_{17} & \\
A_{18} & \\
A_{19} & \end{align*} \]

74LS138

\[ \begin{align*}
A & \\
B & \\
C & \end{align*} \]

\[ \begin{align*}
0 & \\
1 & \\
2 & \\
3 & \\
4 & \\
5 & \\
6 & \\
7 & \end{align*} \]

G2A

G2B

G1

Address Bus

Data Bus

Address space

F8000H-FFFFFH
8088 and 80188 (8-bit) SRAM Memory Interface Example
8086, 80186, 80286 and 80386

Memory Interface

• These machines differ from the 8088/80188 in several ways:
  o The data bus is 16-bitswide.
  o The IO/$\overline{M}$ pin is replaced with M/$\overline{IO}$ (8086/80186) and $\overline{MRDC}$ and $\overline{MWTC}$ for 80286 and 80386SX.
  o $\overline{BHE}$, Bus High Enable, control signal is added.
  o Address pin $A_0$ (or $\overline{BLE}$, Bus Low Enable) is used differently.

• The 16-bit data bus presents a new problem:
  o The microprocessor must be able to read and write data to any 16-bit location in addition to any 8-bit location.
8086, 80186, 80286 and 80386 Memory Interface

- The data bus and memory are divided into banks:

<table>
<thead>
<tr>
<th>High bank</th>
<th>Low bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFD</td>
<td>FFFFFFE</td>
</tr>
<tr>
<td>FFFFFFD</td>
<td>FFFFFFC</td>
</tr>
<tr>
<td>8 bits</td>
<td>8 bits</td>
</tr>
<tr>
<td>Odd bytes</td>
<td>Even bytes</td>
</tr>
<tr>
<td>8 MB</td>
<td>8 MB</td>
</tr>
</tbody>
</table>

- BHE and BLE are used to select one or both:

<table>
<thead>
<tr>
<th>BHE</th>
<th>BLE</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Both banks enabled for 16-bit transfer</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>High bank enabled for an 8-bit transfer</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Low bank enabled for an 8-bit transfer</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>No banks selected</td>
</tr>
</tbody>
</table>
Bank selection can be accomplished in two ways:
  o Separate write decoders for each bank (which drive \(\overline{CS}\)).
  o A separate write signal (strobe) to each bank (which drive \(\overline{WE}\)).

Note that 8-bit read requests in this scheme are handled by the microprocessor (it selects the bits it wants to read from the 16-bits on the bus).

It does not seem to be a big difference between these methods.

Note in either method that \(A_0\) does not connect to memory and bus wire \(A_1\) connects to memory pin \(A_0, A_2\) to \(A_1,\) etc.
80386SX 16-bit Memory Interface Example (Separate Decoders)
80386DX and 80486 Memory Interface

• 80386DX and 80486 have 32-bit data buses and therefore 4 banks of memory.
• 32-bit, 16-bit, and 8-bit transfers are accomplished by different combinations of the bank selection signals $BE3$, $BE2$, $BE1$, $BE0$.
• The Address bits $A_0$ and $A_1$ are used within the microprocessor to generate these signals.
• They are don't cares in the decoding of the 32-bit address outside the chip (using a PLD such as the PAL 16L8).
• The high clock rates of these processors usually require wait states for memory access.
Pentium Memory Interface

- The Pentium, Pentium Pro, Pentium II and III contain a 64-bit data bus.
- Therefore, 8 decoders or 8 write strobes are needed as well as 8 memory banks.
- The write strobes are obtained by combining the bank enable signals ($BE_x$) with the $MWTC$ signal.
- $MWTC$ is generated by combining the $M/IO$ and $W/R$ signals.
Pentium Memory Interface
Pentium Memory Interface Example
Pentium Memory Interface Example

- In order to map the previous memory interface into address space FFF80000H-FFFFFFFFH

Use a 16L8 to do the \( \overline{WR0} - \overline{WR7} \) decoding using \( \overline{MWTC} \) and \( \overline{BE0} - \overline{BE7} \).
Error Detection and Correction in Memory Devices

• Memory devices use error detection and correction methods to detect and correct memory storing errors.

• Error detection and correction methods attach extra bits to data strings which can help in error detection and correction.

• In this section we will discuss:
  o Parity checking
  o Checksum tests
  o Cyclic Redundancy Check
  o Hamming code error correction
Parity Checking

- Parity checking is used to detect single bit errors in the memory.
- The current trend is away from parity checking.
- Parity checking adds 1 bit for every 8 data bits.
  - For EVEN parity, the 9th bit is set to yield an even number of 1's in all 9 bits.
  - For ODD parity, the 9th bit is set to make this number odd.
- For 72-pin SIMMs, the number of data bits is 32 + 4 = 36 (4 parity bits).
Parity for Memory Error Detection

74AS280 Parity Generator/Checker

9-bit parity generator/checker

<table>
<thead>
<tr>
<th>Number of inputs A thru I that are HIGH</th>
<th>Outputs EVEN</th>
<th>ODD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 2, 4, 6, 8</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>1, 3, 5, 7, 9</td>
<td>L</td>
<td>H</td>
</tr>
</tbody>
</table>
Parity for Memory Error Detection

• This circuit generates EVEN or ODD parity for the 9-bit number placed on its inputs.
• Typically, for generation, the 9\textsuperscript{th} input bit is set to 0.
• This circuit also checks EVEN or ODD parity for the 9-bit number.
• In this case, the 9\textsuperscript{th} input bit is connected to the 9\textsuperscript{th} bit of memory.
• For example, if the original byte has an even # of 1's (with 9\textsuperscript{th} bit at GND), the parity bit is set to 1 (from the EVEN output).
• If the EVEN output goes high during the check, then an error occurred.
Parity for Memory Error Detection
Checksum Error Detection

• This parity scheme can only detect a single bit error.
• Block-Check Character (BCC) or Checksum can detect multiple bit errors.
• This is simply the two's complement sum (the negative of the sum) of the sequence of bytes.
• No error occurred if adding the data values and the checksum produces a 0.
**Checksum Error Detection**

- For example

  Given 4 hex data bytes: \(10, 23, 45, 04\)

<table>
<thead>
<tr>
<th>Compute the sum:</th>
<th>Invert and add 1 to get checksum byte:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0111 1100 + 1</td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>1000 0011 + 1</td>
</tr>
<tr>
<td>04</td>
<td>1000 0100 = (84H)</td>
</tr>
<tr>
<td>(7C)</td>
<td>(_{10})</td>
</tr>
</tbody>
</table>

  Check is made by adding and checking for 00 (discard the carry):

<table>
<thead>
<tr>
<th>10</th>
<th>23</th>
<th>45</th>
<th>04</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>23</td>
<td>45</td>
<td>04</td>
</tr>
<tr>
<td>84</td>
<td>04</td>
<td>84</td>
<td>45</td>
</tr>
<tr>
<td>(_{100})</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- If 45 changes to 44 AND 04 changes to 05, the error is missed.
CRC Error Detection

• Cyclic Redundancy Check (CRC) is commonly used to check data transfers in hardware such as hard drives.
• Treats data as a stream of serial data n-bits long.
• The bits are treated as coefficients of a characteristic polynomial, $M(X)$ of the form:

$$M(X) = b_n + b_{n-1}X + b_{n-2}X^2 + \ldots + b_1X^{n-1} + b_0X^n$$

where $b_0$ is the least significant bit while $b_n$ is the most significant bit.
CRC Error Detection

• For example

For the 16-bit data stream: 26F0H = 0010 0110 1111 0000

\[ M(X) = 0 + 0X^1 + 1X^2 + 0X^3 + 0X^4 + 1X^5 + 1X^6 + 0X^7 + 1X^8 + 1X^9 + 1X^{10} + 1X^{11} + 0X^{12} + 0X^{13} + 0X^{14} + 0X^{15} \]

\[ M(X) = 1X^2 + 1X^5 + 1X^6 + 1X^8 + 1X^9 + 1X^{10} + 1X^{11} \]

• The CRC is found by applying the following equation

\[ CRC = \frac{M(X)X^n}{G(X)} = Q(X) + R(X) \]

Q(X) is the quotient
R(X) is the remainder

• G(X) is the called the generator polynomial and has special properties.
CRC Error Detection

- A commonly used polynomial is:

\[ G(X) = X^{16} + X^{15} + X^2 + 1 \]

- The remainder \( R(X) \) is appended to the data block.
- When the CRC and \( R(X) \) is computed by the receiver, \( R(X) \) should be zero.
- Since \( G(X) \) is of power 16, the remainder, \( R(X) \), cannot be of order higher than 15.
- Therefore, no more than 2 bytes are needed independent of the data block size.
CRC Error Detection Example

For the data stream 26F0H:

\[
\begin{align*}
\frac{M(X)X^{16}}{G(X)} &= \frac{X^{27} + X^{26} + X^{25} + X^{24} + X^{22} + X^{21} + X^{18}}{X^{16} + X^{15} + X^2 + 1} \\
&= X^{16} + X^{15} + X^2 + 1 \\
&= X^{27} + X^{26} + X^{25} + X^{24} + X^{22} + X^{21} + X^{18} \\
&= X^{27} + X^{26} + X^{13} + X^{11} \\
&= X^{25} + X^{24} + X^{22} + X^{21} + X^{18} + X^{13} + X^{11} \\
&= X^{25} + X^{24} + X^{13} + X^9 \\
&= X^{22} + X^{21} + X^{18} + X^{13} + X^9 \\
&= X^{22} + X^{21} + X^{18} + X^{13} + X^9 + X^8 + X^6 \\
&= X^{18} + X^{17} + X^{13} + X^9 + X^8 + X^6 \\
&= X^{17} + X^{16} + X^{13} + X^9 + X^8 + X^6 + X^4 + X^2 \\
&= X^{17} + X^{16} + X^3 + X \\
&= \ldots
\end{align*}
\]

Final Solution is:

\[
R(X) = X^{15} + X^{13} + X^9 + X^8 + X^6 + X^4 + X^3 + X + 1
\]

Value appended is the reverse coefficient value 1101 1010 1100 0101 = DAC5H
Error Correction

- Parity, BCC and CRC are only mechanisms for error detection.
- The system is halted if an error is found in memory.
- Error correction is starting to show up in new systems.
- SDRAM has ECC (Error Correction Code).
- Correction will allow the system to continue its operation.
- If two errors occur, they can be detected but not corrected.
- Error correction will of course cost more in terms of extra bits.
Hamming Codes

• Error correction is based on Hamming Codes.
• There is lots of theory here but our focus will be on implementation.
• The objective is to correct single bit errors in an 8-bit data byte.
• We need 4 parity bits to correct single bit errors.
• Note that the parity bits are at bit positions that are powers of 2.
• The data bits of the byte are labeled $X_3$, $X_5$, $X_6$, $X_7$, $X_9$, $X_{10}$, $X_{11}$ and $X_{12}$.
• The parity bits are labeled $P_1$, $P_2$, $P_4$ and $P_8$. 
Calculating the Hamming Code

The key to the Hamming Code is the use of extra parity bits to allow the identification of a single error. Create the code word as follows:

• Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16, etc)
• All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)
• Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.  
  Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1, 3, 5, 7, 9, 11, 13, 15,...)
  Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2, 3, 6, 7, 10, 11, 15,...)
  Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, ...)
  Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15, 24-31, 40-47,...)
• Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.
Hamming Code Example

P1 is generated by computing the parity of X₃, X₅, X₇, X₉, X₁₁, X₁₃, X₁₅. These numbers have a 1 in bit position 1 of the subscript in binary.

Note that each data bit is used in the parity computation of at least 2 P bits.

Given data byte: 11010010

P₁ uses blue bits: 12111097653
P₁ even parity is 1.

P₂ uses brown bits: 12111097653
P₂ even parity is 1.

P₃ uses cyan bits: 12111097653
P₃ even parity is 0.

P₄ uses purple bits: 12111097653
P₄ even parity is 1.

P₁ is assigned even parity using X₃, X₅, X₇, X₉, X₁₁, X₁₃, X₁₅

P₂ is assigned even parity using X₃, X₆, X₇, X₁₀, X₁₁, X₁₄, X₁₅

P₃ is assigned even parity using X₅, X₆, X₇, X₁₂, X₁₃, X₁₄, X₁₅

P₄ is assigned even parity using X₉, X₁₀, X₁₁, X₁₂, X₁₃, X₁₄, X₁₅

Not used since we are correcting byte data.
Hamming Code Example

Parity encoded data: 
110110010011

If $X_{10}$ flips from 0 -> 1, then the check gives the location of the bit error as:

<table>
<thead>
<tr>
<th>P</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$P_1$ even parity is 0.
$P_2$ even parity is now 1.
$P_3$ even parity is 0.
$P_4$ even parity is now 1.

Since these are NOT 0, there was an error.

The position of the bit flip is given by:

$P_4P_3P_2P_1$, which is 1010 or 10 decimal.
Parity for Memory Error Correction

- The 74LS636 corrects errors by storing 5 parity bits with each byte of data.
- The pinout consists of: 8 data I/O pins, 5 check bit I/O pins, 2 control pins, 2 error outputs (Single error flag (SEF), Double error flag (DEF)).