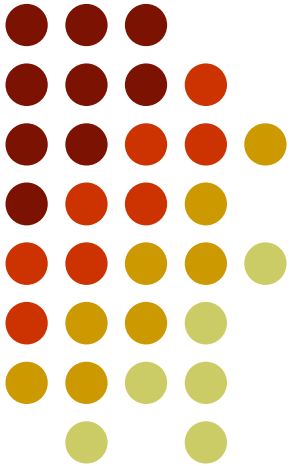
Error Detection and Correction for High Integrity Memory Systems

- Limitations to Dram Reliability.
  - Causes of Dram Failure.

- Simple Error Detection Schemes – Parity.
- Simple Error Detection and Correction Schemes - Hamming Codes.

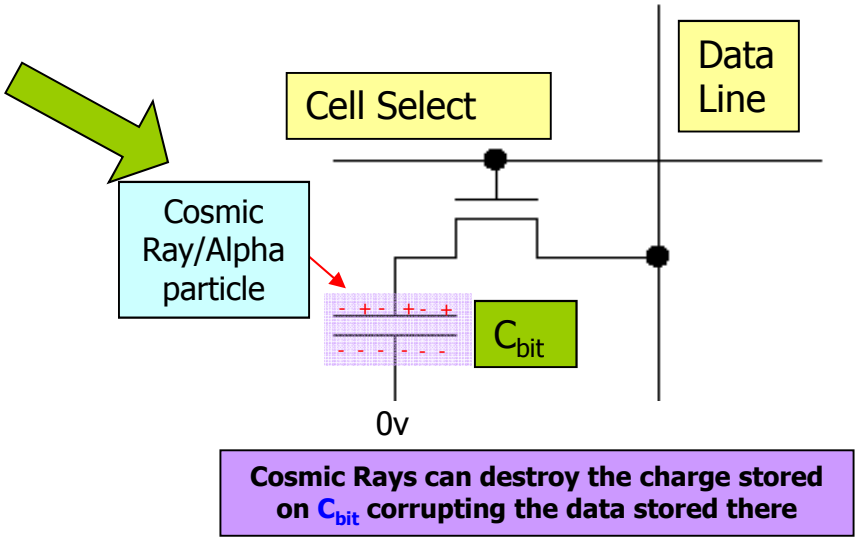
  - Single Bit Error Detect, Single Bit Error Correct Codes (SED/SEC).
  - Double Bit Error Detect, Single Bit Error Correct Codes (DED/SEC).
  - Implementation in Memory Systems.



Error Detection and Correction in Memory Systems

Transient or Soft Errors in Dram Cells

- We know that Dram is the main type of memory used in large, high availability, high reliability computer system (*e.g. servers*).
- However, a significant problem with DRAM Cells is that they are not 100% reliable.
- DRAM data is delicate, because it is maintained as a minute charge stored on a capacitor. Consequently it doesn't take much to upset that delicate balance between a logic 1 and a logic 0.
- This makes Drams susceptible to 'transient' or 'soft' errors where the data can become corrupted by unforeseen external influences (*nothing to do with failing to refresh them or any bad design for example*).
- Such errors can occur as the result of external radiation – Cosmic Rays etc. These are called soft errors as they do not permanently damage the chip, writing to it "fixes" the data.



Error Detection and Correction in Memory Systems

- Soft errors can be a real problem in high availability, high integrity memory systems, not just from an operational point of view (*i.e. the memory fails and the system may need rebooting*) but because they occur at unpredictable times and the failure is hard to replicate and pinpoint.

  The hardware and software engineers often disappear on a "wild goose chase" trying to track down a bug in the software or the Dram controller which often isn't the cause of the problem.

  (*How many times do we blame Microsoft when our PC falls over?*)

- Swapping dram chips doesn't improve matters as the faults are not down to a physical failure of the chip, i.e. it *not* due to wear and tear or manufacturing defects.

- Rather the failure is attributable to the physics of the capacitors used in the devices and is thus an inherent problem that users of Dram have to live with.

- Dram manufacturers have taken steps to "harden" their Drams to shield them better, but the move to higher density and lower voltages has seen a rise in soft error rates for **all** memory technologies: Sram and Dram (as well as multi-level cells in Flash memory)

Error Detection and Correction in Memory Systems

- In your average Home PC, soft errors in Dram memory systems are not a *significant* problem, because data/programs etc. tend not to be left lying around for long enough for the errors to occur or accumulate to the point where they become a problem.

  This is because the memory in your home PC is forever being loaded with new data/programs, so soft errors, if they do arise, tend to be overwritten (*and hence eliminated*). Likewise we often shut down/reboot our PC when it is inactive.

  Note: those who put their PC into standby mode when it's not in use are likely to notice the problem most, followed by those who hibernate their PC (*which saves and reloads any corrupted data to disk*).

- In high availability systems, where servers remain running 24/7 and require 99.9999% availability, it can be a significant problem.

- A system with 1 GByte of RAM can expect an error every two weeks; a hypothetical 1TByte system would experience a soft error every few minutes. (http://www.tezzaron.com/about/papers/soft_errors_1_1_secure.pdf)

Error Detection and Correction in Memory Systems

What are the Causes of Dram Soft Errors?

- Mostly these are caused by two things:-

  - A cell being struck by a stray Cosmic Ray from outer space Ionizing and corrupting the delicate charge stored there. Alpha particles emerging from the material packaging of the chip.

  - A cell being sensitive to patterns of charge stored on neighbouring cells, i.e. the charges on neighbouring cells can all interact (or leak into each other) and cause a bit to 'flip' unintentionally.

  (*these were things that the first generation of Drams were very susceptible too until manufacturers learned to improve fabrication techniques and make them more resilient*).

Note: *Read/write errors are also becoming an issue in Flash memory cells, particularly where we attempt to store more than 1 bit per cell, for the same reasons above, Error Detection and Correction may be needed here also.*

Error Detection and Correction in Memory Systems

How do we deal with these problems?

1. A Memory Controller *could* Detect faulty data *read* from memory and inform the CPU via some kind of hardware 'exception'.
   Verdict : Simple and cheap.

   i.e. the dram controller does not try to correct the error, but simply informs the Operating System (via a CPU exception) that the access is unsafe. The application making the access will most likely be shut down or the system rebooted.

2. Detect the error and attempt to correct it "on-the-fly" in hardware so that the system can continue operation.
   Verdict: More complex and expensive.

- The choice boils down to cost and how important the integrity and availability of your system is.

Error Detection and Correction in Memory Systems

Error Detection Schemes (*no correction*) – Single Bit Parity Checking

- Single Bit Parity schemes have ben around for years and are able to detect the occurrence of single bit errors (and in fact all errors where an odd number of bit errors have occurred) in each memory location, but such a simple scheme cannot *correct* any of the errors.

- Single Bit Parity schemes utilise an additional single bit of storage for each and every memory location in memory and is thus relatively cheap and easy to implement.

- In the case of a byte wide memory system, such a scheme would require 9 bits of storage for each and every memory location, i.e. 8 bits of data + 1 bit of parity per location.

Error Detection and Correction in Memory Systems

Simple (i.e. cheap) Parity Schemes for error detection (*not* correction)

- With a simple Parity scheme, each byte, word or whatever width of data the memory system is organised as, is "tagged" with an additional single Parity bit before it is written to memory.

- This parity bit could be designed to be *even* or *odd parity.*

- That is to say, that the total number of logic 1's in the stored word, including the parity bit itself, is either an even or an odd number.

- With an Even Parity scheme, the parity bit is set to a value that would result in the total number of '1's stored in the memory location being an even number.

- With an Odd Parity scheme, the parity bit is set to a value that would result in the total number of '1's stored in the memory location being an odd number.

Error Detection and Correction in Memory Systems

Parity Bit Generation

- To generate the Parity bit P for an **even** parity scheme, we simply add the data bits together ignoring any carry.
- This turns out to be the same as simply Exclusive-OR'ing the data bits $b_0 - b_7$.
- The resulting parity bit is stored alongside the data in that location.
- Odd parity is simply the inverse of Even parity i.e. the EX-NOR of bits $b_0 - b_7$.

EXOR Gate Truth Table	
A B	Parity Bit
0 0	0
0 1	1
1 0	1
1 1	0

Result is a 1 when number of 1's in A and B is ODD

EX-NOR Truth Table	
A B	Parity Bit
0 0	1
0 1	0
1 0	0
1 1	1

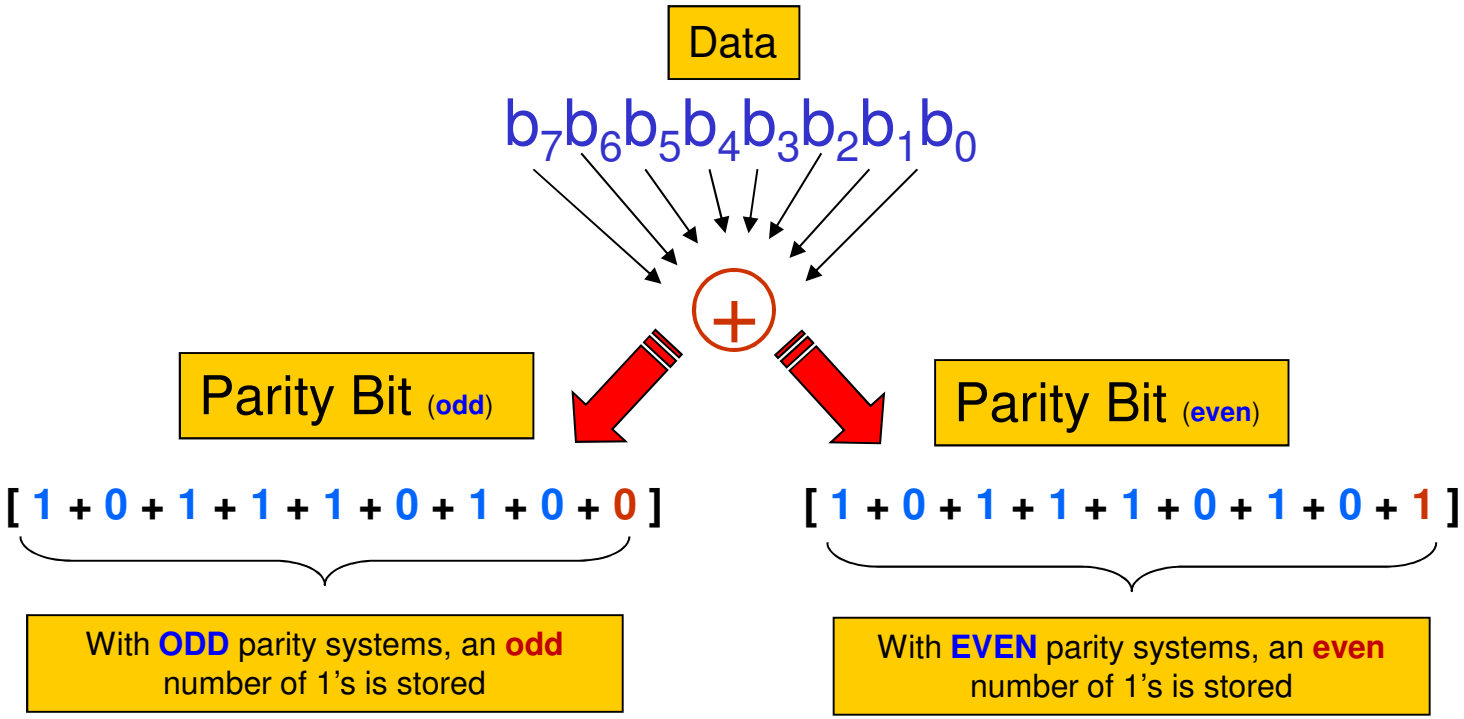
Result is a 1 when number of 1's in A and B is EVEN

Error Detection and Correction in Memory Systems

Example 8 bit Data : [10111010] (Number of '1's in data = 5)

- For an even parity scheme the parity bit P will be set to 1, resulting in the stored word (*data plus parity*) containing 6 logic '1's, i.e. an even number of logic 1's.

- For an odd parity scheme, P will be set to 0, resulting in the stored word containing 5 logic '1's i.e. an odd number of logic '1's



Error Detection and Correction in Memory Systems

Problem

- Generate the value of the Even Parity bit for the following two items of 12 Bit Data:-

  [101101001110]

  [010110010101]

Two Solutions:

- Add up the data bits in binary (*ignoring the carry*) and the result is the value of the Parity Bit stored with the data.

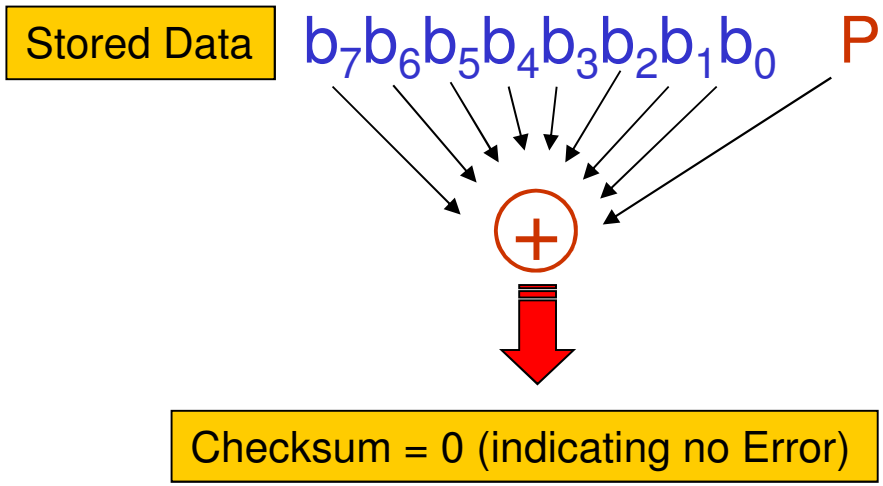
- Alternatively, count the number of 1's in the data bits and assign a value to the Parity Bit such that the total number of 1's stored to an even number.

Error Detection and Correction in Memory Systems

Detecting an Error During a Read Operation

- In effect what we are doing with an EVEN parity scheme is introducing an extra, redundant bit of data such that when the whole word (8 bit data plus 1 parity bit) is read back and all the bits (*data plus parity*) are Added (i.e. EX-OR'ed) together, the result (*called the checksum*) should always be 0.

- If any single bit of data (*or even just the parity bit itself*) has been corrupted, the calculated Checksum will be 1 (*i.e. the checksum bit will be **flipped***) and it is this that allows us to detect an error during a read operation. In fact any number of odd bit errors will result in a Checksum = 1, but an even number of bit errors (2,4,6 etc.) will not be detected at all since it causes a double flipping of the Checksum bit which cancels each other out. This is a limitation of the simple parity scheme.



[1 + 0 + 1 + 1 + 1 + 0 + 1 + 0 + 1] = 0

EX-OR'ing the stored data and Parity results in 0 if **NO** errors (*or an even number of errors*) have occurred.

Checksum

Error Detection and Correction in Memory Systems

Example of Error Detection with Parity

- Suppose the following 9 bit data word i.e. 8 bit data + Even Parity Bit were read from memory. [**10111010** | **1**].

- EX-OR' ing these 9 bits together gives a checksum = logic 0, meaning (*for an even parity scheme*) that the data was read correctly *(or possibly that an even number of errors had occurred)*.

- Suppose instead that we read back the following 9 bit data word [**1011101**1 | **1**] where the data bit $\mathbf{b_0}$ has been corrupted. The result of EX-OR' ing (*adding with no carry*) these 9 bits together to produce the checksum, would result in a logic 1, i.e. an error.

- Suppose instead that we read back the following 9 bit data word [**1011100**1 | **1**] where the data bits $\mathbf{b_0}$ and $\mathbf{b_1}$ have been corrupted. The result of EX-OR' ing these 9 bits together would be a logic 0, i.e. no error for an even parity scheme.

- Note that even with single bit errors, this simple 1-bit parity scheme cannot identify the *location* of the erroneous bit and thus it cannot be used to as a means of correcting the faulty bit.

Error Detection and Correction in Memory Systems

Error Detection and Correction Schemes – Hamming Codes

- Depending upon the chosen level of sophistication, Hamming Codes will enable a memory controller to detect and correct single-bit errors within each memory location.

  Such a scheme may well impose significant additional storage requirements on our system leading to increased cost and complexity.

  For example, by storing an additional 4 bits of hamming code, alongside each and every byte of data, i.e. making a 12 bit wide memory system instead of 8 bit wide, we would be able to detect and correct **all** single bit errors arising within that location, including errors in the hamming code bits themselves.

  The more sophisticated the hamming code scheme, the more erroneous bits can be detected and corrected but at a tradeoff of increased memory cost and a more sophisticated memory controller (*how much is increased reliability worth?*)

Error Detection and Correction in Memory Systems

Error Correcting Techniques: Hamming Codes

- Basic Error Correcting techniques in memory systems borrow concepts from Digital Communications theory.
- The idea is based on Forward Error Codes (F.E.C.s), where multiple Parity bits are encoded into a data stream and transmitted as a Code Word. That is, a code word consisting of [Data Bits + Multiple Parity Bits]
- F.E.C.'s are essentially multiple parity bits, carefully chosen so that they can identify the location of any single bit error (data or parity) in the resultant code word. Hardware *could* use these parity bits to identify and correct bit errors.
- The table opposite highlights the number of parity bits required for a specified data width to enable single bit error detection and single bit error correction (i.e. SED/SEC). We can use this idea in memory systems.
- As you can see this scheme becomes more cost effective with wider data widths.

Data bits	# of Parity Bits
4	3
8	4
12	5
16	5
20	5
24	5
28	6
32	6
64	7
etc	

1 extra byte wide memory chip required to protect 8 bytes of Data.

Error Detection and Correction in Memory Systems

- In communications, where the data is transmitted serially, the extra Parity Bits were transmitted in bit positions corresponding to powers of 2, i.e. bit positions 1, 2, 4, 8, 16, 32 etc. depending on data.

Code Word Bit Position						
7	6	5	4	3	2	1
d_4	d_3	d_2	p_3	d_1	p_2	p_1

- The significance of this, as well as counting from position 1 (*not 0*) will become apparent shortly.

- Using an even parity scheme, as shown below, (where $\oplus$ means EX-OR or "add with no carry"), we generate the values for the parity bits p_1, p_2 and p_3 (assuming just for 4 bits of data) according to these equations.

- This will ensure that during a read, EX-OR'ing the data and the parity bits together in the same way will produce three checksums all = 0.

Each Checksum Bit = 0

$p_1 = d_1 \oplus d_2 \oplus d_4$	i.e. during a read	$p_1 \oplus d_1 \oplus d_2 \oplus d_4 = 0$
$p_2 = d_1 \oplus d_3 \oplus d_4$	i.e. during a read	$p_2 \oplus d_1 \oplus d_3 \oplus d_4 = 0$
$p_3 = d_2 \oplus d_3 \oplus d_4$	i.e. during a read	$p_3 \oplus d_2 \oplus d_3 \oplus d_4 = 0$

Error Detection and Correction in Memory Systems

7	6	5	4	3	2	1
d_4	d_3	d_2	p_3	d_1	p_2	p_1

The data in positions 3, 6, 7 when written in binary have a '1' in bit position 1

The data in positions 3, 5, 7 when written in binary have a '1' in bit position 0

How do we determine the Parity Bit Equations?

- Write out in binary the position numbers occupied by all data bits, i.e. data bits occupying positions 3, 5, 6 and 7, i.e.

d1 = Position 3 = 011
d2 = Position 5 = 101
d3 = Position 6 = 110
d4 = Position 7 = 111

- Identify all those position numbers where there is a '1' in the least significant bit position (*right most column*). These indicate the positions of the data bits that have to be EX-OR'ed to produce parity bit p1.

- Likewise parity bit p2 will be formed by 'EX-OR' ing the data bits occupying positions where there is a 1 in the next l.s.b. of their position number.

011 = 3
101 = 5
111 = 7
} p_1 : d_1, d_2, d_4

011 = 3
110 = 6
111 = 7
} p_2 : d_1, d_3, d_4

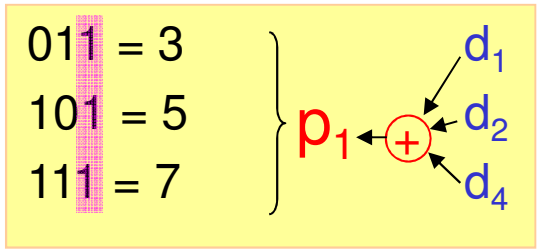
101 = 5
110 = 6
111 = 7
} p_3 : d_2, d_3, d_4

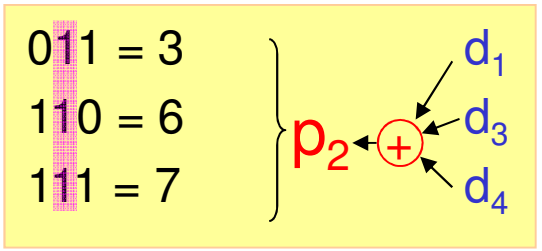
The data in positions 5, 6, 7 when written in binary have a '1' in bit position 2

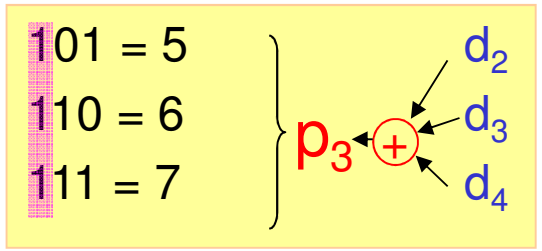
Error Detection and Correction in Memory Systems

7	6	5	4	3	2	1
d_4	d_3	d_2	p_3	d_1	p_2	p_1

- Parity bit p1 is thus formed from the exclusive OR of data bits occupying positions 3, 5 & 7, i.e. the EX-OR of data bits d1, d2 and d4.
- Parity bit p2 is formed from the exclusive OR of data bits occupying positions 3, 6 & 7. i.e. data bits d1, d3 and d4
- Parity bit p3 is formed from the exclusive OR of data bits occupying positions 5, 6 & 7. i.e. data bits d2, d3 and d4
- When we read back the data and the previously stored parity bits and EX-OR them all together (*like we did for the simple parity scheme*) to create a 3 bit checksum, the checksum will be [000] if there were no errors.







Error Detection and Correction in Memory Systems

Problem 1

- How many Parity Bits would be required to enable correction of a single bit error in a 6 bit data word? What positions would the parity and data bits occupy in the code word?

Solution

- First construct the code word as follows placing parity bits at positions in the code word corresponding to powers of two, until you have placed all data bits.

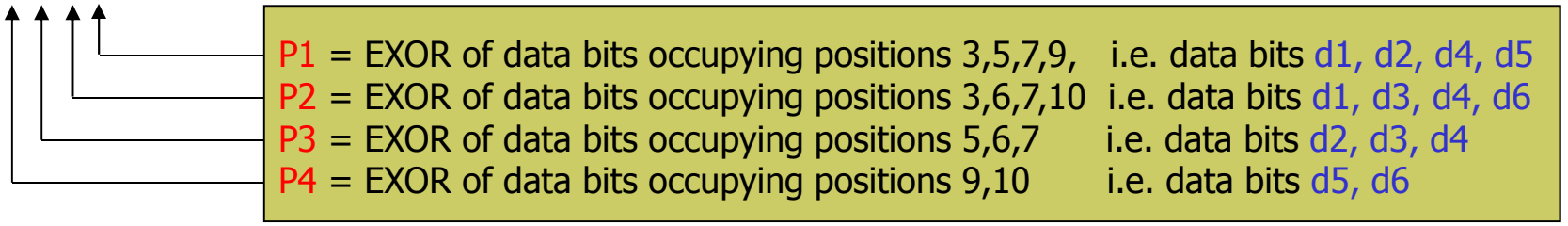
Error Detection and Correction in Memory Systems

Problem 2

- Write down the parity equations for a 6 bit data word.

- First write out in binary the position numbers occupied by the Data Bits in the code word, then look for 1's in the columns.

3	0011	d1
5	0101	d2
6	0110	d3
7	0111	d4
9	1001	d5
10	1010	d6



Error Detection and Correction in Memory Systems

Example Problem using 4 bit Data

- Take the 4 bit data word d_4, d_3, d_2, d_1 = [1011]. We calculate the parity bits according to the equations developed earlier.

$$p_1 = d_1 \oplus d_2 \oplus d_4$$
$$p_2 = d_1 \oplus d_3 \oplus d_4$$
$$p_3 = d_2 \oplus d_3 \oplus d_4$$

- We generate each parity bit thus

$$p_1 = 1 \oplus 1 \oplus 1 \qquad \text{i.e. } p_1 = 1$$
$$p_2 = 1 \oplus 0 \oplus 1 \qquad \text{i.e. } p_2 = 0$$
$$p_3 = 1 \oplus 0 \oplus 1 \qquad \text{i.e. } p_3 = 0$$

- These 3 parity bits are stored such that the resultant code word (*data plus parity*) would be

$$d_4\ d_3\ d_2\ p_3\ d_1\ p_2\ p_1 = [1\ 0\ 1\ 0\ 1\ 0\ 1]$$

Error Detection and Correction in Memory Systems

How do we Check for an Error during Reading?

- We calculate a set of check bits (*the checksum*) using the original equations that were used to create the parity bits in the first instance. For example, assuming the correctly generated code word given earlier (shown below).

$d_4\ \ d_3\ \ d_2\ \ p_3\ \ d_1\ \ p_2\ \ p_1 = [1\ 0\ 1\ 0\ 1\ 0\ 1]$

- We calculate check bit C_1 as the EX-OR of **p1** plus the data bits that were originally used to generate **p1** in the first place, ditto for check bits C_2, C_3 etc. Thus

$$C_1 = p_1 \oplus d_1 \oplus d_2 \oplus d_4 \qquad C_1 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$
$$C_2 = p_2 \oplus d_1 \oplus d_3 \oplus d_4 \qquad C_2 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$$
$$C_3 = p_3 \oplus d_2 \oplus d_3 \oplus d_4 \qquad C_3 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$$

Each Checksum bit will be 0 when there are no errors

- If the Data and Parity Bits (*i.e. the code word stored*) were read correctly then each of the resultant check bits will be 0, (*as we can see above, indicating no corruption of data or parity bits has taken place*).

Error Detection and Correction in Memory Systems

- Suppose an error did occur some time in the future and that instead of reading back the code word that was stored (*repeated again below*)

$d_4\ d_3\ d_2\ p_3\ d_1\ p_2\ p_1 = [1\ 0\ 1\ 0\ 1\ 0\ 1]$

we actually read back a code word with an error in d_3 (*i.e. bit position 6 in the code word*) thus we actually read back :-

$d_4\ d_3\ d_2\ p_3\ d_1\ p_2\ p_1 = [1\ 1\ 1\ 0\ 1\ 0\ 1]$

Error in bit position 6 causing a '0' to be read back as a '1'

Error Detection and Correction in Memory Systems

- To detect the error and locate its position, we calculate the set of check bits by EX-OR'ing the data bits we read back with the parity bits according to the original encoding equations.

$$C_1 = p_1 \oplus d_1 \oplus d_2 \oplus d_4$$
$$C_2 = p_2 \oplus d_1 \oplus d_3 \oplus d_4$$
$$C_3 = p_3 \oplus d_2 \oplus d_3 \oplus d_4$$

- However, in our case we end up with the following check sum bits

$$C_1 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$
$$C_2 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$
$$C_3 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

Bit position 6 [110] meaning d_3 is wrong

- This just so happens to identify the bit position which was in error and thus it can be flipped/inverted in hardware 'on-the-fly' during the read to correct it.

Error Detection and Correction in Memory Systems

Class Exercise

- The following 6 bit data is to be stored [101101]
- Calculate the Parity bits and construct the code word for this data.
- We know from the previous example that 4 parity bits are needed for 6 bits of data in accordance with the equation and code word format below

P_1 = EXOR of data bits occupying positions?
P_2 = EXOR of data bits occupying positions?
P_3 = EXOR of data bits occupying positions?
P_4 = EXOR of data bits occupying positions?

10	9	8	7	6	5	4	3	2	1
d_6	d_5	p_4	d_4	d_3	d_2	p_3	d_1	p_2	p_1

d_6	d_5	p_4	d_4	d_3	d_2	p_3	d_1	p_2	p_1
1	0	?	1	1	0	?	1	?	?

Error Detection and Correction in Memory Systems

Solution

d_6	d_5	p_4	d_4	d_3	d_2	p_3	d_1	p_2	p_1
1	0	1	1	1	0	0	1	0	0

Problem 3

- Now introduce an error into data bit d_4 (bit position 7) and calculate the check bits based on what is read back. Can you detect where the error is?

d_6	d_5	p_4	d_4	d_3	d_2	p_3	d_1	p_2	p_1
1	0	1	0	1	0	0	1	0	0

Error Detection and Correction in Memory Systems

How does it work?

- Take a look at the 4 bit data check sum equations shown again below

$$C_1 = p_1 \oplus d_1 \oplus d_2 \oplus d_4$$
$$C_2 = p_2 \oplus d_1 \oplus d_3 \oplus d_4$$
$$C_3 = p_3 \oplus d_2 \oplus d_3 \oplus d_4$$

7	6	5	4	3	2	1
d_4	d_3	d_2	p_3	d_1	p_2	p_1

- From this, we see that if any single bit is incorrectly read, it will invert those check bits for which the corrupted bit is part of the equation.

- For example, if **d_2** were corrupted, then check bits **C_3** and **C_1** would be inverted to logic 1 and **C_2** would remain logic 0, therefore the check bits would be {101} identifying bit position 5 (i.e. **d_2**) as the faulty bit.

Error Detection and Correction in Memory Systems

- The apparent 'magic' with this scheme is in choosing the Parity equations such that the faulty bit affects the check bits in just such a way as to identify the source of the faulty bit.

- For example, if we are going to detect a fault in data bit d_4 (*code word position 7*), then we have to ensure that all three check bits get flipped to 1 (to give a checksum = [111] i.e. 7_{10}) to identify the error in position 7 i.e d_4.

  This is why d_4 is part of the parity equations for all three check bits as an error in d_4 will flip those check bits to logic 1.

- Likewise an error in d_1 (code word position 3) would have to generate the checksum [011] to identify its location, that is why it is part of the parity equations for check bits C_1 and C_2 and not C_3.

Error Detection and Correction in Memory Systems

What if the Error lies with one of the Parity Bits?

- If one of the parity bits is erroneous then it only affects one check bit (*see below*) and hence it is only able to produce the check sum values [001], [010] or [100] corresponding to bit positions 1, 2 or 4 which corresponds exactly to the positions occupied by the parity bits in the code word (*i.e. positions that are powers of 2*).

$$C_1 = p_1 \oplus d_1 \oplus d_2 \oplus d_4$$
$$C_2 = p_2 \oplus d_1 \oplus d_3 \oplus d_4$$
$$C_3 = p_3 \oplus d_2 \oplus d_3 \oplus d_4$$

An error in a parity bit flips exactly one check bit leading to checksum values of [001], [010] or [100], positions 1,2 or 4 where the parity bits are stored.

- This is why the data and parity bits are stored in the position/order they are and why the bit positions in the code word start off at position 1 (*as a check sum of 0 means no error and thus 0 cannot be used as an erroneous position number*).

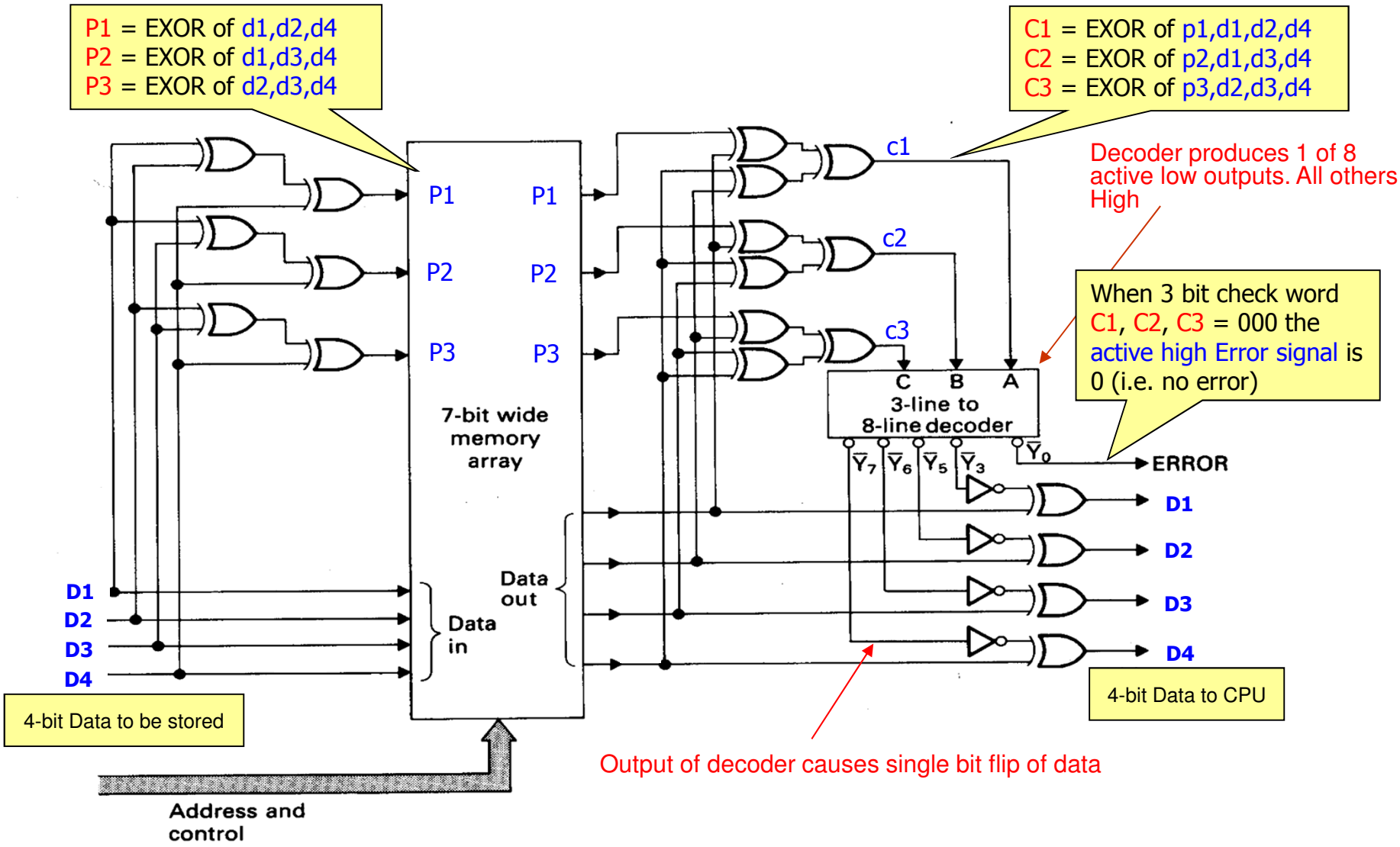
Error Detection and Correction in Memory Systems

- What happens to the check bits if you get a double bit error?
- Obviously, the 3 check bits can only identify one position number so there is no way it could be used to identify the position of 2 incorrect bits.
- However, look at the equations again, what happens if a double bit error were to occur in say d1 and d2 (positions 3 and 5)?

$$C_1 = p_1 \oplus d_1 \oplus d_2 \oplus d_4$$
$$C_2 = p_2 \oplus d_1 \oplus d_3 \oplus d_4$$
$$C_3 = p_3 \oplus d_2 \oplus d_3 \oplus d_4$$

Error Detection and Correction in Memory Systems

- **A Simple Check bit generator plus Error Detector and Corrector Circuit for 4 bit data + 3 bit Parity (normally this circuitry would be integrated into an ECC Dram controller)**



Error Detection and Correction in Memory Systems

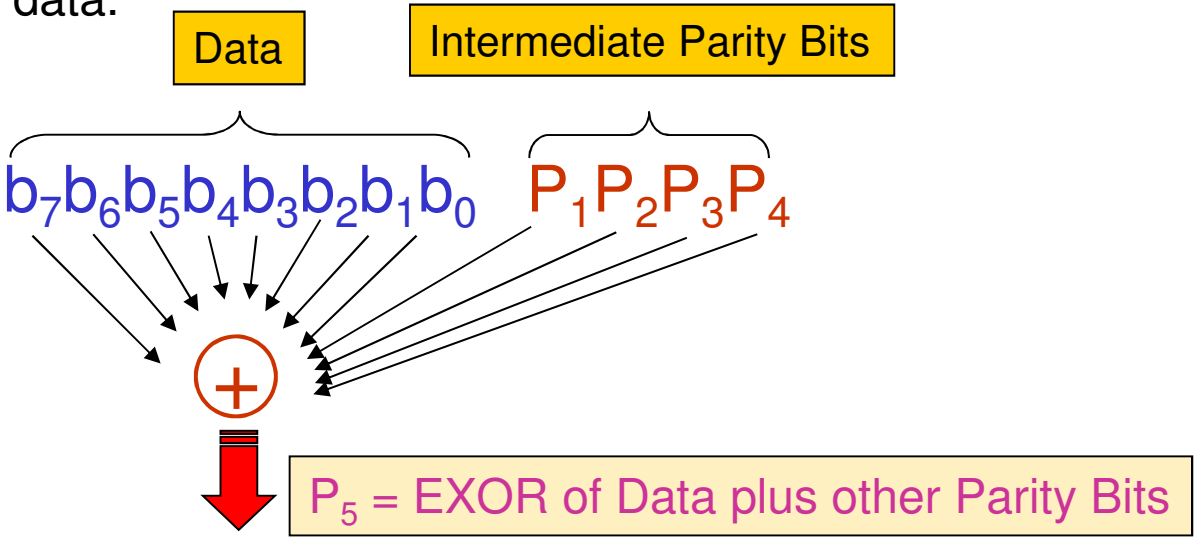
Enhanced Error Correction Schemes

- With the simple addition of an extra parity bit we could also enhance this scheme to include the detection of double bit errors (*but only with the ability to correct single bit errors i.e. DED/SEC*).

- The extra Parity bit is generated by Exclusive OR'ing the original data bits and the intermediate parity bits calculated as previously. We can see this below for 8 bit data.

Data bits	# of Check Bits
4	3
8	4
12	5
16	5
20	5
24	5
28	6
32	6
64	7
etc	

4 bits of hamming code could detect and correct single bit errors in 8 bit wide data



Error Detection and Correction in Memory Systems

Checking and Correcting Errors

- During reading, identical hardware to that used to generate the 5 original parity bits is then used to re-create 5 new check bits (*the checksum*) based on the 8 bits of data and 5 bits of parity actually read from memory.

13	12	11	10	9	8	7	6	5	4	3	2	1	Bit Position
p_5	d_8	d_7	d_6	d_5	p_4	d_4	d_3	d_2	p_3	d_1	p_2	p_1	Parity Bit

- That is, each calculated check bit (C_1-C_4) is formed from EX-OR'ing each parity bit (P_1-P_4) read from memory with the data bits that were used to generate the parity bits in the first place, exactly as before. The result will be a 4 bit checksum. If all is well, this 4 bit checksum should be [0000]

- The fifth check bit (C_5) is calculated as the EX-OR of the 8 data bits plus the 5 parity bits READ BACK FROM MEMORY it should also be 0 when no error has occurred.

- If a single bit error occurs reading back original data + parity, then the value for the 5th check bit C_5 will **always** be flipped to 1 regardless of where that error lies.

- However if a double bit error had occurred during the read, the check bit C_5 will be double flipped i.e. still 0 but the other check bits will be non-zero.

Error Detection and Correction in Memory Systems

Interpreting the results

- If check bit C_5 is 0 and the 4 bit check code C_1- C_4 is 0000, then no errors have occurred and no correction is required.

- If check bit C_5 is 0 and the 4 bit check code C_1- C_4 is **not** 0, then a double bit, non-correctable error has occurred (i.e. no bits can be corrected). In this case the check code cannot be used to identify the location of any of the errors (*i.e. it must be ignored*)

- If check bit C_5 is 1 then a single bit error has occurred in which case, if the 4 bit check code C_1- C_4 is 0000 then the error has occurred only in Parity Bit p_5, the data itself is correct and no correction is required.

  However if the 4 bit check code C_1- C_4 is not 0000, then a single bit error has occurred in the bit position indicated by C_1- C_4 and it can be corrected, this includes the parity bits P_1- P_4.

Error Detection and Correction in Memory Systems

Example DED/SEC Parity Encoding with a 4 bit Data Word

- Take the 4 bit data word d_4, d_3, d_2, d_1 = [1011]. We calculate the parity bits according to the equations developed earlier.

$$p_1 = d_1 \oplus d_2 \oplus d_4$$
$$p_2 = d_1 \oplus d_3 \oplus d_4$$
$$p_3 = d_2 \oplus d_3 \oplus d_4$$

- We generate each parity bit thus

$$p_1 = 1 \oplus 1 \oplus 1 \qquad \text{i.e. } p_1 = 1$$
$$p_2 = 1 \oplus 0 \oplus 1 \qquad \text{i.e. } p_2 = 0$$
$$p_3 = 1 \oplus 0 \oplus 1 \qquad \text{i.e. } p_3 = 0$$

Extra Parity bit p_4 = EX-OR($p_1, p_2, p_3, d_1, d_2, d_3, d_4$) = 0, Thus code word is

$$p_4\ d_4\ d_3\ d_2\ p_3\ d_1\ p_2\ p_1 = [\,0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\,]$$

Error Detection and Correction in Memory Systems

- Let's introduce a single bit error into parity bit p4 so that we read back the code word below :-

$\underline{p_4}$	d_4	d_3	d_2	p_3	d_1	p_2	p_1	= [$\underline{1}$ 1 0 1 0 1 0 1]

- The only bit affected by a change in p4 will be C_4 because no other check bits use p4 in their equation, thus the computed value of the checksum bit C_4 will be

C_4 = EX-OR(p_1, p_2, p_3, p_4, d_1, d_2, d_3, d_4) = 1

The remaining checksum bits C_1- C_3 will be 000, indicating that the error is just in p4 and the data was read correctly.

Error Detection and Correction in Memory Systems

- Let's introduce a single bit error into data bit d4.

p_4	d_4	d_3	d_2	p_3	d_1	p_2	p_1	= [0	0	0	1	0	1	0	1]

- This single bit error will cause C_4 to be flipped to logic 1 since

C_4 = EX-OR(p_1, p_2, p_3, p_4, d_1, d_2, d_3, d_4) = 1

however, d4 is part of the equations for the other 3 checksum/parity bits hence C_1, C_2, C_3 = {111}. Hence we have a single bit correctable error in position 7, i.e. d4.

Error Detection and Correction in Memory Systems

- Let's introduce a double bit error : i.e. d_4 and p_1

p_4	d_4	d_3	d_2	p_3	d_1	p_2	p_1
0	0	0	1	0	1	0	0

- This double bit error will cause C_4 to be logic 0 due to 2 bit flips since

C_4 = EX-OR(p_1, p_2, p_3, p_4, d_1, d_2, d_3, d_4) = 0

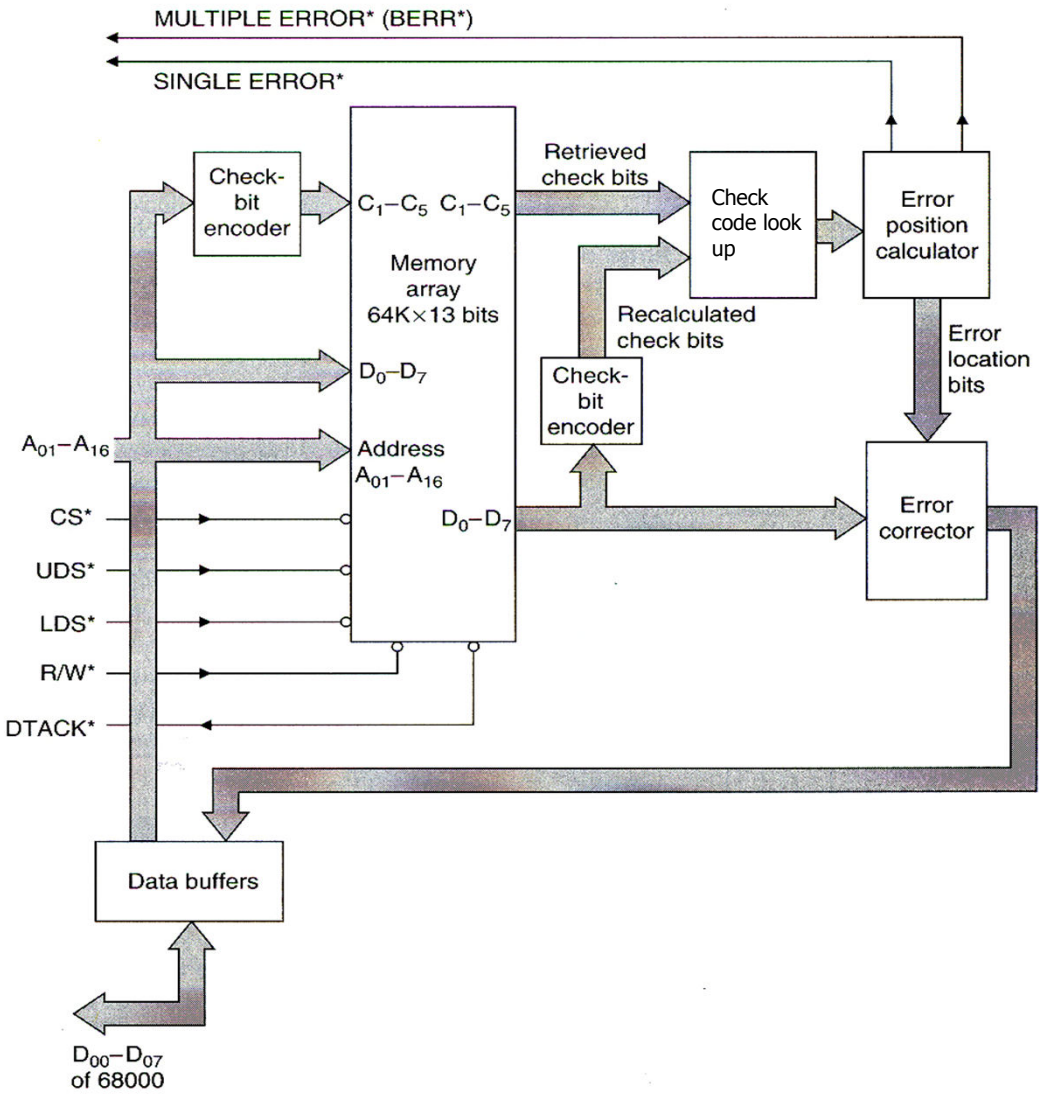
however, d4 (position 7) is part of the equations for the other 3 checksum/parity bits, thus the other 3 checksum bits should in theory indicate {111} highlighting position 7 in the data stream as the source of the corruption, but because p1 (the 2nd faulty bit) is also incorrect, C_1 will be double flipped so position of error is read as {110}, incorrectly identifying position 6 as the source of the error.

However this does not matter since our circuitry will detect this is a double bit non-correctable error (C_4 = 0, $C_3 - C_1$ = non-zero).

Error Detection and Correction in Memory Systems

Example SEC/DED ECC Circuit

- The illustration opposite shows the block diagram of a typical error detection and correction circuit for a byte wide memory array.
- This circuit (taken from Clements) shows the Parity bits labelled as C_1-C_5.
- What about protecting 16 bit data. What are the issues here?
- What is good about using 64 bit wide data as in the PC?



Error Detection and Correction in Memory Systems

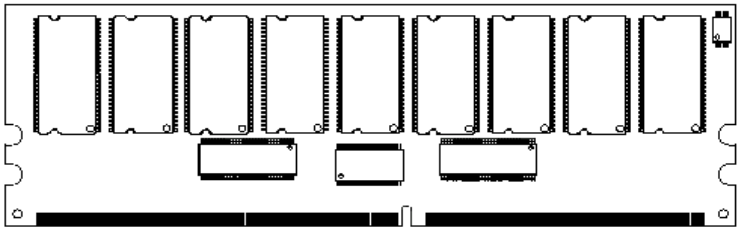
A Typical Micron Memory Module with support for SED/SEC ECC: 64 data bits, 8 Check Bits

Features

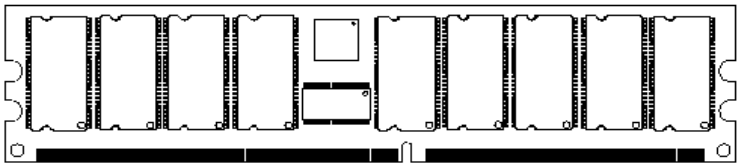
- 184-pin, dual in-line memory modules (DIMM)
- Fast data transfer rates; PC1600 or PC2100
- Utilizes 200 MT/s and 266 MT/s DDR SDRAM stacked components
- ECC, 1-bit error detection and correction
- Registered inputs with one-clock delay
- Phase-lock loop (PLL) clock driver to reduce loading
- 1GB (128 Meg x 72), 2GB (256 Meg x 72), and 4GB (128 Meg x 72)
- Vdd= VddQ = +2.5V
- Vddspd = +2.3V to +3.6V
- 2.5V I/O (SSTL_2 compatible)
- Commands entered on each positive CK edge
- DQS edge-aligned with data for READs; center-aligned with data for WRITEs
- Internal, pipelined double data rate (DDR) architecture; two data accesses per clock cycle
- Bidirectional data strobe (DQS) transmitted/ received with data, i.e., source-synchronous data capture
- Differential clock inputs (CK and CK#)
- Four internal device banks for concurrent operation
- Selectable burst lengths: 2, 4, or 8
- Auto Refresh and Self Refresh Modes
- 7.8125µs maximum average periodic refresh interval
- Serial Presence Detect (SPD) with EEPROM
- Selectable READ CAS latency
- Gold edge contacts

Figure 1: 184-Pin DIMM (MO-206)

Standard PCB (1.7in./43.18mm))



Low Profile PCB (1.2in./30.48mm)



OPTIONS	MARKING
• Package	
184-pin DIMM (Standard)	G
184-pin DIMM (Lead-free)[1]	Y
• Memory Clock/Speed, CAS Latency[2]	
7.5ns (133 Mhz), 266 MT/s, CL = 2	-262
7.5ns (133 Mhz), 266 MT/s, CL = 2	-26A
7.5ns (133 Mhz), 266 MT/s, CL = 2.5	-265
10ns (100 Mhz), 200 MT/s, CL = 2	-202
• Circuit Board	
Standard (1.7in./43.18mm)	See note, page 2
Low Profile (1.2in./30.48mm)	See note, page 2

ECC used with Sram

- Due to the increased density, reduced feature size and lower operating voltages of Sram cells, Sram technology is also becoming increasingly susceptible to the effects of radiation and electrical noise.
- More and more manufacturers are offering ECC, at least as an option, for many of their SRAM chips such as the ISSI Part #**IS61WV51216EDALL** - 512K x 16 SRam (with ECC), as used more recently on the DE2.
- You can see much of what has been discussed in this lecture reflected in this block diagram taken from their data sheet. The ECC circuitry is entirely built into the chip.