COA

Chapter 05: | Error Correction

Prof. Dr. Mostafa Elhosseini

https://youtube.com/drmelhosseini

AGENDA

- Error Correction
  - Parity Code
  - Cyclic Redundancy Check
  - Hamming Code

Error Correction

- A **semiconductor memory** system is **subject** to errors. These can be categorized as
  - Hard failures and
  - Soft errors

Error Correction

- Most modern main memory systems include **logic** for both **detecting** and **correcting** errors
  - **Width** of memory word is increased
  - Additional bits are **parity bits**
  - **Number of parity bits** required depends on the **level** of detection and correction needed

Error Correction

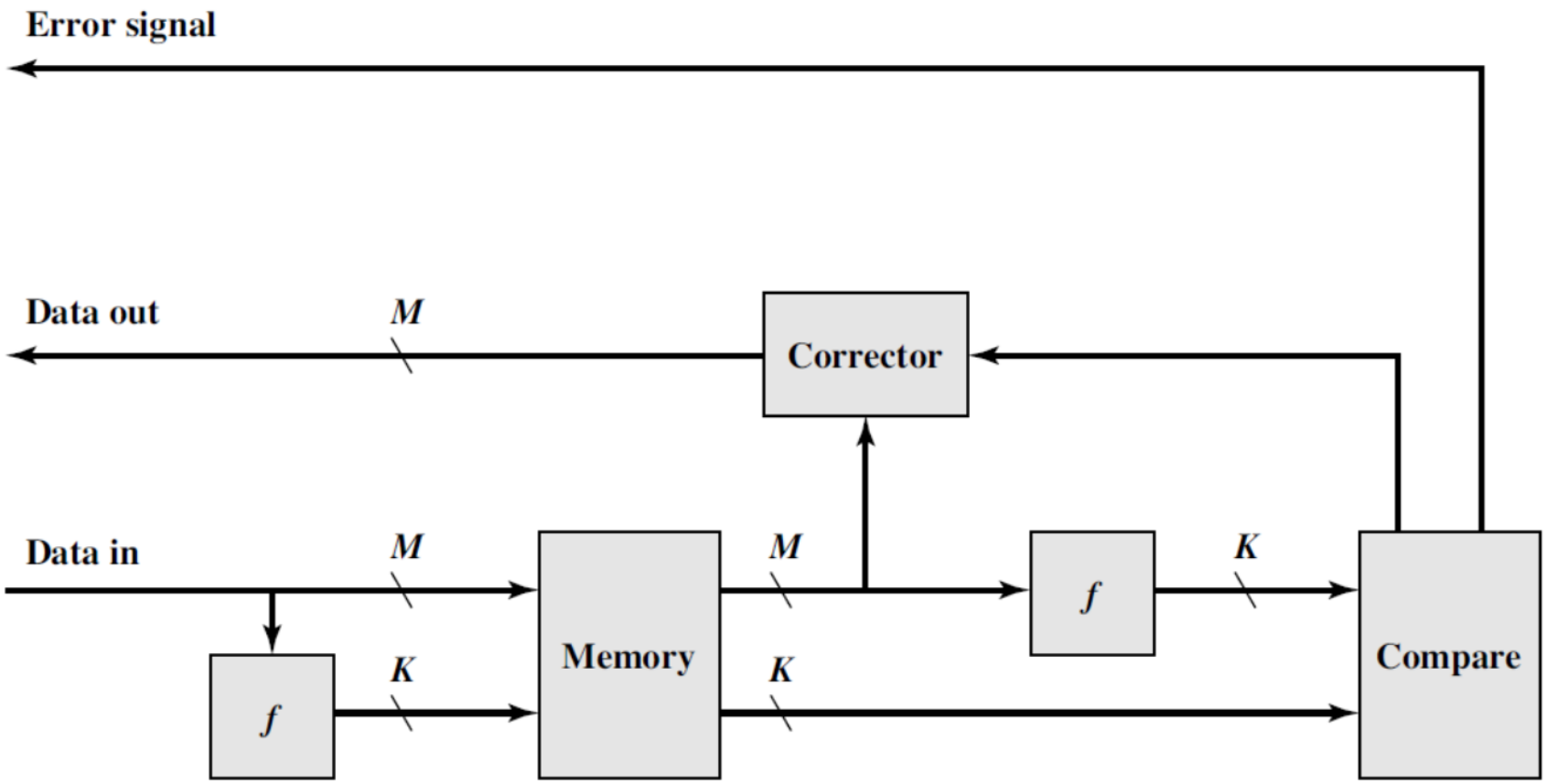

Figure 5.7 Error-Correcting Code Function

Error-correcting codes

- **Codes** that operate in this fashion are referred to as **error-correcting codes**.
- A code is **characterized** by the **number of bit errors** in a word that it can **correct** and **detect**
- Three methods for **adding bits to codes** to detect a single-bit error are:
  - Parity Method
  - Cyclic Redundancy Check (CRC)
  - **Hamming Code**

Parity Bit

- A **Parity Bit** is attached to a group of bits to make the total number of 1s in a group always **even** or always **odd**

The BCD code with parity bits.

Even Parity		Odd Parity	
P	BCD	*P*	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

Cyclic Redundancy Check (CRC)

- The cyclic redundancy check (**CRC**) is a widely used code used for detecting **one- and two-bit transmission** errors
- In CRC, a certain **number of check bits**, sometimes called a **checksum**, are appended to the data bits
- Not every possible error can be identified, but the **CRC is much more efficient than just a simple parity check**

Hamming Distance

- The simplest of the error-correcting codes is the **Hamming code** devised by **Richard Hamming** at Bell Laboratories
- The **Hamming code** is used to **detect and correct** a single-bit error in a transmitted code.
  - Ex. : To accomplish this, **four redundancy bits** are introduced in a **7-bit group of data bits**. **These redundancy bits** are interspersed **at bit positions 2^n** (n = 0, 1, 2, 3) within the original data bits

Error-correcting codes

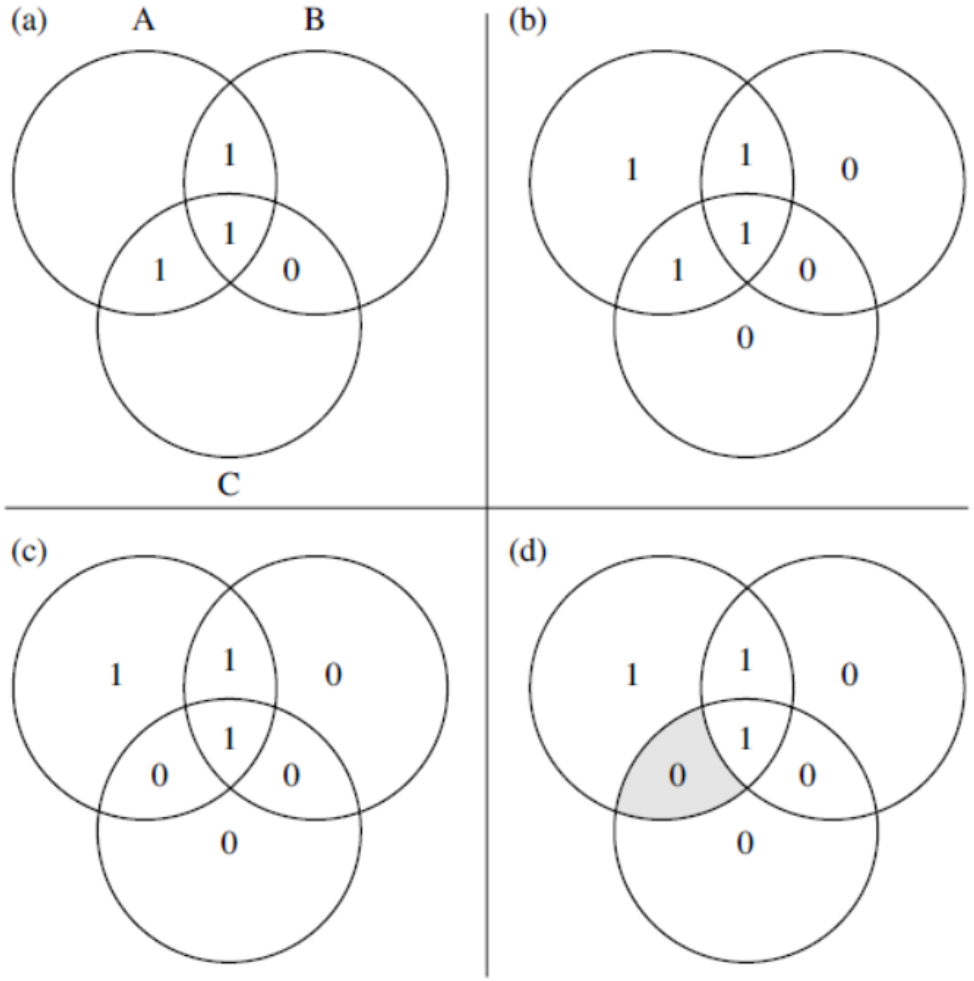

Figure 5.8 Hamming Error-Correcting Code

Single-Error Correction

- Let us determine **how long the code** must be (K = ?)
- the **comparison** logic receives as input **two K-bit values**.
- A bit-by-bit comparison is done by taking the exclusive-OR of the two inputs. The result is called the **syndrome** word
- The **syndrome** word is therefore **K** bits wide and has a range between 0 and 2^K-1.
- The **value 0** indicates that **no error** was detected, leaving **2^K-1** values to indicate, if there is an error, which bit was in error

Single-Error Correction

- because an error could occur on any of the **M data** bits or **K check** bits, we must have

$$2^k - 1 \geq M + K$$

- This inequality gives the **number of bits** needed to correct a single bit error in a word containing **M** data bits
- $k = 3$: $2^3 - 1 \geq 8 + 3$ ×
- $\boldsymbol{k = 4}$: $2^4 - 1 \geq 8 + 4$ √

Single-Error Correction

	Single-Error Correction		Single-Error Correction/ Double-Error Detection	
Data Bits	**Check Bits**	**% Increase**	**Check Bits**	**% Increase**
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91

Single-Error Correction

- For convenience, we would like to generate a 4-bit syndrome for an 8-bit data word with the following characteristics:
  - syndrome contains **all 0s**.
  - syndrome contains **one and only one** bit set to 1.
  - syndrome contains **more than one bit set to 1**.

Single-Error Correction

- The **bit positions** are numbered from **1 to 12**.
- Those **bit positions** whose position numbers are **powers of** 2 are designated as **check bits**

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1

Single-Error Correction

- Each **check bit** operates on every data bit whose **position number contains a 1** in the **same bit position** as the position number of that **check bit**
- Data bit positions **3, 5, 7, 9, and 11** (**D1, D2, D4, D5, D7**) all contain a 1 in the least significant bit of their position **number as does C1**

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1

Single-Error Correction

$$
\begin{aligned}
C1 &= D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \\
C2 &= D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \\
C4 &= D2 \oplus D3 \oplus D4 \oplus D8 \\
C8 &= D5 \oplus D6 \oplus D7 \oplus D8
\end{aligned}
$$

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1

Example

- Assume that the **8-bit input** word is **00111001**, with data bit **D1** in the **rightmost** position
- Suppose now that **data bit 3** sustains an error and is changed from 0 to 1

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$
$$C2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$
$$C4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$
$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$
$$C2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$
$$C4 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$
$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

	C8	C4	C2	C1
	0	1	1	1
$\oplus$	0	0	0	1
	0	1	1	0

Single-error-correcting (SEC) code

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1
Word fetched as	0	0	1	1	0	1	1	0	1	1	1	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Check bit					0				0		0	1

- More commonly, a **single-error-correcting**, **double-error-detecting** (SEC-DED)
  - require **one additional bit** compared with SEC codes

SEC-DED

- An **error-correcting code** enhances the **reliability** of the memory at the **cost of added com**plexity.
- With a 1-bit-per-chip organization, an SEC-DED code is generally considered adequate.
  - **IBM 30xx implementations** used an **8-bit SECDED code** for each **64 bits** of data in main memory. Thus, the size of main memory is actually about **12% larger than** is apparent to the user.
  - The **VAX computers** used a **7-bit SEC-DED** for each **32 bits** of memory, for a **22% overhead**.
  - A number of contemporary DRAMs use **9 check bits** for each **128 bits** of data, for a **7% overhead**