

Discussion 13: ECC/Parity, RAID, I/O

Hamming ECC

Recall the basic structure of a Hamming code. Given bits $1, \dots, m$, the bit at position $2n$ is parity for all the bits with a 1 in position n . For example, the first bit is chosen such that the sum of all odd-numbered bits is even.

Bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	<u>P1</u>	<u>P2</u>	D1	<u>P4</u>	D2	D3	D4	<u>P8</u>	D5	D6	D7	D8	D9	D10	D11
P1	X		X		X		X		X		X		X		X
P2		X	X			X	X			X	X			X	X
P4				X	X	X	X					X	X	X	X
P8								X	X	X	X	X	X	X	X

- i. How many bits do we need to add to 0011_2 to allow single error correction?
Parity Bits: 3
- ii. Which locations in 0011_2 would parity bits be included?
Using P for parity bits: $PP0P011_2$
- iii. Which bits does each parity bit cover in 0011_2 ?
Parity bit #1: 1, 3, 5, 7
Parity bit #2: 2, 3, 6, 7
Parity bit #3: 4, 5, 6, 7
- iv. Write the completed coded representation for 0011_2 to enable single error correction.
1000011₂
- v. How can we enable an additional double error detection on top of this?
Add an additional parity bit over the entire sequence.
- vi. Find the original bits given the following SEC Hamming Code: 0110111_2
Parity group 1: error
Parity group 2: okay
Parity group 4: error
Incorrect bit: $1 + 4 = 5$, change bit 5 from 1 to 0: 0110011_2
0110011₂ \rightarrow 1011_2

vii. Find the original bits given the following SEC Hamming Code: 1001000_2

Parity group 1: error

Parity group 2: okay

Parity group 4: error

Incorrect bit: $1 + 4 = 5$, change bit 5 from 1 to 0: 1001100_2

$\underline{1001100}_2 \rightarrow 0100_2$

viii. Find the original bits given the following SEC Hamming Code: 010011010000110_2

Parity group 1: okay

Parity group 2: error

Parity group 4: okay

Parity group 8: error

Incorrect bit: $2 + 8 = 10$, change bit 10 from 0 to 1: 0100110100110_2

$\underline{0100110100110}_2 \rightarrow 01100100110_2$

RAID

Fill out the following table:

	Configuration	Pro / Good for...	Con / Bad for...
RAID 0	Data disks without check information	No overhead Fast read / write	Reliability
RAID 1	Mirrored Disks: Extra copy of disks	Fast read / write Fast recovery	High overhead → Expensive
RAID 2	Hamming ECC: One check disk per parity group	Smaller overhead	Redundant check disks
RAID 3	Single check disk for error correction (Disk controllers can detect failures) Transfer units are spread over all disks in a group (bit interleaving)	Smallest check information overhead	Read all data disks for small accesses to detect errors
RAID 4	Transfer units = a sector within a single disk. Errors are detected within a single transfer unit Can handle independent reads/writes per disks	Higher throughput of small reads	Still slow small writes (A single check disk is a bottleneck)
RAID 5	Check information is distributed across all disks in a group.	Higher throughput of small writes	

Small accesses = an access to a single disk in a group

I/O

1. Fill this table of polling and interrupts.

Operation	Definition	Pro/Good for	Con
Polling	Forces the hardware to wait on ready bit (alternatively, if timing of device is known – the ready bit can be polled at the frequency of the device).	- Low Latency - Low overhead when data is available - Good For: devices that are always busy or when you can't make progress until the device replies	- Can't do anything else while polling - Can't sleep while polling (CPU always at full speed)
Interrupts	Hardware fires an "exception" when it becomes ready. CPU changes PC to execute code in the interrupt handler when this occurs.	PRO: - Can do useful work while waiting for response - Can wait on many things at once - Good for: Devices that take a long time to respond, especially if you can do other work while waiting.	-nondeterministic when interrupt occurs -interrupt handler has some overhead (e.g. saves all registers, flush pipeline, etc.) - Higher latency/event - Worse throughput

2. Memory Mapped I/O

Certain memory addresses correspond to registers in I/O devices and not normal memory.

0xFFFF0000 – Receiver Control:

LSB is the ready bit, there may be other bits set that we don't need right now.

0xFFFF0004 – Receiver Data:

Received data stored at lowest byte.

0xFFFF0008 – Transmitter Control

LSB is the ready bit, there may be other bit set that we don't need right now.

0xFFFF000C – Transmitter Data

Transmitted data stored at lowest byte.

Write RISC-V code to read a byte from the receiver and immediately send it to the transmitter.

```

        lui t0 0xffff0
receive_wait: lw t1 0(t0)
              andi t1 t1 1                #poll on ready of receiver
              beq t1 x0 receive_wait
              lb t2 4(t0)                #load data
transmit_wait: lw t1 8(t0)                #poll on ready of transmitter
              andi t1 t1 1
              beq t1 x0 transmit_wait    #write to transmitter
              sb t2 12(t0)

```