INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

ELECTRONICS AND COMMUNICATION ENGINEERING

(VLSI SYSTEMS)

ADVANCED ARM ARCHITECTURE-VL 853

IMPLEMENTATION OF HAMMING CODE

UNDER GUIDANCE OF

GIRISH S KUMAR

BY

V.SAHITHI MEENAKSHI

(MT2019528)

Objective

- Generate 31-bit Hamming code from 26-bit Message using Even parity.
- Perform single bit error detection and correction.

Encoding

Step-1

- 26-bit message (d26-d1) 0x376FC17 is taken as test input in R0.

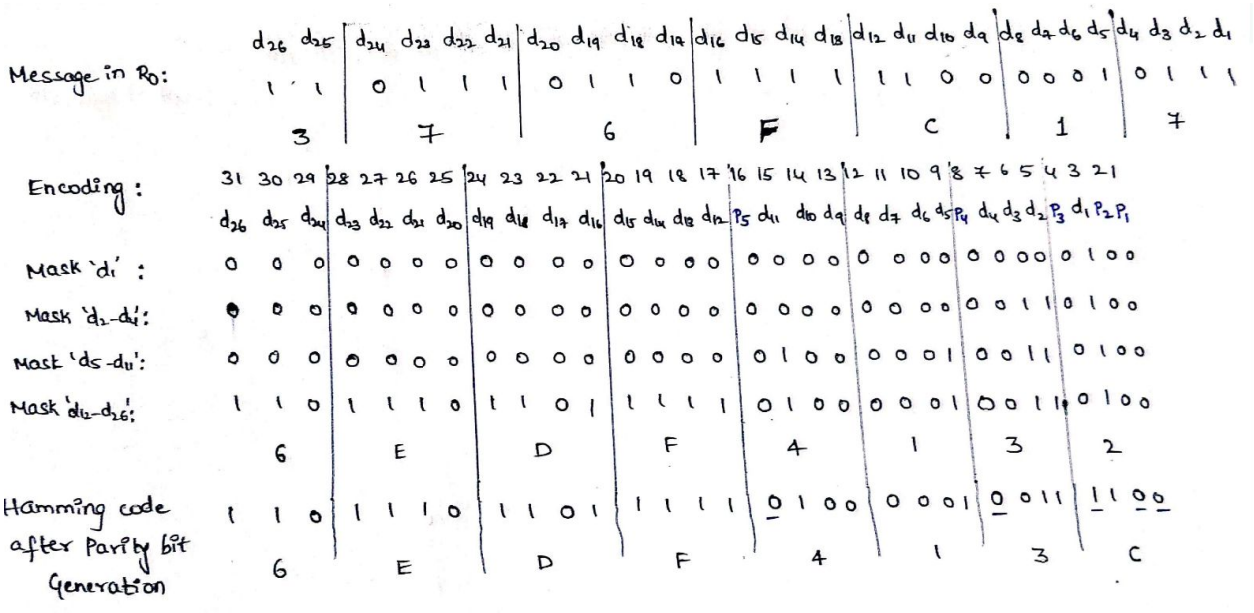
d26	d25	d24	d23	d22	d21	d20	d19	d18	d17	d16	d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1
1	1	0	1	1	1	0	1	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	1

- Mask the message bits and shift them accordingly by the required number to position the bits(d26-d1) at the correct places in the 31-bit codeword.

```
AND R2,R0,#0xE          ;Mask all bits of R0 except d2-d4
ORR R1,R1,R2,LSL #3     ;Align d2-d4 in codeword

AND R2,R0,#0x7F0        ;Mask all bits of R0 except d5-d11
ORR R1,R1,R2,LSL #4     ;Align d5-d11 in codeword
```

- Now the result with message bits aligned in the codeword is present in R1.



Step-2

- Next step is to generate Parity bits for error detection. Parity bits(P1-P5) are inserted at 1,2,4,8,16 positions in codeword respectively.
- The number of parity-bits= 31-26=5 (P1-P5) are generated considering the following bits in codeword(R1).

    - P1-->1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
    - P2-->2,3,6,7,10,11,14,15,18,19,22,23,26,27,30,31
    - P3-->4,5,6,7,12,13,14,15,20,21,22,23,28,29,30,31
    - P4-->8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31
    - P5-->16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31

- Shift the codeword(R1) by the required number and do XOR operation at every iteration with R2.
- The above step is explained in detail using (7,4) hamming code as an example. One can extend the same procedure to (31,26) hamming code.
- Numbers shown in the figure represent bit positions in the codeword. For P1 generation, the 1st-bit position is considered for all XOR iterations.

7	6	5	4	3	2	**1**	R1
9	8	7	6	5	4	**3**	R1>>2
XOR(7,9)	XOR(6,8)	XOR(5,7)	XOR(4,6)	XOR(3,5)	XOR(2,4)	**XOR(1,3)**	R2=R1 xor (R1>>2)
XOR(7,9)	XOR(6,8)	XOR(5,7)	XOR(4,6)	XOR(3,5)	XOR(2,4)	**XOR(1,3)**	R2
XOR(11,13)	XOR(10,12)	XOR(9,11)	XOR(8,10)	XOR(7,9)	XOR(6,8)	**XOR(5,7)**	R2>>4
XOR(7,9,11,13)	XOR(6,8,10,12)	XOR(5,7,9,11)	XOR(4,6,8,10)	XOR(3,5,7,9)	XOR(2,4,6,8)	**XOR(1,3,5,7)**	R2=R2 xor (R2>>4)

- In the (7,4) example considered here, P1 is generated by XOR(1,3,5,7) positions only. Extend the same procedure by doing some more iterations needed for (31,26) code.

P1 generation for (7,4)

```
EOR R2, R1, R1, LSR #2          ; xor of 1,3 bits of  R1
EOR R2, R2, R2, LSR #4          ; xor of 5,7 bits along with 1,3  bits in R1
```

P1 generation for (31,26)

```
EOR R2, R1, R1, LSR #2          ; xor of 1,3 bits of  R1
EOR R2, R2, R2, LSR #4          ; xor of 5,7 bits along with 1,3  bits in R1
EOR R2, R2, R2, LSR #8          ; xor of 9,11,13,15 bits in R2 with previous result
EOR R2, R2, R2, LSR #16         ; xor of 17,19,21,23,25,27,29,31 bits in R2 with previous result
AND R2, R2, #0x1                ; Mask all bits of R2 except P1
ORR R1, R1, R2                  ; Align P1 in codeword
```

- In a similar way, other parity bits are generated.

7	6	5	4	3	**2**	1	R1
8	7	6	5	4	**3**	2	R1>>1
XOR(7,8)	XOR(6,7)	XOR(5,6)	XOR(4,5)	XOR(3,4)	**XOR(2,3)**	XOR(1,2)	R2=R1 xor (R1>>1)
XOR(7,8)	XOR(6,7)	XOR(5,6)	XOR(4,5)	XOR(3,4)	**XOR(2,3)**	XOR(1,2)	R2
XOR(11,12)	XOR(10,11)	XOR(9,10)	XOR(8,9)	XOR(7,8)	**XOR(6,7)**	XOR(5,6)	R2>>4
XOR(7,8,11,12)	XOR(6,7,10,11)	XOR(5,6,9,10)	XOR(4,5,8,9)	XOR(3,4,7,8)	**XOR(2,3,6,7)**	XOR(1,2,5,6)	R2=R2 xor (R2>>4)

Step-3

- Once parity bit(e.g;P1 here) is generated (1st position here in the example), mask that particular bit position and align it in codeword accordingly.

Introducing single bit error

- Flip any bit in R1 to test.

```
EOR R1,R1, #0x4                ;Flip bit 3 to test
```

Error detection

- Calculate C5-C1 check error parity bits similar to the parity bit generation.
- Calculate the decimal equivalent and it gives the position of error bit in the received codeword.
- Error bit position is stored in R10.

```
ADD R10, R10, R2               ; R10 = C1*1 + C2*2 + C3*4 + C4*8 + C5*16
```

Error correction

- Move value #1 into a register(R3). Shift it left by (decimal equivalent-1).
- Now perform logic EXOR between the above register and received codeword.

```
MOV R3,#1
SUB R10,R10,#1
LSL R3,R3,R10
EOR R1,R1,R3
```

- Now R1 contains corrected Hamming code.