Computação Reconfigurável

Aceleração do sistema de deteção e correção de erros de Hamming code [20,15] com dois Co-Processadores de Hardware

Departamento de Eletrónica, Telecomunicações e Informática

Mestrado Integrado em Engenharia de
Computadores e Telemática

Carlos Costa (88755), Ana Rosa (98678)

2022/2023

Índice

1. Descrição da funcionalidade do sistema

O projeto tem como objetivo criar um sistema de aceleração de deteção e correção de erros de Hammig Code de uma mensagem inicial de 15 bits.

No sentido de criar o sistema de deteção e correção de erros durante a transmissão de uma mensagem foram adicionados dois Links Stream ao MicroBlaze e criados dois módulos de hardware.
O módulo EncHammingCode, Figura 1.1, tem como objetivo implementar a codificação de Hamming Code [20,15]. Este módulo tem uma interface de entrada de 32 bits (S00_AXIS) do tipo Stream e modo Slave por onde recebe os dados da interface de saída M0_AXIS do MicroBlaze e uma interface de saída de 32 bits (M00_AXIS) do tipo Stream e modo Master por onde envia os dados codificados para a interface de entrada S0_AXIS do Microblaze.
O módulo DecHammingCode, Figura 1.1, tem como objetivo implementar a descodificação de Hamming Code [20,15]. Este módulo tem uma interface de entrada de 32 bits (S00_AXIS) do tipo Stream e modo Slave por onde recebe os dados codificados da interface de saída M1_AXIS do MicroBlaze e uma interface de saída de 32 bits (M00_AXIS) do tipo Stream e modo Master por onde envia a validação e descodificação dos dados para a interface de entrada S1_AXIS do Microblaze.

O EncHammingCode quando recebe a mensagem de 32 bits apenas considera os 15 bits menos significativos para a codificação. No processo de codificação são obtidos 20 bits e após isso são concatenados 12 bits a zero para se obter 32 bits de saída.
O DecHammingCode ao receber a mensagem codificada de 32 bits apenas considera os 20 bits menos significativos e a partir daí efetua a descodificação. No processo de descodificação são obtidos 17 bits, os dois bits mais significativos indicam a validação da mensagem e os restantes 15 bits representam a mensagem inicial. No final os 17 bits são concatenados com 15 bits a zero para se obter 32 bits de saída.

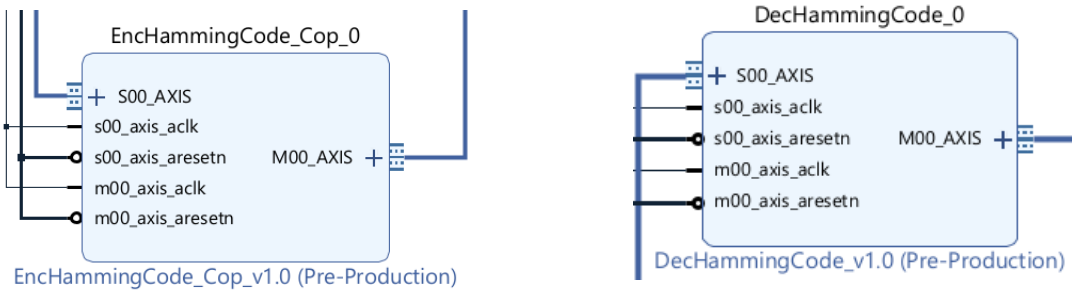

Figura 1.1 - Módulos de Hardware EncHammingCode e DecHammingCode, respetivamente

2. Algoritmo de codificação e descodificação de Hamming Code

2.1 Algoritmo de Codificação

A codificação de Hamming Code baseia-se em adicionar bits de paridade à mensagem original, neste caso considerou-se paridade par.

Assim, o algoritmo é o seguinte:

1. Determinar a quantidade de bits de paridade da mensagem codificada através da fórmula $2^p \geq (p + m) + 1$, sendo ***p*** o número de bits de paridade e ***m*** o número de bits da mensagem inicial.
2. Determinar as posições dos bits de paridade. As posições são determinadas por potências de 2 ($2^0 - 1$, $2^1 - 1$, … $2^{p-1} - 1$).
3. Determinar o valor atribuído a cada bit de paridade. O seu valor é determinado através das combinações destes bits pela operação lógica XOR.
4. Posicionar os bits de dados mais os bits de paridade nas posições de 0 a $(m + p) - 1$.

Na nossa implementação considera-se $m = 15$ sendo, por isso, $p = 5$.
Os bits de paridade têm as posição 15, 7, 3, 1 e 0 e são determinados da seguinte forma:

```
p0 <= m(0) xor m(1) xor m(3) xor m(4) xor m(6) xor m(8) xor m(10) xor m(11) xor m(13);
p1 <= m(0) xor m(2) xor m(3) xor m(5) xor m(6) xor m(9) xor m(10) xor m(12) xor m(13);
p3 <= m(1) xor m(2) xor m(3) xor m(7) xor m(8) xor m(9) xor m(10) xor m(14);
p7 <= m(4) xor m(5) xor m(6) xor m(7) xor m(8) xor m(9) xor m(10);
p15 <= m(11) xor m(12) xor m(13) xor m(14);
```

Figura 2.1.1 - Método de determinação dos bits de paridade (Fórmulas xor)

P15	*P7*	*P3*	*P1*	*P0*	
0	0	0	0	1	**Bit0**
0	0	0	1	0	**Bit1**
0	0	0	1	1	**Bit2**
0	0	1	0	0	**Bit3**
0	0	1	0	1	**Bit4**
0	0	1	1	0	**Bit5**
0	0	1	1	1	**Bit6**
0	1	0	0	0	**Bit7**
0	1	0	0	1	**Bit8**
0	1	0	1	0	**Bit9**
0	1	0	1	1	**Bit10**
0	1	1	0	0	**Bit11**
0	1	1	0	1	**Bit12**
0	1	1	1	0	**Bit13**
0	1	1	1	1	**Bit14**
1	0	0	0	0	**Bit15**
1	0	0	0	1	**Bit16**
1	0	0	1	0	**Bit17**
1	0	0	1	1	**Bit18**
1	0	1	0	0	**Bit19**

Figura 2.1.2 - Método de determinação dos bits de paridade (Tabela de verdade)

Desta forma, a mensagem codificada tem a forma :

m14 m13 m12 m11 p15 m10 m9 m8 m7 m6 m5 m4 p7 m3 m2 m1 p3 m0 p1 p0

2.2 Algoritmo de Descodificação

A descodificação de Hamming Code baseia-se em determinar os checker bits e em função deles verificar se a mensagem descodificada é válida ou não.

1. Calcular os checker bits através da operação XOR das posições dos bits de paridade e dos bits da mensagem. Estes cálculos são determinados novamente através da tabela da Figura 2.1.2. As fórmulas são possíveis de ver na Figura 2.2.1.
2. Determinar o bit de paridade total para a deteção de um único erro ou de um erro duplo. Este bit é determinado através da operação lógica XOR de todos os bits recebidos pelo descodificador (Figura 2.2.2).
3. Se o conjunto dos checker bits for 0, a mensagem é válida. A sequência de bits original pode ser recuperada removendo os bits de paridade adicionados.
4. Se o conjunto dos checker bits for diferente de 0 e o bit de paridade total for 1 significa que existe um único bit da mensagem recebida incorreto. Em função do valor dos checker bits determina-se a posição do bit que está incorreta e aplica-se a inversão desse bit. Esse processo é mostrado na figura 2.2.3
5. Se o conjunto dos checker bits for diferente de 0 e o bit de paridade total for 0 significa que existem dois bits incorretos e, nesse caso, não é possível para o descodificador os corrigir.

```
c0 <= p0 xor codeWord(2) xor codeWord(4) xor codeWord(6) xor codeWord(8) xor
    codeWord(10) xor codeWord(12) xor codeWord(14) xor codeWord(16) xor codeWord(18);
c1 <= p1 xor codeWord(2) xor codeWord(5) xor codeWord(6) xor codeWord(9) xor
    codeWord(10) xor codeWord(13) xor codeWord(14) xor codeWord(17) xor codeWord(18);
c2 <= p3 xor codeWord(4) xor codeWord(5) xor codeWord(6) xor codeWord(11) xor
    codeWord(12) xor codeWord(13) xor codeWord(14) xor codeWord(19);
c3 <= p7 xor codeWord(8) xor codeWord(9) xor codeWord(10) xor codeWord(11) xor
    codeWord(12) xor codeWord(13) xor codeWord(14);
c4 <= p15 xor codeWord(16) xor codeWord(17) xor codeWord(18) xor codeWord(19);
```

Figura 2.2.1 - Método de determinação dos checker bits.

```
s_overallParity <= codeWord(19) xor codeWord(18) xor codeWord(17) xor codeWord(16) xor
       codeWord(15) xor codeWord(14) xor codeWord(13) xor codeWord(12) xor codeWord(11) xor
       codeWord(10) xor codeWord(9) xor codeWord(8) xor codeWord(7) xor codeWord(6) xor
       codeWord(5) xor codeWord(4) xor codeWord(3) xor codeWord(2) xor codeWord(1) xor codeWord(0);
```

Figura 2.2.2 - Método de determinação do bit de paridade total.

```
case c is
     when "00011" => m_valid <= "01" & (s_m xor  "000000000000001");  -- correct bit m0
     when "00101" => m_valid <= "01" & (s_m xor  "000000000000010");  -- correct bit m1
     when "00110" => m_valid <= "01" & (s_m xor  "000000000000100");  -- correct bit m2
     when "00111" => m_valid <= "01" & (s_m xor  "000000000001000");  -- correct bit m3
     when "01001" => m_valid <= "01" & (s_m xor  "000000000010000");  -- correct bit m4
     when "01010" => m_valid <= "01" & (s_m xor  "000000000100000");  -- correct bit m5
     when "01011" => m_valid <= "01" & (s_m xor  "000000001000000");  -- correct bit m6
     when "01100" => m_valid <= "01" & (s_m xor  "000000010000000");  -- correct bit m7
     when "01101" => m_valid <= "01" & (s_m xor  "000000100000000");  -- correct bit m8
     when "01110" => m_valid <= "01" & (s_m xor  "000001000000000");  -- correct bit m9
     when "01111" => m_valid <= "01" & (s_m xor  "000010000000000");  -- correct bit m10
     when "10001" => m_valid <= "01" & (s_m xor  "000100000000000");  -- correct bit m11
     when "10010" => m_valid <= "01" & (s_m xor  "001000000000000");  -- correct bit m12
     when "10011" => m_valid <= "01" & (s_m xor  "010000000000000");  -- correct bit m13
     when "10100" => m_valid <= "01" & (s_m xor  "100000000000000");  -- correct bit m14
```

Figura 2.2.3 - Correção do erro da mensagem recebida.

3. Diagrama de Blocos do Coprocessador

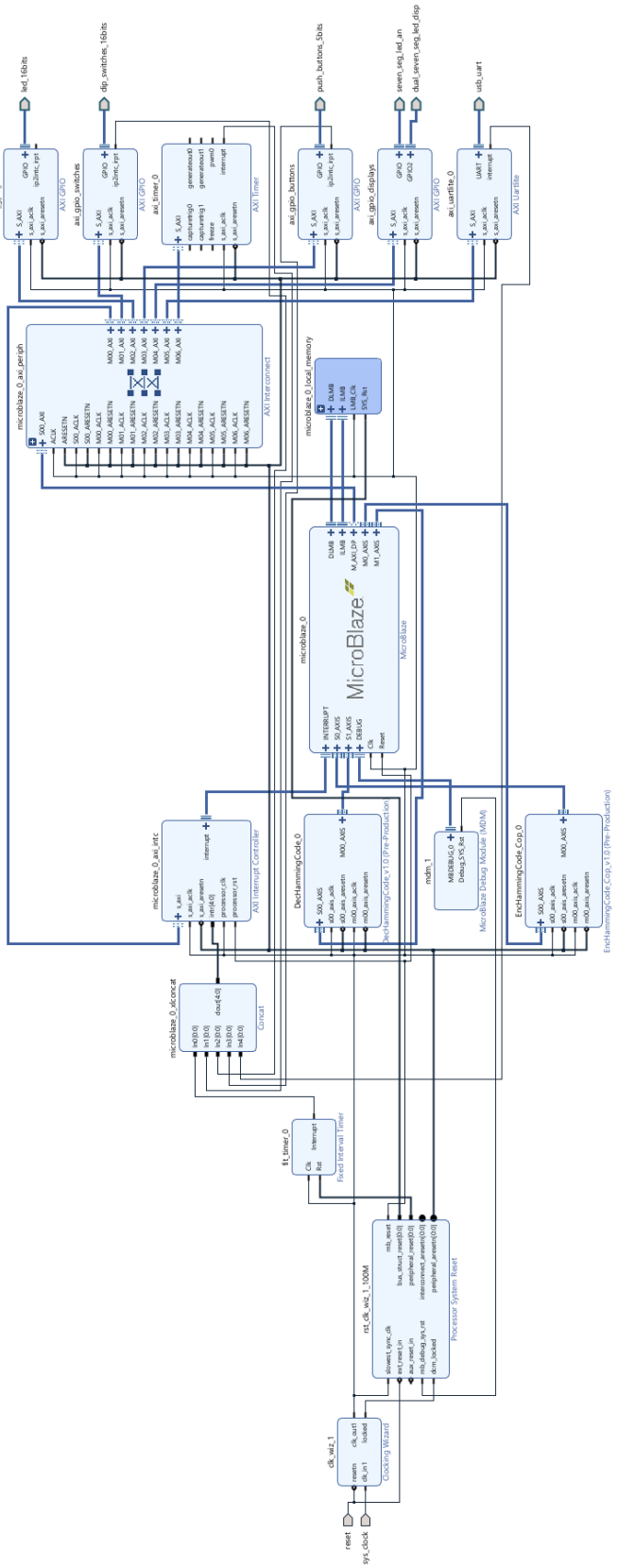

Figura 3.1 - Diagrama de Blocos

4. Resultados Obtidos

Como o algoritmo de Hamming Code é para a transmissão de mensagens de 15 bits gerou-se aleatoriamente números inteiros de 15 bits (de 0 a 32767). Na figura 4.1 é possível observar os oito dos 2000 primeiros valores gerados aleatoriamente.

Num primeiro teste os valores gerados aleatoriamente são inseridos no codificador e os valores codificados resultantes são inseridos no descodificador.
Na figura 4.2 mostra-se os valores codificados tanto por software como por hardware, verificando que por hardware o processo de codificação é muito mais rápido.
Na Figura 4.3 verifica-se que os valores obtidos na descodificação são exatamente iguais aos valores da inicialização do array, pelo que conclui-se que não ocorreu nenhum erro na transmissão da mensagem.

Num segundo teste, para verificar que o descodificador efetua a deteção e correção corretamente inseriu-se os seguintes valores codificados: 0x000EC266 , 0x000AC266 e 0x000AC262. O primeiro valor é válido (2MSB a 0) , o segundo valor representa 1 erro (2MSB a 1) que é corrigido e terceiro valor apresenta dois erros (2MSB a 2) e, por isso, a informação é inválida. O que se pode observar na Figura 4.4

0x000EC266 - 11101100001001100110 (mensagem válida = 00 111010000101101)
0x000AC266 - 10101100001001100110 (1 erro corrigido = 01 111010000101101)
0x000AC262 - 10101100001001100010 (2 erros inválida = 10 101010000101100)

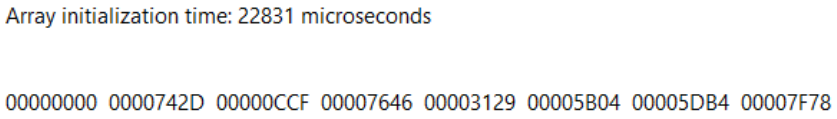

Figura 4.1 - Inicialização do array srcData com 2000 valores aleatórios.

```
---------- Encoder Hamming Code ----------
 Software Only

Software only encoder hamming code time: 31425 microseconds
00000000  000EC266  0001CCFE  000EE4B8  00061246  000BB022  000BDBA8  000F774B
Checking result: OK

 Hardware Assisted

Hardware assisted encoder hamming code time: 801 microseconds
00000000  000EC266  0001CCFE  000EE4B8  00061246  000BB022  000BDBA8  000F774B
Checking result: OK
------------------------------------------------------------------------
```

Figura 4.2 - Resultados dos valores codificados por software e hardware.

```
------------------------------------------------------------------------
 Decoder Hamming Code
 Software Only

Software only decoder hamming code time: 47790 microseconds
00000000  0000742D  00000CCF  00007646  00003129  00005B04  00005DB4  00007F78
Checking result: OK
 Hardware Assisted

Hardware assisted decoder hamming code time: 801 microseconds
00000000  0000742D  00000CCF  00007646  00003129  00005B04  00005DB4  00007F78
Checking result: OK
```

Figura 4.3 - Resultados dos valores codificados por software e hardware.

Figura 4.4 - Resultados dos valores codificados por software e hardware.

5. Contribuição dos Autores

Visto que os requisitos do nosso projeto foram alcançados, a auto-avaliação do grupo é 17. A percentagem de contribuição de cada aluno é de 50%.

Este projeto está organizado em:

- Vivado - diretórios **hamming_code, HammingCode, ip_repo**
- Vitis - diretório **HammingCodeVitis**

6. Bibliografia

http://www.ijlera.com/papers/v2-i11/6.201711550.pdf

https://www.thecrazyprogrammer.com/2017/03/hamming-code-c.html?utm_content=cmp-true

https://www.geeksforgeeks.org/hamming-code-in-computer-network/