UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
NÍVEL MESTRADO PROFISSIONAL

ELCIO KONDO

ESTUDO COMPARATIVO DE ALGORITMOS DE ECC APLICADOS À MEMÓRIA NAND FLASH

SÃO LEOPOLDO
2017

Elcio Kondo

Estudo comparativo de algoritmos de ECC aplicados à memória NAND Flash

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica, pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade do Vale do Rio dos Sinos - UNISINOS.

Orientador: Profa. Dra. Margrit Reni Krug

Coorientador: Prof. Dr. Eduardo Luis Rhod

São Leopoldo
2017

K82e Kondo, Elcio

Estudo comparativo de algoritmos de ECC aplicados à memória NAND Flash/ Elcio Kondo. – 2017.

166 f. : il. ; color.; 30 cm.

Dissertação (Mestrado em Engenharia Elétrica) – Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Engenharia Elétrica, São Leopoldo, RS, 2017.

Orientador: Profa. Dra. Margrit Reni Krug; Coorientador: Prof. Dr. Eduardo Luis Rhod

1. Engenharia Elétrica 2. Código - Correção - Erros (ECC) 3. Memória NAND Flash 4. BCH 5. Reed-Solomon 6. Hamming 7. Armazenamento - Dado digital Título. II. Krug, Margrit Reni. III. Rhod, Eduardo Luis.

CDU 621

Catalogação na Publicação: Bibliotecária: Eliete Mari Doncato Brasil – CRB 10/1184

Elcio Kondo

Estudo comparativo de algoritmos de ECC aplicados à memória NAND Flash

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica, pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade do Vale do Rio dos Sinos - UNISINOS.

Aprovado em 21 de Dezembro de 2016.

BANCA EXAMINADORA:

Prof. Dr. Rodrigo Marques de Figueiredo –
Unisinos
Avaliador

Prof. Dr. Luigi Carro – UFRGS
Avaliador Externo

Profa. Dra. Margrit Reni Krug (Orientador)
Prof. Dr. Eduardo Luis Rhod (Coorientador)

Visto e permitida a impressão
São Leopoldo

Prof. Dr. Eduardo Luis Rhod
Coordenador PPG em Engenharia Elétrica

Este trabalho é dedicado às mulheres
que nos orientam enquanto crianças,
nos incentivam quando adolescentes
e que tentam entender nossas horas
ausentes quando nos tornamos adultos.

AGRADECIMENTOS

À minha família, pelo apoio e paciência.

À memória da minha mãe que sempre me apoiou nos estudos.

Agradecimentos especiais são direcionados ao grupo do itt Chip

À estagiária Bruna Fernandes Flesch pela ajuda na pesquisa deste trabalho.

“O que sabemos é uma gota;
o que não sabemos é um oceano."
(Isaac Newton)

RESUMO

Atualmente vários equipamentos eletrônicos são equipados com memórias NAND *Flash* para armazenar dados. Essas memórias são controladas através de um circuito integrado com um controlador de memória, que internamente possui um sistema para garantir a integridade das informações armazenadas, os quais são conhecidos por *Error Correction Codes* (ECC). Os ECCs são códigos capazes de detectar e corrigir erros através de bits redundantes adicionados à informação. Normalmente, os códigos ECC são implementados em *hardware* dentro do controlador de memória NAND Flash.

Neste trabalho comparou-se alguns códigos de ECC utilizados pela indústria, para as comparações utilizou-se os códigos ECC: Hamming, BCH (Bose-Chaudhuri-Hocquenghem) e Reed-Solomon. Sistematicamente realizou-se comparações entre os ECCs selecionados e escolheu-se os dois mais apropriados (BCH e Hamming), os quais foram implementados em linguagem VHDL, o que possibilitou identificar o código com melhor vantagem econômica no uso em memórias NAND Flash.

Palavras-chaves: ECC. Memória *NAND Flash*. BCH. Reed-Solomon. Hamming.

ABSTRACT

Nowadays several electronic equipment are using NAND Flash memories to store data. These memories are controlled by an integrated circuit with an memory controller embedded that internally has a system to ensure the integrity of the stored information, that are known as Error Correction Codes (ECC). The ECCs are codes that can detect and correct errors by redundant bits added to information. Usually the ECC codes are implemented on NAND Flash memory controller as a hardware block.

On this text ECC codes used by industry, the Hamming code, BCH (Bose-Chaudhuri-Hocquenghem) and Reed-solomon codes were compared.Systemically compare between selected ECCs were done and selected two codes (BCH and Hamming), which were described in VHDL language and allowed to identify the best code with better economical advantage for NAND Flash memories.

Key-words:ECC. Memory *NAND Flash*. BCH. Reed-Solomon. Hamming.

LISTA DE FIGURAS

Figura 1 – Comparativo Entre Estruturas do Transistor MOS e com *Floating Gate*. . . 29
Figura 2 – Operações do Transistor com *Floating Gate*. 30
Figura 3 – Arquitetura Lógica Memória NAND Flash 32
Figura 4 – Distribuição de V_t para SLC, MLC e TLC 33
Figura 5 – Estrutura Básica de ECC. 35
Figura 6 – Exemplo de Notação de Códigos ECC. 36
Figura 7 – Localização do ECC e Tabela Comparativa 36
Figura 8 – Paridade. 37
Figura 9 – Circuito Gerador de Paridade 38
Figura 10 – *Datasheet* Módulo de Memória DRAM com ECC 40
Figura 11 – Estrutura Código BCH 46
Figura 12 – Diagrama de Divisor Polinomial e Tabela de Estados 48
Figura 13 – Circuito Divisor Polinomial para φ_1=x^5+x^2+1 54
Figura 14 – Circuito Divisor Polinomial para φ_3=x^5+x^4+x^3+x^2+1 54
Figura 15 – Codificação - Código de Hamming 58
Figura 16 – Decodificação para Hamming (14,9) 61
Figura 17 – Exemplo do Código de Reed Solomon 63
Figura 18 – Exemplo Informação de 9 bits Dividida em 3 Símbolos de m=3 bits 64
Figura 19 – Tabela do Corpo de Galois GF(8) e o Circuito Gerador 65
Figura 20 – Codificação em Símbolos para o Código R/S 65
Figura 21 – Tabela Resultado Artigo 74
Figura 22 – Tabela Resumo - Trabalhos Relacionados 74
Figura 23 – Comparação entre os 4 códigos 78
Figura 24 – Comparação dos 3 Códigos para NAND Flash 80
Figura 25 – Comparação BCH / R/S - Capacidade de Correção vs Quantidade de bits de Paridade 81
Figura 26 – Comparação dos Códigos BCH vs 4xHamming para NAND Flash 82
Figura 27 – Comparação Hamming 4096 bits vs Hamming 4 Blocos 1024 bits 84
Figura 28 – Diagrama de Blocos - Código Hamming para 1024 bits 85
Figura 29 – Distribuição da Página - Código Hamming 1024 bits 86
Figura 30 – Fluxograma - Bloco Codificador - Código Hamming para 1024 bits 87
Figura 31 – Fluxograma - Bloco Decodificador - Código Hamming para 1024 bits . . . 88
Figura 32 – Diagrama de Blocos - Código BCH para 4096 bits 89
Figura 33 – Circuito Codificador - Código BCH para 4096 bits 91
Figura 34 – Diagrama de Blocos - Cálculo das Síndromes 91
Figura 35 – Diagrama de Blocos - Cálculo do Polinômio de Erros 92

Figura 36 – Diagrama de Blocos - Cálculo da Localização dos Erros 93

LISTA DE TABELAS

Tabela 1 – Principais Características dos Códigos BCH para m≥3 41
Tabela 2 – Tabela Característica de BCH de m=3 a m=8 42
Tabela 3 – Tabela de Polinômios Primitivos em Função de m 43
Tabela 4 – Tabela de Corpo de Galois para $x^5 + x^2 + 1$ 44
Tabela 5 – Tabela de Polinômios Mínimos para GF(32) 45
Tabela 6 – Coeficientes do Polinômio Localizador de Erros em Função das Síndromes para $1 \leq t \leq 5$ 52
Tabela 7 – Principais Características dos Códigos de Hamming 57
Tabela 8 – Tabela de Possíveis Códigos de Hamming 57
Tabela 9 – Detecção de Erro Simples e Erro Duplo 60
Tabela 10 – Principais Características do Código de Reed-Solomon 62
Tabela 11 – Tabelas de Adição e Multiplicação para GF(8) 66
Tabela 12 – Estudo de Cobertura do Código de Hamming para 4096 bits 81
Tabela 13 – Primitivas Utilizadas Pela Implementação em VHDL dos Códigos 4xHamming 1024 e BCH 4096 96
Tabela 14 – Comparação Entre Primitivas 97
Tabela 15 – Transistores Utilizados Pela Implementação em VHDL dos Códigos 4x Hamming 1024 e BCH 4096 98
Tabela 16 – Distribuição de Transistores 99
Tabela 17 – Tempos de Computação para Código 4xHamming1024 e Código BCH4096 100

LISTA DE ABREVIATURAS E SIGLAS

BCH	Bose-Chaudhuri-Hocquenghem
CD	*Compact Disc* - Disco Compacto
DED	*Double Error Detection* - Detecção de erro duplo
DRAM	*Dynamic Random Access Memory* - Memória dinâmica de acesso randômico
DVD	*Digital Video Disc* - Disco de Vídeo Digital
ECC	*Error Correction Code* - Código de Correção de Erros
EEPROM	*Electrical Erasable Read Only Memory* - Memória de Somente Leitura Elétricamente Apagável
eMMC	*embedded Multi Media Card* - Cartão de Multi Mídia Embutido
FPGA	*Field Programmable Gate Array* - Matriz de Campos Programáveis
IC	*Integrated Circuit* - Circuito Integrado
I/O	*Input/Output* - Entrada/Saída
LDPC	*Low Density Parity Check* - Verificação de Paridade de Baixa Densidade
LFSR	*Linear Feedback Shift Register* - Registrador de Deslocamento de Realimentação Linear
MLC	*Multi Level Cell* - Célula de Múltiplos Níveis
MMC	Mínimo Múltiplo Comum
MOS	*Metal Oxide Semiconductor* - Metal Óxido Semicondutor
NAND	*Not And* - Não E
NOR	*Not Or* - Não Ou
NVM	*Non Volatile Memory* - Memória Não Volátil
R/S	Reed-Solomon
SEC	*Single Error Correction* - Correção de Erro Simples
SLC	*Single Level Cell* - Célula de Nível Único

TLC *Triple Level Cell* - Célula de Três Níveis

USB *Universal Serial Bus* - Barramento Universal Serial

VHDL *Very Fast Hardware Description Language* - Linguagem Rápida de Descrição de Hardware

V_t *Threshold Voltage* - Tensão de Disparo

XOR Ou exclusivo

SUMÁRIO

1 INTRODUÇÃO . . . 25
1.1 Justificativa . . . 26
1.2 Objetivos . . . 26
1.2.1 Objetivos gerais . . . 26

2 MEMÓRIAS NAND FLASH . . . 29
2.1 História da Célula com *Floating Gate* . . . 29
2.2 Funcionamento Básico do Transistor de *Floating Gate* . . . 30
2.3 A origem da NAND Flash . . . 30
2.4 Arquitetura Lógica e Características de uma Memória NAND Flash . . . 31
2.5 Single Level Cell (SLC) e Multi Level Cell (MLC) . . . 32

3 CÓDIGOS CORRETORES DE ERRO . . . 35
3.1 Paridade . . . 37
3.1.1 Codificação . . . 38
3.1.1.1 Exemplo de Codificação . . . 38
3.1.2 Decodificação . . . 39
3.1.2.1 Exemplo de Decodificação . . . 39
3.1.3 ECC em Módulos de Memória DRAM . . . 39
3.2 Código de Bose-Chaudhuri-Hocquenghem (BCH) . . . 40
3.2.1 Códigos Cíclicos Binários . . . 41
3.2.2 Nomenclaturas Utilizadas pelo Código BCH . . . 41
3.2.3 O Polinômio Gerador . . . 42
3.2.4 Corpo de Galois ou *Galois Field* (GF) . . . 43
3.2.5 Operações Utilizando Corpo de Galois . . . 45
3.2.6 Polinômios Mínimos . . . 45
3.2.7 Divisão de Polinômios . . . 46
3.2.8 Codificação . . . 49
3.2.8.1 Exemplo de Codificação . . . 50
3.2.9 Decodificação . . . 51
3.2.9.1 Exemplo de Decodificação . . . 52
3.3 Código de Hamming . . . 56
3.3.1 Codificação . . . 58
3.3.1.1 Exemplo de Codificação . . . 59
3.3.2 Decodificação . . . 59
3.3.2.1 Exemplo de Decodificação . . . 61

3.4 Código de Reed-Solomon . . . 62
3.4.1 Transformando a Informação Binária em Símbolos . . . 63
3.4.1.1 Exemplo de Transformação de Informação Binária em Símbolos . . . 64
3.4.2 Operações com o Corpo de Galois para GF(8) . . . 65
3.4.3 Polinômio Gerador . . . 66
3.4.4 Codificação . . . 66
3.4.4.1 Exemplo de Codificação . . . 66
3.4.5 Decodificação . . . 67
3.4.5.1 Exemplo de Cálculo de Sindromes para Um Código Sem Erros . . . 67
3.4.5.2 Exemplo de Decodificação: . . . 68

4 TRABALHOS RELACIONADOS . . . 73

5 MATERIAIS, FERRAMENTAS E METODOLOGIA . . . 77
5.1 Premissas . . . 77
5.2 Análise . . . 77
5.2.1 Análise - Parte 1 . . . 77
5.2.2 Análise - parte 2 . . . 79
5.2.3 Análise - parte 3 . . . 81
5.3 Ferramentas . . . 83
5.4 Metodologia . . . 83
5.4.1 Implementação em VHDL código de Hamming com 4 blocos de 1024 bits . 83
5.4.2 Implementação em VHDL do Código BCH para página de 4096 bits . . . 89

6 RESULTADOS . . . 95
6.1 Comparação entre os Recursos Utilizados - Primitivas . . . 95
6.2 Comparação entre os Recursos Utilizados - Transistores . . . 97
6.2.1 Comparação entre os Tempos de Computação . . . 100

7 CONCLUSÃO . . . 103
7.1 Trabalhos futuros . . . 104

REFERÊNCIAS . . . 105

APÊNDICES 109

APÊNDICE A – CIRCUITO DE DIVISÃO POLINOMIAL - BCH 4096 BITS . . . 111

APÊNDICE B – QTDE DE TRANSISTORES EM CIRCUITOS PRIMITIVOS 113

APÊNDICE C – RESULTADOS - SÍNTESE ISE XILINX 119

1 INTRODUÇÃO

As memórias NAND Flash a cada dia se tornam mais importantes em nossas vidas, pois estão em praticamente todos os dispositivos eletrônicos do nosso cotidiano desde os celulares e computadores até nos carros e *smart* tvs, armazenando informações, fotos e mensagens. Com o aumento da dependência sobre as memórias NAND Flash, cada vez mais o aumenta a necessidade de memórias com melhor desempenho, mais velocidade e mais capacidade de armazenamento e também a necessidade de dados íntegros para garantir o bom funcionamento. Por isso, garantir a detecção e correção de erros em memórias torna-se cada vez mais importante.

Segundo Rino Micheloni Luca (2010) a aplicação que alavancou as memórias NAND Flash foi a fotografia digital e o armazenamento digital. Em 1995 foi lançada, a primeira mídia removível utilizando NAND Flash em formato de cartão e a partir de então surgiram outros tipos e formatos de cartões. Os cartões não armazenam somente as imagens, mas também áudio e vídeo o que viabiliza o transporte e a transferência de arquivos de um dispositivo para outro. Outro dispositivo que tem como componente principal a NAND Flash é o *pendrive* que foi introduzido no ano 2000 pelas empresas Trek e IBM e que foi responsável pela extinção do *floppy disk* (RINO MICHELONI LUCA, 2010).

A memória NAND Flash é um dispositivo de armazenamento de dados digitais do tipo não volátil, ou seja, não perde os dados quando fica sem energia. Além disso apresenta uma série de vantagens sobre as outras mídias utilizadas para armazenamento, tais como: alta capacidade de armazenamento; alta densidade e um baixo custo por bit. Atualmente, um chip pode guardar até 64 GBytes, contra 8 GBytes do DVD. Os fabricantes de memória utilizam códigos corretores de erros ou ECC para detectar e corrigir os bits lidos e escritos na memória. A literatura apresenta diversas estratégias criadas pelos fabricantes de memória com a intenção de garantir a confiabilidade deste tipo de dispositivo (RINO MICHELONI LUCA, 2010) e Chen (2007).

As memórias NAND Flash estão entre os dispositivos que utilizam o último nó tecnológico da miniturização de circuitos para aumentar a densidade e capacidade de armazenamento. Junto com a miniaturização dos circuitos, há o aumento da frequência dos sinais e a também a probabilidade de erros durante o funcionamento da memória. Existem também algumas características intrínsecas da memória NAND Flash como a limitação da quantidade de escritas e apagamentos, que quando estão no final de sua vida útil a quantidade de erros aumenta. Devido a isso os fabricantes adaptaram o ECC para memórias para permitir o seu correto funcionamento. Um outro fator que pode causar erros são os ruídos internos, como por exemplo o *crosstalk*. Mutlu (2014) diz que as memórias NAND Flash em funcionamento são muito ruidosas e por isso necessitam de ECC.

Para Lin, Chen e Wang (2006) os códigos de ECCs para memórias NAND Flash mais comuns são: códigos de Hamming, Reed-Solomon e Bose Chaudhuri Hocquenghem (BCH). Os estudos apontam algumas vantagens e desvantagens no uso de cada uma (LIN; CHEN; WANG, 2006). Portanto, neste trabalho, realizou-se um estudo que envolve os algoritmos de ECC encontrados na literatura para identificar aspectos relacionados ao custo e benefícios da aplicação de cada uma, como por exemplo: velocidade de codificação e correção, área estimada ocupada por cada solução e velocidades de codificação e decodificação.

Este trabalho foi dividido em capítulos. O capítulo 2 fala sobre as memórias NAND Flash explicando a origem da célula básica da memória, o princípio de funcionamento e sua arquitetura interna. No capítulo 3 o leitor é apresentado aos 3 códigos de ECCs (BCH, Hamming e Reed-Solomon) e também à paridade. Para cada um deles é explicado o algoritmo, o processo de codificação e decodificação e cada um com exemplos para ilustrar o funcionamento de cada código. O leitor vai notar que para todos os exemplos é utilizada a mesma informação de 9 bits e essas informações serão consolidadas mais adiante no capítulo 6, capítulo dos resultados. O capítulo 4 trata dos trabalhos relacionados em que foram pesquisados os trabalhos e artigos que possuem alguma relação com a comparação entre os ECCs dedicados à memória NAND Flash. Dentre os trabalhos pesquisados não foi encontrado comparações entre os códigos de Hamming e BCH aplicados à memória NAND Flash, principal objetivo deste trabalho.

1.1 Justificativa

As memórias NAND Flash vêm ganhando cada vez mais mercado, pois além de serem memórias do tipo não voláteis apresentam alta densidade e custo baixo (RINO MICHELONI LUCA, 2010). Porém, devido ao fato da importância dos dados, além do grande volume armazenado, esses dispositivos precisam garantir confiabilidade e, ao mesmo tempo, garantir que as velocidades de leitura e gravação sejam aceitáveis. Os algoritmos de detecção e correção de erros (ECC) podem auxiliar nesta atividade (BARROS, 2011).

1.2 Objetivos

1.2.1 Objetivos gerais

Este projeto tem como objetivo geral a realização da comparação entre alguns algoritmos de ECC aplicados à memória NAND Flash, a fim de identificar qual melhor se adequa comercialmente, levando em conta a cobertura de falhas e o impacto da implementação na velocidade de operação da memória. Para alcançar o objetivo geral, os seguintes objetivos específicos foram necessários:

a) Analise e identificação de qual(is) dos algoritmos foram mais adequados à NAND Flash, levando-se em consideração a cobertura de falhas e o tempo total de teste;

b) Implementação e validação de cada um dos algoritmos de ECC identificados;

c) Coleta dos dados relacionados à detecção e correção de erros na execução e a velocidade de cada algoritmo.

2 MEMÓRIAS NAND FLASH

Este capítulo tem como finalidade apresentar um breve referencial teórico sobre as memórias NAND Flash, para contextualizar sua importância, além de caracterizá-las.

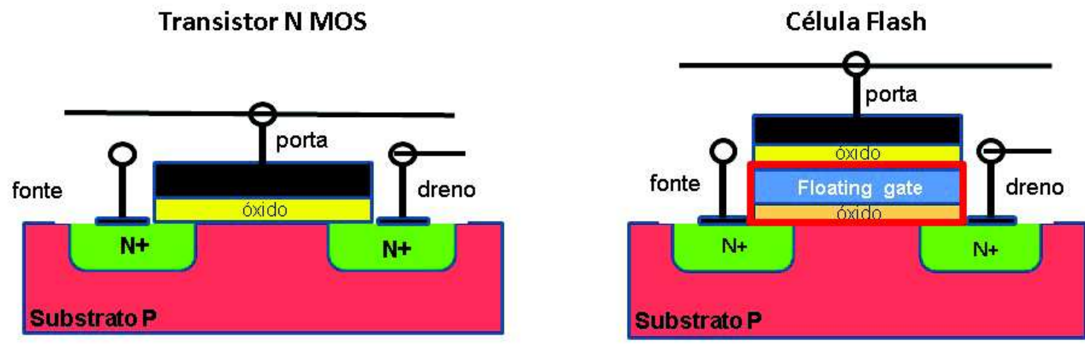
2.1 História da Célula com *Floating Gate*

Historicamente, a célula básica utilizada em memórias das famílias EEPROM, NOR e NAND, foi criada nos anos 60. A ideia de utilizar um transistor MOS e guardar informação na porta surgiu através de um defeito dos transistores MOS pois, quando um elétron atraído pelo campo elétrico do *gate* penetra e atravessa o óxido ficando armazenado dentro do *gate* altera o funcionamento do transistor deslocando o valor da tensão de formação do canal, chamado de *Vthreshold*, ou simplesmente V_t (RINO MICHELONI LUCA, 2010). Observando-se este tipo de defeito criou-se um transistor com a porta flutuante (*floating gate*), que consiste em construir dentro do óxido da porta do transistor uma segunda porta, sem conexão elétrica com a outra. Desta forma os elétrons são atraídos pela porta flutuante, atravessam o óxido e ficam armazenados na porta flutuante, e fisicamente a estrutura da porta não é afetada por esse processo(RINO MICHELONI LUCA, 2010) .

O mecanismo de armazenamento do transistor de *floating gate* é baseado no armazenamento de elétrons dentro da porta flutuante e que alteram o valor de V_t do transistor, ou seja, quando há elétrons dentro da porta flutuante, tem-se o valor lógico 1 e quando não há elétrons um V_t equivalente ao nível lógico zero. (RINO MICHELONI LUCA, 2010).

A Figura 1 mostra um comparativo entre o transistor MOS e um transistor com *floating gate*, onde pode-se observar que no transistor com *floating gate* há um bloco imerso no óxido de porta.

Figura 1 – Comparativo Entre Estruturas do Transistor MOS e com *Floating Gate*.



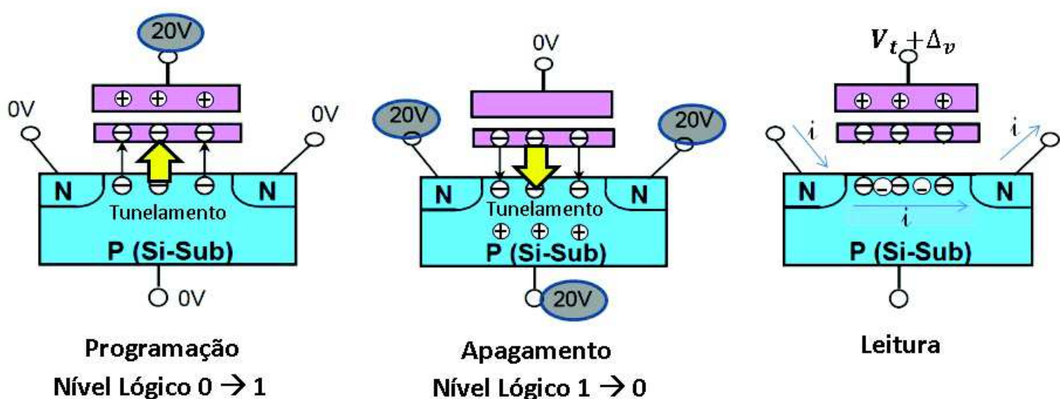
Fonte: adaptado de Chen (2007).

2.2 Funcionamento Básico do Transistor de *Floating Gate*

Segundo Rino Micheloni Luca (2010) o funcionamento do transistor de *floating gate* pode ser resumido em 3 tipos de operações: gravação, leitura e apagamento. Na gravação é aplicada uma alta tensão no gate, enquanto o dreno e a fonte são aterrados. Os elétrons são então atraídos para o óxido da porta, atravessam o óxido e chegam no *floating gate* e lá ficam armazenados.

As três operações do transistor com o *floating gate* são observadas na Figura 2.

Figura 2 – Operações do Transistor com *Floating Gate*.



Fonte: Adaptado de Rino Micheloni Luca (2010).

Na Figura 2 observa-se que na programação os elétrons são atraídos pelo campo elétrico da porta que é ligado à fonte de alta tensão e armazenados no *floating gate*. No apagamento o substrato e os poços N são ligados à fonte de alta tensão e os elétrons são atraídos para fora do *floating gate* pelo campo elétrico do substrato e dos poços N. Já na leitura, a tensão no porta determina a formação do canal de acordo com com a quantidade de elétrons presos no *floating gate*, caracterizando os níveis lógicos 0 e 1.

2.3 A origem da NAND Flash

A memória NAND Flash foi criada em 1984 pelo professor Fujio Masuoka que trabalhava no laboratório da Toshiba do Japão e consiste em ligar os transistores de *gate* flutuante em série (MASUOKA et al., 1987).

Segundo Rino Micheloni Luca (2010), a configuração atual da NAND Flash foi lançada comercialmente em 1991 pela Intel, e a cada ano o seu uso aumenta, especialmente pelo fato da velocidade de operação ser superior às apresentadas pelos discos rígidos, do inglês *hard disks* (HD). Caracterizam-se por poderem ser montadas em unidades de estado sólido, do inglês *solid state drives* (SSD), que vem gradativamente substituindo com vantagens os HDs mecânicos, mas que ainda possuem custo maior por Giga Byte (RINO MICHELONI LUCA, 2010).

A NAND Flash também vem se popularizando cada vez mais nas memórias assim chamadas de USB ,*Universal Serial Bus*, os pendrives, cartões de memória SD, do inglês *secure digital card*(SD card), e o cartão de multi media embutida, do inglês *embedded multi midia card* (eMMC), entre outros. A cada ano a densidade dos circuitos integrados (CIs) de NAND Flash vem aumentando e o preço por Gigabyte diminuindo. Por possuírem baixíssimo consumo de corrente, o calor dissipado é muito baixo, assim permite-se o encapsulamento do tipo 3D (encapsulamentos com empilhamento de mais de um CI), cujas ligações dos pinos de endereço e I/O são feitas em paralelo.(MICHELONI; MARELLI; RAVASIO, 2008)

2.4 Arquitetura Lógica e Características de uma Memória NAND Flash

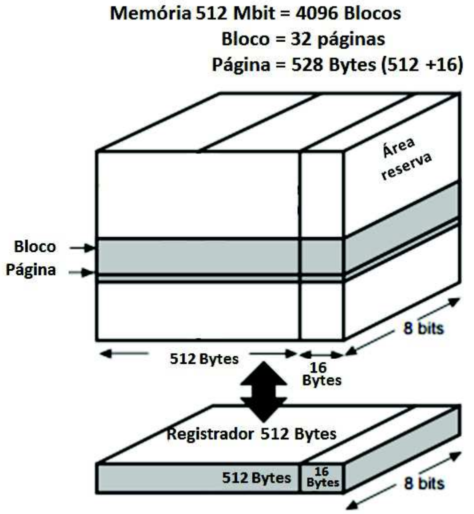
A memória NAND Flash tem suas células organizadas formando Páginas ou *page*, do inglês, que é a menor unidade endereçável, e Blocos ou *blocks* (RINO MICHELONI LUCA, 2010).

Utilizando como exemplo uma memória comercial NAND Flash de 512M bits do fabricante Micron modelo NAND512W3A (Micron, 2012), tem-se:

- Página: composta por 512M Bytes mais 16 Bytes na Área Reserva;
- Área Reserva: área da memória compartilhada e utilizada para guardar os bits de paridade do ECC, as marcações de *bad blocks* e também do *wear leveling* e,
- Bloco: composto por 32 páginas.

A Figura 3 mostra a menor unidade endereçável de uma memória NAND Flash que, é a página com 512 Bytes + 16 Bytes = 528 Bytes ou 4224 bits.

Figura 3 – Arquitetura Lógica Memória NAND Flash



Fonte: adaptado de Micron (2012).

A Figura 3 mostra a arquitetura lógica de uma memória NAND Flash comercial de 512M bits do fabricante Micron, em que uma página é composta por 4096 bits + 128 bits reserva ou 528 Bytes no total. Cada Bloco é composto por 32 páginas (32 páginas x 528 Bytes = 16896 Bytes) e no total há 4096 Blocos de 16896 Bytes (4096 Blocos x 16896 Bytes = 69 206 016 Bytes ou 64 GBytes.

Uma característica das memórias baseadas em transistores com *floating gate* é a resistência ao desgaste, do inglês *endurance*. O óxido de porta é danificado fisicamente a cada gravação e a cada apagamento. Por esse motivo, existe uma quantidade finita de gravações e apagamentos que uma memória pode realizar, e que hoje encontram-se na casa dos 10 milhões de escritas. Assim, uma leitura pode ser afetada por uma célula cujo óxido já esteja danificado (RINO MICHELONI LUCA, 2010).

Para minimizar o impacto do desgaste, os controladores de NAND possuem a função de nivelamento de desgaste, do inglês *wear leveling*, que consiste em distribuir uniformemente as gravações entre as células de forma a nivelar o desgaste do óxido das células e maximizar a vida da memória (LOPES, 2015).

Blocos com bits com falhas são marcados e substituídos por blocos reservas. Para isso os controladores de memória são equipados com a função de controle de *bad blocks* e a identificação desse é guardada na área reserva(MICHELONI; MARELLI; RAVASIO, 2008).

2.5 Single Level Cell (SLC) e Multi Level Cell (MLC)

A memória NAND Flash com células que permitem somente 2 estados é denominada *Single-Level-Cell* (SLC) ou célula de nível único. Controlando a variação da tensão aplicada

ao *gate* pode-se ter mais de 2 estados. Se a tensão resultante de V_t tiver uma distribuição que tenha 4 estados então a célula pode armazenar 2 bits por célula. Esse componente é conhecido como *Multi-Level-Cell* (MLC) ou célula de múltiplos níveis. Similarmente um componente com 8 níveis de V_t pode armazenar 3 bits por célula e é conhecida como *Triple-Level-Cell* (TLC) ou célula de três níveis (REGULAPATI, 2015). A Figura 4 mostra a distribuição de v_t para SLC, MLC e TLC.

Figura 4 – Distribuição de V_t para SLC, MLC e TLC

Fonte: adaptado de Regulapati (2015).

Na Figura 4 observa-se à esquerda o estado apagado quando não há informação guardada na célula. Quando há informação, a célula de SLC guarda apenas 1 bit e tem apenas 1 nível de Vt. Na célula MLC há 4 níveis de Vt que correspondem aos 4 níveis lógicos: 11, 01, 00 e 10 e na célula TLC há 8 níveis lógicos para 8 níveis de Vt correspondendo aos níveis lógicos 111, 110, 010, 011, 001, 000, 100 e 101.

3 CÓDIGOS CORRETORES DE ERRO

Neste capítulo será realizada uma breve introdução aos códigos corretores de erros utilizados neste trabalho. O primeiro código é a Paridade, que foi introduzida aqui por motivos históricos porque não se classifica como um código corretor de erros como será visto mais a diante. Os códigos BCH, Hamming e Reed-Solomon possuem diferentes níveis de detecção e correção que serão descritos em seguida.

Por definição, ECC são códigos que possuem a capacidade de detecção e correção de erros ocorridos sobre um determinado conjunto de dados. No geral, os códigos corretores estão baseados em um mesmo princípio: dados de redundância são adicionados à informação para corrigir possíveis erros que podem ocorrer em processos de armazenamento e transmissão de dados. Na forma mais usual, símbolos de redundância são anexados aos símbolos de informação para obter uma sequência de código denominada *codeword*. Os símbolos de redundância são também identificados de paridade e são chamados assim por razões históricas devido a sua origem vir da paridade que será apresentada na próxima seção (FREGNI; SARAIVA, 1995)

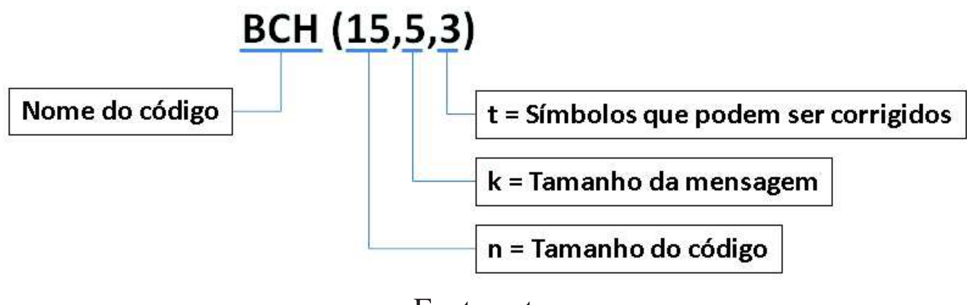
A Figura 5 mostra a estrutura do ECC, composta por: informação (*user data*) e a paridade (*Parity*), a soma desses dois campos chama-se palavra de código, em inglês *Codeword*.

Figura 5 – Estrutura Básica de ECC.

Fonte:Tanakamaru, Yanagihara e Takeuchi (2013).

Sobre a notação utilizada em ECC a mais usual é escrever o nome do código e entre parênteses o tamanho n do *codeword*, o tamanho k da informação e algumas vezes o tamanho t de símbolos detectados e corrigidos), conforme se observa na Figura 6.

Figura 6 – Exemplo de Notação de Códigos ECC.



Fonte:autor

A implementação do ECC pode ser feita basicamente de 3 formas:

- Internamente ao CI do controlador de memória Flash;
- Internamente ao CI de memória NAND Flash;
- Em software implementado fora da placa.

A Figura 7 mostra a comparação das 3 localizações do ECC e a tabela comparativa com as informações do local de implementação, velocidade e o custo.

Figura 7 – Localização do ECC e Tabela Comparativa

Localização do ECC

Controlador de NAND com ECC
Controlador de NAND
ECC
Memória NAND

Controlador de NAND
ECC
Memória NAND
Memória NAND com ECC

Controlador de NAND
ECC
Memória NAND

Localização	Dentro do controlador de NAND	Dentro da memória NAND	Externo
implementação	hardware	hardware	Software

Fonte:baseado em (MICHELONI; MARELLI; RAVASIO, 2008)

Na Figura 7, mostram-se as possibilidades de implementação do ECC em memórias NAND Flash, a qual possui alta velocidade de codificação e decodificação, porém possui custo elevado em relação a implementação externa feita em software.

A implementação do ECC em software é feita externamente utilizando a CPU e as memórias do computador para calcular e armazenar os bits durante os cálculos. Já a implementação em hardware do ECC dentro do CI de memória NAND Flash possui alta velocidade de codificação e decodificação como descreve Tanzawa et al. (1997) em seu artigo, porém o CI apresenta um custo maior em relação à implementação do ECC dentro do controlador por ser ter uma aplicação mais restrita e baixo volume de venda. Existem também as implementações mistas com parte do ECC no controlador e parte externa, ou parte do ECC dentro do CI da NAND Flash e parte no controlador.

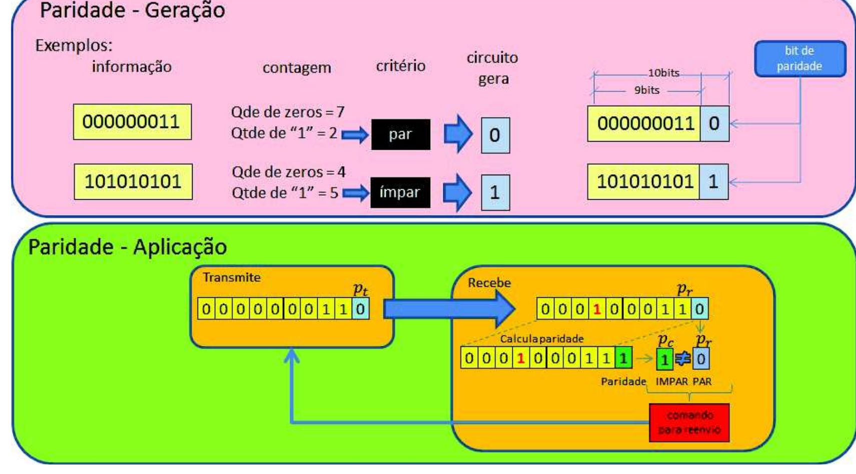
3.1 Paridade

A paridade é a forma mais simples e tradicional de conferir a ocorrência de erros simples na leitura ou armazenamento de alguma unidade de memória (FREGNI; SARAIVA, 1995).

A ideia básica da paridade é adicionar-se ao conjunto de bits de informação um bit adicional: o bit de paridade. Seu valor, zero ou um, será ajustado de acordo com os valores dos demais bits de forma que o total de bits de nível do conjunto seja par ou ímpar de acordo com a convenção adotada. Na leitura, se algum bit se inverter, a paridade do conjunto ficará incorreta e indica a existência de um erro. Entretanto se dois bits se inverterem não haverá detecção pela paridade (FREGNI; SARAIVA, 1995).

A paridade mostrada na Figura 8 mostra uma informação de 9 bits que é codificada gerando-se o bit de paridade e depois é feita a leitura e a verificação se a informação recebida está correta.

Figura 8 – Paridade.



Fonte: Autor

A Figura 8 mostra o uso da paridade, sendo 9 bits de informação e 1 bit de paridade. Antes do envio da informação, é feita uma codificação com os bits de dados e é gerado um bit de paridade que é adicionado aos bits de informação. Na recepção é feita uma nova codificação com

os bits de dados que são comparados com o bit de paridade. Se os bits são iguais não há erro, se forem diferentes a mensagem é desprezada e é requisitado o um novo envio da mensagem.

3.1.1 Codificação

No processo de codifição utilizou-se o circuito mais comum composto por portas do tipo ou-exclusivo (XOR). Neste circuito o bit de paridade é produzido através da lógica XOR com todos os bits da informação. A Figura 9 mostra o circuito de codificação de paridade para uma informação de 9 bits e a geração do bit de paridade P. A informação x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 gera o bit de paridade P_t onde o índice t denomina a transmissão.

Figura 9 – Circuito Gerador de Paridade

Fonte: adaptado de Fregni e Saraiva (1995)

O circuito original em Fregni e Saraiva (1995) utilizava 8 bits e foi adaptado para 9 bits para o exemplo da Figura 8, o qual foi utilizado no estudo apresentado no capítulo 4 desta dissertação.

3.1.1.1 Exemplo de Codificação

Utilizando-se uma informação de 9 bits, como exemplo 101010101_2. A codifição é feita calculando-se XOR 2 a 2 com os bits da informação. Assim chega-se a seguinte situação:

1 xor 0 xor 1 xor 0 xor 1 xor 0 xor 1 xor 0 xor 1 = 1

O valor de P é igual a 1, conforme demonstrado, a informação codificada resulta em: $101010101\mathbf{1}_2$

3.1.2 Decodificação

Na decodificação utiliza-se o mesmo método da codificação, utilizando o circuito da Figura 9 para obter um novo valor de P que é chamado de P_c de c=calculado. Compara-se o valor da paridade recebida P_r com o valor obtido P_c, se o valor for igual conclui-se que não houve erros, se forem diferentes há um erro na informação recebida que é desprezada e o circuito decodificador pede para que a informação seja reenviada (FREGNI; SARAIVA, 1995).

Este método apresenta a limitação de detecção de apenas 1 erro, pois se existirem 2 erros ou mais, um erro pode mascarar o outro não havendo detecção e correção.

3.1.2.1 Exemplo de Decodificação

Supondo que seja recebida a informação: $101010101\mathbf{1}_2$ obtém-se $P_r=1_2$ e a informação 101010101_2 . Na decodificação é calculado um novo bit de paridade, P_c resultando na expressão: 1 xor 0 xor 1 xor 0 xor 1 xor 0 xor 1 xor 0 xor 1 = 1 = P_c . Assim como $P_c=P_r=1$ não há erro e não há necessidade da informação ser reenviada.

3.1.3 ECC em Módulos de Memória DRAM

A paridade é chamada erroneamente de ECC pelos fabricantes de módulos de memória DRAM, como mostrado no *datasheet* da Figura 10, entretanto nessas memórias não há correção da informação, apenas a detecção.

Figura 10 – *Datasheet* Módulo de Memória DRAM com ECC

Micron

8GB (x72, ECC, SR) 288-Pin DDR4 RDIMM
Features

DDR4 SDRAM RDIMM

MTA18ASF1G72PZ – 8GB

Features

- DDR4 functionality and operations supported as defined in the component data sheet
- 288-pin, registered dual in-line memory module (RDIMM)
- Fast data transfer rates: PC4-2666, PC4-2400, or PC4-2133
- 8GB (1 Gig x 72)
- V_{DD} = 1.20V (NOM)
- V_{PP} = 2.5V (NOM)
- V_{DDSPD} = 2.5V (NOM
- Supports ECC error detection and correction
- Nominal and dynamic on-die termination (ODT) for data, strobe, and mask signals
- Low-power auto self refresh (LPASR)
- On-die V_{REFDQ} generation and calibration
- Single-rank
- Onboard I²C temperature sensor with integrated serial presence-detect (SPD) EEPROM
- 16 internal banks; 4 groups of 4 banks each
- Fixed burst chop (BC) of 4 and burst length (BL) of 8 via the mode register set (MRS)
- Selectable BC4 or BL8 on-the-fly (OTF)
- Gold edge contacts

Figure 1: 288-Pin RDIMM (MO-309, R/C-C)

Module height: 31.25mm (1.23in)

Figure 2: 288-Pin RDIMM (MO-309, R/C-C1)

Module height: 31.25mm (1.23in)

Options	Marking
• Operating temperature	
– Commercial (0°C ≤ T_{OPER} ≤ 95°C)	None
• Package	
– 288-pin DIMM (halogen-free)	Z
• Frequency/CAS latency	
– 0.75ns @ CL = 19 (DDR4-2666)	-2G6
– 0.83ns @ CL = 17 (DDR4-2400)	-2G3
– 0.93ns @ CL = 15 (DDR4-2133)	-2G1

Fonte: (Micron, 2013)

A Figura 10 mostra o *datasheet*, de um módulo de memória DRAM com ECC do fabricante Micron. O *datasheet* menciona detecção e correção, a detecção é feita através de 1 bit no final de cada palavra, porém na correção a recuperação da informação não é feita utilizando o bit de paridade, um circuito requisita que a mensagem seja reenviada em caso de detecção de falha. As informações podem ser encontradas no *website* do fabricante Micron (www.micron.com) no *datasheet* Micron (2012).

3.2 Código de Bose-Chaudhuri-Hocquenghem (BCH)

O código BCH é capaz de múltiplas detecções e correções de erros e foi primeiramente criado por A. Hocquenghem em 1959 e independentemente em 1960 por R. C. Bose e D.K. Ray-Chaudhuri (BOSE; RAY-CHAUDHURI, 1960).

Neste trabalho não serão apresentadas as demonstrações matemáticas do código BCH por não fazer parte do seu escopo. Entretanto, para um melhor entendimento do código são necessários alguns conceitos matemáticos que serão descritos a seguir.

3.2.1 Códigos Cíclicos Binários

Os códigos cíclicos binários são uma classe de códigos corretores de erros que são eficientemente codificados e decodificados utilizando simples registradores de deslocamentos e elementos de lógica combinacional, sendo sua representação baseada em polinômios (ALMEIDA, 2011).

A seguir é mostrado um exemplo de informação binária escrita na forma polinomial.

A informação 1011_2 na forma polinomial é escrita como:

$$1*x^3+0*x^2+1*x^1+1*x^0$$

3.2.2 Nomenclaturas Utilizadas pelo Código BCH

Conforme definido em Almeida (2011) **m** é o grau do polinômio primitivo, **k** é o comprimento da informação, **t** é o número máximo de erros corrigíveis, **n** é a largura do *codeword* e **n-k** é o número de bits de paridade. Portanto, resumidamente os elementos utilizados na codificação BCH são:

- **m** = grau do polinômio primitivo
- **k** = comprimento da informação
- **t** = número máximo de erros corrigíveis
- **n** = 2^m-1 largura da *codeword*
- n-k $\leq$ m·t número de bits de paridades

Jiang (2010) em seu trabalho, mostra em uma tabela as principais características dos códigos BCH, reproduzida na Tabela 1.

Tabela 1 – Principais Características dos Códigos BCH para m$\geq$3

Comprimento do Código	n=2^m-1
Número de bits de Informação	$k \geq 2^m-1-m\cdot t$
Número de bits de Paridade	$n-k \leq m\cdot t$
Capacidade de Correção	t erros

Fonte: (JIANG, 2010)

Para demonstrar as principais características do código BCH mostradas na Tabela 1 aplicada para m=3 a m=8, foi construida a Tabela 2 em que mostra a variação de n, k e t em função de m variando de 3 a 8.

Tabela 2 – Tabela Característica de BCH de m=3 a m=8

m	n	k	t	m	n	k	t	m	n	k	t	m	n	k	t
3	7	4	1			120	1			247	1			115	22
		11	1			113	2			239	2			107	23
4	15	7	2			106	3			231	3			99	24
		5	3			99	4			223	4			91	25
		26	1			92	5			215	5			87	26
		21	2			85	6			207	6			79	27
5	31	16	3			78	7			199	7			71	29
		11	5			71	9			191	8			63	30
		6	7	7	127	64	10	8	255	187	9	8	255	55	31
		57	1			57	11			179	10			47	42
		51	2			50	13			171	11			45	43
		45	3			43	14			163	12			37	45
		39	4			36	14			155	13			29	47
		36	5			29	21			147	14			21	55
6	63	30	6			22	23			139	18			13	59
		24	7			15	27			131	19			9	63
		18	10			8	31			123	21				
		16	11												
		10	13												
		7	15												

Fonte: adaptado de Jiang (2010)

A Tabela 2 mostra os valores de com as características do código BCH com os valores de m=3 a m=8.

A seguir serão introduzidos alguns termos que serão necessários tanto para a codificação e decodificação do código BCH quanto para o Reed-Solomon, descrito posteriormente.

3.2.3 O Polinômio Gerador

O polinômio gerador é especificado em termos das raízes do Corpo de Galois GF(2^m) e é o polinômio de menor grau sobre GF(2^m) que tem como raízes: $\alpha, \alpha^2, \alpha^3, ... \alpha^{2t}$ e é utilizando para determinar os valores das paridades (JIANG, 2010).

3.2.4 Corpo de Galois ou *Galois Field* (GF)

O corpo de Galois GF(2^m) é baseado em polinômios primitivos de ordem m, onde polinômios primitivos são polinômios que não podem ser fatorados (ALMEIDA, 2011).

Tabela 3 – Tabela de Polinômios Primitivos em Função de m

m		m	
3	x^3+x+1	14	$x^{14}+x^{10}+x^6+x+1$
4	x^4+x+1	15	$x^{15}+x+1$
5	x^5+x^2+1	16	$x^{16}+x^{12}+x^3+x+1$
6	x^6+x+1	17	$x^{17}+x^3+1$
7	x^7+x^3+1	18	$x^{18}+x^7+1$
8	$x^8+x^4+x^3+x^2+1$	19	$x^{19}+x^5+x^2+x+1$
9	x^9+x^4+1	20	$x^{20}+x^3+1$
10	$x^{10}+x^3+1$	21	$x^{21}+x^3+1$
11	$x^{11}+x^2+1$	22	$x^{22}+x+1$
12	$x^{12}+x^6+x^4+x+1$	23	$x^{23}+x^5+1$
13	$x^{13}+x^4+x^3+x+1$	24	$x^{24}+x^7+x^2+x+1$

Fonte: (ALMEIDA, 2011)

A Tabela 3 mostra e polinômios primitivos para m de 3 a 24.

Os polinômios primitivos são conhecidos e tabelados, e podem ser encontrados na literatura como por exemplo em (ALMEIDA, 2011), (PETERSON; BROWN, 1961)

Como exemplo utilizou-se a construção da tabela com os valores do Corpo de Galois, para m = 5. Pela Tabela 3 o polinômio primitivo correspondente para m=5 é x^5+x^2+1, de onde são calculados os valores da Tabela 4 do Corpo de Galois (ALMEIDA, 2011). A seguir, apresenta-se a Tabela 4 para o qual utilizou-se os valores: m=5, GF(2^5)=GF(32) .

Tabela 4 – Tabela de Corpo de Galois para x^5+x^2+1

Representação por Potência	Representação Polinomial	Representação Vetorial
0	0	00000
$\alpha^0=\alpha^{32}=1$	1	00001
α	α	00010
α^2	α^2	00100
α^3	α^3	01000
α^4	α^4	10000
α^5	α^2+1	00101
α^6	$\alpha^3+\alpha$	01010
α^7	$\alpha^4+\alpha^2$	10100
α^8	$\alpha^3+\alpha^2+1$	01101
α^9	$\alpha^4+\alpha^3+\alpha$	11010
α^{10}	α^4+1	10001
α^{11}	$\alpha^2+\alpha+1$	00111
α^{12}	$\alpha^3+\alpha^2+\alpha$	01110
α^{13}	$\alpha^4+\alpha^3+\alpha^2$	11100
α^{14}	$\alpha^4+\alpha^3+\alpha^2+1$	11101
α^{15}	$\alpha^4+\alpha^3+\alpha^2+\alpha+1$	11111
α^{16}	$\alpha^4+\alpha^3+\alpha+1$	11011
α^{17}	$\alpha^4+\alpha+1$	10011
α^{18}	$\alpha+1$	00011
α^{19}	$\alpha^2+\alpha$	00110
α^{20}	$\alpha^3+\alpha^2$	01100
α^{21}	$\alpha^4+\alpha^3$	11000
α^{22}	α^2+1	10101
α^{23}	$\alpha^3+\alpha^2+\alpha+1$	01111
α^{24}	$\alpha^4+\alpha^3+\alpha^2+\alpha$	11110
α^{25}	$\alpha^4+\alpha^2+\alpha+1$	11001
α^{26}	$\alpha^4+\alpha^2+\alpha+1$	10111
α^{27}	$\alpha^3+\alpha+1$	01011
α^{28}	$\alpha^4+\alpha^2+\alpha$	10110
α^{29}	α^3+1	01001
α^{30}	$\alpha^4+\alpha$	10010
α^{31}	α^3+1	01001

Fonte: adaptado de Mozhiarasi e Gayathri (2014)

A Tabela 4 de GF(32) foi construída baseada no polinômio mínimo $x^5 + x^2 + 1$ e desenvolvida conforme a sequência demonstrada a seguir:

$$
\begin{aligned}
\alpha^0 &= 1 \rightarrow 00001_2 \\
\alpha^1 &= \alpha \rightarrow 00010_2 \\
\alpha^2 &= \alpha^2 \rightarrow 00100_2 \\
\alpha^3 &= \alpha^3 \rightarrow 01000_2 \\
\alpha^4 &= \alpha^4 \rightarrow 10000_2 \\
x^5 = x^2 + 1 \rightarrow \alpha^5 &= \alpha^2 + 1 \rightarrow 00101_2 \\
x^6 = x^3 + x \rightarrow \alpha^6 &= \alpha^3 + \alpha \rightarrow 01010_2 \\
&. \\
&. \\
&. \\
\alpha^{31} &= \alpha^3 + 1 \rightarrow 01001_2
\end{aligned}
$$

3.2.5 Operações Utilizando Corpo de Galois

As operações com os elementos do Corpo de Galois não são usuais. Para demonstrar as operações de multiplicação, soma e módulo, utilizando-se os valores da Tabela 4 conforme mostrado no exemplo:
Para valores de m=5, GF(2^5=32) da Tabela 4:

$$
\begin{aligned}
multiplicao\ \ & \alpha^2 \times \alpha^3 = \alpha^{2+3} = \alpha^5 \\
soma\ \ & \alpha^2 + \alpha^3 = 00100 \oplus 01000 = 01100 = \alpha^{20} \\
modulo\ \ & \alpha^{35} = \alpha^{32+3} = \alpha^{32} \times \alpha^3 = \alpha^3
\end{aligned}
$$

3.2.6 Polinômios Mínimos

Os polinômios mínimos para um corpo finito podem ser encontrados em tabelas como no apêndice de Justesen e Hoholdt (2004) como mostrado na Tabela 5.

Tabela 5 – Tabela de Polinômios Mínimos para GF(32)

	Polinômios Mínimos m=5
$\varphi_1(x)$	$x^5 + x^2 + 1$
$\varphi_3(x)$	$x^5 + x^4 + x^3 + x^2 + 1$
$\varphi_5(x)$	$x^5 + x^4 + x^2 + x + 1$
$\varphi_7(x)$	$x^5 + x^3 + x^2 + x + 1$
$\varphi_{11}(x)$	$x^5 + x^4 + x^3 + x + 1$
$\varphi_{15}(x)$	$x^5 + x^3 + 1$

Fonte: adaptado de (JUSTESEN; HOHOLDT, 2004)

A Tabela 5 mostra os polinômios mínimos para o Corpo de Galois GF(32). A Figura 11 mostra a estrutura básica de um código BCH.

Figura 11 – Estrutura Código BCH

Fonte: adaptado de Micheloni, Marelli e Ravasio (2008)

Na Figura 11 observa-se na entrada do bloco codificador a informação **m(x)**, sua codificação em **c(x)** na saída do bloco e o armazenamento na memória *Flash*. A decodificação é feita em várias etapas começando pelo bloco de Cálculo das Síndromes, que recebe a informação **r(x)** da memória e calcula as síndromes **S**. As síndromes entram no bloco de Cálculo do Polinômio de Erros que calculam os fatores σ para o polinômio de erros utilizando pelo bloco Cálculo da Localização dos Erros que calcula as raízes **x** do polinômio de erros e indicam a posições dos erros no *codeword* c(x). No final os erros são corrigidos invertendo-se os valores dos bits errados, os bits de paridade são retirados e na saída tem-se novamente a informação **m(x)**.

3.2.7 Divisão de Polinômios

Para a codificação é necessário realizar uma divisão polinomial da informação pelo polinômio gerador para o cálculo da paridade. Para cálculos com polinômios com poucos termos pode-se fazer o cálculo manualmente como mostrado em a), porém quando o polinômio tem centenas ou milhares de termos o método manual não é mais viável e normalmente utilizam-se recursos de hardware ou computacionais para fazer a divisão. Existem vários métodos de divisão de polinômios entre os quais o algoritmo manual e do registrador de deslocamento. A seguir será mostrado uma comparação de divisão polinomial feita por 2 métodos diferentes; o método manual no item a) e o método utilizando o registrador de deslocamento no item b):

- a) Divisão polinomial
  Exemplo Divisão Polinomial: x^{18}+x^{16}+x^{14}+x^{12}+x^{10} mod $x^{10}+x^9+x^8+x^6+x^5+x^3+1$

$$
\begin{array}{l|l}
x^{18}+x^{16}+x^{14}+x^{12}+x^{10} & x^{10}+x^{9}+x^{8}+x^{6}+x^{5}+x^{3}+1 \\
\underline{x^{18}+x^{17}+x^{16}+x^{14}+x^{13}+x^{11}+x^{8}} & x^{8}+x^{7}+x^{6}+x^{4}+x^{3}+x^{2} \\
\quad x^{17}+x^{13}+x^{12}+x^{11}+x^{10}+x^{8} & \\
\quad \underline{x^{17}+x^{16}+x^{15}+x^{13}+x^{12}+x^{10}+x^{7}} & \\
\qquad x^{16}+x^{15}+x^{11}+x^{8}+x^{7} & \\
\qquad \underline{x^{16}+x^{15}+x^{14}+x^{12}+x^{11}+x^{9}+x^{6}} & \\
\qquad\quad x^{14}+x^{12}+x^{9}+x^{8}+x^{7}+x^{6} & \\
\qquad\quad \underline{x^{14}+x^{13}+x^{12}+x^{10}+x^{9}+x^{7}+x^{4}} & \\
\qquad\qquad x^{13}+x^{10}+x^{8}+x^{6}+x^{4} & \\
\qquad\qquad \underline{x^{13}+x^{12}+x^{11}+x^{9}+x^{8}+x^{6}+x^{3}} & \\
\qquad\qquad\quad x^{12}+x^{11}+x^{10}+x^{9}+x^{4}+x^{3} & \\
\qquad\qquad\quad \underline{x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{2}} & \\
\qquad\qquad\qquad x^{9}+x^{8}+x^{7}+x^{5}+x^{4}+x^{3}+x^{2} &
\end{array}
$$

resto da divisão: $r_a(x) = x^9+x^8+x^7+x^5+x^4+x^3+x^2 = 1110111100_2$

A codificação final fica:

$$c_a(x) = x^{n-k} \times m(x) + r_a(x)$$
$$101010101\ 00000000_2 + \mathit{1110111100_2} \rightarrow c_a(x) = 101010101\ \mathit{1110111100_2}$$

- b) Divisão com registradores de deslocamento de realimentação linear ou em inglês *Linear Feedback Shift Register* (LFSR). Peterson e Brown (1961) fizeram estudos com registradores de deslocamento para divisão de polinômios, como mostrado na Figura 12.

Figura 12 – Diagrama de Divisor Polinomial e Tabela de Estados

divisor
(polinômio gerador)

$x^{10}+x^9+x^8+x^6+x^5+x^3+1$

entrada de bits

A B C D E F G H I J

x^{10} x^9 x^8 x^6 x^5 x^3 1

	Estado do Registrador											
passo	A	B	C	D	E	F	G	H	I	J	bit de entrada	
cond inicial	0	0	0	0	0	0	0	0	0	0		
1	0	0	0	0	0	0	0	0	0	0	0	31
2	0	0	0	0	0	0	0	0	0	0	0	30
3	0	0	0	0	0	0	0	0	0	0	0	29
4	0	0	0	0	0	0	0	0	0	0	0	28
5	0	0	0	0	0	0	0	0	0	0	0	27
6	0	0	0	0	0	0	0	0	0	0	0	26
7	0	0	0	0	0	0	0	0	0	0	0	25
8	0	0	0	0	0	0	0	0	0	0	0	24
9	0	0	0	0	0	0	0	0	0	0	0	23
10	0	0	0	0	0	0	0	0	0	0	0	22
11	0	0	0	0	0	0	0	0	0	0	0	21
12	0	0	0	0	0	0	0	0	0	0	0	20
13	0	0	0	0	0	0	0	0	0	1	1	19
14	0	0	0	0	0	0	0	0	1	0	0	18
15	0	0	0	0	0	0	0	1	0	1	1	17
16	0	0	0	0	0	0	1	0	1	0	0	16
17	0	0	0	0	0	1	0	1	0	1	1	15
18	0	0	0	0	1	0	1	0	1	0	0	14
19	0	0	0	1	0	1	0	1	0	1	1	13
20	0	0	1	0	1	0	1	0	1	0	0	12
21	0	1	0	1	0	1	0	1	0	1	1	11
22	1	0	1	0	1	0	1	0	1	0	0	10
23	1	0	0	0	1	1	1	1	0	1	0	9
24	1	1	0	0	0	1	0	0	1	1	0	8
25	0	1	0	1	0	0	1	1	1	1	0	7
26	1	0	1	0	0	1	1	1	1	0	0	6
27	1	0	0	1	0	1	0	1	0	1	0	5
28	1	1	1	1	0	0	0	0	1	1	0	4
29	0	0	1	1	1	0	1	1	1	1	0	3
30	0	1	1	1	0	1	1	1	1	0	0	2
31	1	1	1	0	1	1	1	1	0	0	0	1

dividendo
(mensagem)

resto
(paridade)

Fonte: autor baseado em Peterson e Brown (1961)

O polinômio gerador (divisor) é representado pelo circuito com os registradores de deslocamento, onde o número de flip-flops é igual ao grau do polinômio gerador e cada termo do polinômio gerador g(x): x^{10}, x^9, x^8, x^6, x^5, x^3 e 1 são realimentações de x^{10} representados pelos fios verticais . A informação é o dividendo que entra pelo lado direito do circuito. Na tabela com os estados do registrador observam-se os bits da informação na coluna "bits de entrada"e a última linha da tabela é o resto da divisão que o valor da paridade procurada.

Resultado da divisão: $r_b(x) = 1110111100_2$

A codificação final fica:

$$c_b(x) = x^{n-k} \times m(x) + r_b(x)$$

$$101010101\ 00000000_2 + \mathit{1110111100}_2 \rightarrow c_b(x) = 101010101\ \mathit{1110111100}_2$$

O valor de c_b(x) obtido é o mesmo de c_a(x) do item a) através da divisão manual.
item a) c_a(x) = 101010101 *1110111100*$_2$
item b) c_b(x) = 101010101 *1110111100*$_2$

Observa-se que os 13 primeiros termos da tabela da Figura 12 são iguais a zero, isso é deve-se ao fato de se estar utilizando BCH(31,21,2) com n=31 e k=21. O grau do polinômio do dividendo é 31 e a informação a ser codificada é 18 [$x^{n-k} = x^{10} \times$m(x)], a diferença de n-k (n-k = 31-18 = 13) são 13 termos iguais a zero, porém t=2 e a quantidade de bits de paridade n-k=10 são mantidos, assim conclui-se que o BCH(31,21,2)=BCH(19,9,2).

3.2.8 Codificação

Para fazer a codificação de uma informação no código BCH, calcula-se os bits de paridade fazendo a divisão polinomial entre a informação e um polinômio gerador, o resto da divisão serão os bits de paridade. Para codificar uma informação m(x) em BCH, seguem os procedimentos conforme Bose (2008):
- Escolher um polinômio primitivo de grau m e construir GF(2^m);
- Encontrar f_i(x) o polinômio mínimo de α^i para i=1,..., 2t;
- Achar o polinômio gerador para corrigir t erros que é dado por g(x)=MMC[f_1(x)... f_{2t}(x)];
- Multiplicar a informação m(x) por x^{n-k};
- Achar o resto da divisão de [$x^{n-k} \times$ m(x)] mod g(x) = b(x);
- Somar [$x^{n-k} \times$m(x)] com o resto da divisão b(x) = [$x^{n-k} \times$m(x)] mod g(x);
- Obtém-se o *codeword* c(x) = m(x) + b(x).

Para codificação do código BCH utiliza-se de um polinômino gerador g(x) que é obtido através da Tabela 5 e executa-se;

- Para corrigir 2 erros: g_2(x)=MMC $\varphi_1(x)$, $\varphi_3(x)$

  $g_2(x) = \varphi_1(x) * \varphi_3(x) = (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + 1)$

$$g_2(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1 \tag{1}$$

- Para corrigir 3 erros: g_3(x)=MMC $\varphi_1(x)$, $\varphi_3(x)$, $\varphi_5(x)$

  $g_3(x) = \varphi_1(x) * \varphi_3(x) * \varphi_5(x) = (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + 1)(x^5 + x^4 + x^2 + x + 1)$

$$g_3(x) = x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1 \tag{2}$$

3.2.8.1 Exemplo de Codificação

Para exemplificar a codificação de uma informação de 9 bits $(101010101)_2$ com correção de 2 bits, utiliza-se a Tabela 2, para k=9 e t=2, encontram-se m=5 e n=31
Assim: Para BCH(31,21,2) = BCH(n,k,t) tem-se:

- n=15 bits
- k=9 bits –>101010101_2
- t=2 bits –> correção de 2 erros
- m=5 –> da Tabela 11 para BCH(31,21,2)

Resolução:

Para a correção de 2 bits errados, da equação 25 tem-se o polinômio gerador:

$$g_2(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$$

Da informação 101010101_2 obtém-se o polinômio m(x):

$$\text{m(x)} = 1 * x^8 + 0 * x^7 + 1 * x^6 + 0 * x^5 + 1 * x^4 + 0 * x^3 + 1 * x^2 + 0 * x^1 + 1 * x^0$$
$$m(x) = x^8 + x^6 + x^4 + x^2 + 1$$

O grau do polinômio m(x) é menor que o grau de $g_2(x)$, para isso é necessário fazer o ajuste do grau do polinômio m(x) para a divisão por $g_2(x)$, multiplicando m(x) por x^{n-k}:

$$x^{n-k} = x^{31-21} = x^{10}$$
$$x^{n-k} \times m(x) = x^{10} \times (x^8+x^6+x^4+x^2+1) = x^{18}+x^{16}+x^{14}+x^{12}+x^{10}$$

Para determinar os bits de paridade a informação é dividida pelo polinômio gerador para obter o resto da divisão:

Dividir $[x^{n-k} \times m(x)]$ por $g_2(x)$ para obter o resto
$$x^{18}+x^{16}+x^{14}+x^{12}+x^{10} \text{ mod } x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$$

Obtém-se os 10 bits de paridade:

$$x^9 + x^8 + x^7 + x^5 + x^4 + x^3 + x^2 = \mathit{1110111100}_2$$

Assim chega-se ao *codeword* c(x) com n=15 bits:

$$c(x) = \mathbf{m(x)} + \mathit{paridade} = \mathbf{101010101}_2 + \mathit{1110111100}_2$$
$$\mathit{codeword}\ c(x) = \underbrace{101010101}_{m(x)}\ \underbrace{1110111100_2}_{paridade}$$

3.2.9 Decodificação

A decodificação de uma informação ou *codeword* é feita em 3 etapas, são elas:

- a) Cálculo das síndromes
  - se $S_1 = S_3 = S_5 = 0$ não há erro na mensagem recebida
  - se S_1 ou S_3 ou $S_5 \neq 0$ há erro(s) na mensagem recebida
- b) Cálculo do polinômio de erros
- c) Cálculo das posições de erros

Para o cálculo das síndromes do item a) **Cálculo das Síndromes**
As síndromes podem ser calculadas de 2 formas:

- a1) através de α^n, onde n é ímpar e utilizando Corpo de Galois, normalmente utilizado para polinômios de grau menores que 20:

$$S_1 = r(\alpha^1) \tag{3}$$
$$S_3 = r(\alpha^3) \tag{4}$$
$$S_5 = r(\alpha^5) \tag{5}$$

- a2) através de divisão polinomial com LFSR utilizada para graus grandes (acima de 20):

$$S_1 = r(\alpha^1) mod f_1(\alpha^1) \qquad (resto\ da\ divisao\ de\ r(x)\ por\ f_1(x)) \tag{6}$$
$$S_3 = r(\alpha^3) mod f_3(\alpha^3) \qquad (resto\ da\ divisao\ de\ r(x)\ por\ f_3(x)) \tag{7}$$
$$S_5 = r(\alpha^5) mod f_5(\alpha^5) \qquad (resto\ da\ divisao\ de\ r(x)\ por\ f_5(x)) \tag{8}$$

Onde f_1(x), f_3(x) e f_5(x) são polínômios mínimos de GF(2^m).

b) **Cálculo do Polinômio de Erros**
O polinômio de erros pode ser calculado através do algoritmo de Peterson demonstrado em (PETERSON, 1960) que consiste em calcular os valores de σ do polinômio de erros:

$$\sigma(x) = 1 + \sigma_1 * x + \sigma_2 * x^2 ... \sigma_n * x^n$$

A Tabela 6 mostra as fórmulas dos termos do polinômio de erros, σ para valores de **t** entre 1 e 5.

Tabela 6 – Coeficientes do Polinômio Localizador de Erros em Função das Síndromes para $1 \leq t \leq 5$

t	$\sigma(S_t)$
1	$\sigma_1 = S_1$
2	$\sigma_1 = S_1$
	$\sigma_2 = \frac{S_3+S_1^3}{S_1}$
3	$\sigma_1 = S_1$
	$\sigma_2 = \frac{S_1^2S_3+S_5}{S_1^3+S_3}$
	$\sigma_3 = \left(S_1^3+S_3\right)+S_1\sigma_2$
4	$\sigma_1 = S_1$
	$\sigma_2 = \frac{S_1\left(S_7+S_1^7\right)+S_3\left(S_1^5+S_5\right)}{S_3\left(S_1^3+S_3\right)+S_1\left(S_1^5+S_5\right)}$
	$\sigma_3 = \left(S_1^3+S_3\right)+S_1\sigma_2$
	$\sigma_4 = \frac{\left(S_5+S_1^2S_3\right)+\left(S_1^3+S_3\right)\sigma_2}{S_1}$

Fonte: (PETERSON, 1960)

c) **Cálculo das Posições dos Erros**

Para calcular as posições dos erros utiliza-se o algoritmo interativo de Chien, que consiste em substituir em $\sigma(x)$ todos os valores de α para achar as raízes que indicam a posição da falha.

$$\sigma(x) = 1+\sigma_1 * x+\sigma_2 * x^2 ... \sigma_n * x^n$$
$$\sigma(\alpha^0) = 1+\sigma_1 * \alpha^0+\sigma_2 * \alpha^{0*2} ... \sigma_n * \alpha^{0*n}$$
$$\sigma(\alpha^1) = 1+\sigma_1 * \alpha^1+\sigma_2 * \alpha^{1*2} ... \sigma_n * \alpha^{1*n}$$
$$.$$
$$.$$
$$\sigma(\alpha^n) = 1+\sigma_1 * \alpha^n+\sigma_2 * \alpha^{n*2} ... \sigma_n * \alpha^{n*n}$$

Quando o valor da equação for zero, α^n é a raiz da equação, e indica a posição da falha procurada. A posição da falha é indicada por:

Sendo α^n raiz da equação; $1/\alpha^n = \alpha^{-n} \rightarrow$ -n indica a posição do bit com erro em r(x).

Para corrigir o bit com erro, inverte-se o valor para corrigi-lo.

3.2.9.1 Exemplo de Decodificação

Recebeu-se um *codeword* BCH(31,21,2) com os seguintes valores: 10101010**0** 1110111100_2 onde tem-se um erro no bit 10.

Chamando de r(x) a *codeword* recebida:

r(x) = 10101010**0** 1110111100$_2$ = $x^{18}+x^{16}+x^{14}+x^{12}+x^9+x^8+x^7+x^5+x^4+x^3+x^2$

a) **Cálculo das Síndromes:**

Método 1

Substituindo α^1 em r(x) para determinar a síndrome S_1 conforme a equação 3 e utilizando os valores da tabela de Galois de GF(32) da Tabela 4:

$S_1(\alpha) = r(\alpha) = \alpha^{18}+\alpha^{16}+\alpha^{14}+\alpha^{12}+\alpha^9+\alpha^8+\alpha^7+\alpha^5+\alpha^4+\alpha^3+\alpha^2$

$= 00011 \oplus 11011 \oplus 11101 \oplus 01110 \oplus 11010 \oplus 01101 \oplus 10100 \oplus 00101 \oplus 10000 \oplus 01000 \oplus 00100$

=10001=α^{10}

$S_3(\alpha^3) = r(\alpha^3) = \alpha^{3*18}+\alpha^{3*16}+\alpha^{3*14}+\alpha^{3*12}+\alpha^{3*9}+\alpha^{3*8}+\alpha^{3*7}+\alpha^{3*5}+\alpha^{3*4}+\alpha^{3*3}+\alpha^{3*2} = [S(\alpha)]^3 = [\alpha^{10}]^3 = \alpha^{30}$

Resulta em:

$$S_1=\alpha^{10}$$
$$S_3=\alpha^{30}$$

Método 2

Utilizando registradores de deslocamento para fazer a divisão polinomial de r(x) por φ_1(x) para determinar S_1 e dividir r(x) por φ_3(x) para determinar S_3.

Cálculo de S_1:

As equações do polinômios mínimos φ_1(x) e φ_3(x) são obtidos através da Tabela 5 - Tabela de polinômios mínimos para GF(32).

S_1 = r(x) mod φ_1(x)

r(x) = $x^{18}+x^{16}+x^{14}+x^{12}+x^9+x^8+x^7+x^5+x^4+x^3+x^2$

onde φ_1(x) = x^5+x^2+1

Divisão polinominal através LFSR para obter o resto da divisão de r(x) por φ_1(x).

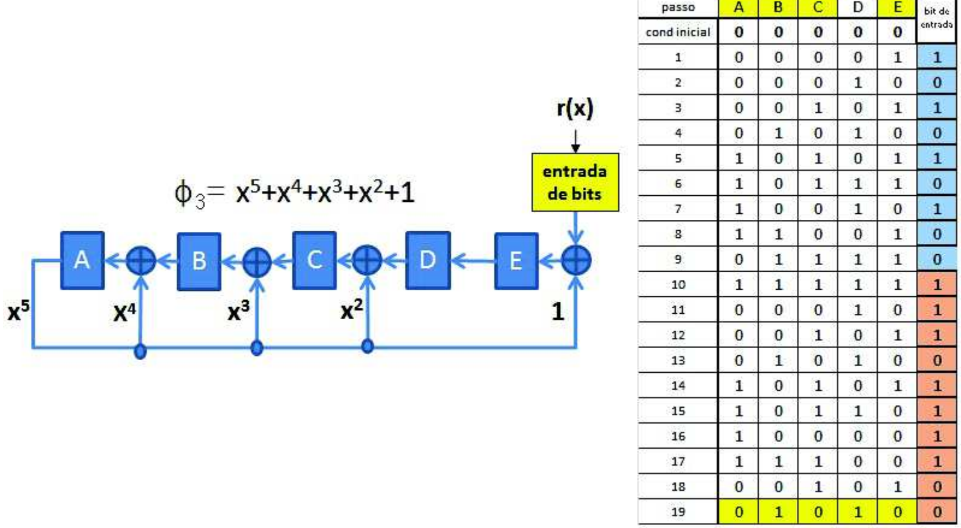
Figura 13 – Circuito Divisor Polinomial para φ_1=x^5+x^2+1

passo	A	B	C	D	E	bit de entrada
cond inicial	0	0	0	0	0	
1	0	0	0	0	1	1
2	0	0	0	1	0	0
3	0	0	1	0	1	1
4	0	1	0	1	0	0
5	1	0	1	0	1	1
6	0	1	1	1	1	0
7	1	1	1	1	1	1
8	1	1	0	1	1	0
9	1	0	0	1	1	0
10	0	0	0	1	0	1
11	0	0	1	0	1	1
12	0	1	0	1	1	1
13	1	0	1	1	0	0
14	0	1	0	0	0	1
15	1	0	0	0	1	1
16	0	0	1	1	0	1
17	0	1	1	0	1	1
18	1	1	0	1	0	0
19	1	0	0	0	1	0

Fonte: adaptado de Peterson e Brown (1961)

O resultado da divisão de r(x) por φ_1(x) resulta como resto 10001_2, que pela Tabela 4 chega-se a α^{10} que é o valor procurado de S_1. O valor de S_3 é obtido dividindo r(x) por φ_3(x) utilizando o circuito da Figura 14.

Figura 14 – Circuito Divisor Polinomial para φ_3=x^5+x^4+x^3+x^2+1



passo	A	B	C	D	E	bit de entrada
cond inicial	0	0	0	0	0	
1	0	0	0	0	1	1
2	0	0	0	1	0	0
3	0	0	1	0	1	1
4	0	1	0	1	0	0
5	1	0	1	0	1	1
6	1	0	1	1	1	0
7	1	0	0	1	0	1
8	1	1	0	0	1	0
9	0	1	1	1	1	0
10	1	1	1	1	1	1
11	0	0	0	1	0	1
12	0	0	1	0	1	1
13	0	1	0	1	0	0
14	1	0	1	0	1	1
15	1	0	1	1	0	1
16	1	0	0	0	0	1
17	1	1	1	0	0	1
18	0	0	1	0	1	0
19	0	1	0	1	0	0

Fonte: adaptado de (PETERSON; BROWN, 1961)

O resto da divisão de r(x) por φ_3(x) é 01010_2 = x^3+x = $\alpha^{3*3} + \alpha^{3*1}$ = $\alpha^9 + \alpha^3$ da Tabela 4: $\alpha^9 \oplus \alpha^3$ = $11010_2 \oplus 01000_2$ = 10010_2 = α^{30} = S_3 .

As síndromes $S_1 = \alpha^{10}$ e $S_3 = \alpha^{30}$ são as mesmas das obtidas pelo Método 1.

b) **Cálculo dos σ**

Para t=2 as equações de $\sigma_1 e \sigma_2$ são obtidos da Tabela 6.

$$\sigma_1 = S_1 = \alpha^{10}$$
$$\sigma_2 = \frac{S_3 + S_1^3}{S_1} = \frac{\alpha^{30} + \alpha^{10\times 3}}{\alpha^{10}} = \frac{\alpha^{30} + \alpha^{30}}{\alpha^{10}} = \frac{0}{\alpha^{10}} = 0$$

Assim chega-se ao polinômio de erros:

$$\begin{aligned} \sigma(x) &= 1 + \sigma_1 x + \sigma_2 x^2 \\ &= 1 + \alpha^{10} x + 0x^2 \\ &= 1 + \alpha^{10} x \end{aligned}$$

c) **Cálculo da Posição dos Erros**

O cálculo das raízes de $\sigma(x) = 1 + \alpha^{10}x$ através do algoritmo interativo de Chien conforme apresentado a seguir (MOZHIARASI; GAYATHRI, 2014):

$$\sigma(\alpha^0) = 1 + \alpha^{10} \times \alpha^0 = 1 + \alpha^{10+0} = \alpha^0 + \alpha^{10} \rightarrow tabela \rightarrow \alpha^4$$
$$\sigma(\alpha^1) = 1 + \alpha^{10} \times \alpha^1 = 1 + \alpha^{10+1} = \alpha^0 + \alpha^{11} \rightarrow tabela \rightarrow \alpha^{19}$$
$$\sigma(\alpha^2) = 1 + \alpha^{10} \times \alpha^2 = 1 + \alpha^{10+2} = \alpha^0 + \alpha^{12} \rightarrow tabela \rightarrow \alpha^{23}$$
$$\sigma(\alpha^3) = 1 + \alpha^{10} \times \alpha^3 = 1 + \alpha^{10+3} = \alpha^0 + \alpha^{13} \rightarrow tabela \rightarrow \alpha^{14}$$
$$\sigma(\alpha^4) = 1 + \alpha^{10} \times \alpha^4 = 1 + \alpha^{10+4} = \alpha^0 + \alpha^{14} \rightarrow tabela \rightarrow \alpha^{13}$$
$$\sigma(\alpha^5) = 1 + \alpha^{10} \times \alpha^5 = 1 + \alpha^{10+5} = \alpha^0 + \alpha^{15} \rightarrow tabela \rightarrow \alpha^{24}$$
$$\sigma(\alpha^6) = 1 + \alpha^{10} \times \alpha^6 = 1 + \alpha^{10+6} = \alpha^0 + \alpha^{16} \rightarrow tabela \rightarrow \alpha^9$$
$$\sigma(\alpha^7) = 1 + \alpha^{10} \times \alpha^7 = 1 + \alpha^{10+7} = \alpha^0 + \alpha^{17} \rightarrow tabela \rightarrow \alpha^{30}$$
$$\sigma(\alpha^8) = 1 + \alpha^{10} \times \alpha^8 = 1 + \alpha^{10+8} = \alpha^0 + \alpha^{18} \rightarrow tabela \rightarrow \alpha^1$$
$$\sigma(\alpha^9) = 1 + \alpha^{10} \times \alpha^9 = 1 + \alpha^{10+9} = \alpha^0 + \alpha^{19} \rightarrow tabela \rightarrow \alpha^{11}$$
$$\sigma(\alpha^{10}) = 1 + \alpha^{10} \times \alpha^{10} = 1 + \alpha^{10+10} = \alpha^0 + \alpha^{20} \rightarrow tabela \rightarrow \alpha^8$$
$$\sigma(\alpha^{11}) = 1 + \alpha^{10} \times \alpha^{11} = 1 + \alpha^{10+11} = \alpha^0 + \alpha^{21} \rightarrow tabela \rightarrow \alpha^{25}$$
$$\sigma(\alpha^{12}) = 1 + \alpha^{10} \times \alpha^{12} = 1 + \alpha^{10+12} = \alpha^0 + \alpha^{22} \rightarrow tabela \rightarrow \alpha^7$$
$$\sigma(\alpha^{13}) = 1 + \alpha^{10} \times \alpha^{13} = 1 + \alpha^{10+13} = \alpha^0 + \alpha^{23} \rightarrow tabela \rightarrow \alpha^{12}$$
$$\sigma(\alpha^{14}) = 1 + \alpha^{10} \times \alpha^{14} = 1 + \alpha^{10+14} = \alpha^0 + \alpha^{24} \rightarrow tabela \rightarrow \alpha^{15}$$
$$\sigma(\alpha^{15}) = 1 + \alpha^{10} \times \alpha^{15} = 1 + \alpha^{10+15} = \alpha^0 + \alpha^{25} \rightarrow tabela \rightarrow \alpha^{21}$$

$$\sigma(\alpha^{16}) = 1+\alpha^{10}\times\alpha^{16} = 1+\alpha^{10+16} = \alpha^0+\alpha^{26} \rightarrow tabela \rightarrow \alpha^{28}$$
$$\sigma(\alpha^{17}) = 1+\alpha^{10}\times\alpha^{17} = 1+\alpha^{10+17} = \alpha^0+\alpha^{27} \rightarrow tabela \rightarrow \alpha^{6}$$
$$\sigma(\alpha^{18}) = 1+\alpha^{10}\times\alpha^{18} = 1+\alpha^{10+18} = \alpha^0+\alpha^{28} \rightarrow tabela \rightarrow \alpha^{26}$$
$$\sigma(\alpha^{19}) = 1+\alpha^{10}\times\alpha^{19} = 1+\alpha^{10+19} = \alpha^0+\alpha^{29} \rightarrow tabela \rightarrow \alpha^{3}$$
$$\sigma(\alpha^{20}) = 1+\alpha^{10}\times\alpha^{20} = 1+\alpha^{10+20} = \alpha^0+\alpha^{30} \rightarrow tabela \rightarrow \alpha^{17}$$
$$\sigma(\alpha^{21}) = 1+\alpha^{10}\times\alpha^{21} = 1+\alpha^{10+21} = \alpha^0+\alpha^{31} = \alpha^0+\alpha^0 \rightarrow \mathbf{0}\ \ (\alpha^{21}\ \ raiz)$$
$$\sigma(\alpha^{22}) = 1+\alpha^{10}\times\alpha^{22} = 1+\alpha^{10+22} = \alpha^0+\alpha^{32} = \alpha^0+\alpha^1 \rightarrow tabela \rightarrow \alpha^{18}$$
$$\sigma(\alpha^{23}) = 1+\alpha^{10}\times\alpha^{23} = 1+\alpha^{10+23} = \alpha^0+\alpha^{33} = \alpha^0+\alpha^2 \rightarrow tabela \rightarrow \alpha^{5}$$
$$\sigma(\alpha^{24}) = 1+\alpha^{10}\times\alpha^{24} = 1+\alpha^{10+24} = \alpha^0+\alpha^{34} = \alpha^0+\alpha^3 \rightarrow tabela \rightarrow \alpha^{29}$$
$$\sigma(\alpha^{25}) = 1+\alpha^{10}\times\alpha^{25} = 1+\alpha^{10+25} = \alpha^0+\alpha^{35} = \alpha^0+\alpha^4 \rightarrow tabela \rightarrow \alpha^{10}$$
$$\sigma(\alpha^{26}) = 1+\alpha^{10}\times\alpha^{26} = 1+\alpha^{10+26} = \alpha^0+\alpha^{36} = \alpha^0+\alpha^5 \rightarrow tabela \rightarrow \alpha^{2}$$
$$\sigma(\alpha^{27}) = 1+\alpha^{10}\times\alpha^{27} = 1+\alpha^{10+27} = \alpha^0+\alpha^{37} = \alpha^0+\alpha^6 \rightarrow tabela \rightarrow \alpha^{27}$$
$$\sigma(\alpha^{28}) = 1+\alpha^{10}\times\alpha^{28} = 1+\alpha^{10+28} = \alpha^0+\alpha^{38} = \alpha^0+\alpha^7 \rightarrow tabela \rightarrow \alpha^{22}$$
$$\sigma(\alpha^{29}) = 1+\alpha^{10}\times\alpha^{29} = 1+\alpha^{10+29} = \alpha^0+\alpha^{39} = \alpha^0+\alpha^8 \rightarrow tabela \rightarrow \alpha^{20}$$
$$\sigma(\alpha^{30}) = 1+\alpha^{10}\times\alpha^{30} = 1+\alpha^{10+30} = \alpha^0+\alpha^{40} = \alpha^0+\alpha^9 \rightarrow tabela \rightarrow \alpha^{16}$$

Como α^{21} é raíz da equação $\sigma(x)$, então a posição do bit com erro é dada por: $\frac{1}{\alpha^{21}} = \alpha^{-21} = \alpha^{(31-21)=10}$ –> bit 10 é o bit com erro.

3.3 Código de Hamming

O código de Hamming é um cíclico binário surgido nos anos 40 e foi criado dentro da Bell Labs por Richard Hamming para corrigir os problemas com leitura dos cartões perfurados utilizados para programar computadores (SWEENEY, 2002). Segundo Lin, Chen e Wang (2006) o código de Hamming quando comparado a outros algoritmos como BCH e Reed-Solomon necessita de pouca implementação e apresenta baixa complexidade, assim necessita de poucos recursos de computação. Além disso, o código de Hamming apresenta a limitação de detectar até 2 bits ou corrigir 1 bit.

O código de Hamming é um caso particular do código BCH quando detecta e corrige 1 bit. (RINO MICHELONI LUCA, 2010)

Segundo Hamming (1950) as condições para o código de Hamming, estão conforme mostrados na Tabela 7.

O comprimento do *codeword* é dado por **n** = 2^m-1, onde **m** = n-k são os bits de paridade. O número de bits de informação é dado por **k** = 2^m-1-m e **t** =1 é a capacidade de correção, que é de apenas 1 bit. Portanto, resumindo-se:

- **m** = bits de paridade

Tabela 7 – Principais Características dos Códigos de Hamming

Comprimento do Código	n=2^m-1
Número de bits de Informação	$k = 2^m - 1 - m$
Número de bits de Paridade	n-k = m
Capacidade de Correção	t = 1

Fonte: (JIANG, 2010)

- **k** = bits de informação
- **n** = codeword

Baseado nas condições da Tabela 7, Hamming (1950) demonstra que:

$$2^m \geq m + k + 1 \tag{9}$$

Substituindo: n = m + k

$$2^k \leq \frac{2^n}{n+1} \tag{10}$$

Resolvendo a inequação, obtem-se os valores de n e k para valores de m variando de 1 a 14. A Tabela 8 mostra os valores de n, k e m.

Tabela 8 – Tabela de Possíveis Códigos de Hamming

n	**k**	**m correspondente**
1	0	1
2	0	2
3	1	2
4	1	3
5	2	3
6	3	3
7	4	3
8	4	4
9	5	4
10	6	4
11	7	4
12	8	4
13	9	4
14	10	4
15	11	4
16	11	5
4110	4096	14

Fonte: adaptado de (HAMMING, 1950)

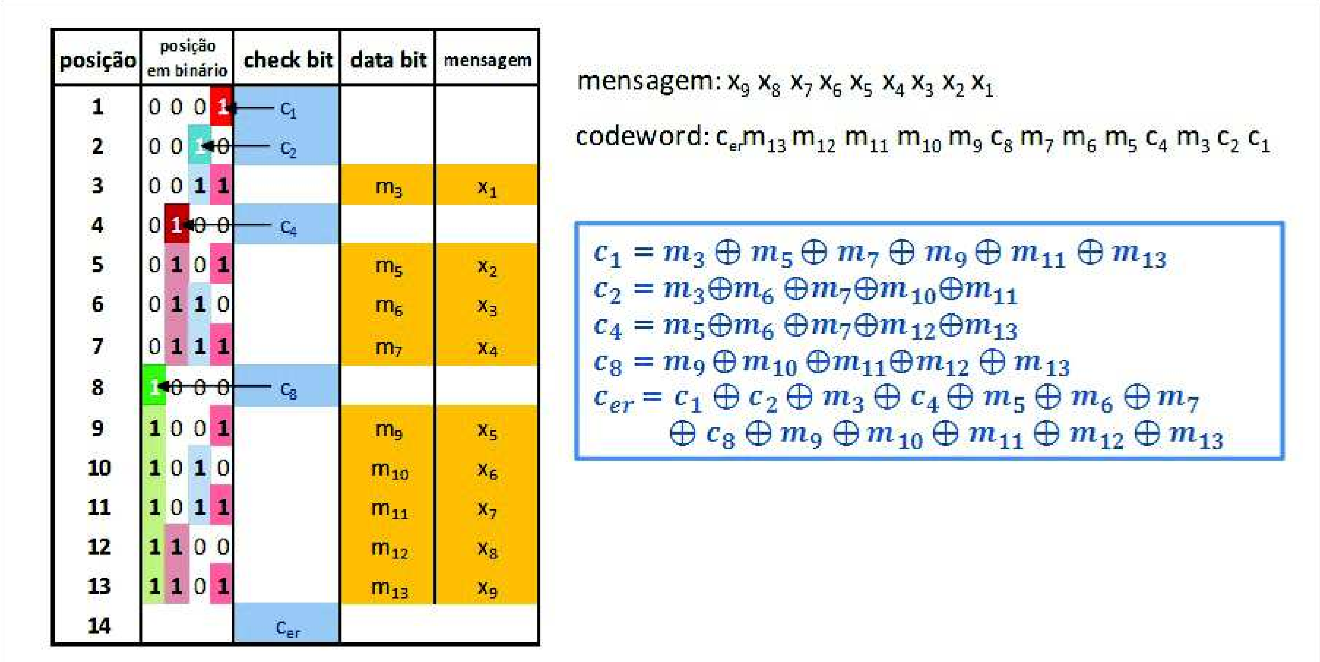
Na adaptação da Tabela 8 foi adicionada a última linha com n=4110, k=4096 e m=14 que serão utilizados nos itens seguintes quando o código de Hamming for aplicado para as memórias NAND Flash.

3.3.1 Codificação

No seu trabalho, Hamming (1950) descreve que o código pode detectar e corrigir 1 erro e denomina de: *single error correction* (SEC) ou detectar 2 erros e denomina de: *double error detection* (DED). Neste trabalho será implementado somente o SEC pois, no caso do DED não há correção apenas detecção.

Para explicar o código será utilizado como exemplo uma informação de 9 bits 101010101_2, conforme mostrado na Figura 15.

Figura 15 – Codificação - Código de Hamming



posição	posição em binário	check bit	data bit	mensagem
1	0 0 0 1	c_1		
2	0 0 1 0	c_2		
3	0 0 1 1		m_3	x_1
4	0 1 0 0	c_4		
5	0 1 0 1		m_5	x_2
6	0 1 1 0		m_6	x_3
7	0 1 1 1		m_7	x_4
8	1 0 0 0	c_8		
9	1 0 0 1		m_9	x_5
10	1 0 1 0		m_{10}	x_6
11	1 0 1 1		m_{11}	x_7
12	1 1 0 0		m_{12}	x_8
13	1 1 0 1		m_{13}	x_9
14		c_{er}		

mensagem: $x_9\ x_8\ x_7\ x_6\ x_5\ x_4\ x_3\ x_2\ x_1$

codeword: $c_{er}\ m_{13}\ m_{12}\ m_{11}\ m_{10}\ m_9\ c_8\ m_7\ m_6\ m_5\ c_4\ m_3\ c_2\ c_1$

$$c_1 = m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \oplus m_{13}$$
$$c_2 = m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11}$$
$$c_4 = m_5 \oplus m_6 \oplus m_7 \oplus m_{12} \oplus m_{13}$$
$$c_8 = m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13}$$
$$c_{er} = c_1 \oplus c_2 \oplus m_3 \oplus c_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus c_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13}$$

Fonte:Elaborado pelo autor baseado em (HAMMING, 1950)

A Figura 15 demonstra a codificação estendida de Hamming (14,9) de uma informação de 9 bits, no qual são calculados os 4 bits da paridade mais um 1 bit utilizado para a detecção de erros. Os bits de paridade estão localizados nas posições: 1, 2, 4 e 8 e são representados por: c_1, c_2, c_4 e c_8 respectivamente na tabela. O bit de detecção de erros é indicado por c_{er} e está localizado na última posição. Os bits da informação são re-arranjados no *codeword* distribuindo-se os bits da informação conforme mostrado na coluna *data bit*. Para calcular os bits de paridade, observa-se por exemplo que o bit 1 da paridade c_1 está localizado na coluna A, então c_1 será o XOR de todos os bits localizados nas posições que contém 1 na primeira coluna que são: $3_{10}=011_2$, $5_{10}=101_2$, $7_{10}=111_2$, $9_{10} = 1001_2$, $11_{10} = 1011_2$ e $13_{10} = 1101_2$, o mesmo é feito para o cálculo de c_2, c_4 e c_8. C_{er}.

Assim chega-se as equações para c_1, c_2 e c_4 conforme mostrado a seguir:

$$c_1 = m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \oplus m_{13} \quad (11)$$

$$c_2 = m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11} \quad (12)$$

$$c_4 = m_5 \oplus m_6 \oplus m_7 \oplus m_{12} \oplus m_{13} \quad (13)$$

$$c_8 = m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13} \quad (14)$$

$$c_{er} = c_1 \oplus c_2 \oplus m_3 \oplus c_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus c_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13} \quad (15)$$

3.3.1.1 Exemplo de Codificação

Para exemplificar a codificação do código de Hamming, utilizando-se a informação recebida:101010101_2 com 9 bits. Para melhorar a visualização dos bits de informação e suas posições: Indexando os 9 bits da informação:

m_{13}	m_{12}	m_{11}	m_{10}	m_9	m_7	m_6	m_5	m_3
1	0	1	0	1	0	1	0	1

Utilizando as equações 11, 12, 13, 14 e 15

$c_1 = m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \oplus m_{13} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = \mathbf{0}$

$c_2 = m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = \mathbf{1}$

$c_4 = m_5 \oplus m_6 \oplus m_7 \oplus m_{12} \oplus m_{13} = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = \mathbf{0}$

$c_8 = m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13} = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = \mathbf{1}$

$c_{er} = c_1 \oplus c_2 \oplus m_3 \oplus c_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus c_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13} = \mathbf{0} \oplus \mathbf{1} \oplus 1 \oplus \mathbf{0} \oplus 0 \oplus 1 \oplus 0 \oplus \mathbf{1} \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = \mathbf{1}$

O codeword completo fica: $\mathbf{1}10101\mathbf{1}010\mathbf{0}1\mathbf{10}_2$

3.3.2 Decodificação

Para a decodificação a informação recebida conforme Figura 16:

$$y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12} y_{13} y_{14} \quad (16)$$

Se não houver erro, para x_i tem-se y_i.

Então:

$$y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12} y_{13} y_{14} = x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} \quad (17)$$

Para a detecção e correção calcular as síndromes $k_1, k_2, k_4, k_8 \ \ e \ \ k_{14}$:

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11} \oplus y_{13} \tag{18}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11} \tag{19}$$

$$k_4 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12} \oplus y_{13} \tag{20}$$

$$k_8 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12} \oplus y_{13} \tag{21}$$

$$k_{14} = y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12} \oplus y_{13} \oplus y_{14} \tag{22}$$

Identifica-se que não há erro no código se as síndromes $k_1 = k_2 = k_4 = k_8 = 0$, em caso de erro o binário formado por k_1, k_2, k_4 e k_8 indicará a posição do bit com erro.

O bit extra de paridade k_{14} em conjunto com os bits de paridade indicam se ocorreu um erro duplo ou simples, conforme mostra a Tabela 9

Tabela 9 – Detecção de Erro Simples e Erro Duplo

k_{14}	$k_8 \oplus k_4 \oplus k_2 \oplus k_1$	Resultado
0	0	sem erros
0	1	duplo erro
1	0	erro bit extra
1	1	único bit erro

Fonte:Elaborado pelo autor baseado em (HAMMING, 1950)

Para um erro em um único bit k_{14}=1 e $k_8 \oplus k_4 \oplus k_2 \oplus k_1$=1, para erro duplo k_{14}=0 e $k_8 \oplus k_4 \oplus k_2 \oplus k_1$=1, k_{14}=1 e $k_8 \oplus k_4 \oplus k \oplus_2 \oplus k_1$=0 indica erro no bit extra de paridade e quando todas as síndromes são zero não há erro na informação.

A Figura 16 mostra a decodificação de uma mensagem recebida de 14 bits para Hamming (14,9). Nota-se que o algoritmo é praticamente o mesmo da codificação, exceto que agora para o cálculo de k_1, k_2, k_4, k_8 e k_{14} os bits da posição de paridade são incluídos, como por exemplo para o cálculo de k_1 é feito o XOR entre as posições que contém 1 na coluna de k_1 que são as posições 1, 3, 5, 7, 9, 11 e 13 ou y_1, y_3, y_5, y_7, y_9, y_{11} e y_{13}.

Figura 16 – Decodificação para Hamming (14,9)

		check bit	data bit	mensagem
1	0 0 0 1	k_1	y_1	
2	0 0 1 0	k_2	y_2	
3	0 0 1 1		y_3	x_1
4	0 1 0 0	k_4	y_4	
5	0 1 0 1		y_5	x_2
6	0 1 1 0		y_6	x_3
7	0 1 1 1		y_7	x_4
8	1 0 0 0	k_8	y_8	
9	1 0 0 1		y_9	x_5
10	1 0 1 0		y_{10}	x_6
11	1 0 1 1		y_{11}	x_7
12	1 1 0 0		y_{12}	x_8
13	1 1 0 1		y_{13}	x_9
14		k_{14}	y_{14}	

codeword recebido: $y_{14}y_{13}y_{12}y_{11}y_{10}y_9y_8\ y_7\ y_6\ y_5\ y_4\ y_3\ y_2\ y_1$
informação: $y_{13}y_{12}y_{11}y_{10}y_9\ y_7\ y_6\ y_5\ y_3$

$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11} \oplus y_{13}$
$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$
$k_4 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12} \oplus y_{13}$
$k_8 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12} \oplus y_{13}$
$k_{14} = y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12} \oplus y_{13} \oplus y_{14}$

Fonte:Elaborado pelo autor baseado em (HAMMING, 1950)

3.3.2.1 *Exemplo de Decodificação*

Para explicar a decodificação do código de Hamming, toma-se como exemplo a informação: **1**10101**1**010**0**1**10**$_2$, e inserindo um erro no bit 13 tem-se: **1**00101**1**010**0**1**10**$_2$
Para melhor identificar os bits recebidos identifica-se os bits em função de y:

y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1
1	0	0	1	0	1	1	0	1	0	0	1	1	0

Utilizando as equações 20, 21, 22, 23 e 24 são calculadas as síndromes:

$$k_1 = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$k_2 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0$$

$$k_4 = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$k_8 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

$$k_{14} = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1$$

Chega-se a:

- bit extra k_{14}=1
- $k_1+k_2+k_4+k_8$=1+0+1+1=1

Consultando a Tabela 9 para (k_{14}=1) e (k_1 ou k_2 ou k_4 ou k_8=1) verifica-se que há um único bit de erro.

Para a localização da posição do bit com erro, as síndromes são rearranjadas de forma a gerar um número binário de 4 bits:
$k_8k_4k_2k_1 = 1101$, tem-se o número em binário: $1101_2 = 13_{10}$ que indica que há um erro no bit 13.

3.4 Código de Reed-Solomon

O código de Reed-Solomon (R/S) é um código cíclico não binário que foi criado nos anos 60 por Irving S. Reed e Gustave Solomon e tem como característica o uso de síndromes que são um conjunto de bits para fazer a codificação e decodificação (RINO MICHELONI LUCA, 2010).

Neste código os bits da informação são divididos em pequenos grupos chamados símbolos de tamanho m sabendo-se que m é qualquer inteiro positivo maior que 2. Os códigos R/S obedecem a inequação:

$$0 < k < n < 2^m + 2 \tag{23}$$

Onde o valor de k é o número de símbolos de dados que estão sendo codificados e n é o número de símbolos códigos em um bloco codificado.(SKLAR, 1960).

O código Reed Solomom capaz de corrigir vários bits em sequência, isso é denominado erros em *burst*, em inglês ou rajada em português, o que o torna mais adequado para o uso em transmissões de informações por satélites e em mídias de DVD e CD por apresentarem erros agrupados (SKLAR, 1960).

As principais características do código Reed-Solomon estão resumidas na Tabela 10.

Tabela 10 – Principais Características do Código de Reed-Solomon

Comprimento do Código	n=2^m-1
Número de símbolos de Informação	$k = 2^m - 1 - 2t$
Número de símbolos de Paridade	n-k = 2t
Capacidade de Correção	$t = \left\lfloor \frac{n-k}{2} \right\rfloor$
Comprimento do símbolo	m

Fonte: adaptado de Sklar (1960)

Pela Tabela 10 observa-se que o comprimento do código em símbolos é dado por **n** = 2^m-1, o comprimento da informação em símbolos é dado por **k** = 2^m-1-2t, os símbolos de paridade são n-k = 2t, a capacidade de correção é t = [n-k/2] e o comprimento do símbolo é **m**.

A Figura 17 mostra um exemplo do Código de Reed-Solomon com os seus principais blocos desde a codificação, armazenamento na memória *Flash* e os blocos utilizados na decodificação.

Nota-se que a quantidade blocos na decodificação é maior que a codificação. Conclui-se que a etapa mais complexa e demorada do código Reed-Solomon é a decodificação. Segundo Rino Micheloni Luca (2010) para código Reed-Solomon se a codificação tivesse peso 1 a decodificação teria peso 8.

Figura 17 – Exemplo do Código de Reed Solomon

Fonte: (MICHELONI; MARELLI; RAVASIO, 2008)

A Figura 17 mostra a estrutura do código de Reed-Solomon que tem arquitetura similar ao código BCH exceto pelo bloco de cálculo de erros (MICHELONI; MARELLI; RAVASIO, 2008). Para a codificação, a informação m(x) entra no codificador e sai o *codeword* c(x) codificado com a informação mais os símbolos de paridade. Na decodificação utiliza-se a informação vinda da memória r(x) para calcular as síndromes e se não houver erro, os símbolos de paridade são separados e retirados da informação e enviados para fora como m(x) novamente. Se houver algum erro, as síndromes S são utilizadas para o cálculo dos coeficientes σ do polinômio de erros no bloco Cálculo do Polinômio de Erros e depois os X que são as raízes do polinômio localizador de erros são calculados pelo blocos Cálculo da Localicação dos Erros. Após a localização dos erros, são calculados os valores dos erros pelo bloco Cálculo dos Erros e em seguida os erros são corrigidos no Multiplicador para, na sequência os bits de paridade serem retirados e na saída tem-se a informação m(x) novamente.

3.4.1 Transformando a Informação Binária em Símbolos

Para codificar uma informação binária em símbolos é necessário descobrir primeiramente o valor de m. Normalmente são conhecidos os valores de k, que é o tamanho da informação em símbolos, e t que é a quantidade de símbolos que se quer corrigir, assim através da equação k=2^m-1-2t da Tabela 10 chega-se à equação:

$$2^m = k + 1 + 2t \tag{24}$$

Com o valor de m determina-se o polinômio primitivo e gera-se a tabela do Corpo de Galois. Como já visto anteriormente na seção 3.2.4 Corpo de Galois, o polinômio primitivo

baseado em m na Tabela 3. Tabela do Corpo de Galois pode ser obtido através de circuitos LFSR com o polinômio primitivo.

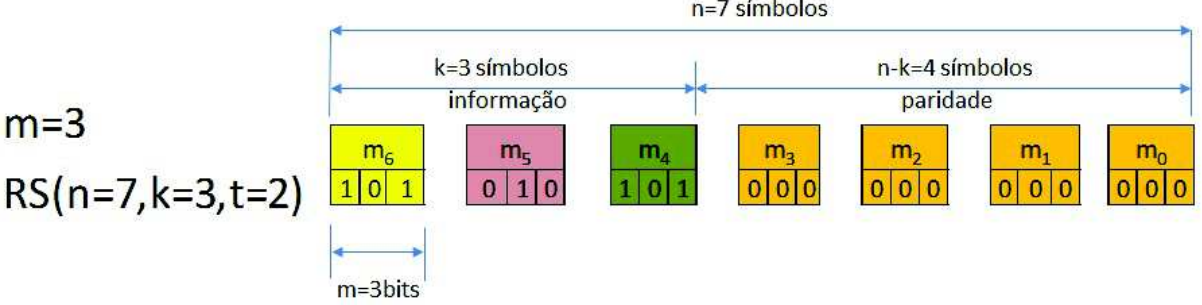
3.4.1.1 Exemplo de Transformação de Informação Binária em Símbolos

Para explicar a transformação da informação binária em símbolos do código de Reed-Solomon, toma-se como exemplo a informação 101010101_2 com 9 bits e t=2. Pela equação 24 tem-se:

$$
\begin{aligned}
2^m &= k + 1 + 2 * t \\
2^m &= \frac{9}{m} + 1 + 2 * 2 \\
2^m &= \frac{9}{m} + 5 \\
m &= 3
\end{aligned}
$$

Para m=3 a informação de 9 bits fica com k=3 símbolos de 3 bits cada e a quantidade de símbolos de paridade é dada por n-k=2t que vem da Tabela 10, e para t=2, tem-se 2t = 4 símbolos ou 4*3 = 12 bits, totalizando um *codeword* de 9+12 = 21 bits ou 7 símbolos conforme mostrado na Figura 18.

Figura 18 – Exemplo Informação de 9 bits Dividida em 3 Símbolos de m=3 bits



Fonte: autor baseado em Sklar (1960)

Para a codificação e decodificação é necessário o uso do Corpo de Galois, o qual diz que para m=3 o polinômio gerador deverá ser um polinômio primitivo de grau m o qual é: $x^3 + x + 1 = 0$ conforme Tabela de polinômios primitivos Tabela 3.

O Corpo de Galois para m=3 GF(2^m=8) é como mostrado na Figura 19 onde pode-se visualizar a tabela binária para os 8 termos e também o circuito gerador (RUMSEY; WATKINSON, 2004).

Figura 19 – Tabela do Corpo de Galois GF(8) e o Circuito Gerador

Tabela do Campo de Galois para $x^3 + x + 1 = 0$

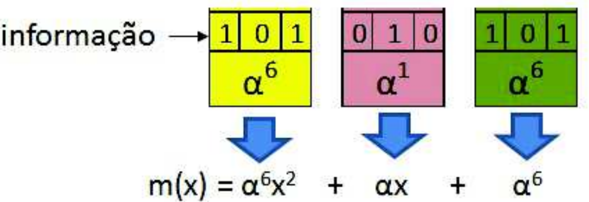
	dec	bin		
α^1	2	0	1	0
α^2	4	1	0	0
α^3	3	0	1	1
α^4	6	1	1	0
α^5	7	1	1	1
α^6	5	1	0	1
α^7	1	0	0	1

Circuito Gerador

Fonte: (RUMSEY; WATKINSON, 2004)

Utilizando a tabela de GF(8) da Figura 19 a informação com 9 bits fica com 3 símbolos sendo $101_2 = \alpha^6, 010_2 = \alpha^1$ e $101_2 = \alpha^6$ e polinômio da informação fica: m(x) = $\alpha^6 x^2 + \alpha x + \alpha^6$, conforme Figura 20.

Figura 20 – Codificação em Símbolos para o Código R/S



Fonte:baseado em (SKLAR, 1960)

3.4.2 Operações com o Corpo de Galois para GF(8)

Como visto anteriormente, as operações com o Corpo de Galois não são triviais, assim vários autores recorrem às tabelas para simplificar e poupar tempo nos cálculos (RINO MICHELONI LUCA, 2010) (SKLAR, 1960).

A Tabela 11 mostra as tabelas de soma e produto para GF(8) gerados pela tabela de Corpo de Galois para m=3.

Tabela 11 – Tabelas de Adição e Multiplicação para GF(8)

Tabela de Soma

+	a0	a1	a2	a3	a4	a5	a6
a0	0	a3	a6	a1	a5	a4	a2
a1	a3	0	a4	a0	a2	a6	a5
a2	a6	a4	0	a5	a1	a3	a0
a3	a1	a0	a5	0	a6	a2	a4
a4	a5	a2	a1	a6	0	a0	a3
a5	a4	a6	a3	a2	a0	0	a1
a6	a2	a5	a0	a4	a3	a1	0

Tabela de Multiplicação

x	a0	a1	a2	a3	a4	a5	a6
a0	a0	a6	a5	a4	a3	a2	a1
a1	a1	a0	a6	a5	a4	a3	a2
a2	a2	a1	a0	a6	a5	a4	a3
a3	a3	a2	a1	a0	a6	a5	a4
a4	a4	a3	a2	a1	a0	a6	a5
a5	a5	a4	a3	a2	a1	a0	a6
a6	a6	a5	a4	a3	a2	a1	a0

Fonte: (SKLAR, 1960)

A Tabela 11 mostra as tabelas de soma e multiplicação entre os fatores do Corpo de Galois de GF(8). Os valores do Corpo de Galois GF(8) são mostrados na tabela da Figura 19.

3.4.3 Polinômio Gerador

O Polinômio Gerador para a codificação do código R/S é dada por g(x), no qual a quantidade de raízes é dada por n-k=2t, assim para o polinômio gerador tem-se (SKLAR, 1960):

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)...(x-\alpha^{2t}) \tag{25}$$

3.4.4 Codificação

Para a codificação do código de Reed-Solomon, utiliza-se o mesmo procedimento da codificação BCH:

a) determinar o polinômio gerador g(x)

b) multiplicar o polinômio da informação por x^{n-k}

c) dividir o $x^{n-k} \times m(x)$ pelo polinômio gerador g(x) para obter o resto da divisão b(x)

d) somar $x^{n-k} \times m(x)$ com o resto para obter o *codeword*

3.4.4.1 Exemplo de Codificação

Para exemplificar a codificação do código R/S, será mostrado as quatro etapas para codicação conforme o item 3.4.4 . a) Determinado o Polinômio Gerador:

Para explicar a codificação do código R/S através do polinômio gerador é utilizado como exemplo o código RS(7,3,2) que possui capacidade de correção de dois erros e para isso necessita de um polinômio gerador com n-k = 2t = 4 raízes.

a) Determinar o polinômio gerador, utilizando a equação 25 para 2t = 4:

$$\begin{aligned} g(x) =& (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) \\ =& (x^2-(\alpha+\alpha^2)x+\alpha^3)(x^2-(\alpha^3+\alpha^4)x+\alpha^7) \\ =& (x^2-\alpha^4x+\alpha^3)(x^2-\alpha^6x+\alpha^0) \\ =& (x^4-(\alpha^4+\alpha^6)x^3+(\alpha^3+\alpha^{10}+\alpha^0)x^2+(\alpha^4+\alpha^9)x+\alpha^3) \\ =& x^4-\alpha^3x^3+\alpha^0x^2-\alpha^1x+\alpha^3 \end{aligned}$$

Trocando os sinais negativos por positivos, chega-se ao polinômio gerador para 2 erros:

$$g(x) = x^4+\alpha^3x^3+\alpha^0x^2+\alpha^1x+\alpha^3 \tag{26}$$

b) Multiplicar o Polinômio da Informação por x^{n-k}

Para x$^{n-k} \times m(x) = x^4 \times (\alpha^6x^2+\alpha x+\alpha^6) \rightarrow \alpha^6x^6+\alpha x^5+\alpha^6x^4$

c) Dividir o x$^{n-k} \times m(x)$ Pelo Polinômio Gerador g(x) Para Obter o Resto da Divisão.

$$\begin{aligned} x^{n-k} \times m(x) \;\; & mod \;\; g(x) \\ \alpha^6x^6+\alpha x^5+\alpha^6x^4 \;\; & mod \;\; x^4+\alpha^3x^3+\alpha^0x^2+\alpha^1x+\alpha^3 \\ obtem-se \;\; o \;\; resto \;\; da \;\; divisao: \;\; & b(x) = \alpha^2x^3+\alpha x^2+\alpha^3x+\alpha^3 \end{aligned}$$

d) Somar x$^{n-k} \times m(x)$ com o Resto da Divisão para obter o *codeword*.
Assim o *codeword* c(x) = m(x) + b(x) fica:

$$\text{c(x)} = \underbrace{\alpha^6x^6+\alpha x^5+\alpha^6x^4}_{m(x)} + \underbrace{\alpha^2x^3+\alpha x^2+\alpha^3x+\alpha^3}_{b(x)}$$

3.4.5 Decodificação

Para a decodificação conforme visto anteriormente no início do item 3.4, são necessárias várias etapas:
a) Cálculo das Síndromes
b) Cálculo do polinômio de de erros
c) Localicação dos erros
d) Cálculo dos erros

3.4.5.1 Exemplo de Cálculo de Sindromes para Um Código Sem Erros

Para exemplificar o cálculo das síndromes para um código sem erros, é utilizado o *codeword* a informação codificada no exemplo anterior. Para o cálculo das síndromes Sklar

(1960) demonstra que as equações são:

$$S_1(\alpha^1) = r(\alpha^1) \quad (27)$$

$$S_2(\alpha^2) = r(\alpha^2) \quad (28)$$

$$S_3(\alpha^3) = r(\alpha^3) \quad (29)$$

$$S_4(\alpha^4) = r(\alpha^4) \quad (30)$$

Aplicando para c(x) = r(x) = $\alpha^6x^6 + \alpha x^5 + \alpha^6x^4 + \alpha^2x^3 + \alpha x^2 + \alpha^3x + \alpha^3$ codificado do exemplo:

$$\begin{aligned}
S_1(\alpha^1) = r(\alpha^1) &= \alpha^6\alpha^{1\cdot6} + \alpha\alpha^{1\cdot5} + \alpha^6\alpha^{1\cdot4} + \alpha^2\alpha^{1\cdot3} + \alpha\alpha^{1\cdot2} + \alpha^3\alpha^1 + \alpha^3 \\
&= \alpha^{12} + \alpha^6 + \alpha^{10} + \alpha^5 + \alpha^3 + \alpha^4 + \alpha^3 \\
&= \alpha^5 + \alpha^6 + \alpha^3 + \alpha^5 + \alpha^3 + \alpha^4 + \alpha^3 \\
&= 0 \\
S_2(\alpha^2) = r(\alpha^2) &= \alpha^6\alpha^{2\cdot6} + \alpha\alpha^{2\cdot5} + \alpha^6\alpha^{2\cdot4} + \alpha^2\alpha^{2\cdot3} + \alpha\alpha^{2\cdot2} + \alpha^3\alpha^2 + \alpha^3 \\
&= \alpha^{18} + \alpha^{11} + \alpha^{14} + \alpha^8 + \alpha^5 + \alpha^5 + \alpha^3 \\
&= \alpha^4 + \alpha^4 + \alpha^1 + \alpha^1 + \alpha^5 + \alpha^5 + \alpha^3 \\
&= 0 \\
S_3(\alpha^3) = r(\alpha^3) &= \alpha^6\alpha^{3\cdot6} + \alpha\alpha^{3\cdot5} + \alpha^6\alpha^{3\cdot4} + \alpha^2\alpha^{3\cdot3} + \alpha\alpha^{3\cdot2} + \alpha^3\alpha^3 + \alpha^3 \\
&= \alpha^{24} + \alpha^{16} + \alpha^{18} + \alpha^{11} + \alpha^7 + \alpha^6 + \alpha^3 \\
&= \alpha^3 + \alpha^2 + \alpha^4 + \alpha^4 + \alpha^0 + \alpha^6 + \alpha^3 \\
&= 0 \\
S_4(\alpha^4) = r(\alpha^4) &= \alpha^6\alpha^{4\cdot6} + \alpha\alpha^{4\cdot5} + \alpha^6\alpha^{4\cdot4} + \alpha^2\alpha^{4\cdot3} + \alpha\alpha^{4\cdot2} + \alpha^3\alpha^4 + \alpha^3 \\
&= \alpha^{30} + \alpha^{21} + \alpha^{22} + \alpha^{14} + \alpha^9 + \alpha^7 + \alpha^3 \\
&= \alpha^2 + \alpha^0 + \alpha^1 + \alpha^0 + \alpha^2 + \alpha^0 + \alpha^3 \\
&= 0
\end{aligned}$$

Verifica-se então que $S_1(\alpha^1) = S_2(\alpha^2) = S_3(\alpha^3) = S_4(\alpha^4) = 0$ não há erros na informação r(x) recebida.

3.4.5.2 Exemplo de Decodificação:

Como exemplo para a decodificação vamos alterar as símbolos de x^4 e x^2 de α^6 para α^2 e α para α^3 respectivamente, resultando em:

$$\text{r(x)} = \alpha^6x^6 + \alpha x^5 + \alpha^2x^4 + \alpha^2x^3 + \alpha^3x^2 + \alpha^3x + \alpha^3$$

a) Cálculo as síndromes:

$$
\begin{aligned}
S_1 = r(\alpha^1) &= \alpha^6\alpha^{1\cdot 6} + \alpha\alpha^{1\cdot 5} + \alpha^2\alpha^{1\cdot 4} + \alpha^2\alpha^{1\cdot 3} + \alpha^3\alpha^{1\cdot 2} + \alpha^3\alpha^1 + \alpha^3 \\
&= \alpha^{12} + \alpha^6 + \alpha^6 + \alpha^5 + \alpha^5 + \alpha^4 + \alpha^3 \\
&= \alpha^5 + \alpha^6 + \alpha^6 + \alpha^5 + \alpha^5 + \alpha^4 + \alpha^3 \\
&= \alpha^1 \\
S_2 = r(\alpha^2) &= \alpha^6\alpha^{2\cdot 6} + \alpha\alpha^{2\cdot 5} + \alpha^2\alpha^{2\cdot 4} + \alpha^2\alpha^{2\cdot 3} + \alpha^3\alpha^{2\cdot 2} + \alpha^3\alpha^2 + \alpha^3 \\
&= \alpha^{18} + \alpha^{11} + \alpha^{10} + \alpha^8 + \alpha^7 + \alpha^5 + \alpha^3 \\
&= \alpha^4 + \alpha^4 + \alpha^3 + \alpha^1 + \alpha^0 + \alpha^5 + \alpha^3 \\
&= \alpha^2 \\
S_3 = r(\alpha^3) &= \alpha^6\alpha^{3\cdot 6} + \alpha\alpha^{3\cdot 5} + \alpha^2\alpha^{3\cdot 4} + \alpha^2\alpha^{3\cdot 3} + \alpha^3\alpha^{3\cdot 2} + \alpha^3\alpha^3 + \alpha^3 \\
&= \alpha^{24} + \alpha^{16} + \alpha^{14} + \alpha^{11} + \alpha^9 + \alpha^6 + \alpha^3 \\
&= \alpha^3 + \alpha^2 + \alpha^0 + \alpha^4 + \alpha^2 + \alpha^6 + \alpha^3 \\
&= \alpha^1 \\
S_4 = r(\alpha^4) &= \alpha^6\alpha^{4\cdot 6} + \alpha\alpha^{4\cdot 5} + \alpha^2\alpha^{4\cdot 4} + \alpha^2\alpha^{4\cdot 3} + \alpha^3\alpha^{4\cdot 2} + \alpha^3\alpha^4 + \alpha^3 \\
&= \alpha^{30} + \alpha^{21} + \alpha^{18} + \alpha^{14} + \alpha^{11} + \alpha^7 + \alpha^3 \\
&= \alpha^2 + \alpha^0 + \alpha^4 + \alpha^0 + \alpha^4 + \alpha^0 + \alpha^3 \\
&= \alpha^4
\end{aligned}
$$

b) Cálculo do Polinômio de Erros.

Para o cálculo do polinômio de erros, necessita-se encontrar os termos σ da equação de erros, O polinômio de erros é dado pela equação:

$$\sigma(x) = 1 + \sigma_1 x^1 + \sigma_2 x^2 \tag{31}$$

Para calcular os índices σ_1 e σ_2 do polinômio de erros, utizam-se as fórmulas descritas em Sklar (1960):

$$\begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} \begin{vmatrix} \sigma_2 \\ \sigma_1 \end{vmatrix} = \begin{vmatrix} S_3 \\ S_4 \end{vmatrix}$$

Chega-se a:

$$\sigma_2 = \frac{\begin{vmatrix} S_3 & S_2 \\ S_4 & S_3 \end{vmatrix}}{\begin{vmatrix} S_1 & S_2 \\ S_2 & S_3 \end{vmatrix}} = \frac{S_3^2 - S_4 S_2}{S_1 S_3 - S_2^2} \tag{32}$$

$$\sigma_1 = \frac{\begin{vmatrix} S_1 & S_3 \\ S_2 & S_4 \end{vmatrix}}{\begin{vmatrix} S_1 & S_2 \\ S_s & S_3 \end{vmatrix}} = \frac{S_1 S_4 - S_2 S_3}{S_1 S_3 - S_2^2} \tag{33}$$

Substituindo os valores do exemplo com $S_1 = \alpha^1$, $S_2 = \alpha^2$, $S_3 = \alpha^1$ e $S_4 = \alpha^4$

$$\sigma_1 = \frac{S_1S_4 - S_2S_3}{S_1S_3 - S_2^2} = \frac{\alpha\alpha^4 - \alpha^2\alpha}{\alpha} = \frac{\alpha^5 - \alpha^3}{\alpha} = \frac{\alpha^2}{\alpha} = \alpha^2 - \alpha^{-1} = \alpha$$
$$\sigma_2 = \frac{S_3^2 - S_4S_2}{S_1S_3 - S_2^2} = \frac{\alpha^2 - \alpha^4\alpha^2}{\alpha\alpha - \alpha^{2\cdot 2}} = \frac{\alpha^2 - \alpha^6}{\alpha^2 - \alpha^4} = \frac{\alpha^0}{\alpha} = \alpha^0\alpha^{-1} = \alpha^{-1}\alpha^7 = \alpha^6$$

c) Cálculo das Raízes do Polinômios de Erros

Para fazer a localização dos erros, procura-se as raízes do polinômio de erros e para isso vamos utilizar o algoritmo iterativo de Chien que consiste em substituir os valores possíveis de α no polinômio de erros e achar qual valor de x é igual a zero:

$$x = \alpha^0 \to \sigma(\alpha^0) = 1 + \sigma_1\alpha^{0\cdot 1} + \sigma_2\alpha^{0\cdot 2}$$
$$x = \alpha^1 \to \sigma(\alpha^1) = 1 + \sigma_1\alpha^{1\cdot 1} + \sigma_2\alpha^{1\cdot 2}$$
$$\vdots$$
$$x = \alpha^6 \to \sigma(\alpha^6) = 1 + \sigma_1\alpha^{6\cdot 1} + \sigma_2\alpha^{6\cdot 2}$$

Para o exemplo:
Com $\sigma_1 = \alpha$ e $\sigma_2 = \alpha^6$ utilizando o algoritmo de Chien:

$$x = \alpha^0 \to \sigma(\alpha^0) = 1 + \alpha\cdot\alpha^{0\cdot 1} + \alpha^6\alpha^{0\cdot 2} = \alpha^4$$
$$x = \alpha^1 \to \sigma(\alpha^1) = 1 + \alpha\cdot\alpha^{1\cdot 1} + \alpha^6\alpha^{1\cdot 2} = 1 + \alpha^2 + \alpha^1 = \alpha^5$$
$$x = \alpha^2 \to \sigma(\alpha^2) = 1 + \alpha\cdot\alpha^{2\cdot 1} + \alpha^6\alpha^{2\cdot 2} = 1 + \alpha^3 + \alpha^3 = 1$$
$$x = \alpha^3 \to \sigma(\alpha^3) = 1 + \alpha\cdot\alpha^{3\cdot 1} + \alpha^6\alpha^{3\cdot 2} = 1 + \alpha^4 + \alpha^5 = 1 + 1 = 0$$
$$x = \alpha^4 \to \sigma(\alpha^4) = 1 + \alpha\cdot\alpha^{4\cdot 1} + \alpha^6\alpha^{4\cdot 2} = 1 + \alpha^5 + \alpha^0 = \alpha^5$$
$$x = \alpha^5 \to \sigma(\alpha^5) = 1 + \alpha\cdot\alpha^{5\cdot 1} + \alpha^6\alpha^{5\cdot 2} = 1 + \alpha^6 + \alpha^2 = 1 + 1 = 0$$
$$x = \alpha^6 \to \sigma(\alpha^6) = 1 + \alpha\cdot\alpha^{6\cdot 1} + \alpha^6\alpha^{6\cdot 2} = 1 + \alpha^0 + \alpha^4 = \alpha^4$$

As posições dos erros são:

$$\alpha^3 \to \frac{1}{\alpha^3} = \alpha^{-3}\alpha^7 = \boldsymbol{\alpha^4} \to posicao\ \ 4$$
$$\alpha^5 \to \frac{1}{\alpha^5} = \alpha^{-5}\alpha^7 = \boldsymbol{\alpha^2} \to posicao\ \ 2$$

d) Cálculo dos valores dos Erros:
Este bloco não é utilizado no código BCH, pois o código BCH é binário e como o código

Reed-Solomon utiliza símbolos é necessário descobrir o valor correto das síndromes com erro. Determinados a localização dos erros, (posição 2 e 4) os valores dos erros são calculados como:

$$S_1(x) = e_4x^4 + e_2x^2 \tag{34}$$

$$S_2(x) = e_4x^4 + e_2x^2 \tag{35}$$

Calculando para os valores do exemplo:
As posições das falhas que foram determinadas pelo polinômio de erros, x^4 e x^2, utilizando as equações 34 e 35.

$$S_1(\alpha^1) = \alpha^1 = e_4\alpha^{1\cdot4} + e_2\alpha^{1\cdot2} = e_4\alpha^4 + e_2\alpha^2$$
$$S_2(\alpha^2) = \alpha^2 = e_4\alpha^{2\cdot4} + e_2\alpha^{2\cdot2} = e_4\alpha^8 + e_2\alpha^4$$

Resolvendo o sistema de 2 equações a 2 incógnitas:

$$(i)\ \alpha^1 = e_4\alpha^4 + e_2\alpha^2 \Rightarrow e_2 = \frac{\alpha - e_4\alpha^4}{\alpha^2}$$
$$(ii)\ \alpha^2 = e_4\alpha^8 + e_2\alpha^4 \Rightarrow \alpha^2 = e_4\alpha^8 + (\frac{\alpha - e_4\alpha^4}{\alpha^2})\alpha^4 \Rightarrow e_4 = (\frac{\alpha^2 - \alpha^3}{\alpha^1 - \alpha^6}) = \frac{\alpha^5}{\alpha^5} = \alpha^0$$
$$(iii)\ e_2 = \frac{\alpha - \alpha^0\alpha^4}{\alpha^2} = \alpha^2$$

Substituindo:

$$e(x) = 00x^6 + 00x^5 + \alpha^0x^4 + 00x^3 + \alpha^2x^2 + 00x + 0$$
$$+$$
$$r(x) = \alpha^6x^6 + \alpha x^5 + \alpha^2x^4 + \alpha^2x^3 + \alpha^3x^2 + \alpha^3x + \alpha^3$$
$$=$$
$$c(x) = \alpha^6x^6 + \alpha x^5 + \alpha^6x^4 + \alpha^2x^3 + \alpha x^2 + \alpha^3x + \alpha^3$$

Verifica-se que o valor de c(x) corrigido é o mesmo valor de c(x) recebido.

4 TRABALHOS RELACIONADOS

O trabalho de Regulapati (2015) com o título de "Error Correction Codes in NAND Flash Memory"faz a comparação entre os códigos de Hamming, BCH, Reed-Solomon e LDPC (*Low Density Parity Check*) que significa Verificação de Paridade de Baixa Densidade.

As simulações foram realizadas para os códigos BCH e Reed-Solomon descritos em software e utilizando o simulador de transmissões espaciais, AWGN (*Additive White Gaussian Noise*), que é utiliza um modelo que gera ruído branco.

Regulapati (2015) concluiu que o para memórias NAND Flash SLC o código de Hamming é mais indicado por ter correção de erros simples e quando é utilizado em grandes blocos, divide-se o bloco grande em blocos menores, para correção de múltiplos bits o código BCH é superior ao código de Reed-Solomon por utilizar menos bits de paridade. (REGULAPATI, 2015)

No artigo "A Compact On-Chip ECC for Low Cost Flash Memories"(TANZAWA et al., 1997) a equipe de pesquisa do laboratório da Toshiba do Japão implementa ECC baseado no algoritmo de Hamming em um CI de NAND Flash de 64M bit. O resultado obtido foi acréscimo de 2% em área do CI com um acréscimo de consumo de menos de 1mA (TANZAWA et al., 1997).

O artigo "New Ultra High Density EPROM and Flash EEPROM with NAND Structure Cell"do Dr. Fujio Masuoka (MASUOKA et al., 1987) que é o inventor da *NAND Flash* a descreve no seu artigo de 1987. Masuoka descreve a nova Flash EEPROM com estrutura NAND utilizando a mesma célula com *floating gate* é interligada em série de formar NANDs e possui vantagens em economia de área de célula, aumento da densidade na construção dos transistores. O artigo descreve a estrutura, o princípio de operação, e a confiabilidade da nova estrutura. (MASUOKA et al., 1987)

O artigo de Eitan Yaakobi com o título de "Error Correction Code for Memories Flash"faz a comparação entre os códigos Reed-Solomon (R/s) e BCH em memórias NAND Flash através de experimentos empíricos. Yaakobi conclui que os bit com erros ocorrem tanto em memórias SLC como MLC de forma uniforme sem preferência por simetria ou dependência de localização. A conclusão é que o código BCH é melhor para erros aleatórios e que estão distribuídos e o código R/S é melhor para erros do tipo rajada em que os erros estão concentrados. (YAAKOBI et al., 2009)

(PANDA; SARIK; AWASTHI, 2012) em seu artigo "FPGA Implementation of Encoder for (15, k) Binary BCH Code Using VHDL and Performance Comparison for Multiple Error Correction Control"faz a descrição em VHDL utilizando FPGA Xilinx com 3 variações do código BCH para detectar 1, 2 e 3 erros. O código foi descrito para n=15 bits e chega a conclusão que a descrição que detecta e corrige 3 erros BCH (15,5,3) é o mais vantajoso pelo fato de ser

mais rápido em relação aos 2 outros, conforme mostra a Figura 21 (PANDA; SARIK; AWASTHI, 2012).

Figura 21 – Tabela Resultado Artigo

TABLE I. DEVICE UTILIZATION AND TIMING SUMMARY

Component Utilization/ Time	(15, 11, 1) BCH Encoder	(15, 7, 2) BCH Encoder	(15, 5, 3) BCH Encoder
No. of Slices	6	8	9
No. of Slice FF	9	12	15
4 input LUTs	12	14	16
Number of IOs	5	5	5
Simulation Clock	20 ns	20 ns	20 ns
Maxm. Combinational path delay	9.159 ns	9.159 ns	9.159 ns
Maxm output required	8.81 ns	8.73 ns	8.57 ns
Total CPU time to Xst completion	6.13 sec	5.9 sec	5.7 sec

Synthesis report for the targeted device Xilinx Spartan 3S1000 FPGA

Fonte: Panda, Sarik e Awasthi (2012).

Tabela do artigo de Panda, Sarik e Awasthi em que mostram que o código BCH com cobertura de 3 bits é o mais rápido em relação as códigos com cobertura de 1 e 2 bits.

Para melhor entendimento e análise dos trabalhos, construiu-se a tabela da Figura 22.

Figura 22 – Tabela Resumo - Trabalhos Relacionados

Autor	Título	Assunto	Conclusão
TANZAWA (1997)	"A Compact On-Chip ECC for Low Cost Flash Memories"	implementa ECC baseado no algoritmo de Hamming em um CI de NAND Flash de 64M bit.	Acréscimo de 2% em área do CI com um acréscimo de consumo de menos de 1mA
YAAKOBI (2009)	"Error Correction Code for Memories Flash"	comparação entre os códigos Reed-Solomon (R/s) e BCH em memórias NAND Flash através de experimentos empíricos.	• Conclui que os bit com erros ocorrem tanto em memórias SLC como MLC de forma uniforme sem preferência por simetria ou dependência de localização. • A conclusão é que o código BCH é melhor para erros aleatórios e que estão distribuídos e o código R/S é melhor para erros do tipo rajada em que os erros estão concentrados
PANDA; SARIK; AWASTHI 2012)	"FPGA Implementation of Encoder for (15, k) Binary BCH Code Using VHDL and Performance Comparison for Multiple Error Correction Control"	faz a descrição em VHDL utilizando FPGA Xilinx com 3 variações do código BCH para detectar 1, 2 e 3 erros para n=15 bits	A descrição que detecta e corrige 3 erros BCH (15,5,3) é o mais vantajoso pelo fato de ser o mais rápido em relação aos 2 outros.
Regulapati (2015)	"Error Correction Codes in NAND Flash Memory"	comparação entre os códigos de Hamming, BCH, Reed-Solomon e LDPC (Low Density Parity Check)	• Para memórias NAND Flash SLC o código de Hamming é mais indicado por ter correção de erros simples e quando é utilizado em grandes blocos, divide-se o bloco grande em blocos menores. • Para correção de múltiplos bits o código BCH é superior ao código de Reed-Solomon por utilizar menos bits de paridade.

Fonte: autor

Tanzawa implementa o algoritmo de Hamming no *hardware* da memória NAND Flash e conclui que o aumento de área é de 2% e o aumento de consumo é de menos de 1mA.

Panda, Sarik e Awasthi implementam o código BCH em FPGA e fazem a comparação do código BCH para a cobertura de 1, 2 e 3 bits e concluem que o código BCH com cobertura de 3 bits é o mais rápido.

Analisando os resultados obtidos por Regulapati em que para as memórias NAND Flash SLC, o código de Hamming é mais indicado por ter correção de erros simples e quando utilizado em grandes blocos, divide-se o bloco grande em blocos menores. Porém Yaakobi, em seu trabalho diz que os bits com erros ocorrem tanto em memórias SLC como MLC de forma uniforme sem preferência por simetria ou dependência de localização, baseado nestes trabalhos conclui-se que o código de Hamming pode ser aplicado tanto para memórias SLC como para MLC.

Yaakobi e Regulapati chegam a mesma conclusão quando comparam os códigos de R/S com BCH. Ambos concluem que o código BCH é superior ao R/S, mesmo resultado deste trabalho.

Nas pesquisas realizadas e nos trabalhos relacionados, não foi encontrado nenhum trabalho comparando o código BCH com o código de Hammming, que é o foco e a motivação deste trabalho.

5 MATERIAIS, FERRAMENTAS E METODOLOGIA

5.1 Premissas

Neste trabalho o uso área reserva da página para o ECC foi limitado em 50%, pois essa área é compartilhada com outras funções do controlador como: o *endurance* ou desgaste, marcação de *bad block*, *wear leveling* ou gerenciamento de desgaste.

5.2 Análise

Para um melhor entendimento dividiu-se a análise em 3 partes, descritas a seguir:

- Parte 1 - Análise dos 4 códigos em relação à cobertura para uma informação de 9 bits: Paridade(detecção de 1 bit) vs Hamming(detecção e correção de 1 bit) e BCH(detecção e correção de 2 bits) vs R/S(detecção e correção de 2 bits).
- Parte 2 - Análise dos códigos de Hamming(detecção e correção de 1 bit), BCH(detecção e correção de 4 bits) e R/S(detecção e correção de 3 bits) aplicados à memória NAND Flash.
- Parte 3 - Análise dos códigos Hamming com 4 blocos e detecção e correção de 4 bits e BCH com detecção e correção e correção de 4 bits aplicados à memória NAND Flash .

5.2.1 Análise - Parte 1

A Figura 23 mostra a tabela comparativa entre os 4 códigos que foram estudados neste trabalho utilizando a mesma informação de 9 bits 101010101_2, cuja codificação foi feita através dos exemplos dos códigos ao longo do texto. Para esta primeira análise, os códigos foram comparados pela sua cobertura, assim foram dividos em 2 blocos:
- primeiro bloco - códigos com detecção e correção de 1 erro - Paridade vs Hamming
- segundo bloco - códigos com detecção e correção de 2 erros - BCH vs R/S

Figura 23 – Comparação entre os 4 códigos

Comparação entre os códigos:

informação de 9 bits: 1 0 1 0 1 0 1 0 1

Código	tamanho da informação	paridade (bits)	overhead	detecção (bits)	correção (bits)	comentário
PARIDADE	9	1	11%	1	0	não é ECC pois não corrige informação
BCH BCH(31,21,2)	9	10	111%	2	2	precisa de 10 bits de paridade para detectar e corrigir 2 bits
HAMMING Hamming(13,9)	9	4	44%	1	1	detecta e corrige 1 bit e usa 4 bits de paridade
REED-SOLOMON R/S(7,3,2)	9	12	133%	2	2	usa 12 bits de paridade e corrige 2 símbolos de 3 bits **Nota -** considerando detecção de 1 bit por símbolo

REED-SOLOMON: m=3; 1 símbolo = 3 bits; 3 símbolos; 7 símbolos

Fonte: autor

A Figura 23 mostra a comparação entre os 4 códigos para uma informação de 9 bits que foi codificada para cada um dos códigos estudados.

Decidiu-se pela informação de 9 bits devido ao menor valor de m ser igual a 3. Para o código Reed-Solomon com 9 bits e m igual a 3 formam-se 3 símbolos de 3 bits, o que não seria possível caso tivesse sido usado uma informação com 8 bits ou 1 Byte.

Na primeira coluna, o tamanho da informação refere-se a quantidade de bits codificados, a coluna paridade mostra a quantidade de bits de paridade necessários e a coluna seguinte mostra o percentual de acréscimo de bits, a coluna detecção mostra a quantidade de bits que é possível detectar e a coluna correção mostra a quantidade de bits que é possível corrigir.

Paridade vs Hamming

Os dois códigos detectam o erro de 1 bit, porém apenas o código de Hamming corrige o bit, enquanto na paridade não há correção. A paridade conforme já explicado na seção 3.1 não é um código ECC por apenas detectar e não corrigir. Portanto conclui-se que apenas o código de Hamming é um ECC para detectar e corrigir 1 bit.

BCH vs R/S

Os códigos de BCH e R/S da Tabela 23 detectam e corrigem até 2 erros, mas a quantidade de bits de paridade necessária para o código BCH é menor do que o código R/S, sendo 10 bits para o código BCH e 12 bits para o R/S. Em termos de implementação o código R/S necessita do bloco a mais do que o código BCH para o cálculo dos valores dos erros, por trabalhar com símbolos equanto que o código BCH não necessita deste bloco por ser um código binário.

Micheloni, Marelli e Ravasio (2008) e Regulapati (2015) também fizeram comparações entre BCH e R/S e concluíram que o código BCH é mais vantajoso do que o código R/S por necessitar de menos bits de paridade, a exceção é feita quando os erros estão agrupados e são do tipo rajada ou *burst* em inglês enquanto que para erros com distribuição randômica como no caso das memórias NAND Flash o código BCH é mais efetivo.

Concluiu-se que o código de Reed-Solomon é o que necessita de mais bits de paridade e recursos (pois necessita de um bloco a mais) para uma mesma cobertura, quando comparado com o código BCH, o que restringe o seu uso quando aplicado em memórias NAND Flash, pois a quantidade de bits reservados para a paridade é limitada.

5.2.2 Análise - parte 2

Através da análise da parte 1 restaram os códigos de Hamming e BCH, porém os códigos possuem coberturas diferentes não sendo possível uma comparação direta entre eles.

Para a análise parte 2, comparou-se entre os códigos de Hamming, BCH e R/S aplicados à memória NAND Flash para determinar qual código seria o mais indicado para o uso comercial.

Análise dos códigos aplicada à memória NAND Flash de 4Gbits

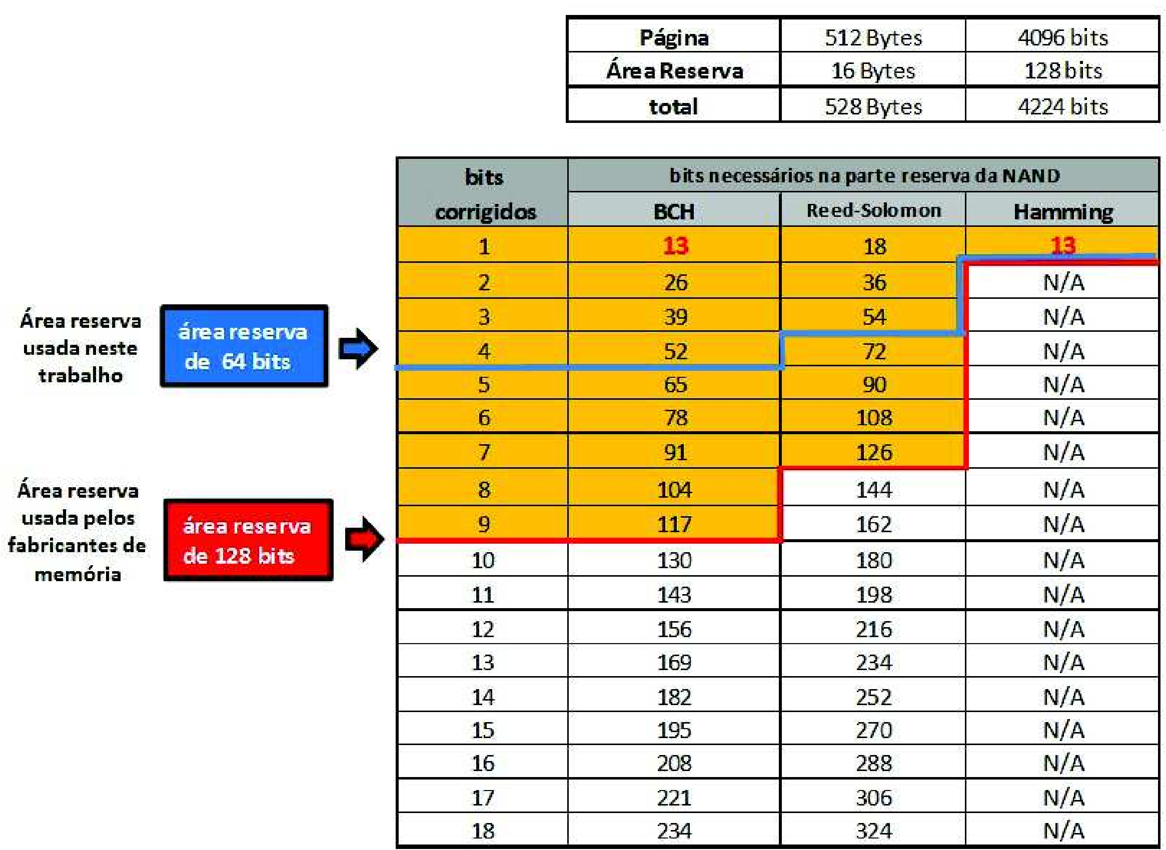
Como já visto anteriormente no item 2.4, a menor unidade endereçável de uma memória NAND Flash é a página, que para uma memória comercial de 4Gbits tem o tamanho de 512GBytes ou 4096 Gbits mais 16 Bytes ou 128 bits (Micron, 2012).

Para este trabalho o uso da área reserva foi limitada em 64 bits que é 50% da área reserva conforme explicado no item 5.1 de premissas. As fabricantes de memória NAND Flash Micron (Micron, 2005) e Macronix (Macronix, 2014) em seus *white papers* utilizam todos os bits da área reserva para a paridade dos ECCs não deixando espaço para outras funções do controlador

como marcação de *bad blocks*, *wear leveling* e *endurance* e com isso pode prejudicar o correto funcionamento da memória.

Com a informação de k = 4096 bits e 64 bits de paridade, verifica-se que o código BCH tem a melhor cobertura corrigindo 4 bits e utilizando 54 bits de paridade enquanto o código R/S corrige somente 3 bits e para isso utiliza 52 bits. O código de Hamming utiliza 13 bits para corrigir somente 1 bit conforme mostra a Figura 24

Figura 24 – Comparação dos 3 Códigos para NAND Flash



Página	512 Bytes	4096 bits
Área Reserva	16 Bytes	128 bits
total	528 Bytes	4224 bits

bits corrigidos	bits necessários na parte reserva da NAND		
	BCH	Reed-Solomon	Hamming
1	13	18	13
2	26	36	N/A
3	39	54	N/A
4	52	72	N/A
5	65	90	N/A
6	78	108	N/A
7	91	126	N/A
8	104	144	N/A
9	117	162	N/A
10	130	180	N/A
11	143	198	N/A
12	156	216	N/A
13	169	234	N/A
14	182	252	N/A
15	195	270	N/A
16	208	288	N/A
17	221	306	N/A
18	234	324	N/A

Fonte: adaptado de Regulapati (2015)

Observa-se na Figura 24 quantidade de bits corrigidos e os bits de paridade necessários para uma área reserva de 128 bits, limitado pela linha vermelha e para uma área reserva de 64 bits, limitado pela linha azul.

Para correção de 1 bit, o código de Hamming é um caso particular do código BCH e necessita de 13 bits de paridade. (RINO MICHELONI LUCA, 2010)

A Figura 25 mostra a comparação entre os códigos BCH e R/S em relação à capacidade de correção e à quantidade de bits de paridade necessários. Para uma cobertura de 9 bits e área reserva de 128 bits o código BCH necessita de 117 bits de paridade, enquanto o código R/S necessita de 162 bits . Com a área reserva limitada em 64 bits novamente o código BCH necessita menos bits de paridade do que o código R/S, 52 bits comparado com os 72 bits do R/S.

Figura 25 – Comparação BCH / R/S - Capacidade de Correção vs Quantidade de bits de Paridade

Fonte: adaptado de Regulapati (2015)

Observando a Figura 25 evidencia-se novamente que o código BCH utiliza menor quantidade de bits de paridade se comparado com a aplicação do código Reed-Solomon para uma mesma capacidade de correção.

5.2.3 Análise - parte 3

O código de Hamming tem capacidade de correção de apenas 1 bit, enquanto o código BCH tem capacidade correção de 4 bits para a informação de 4096 bits. Desta forma a comparação entre os códigos não é possível devido à diferença de cobertura.
Analizando-se o código de Hamming aplicado à 4096 bits:

Tabela 12 – Estudo de Cobertura do Código de Hamming para 4096 bits

Qtde Blocos	Tamanho do Bloco	Total bits de Paridade	Cobertura	Notas
1 bloco	4096 bits	13 bits	1 bit	cobertura de 1 bit
2 blocos	2048 bits	24 bits	2 bits	cobertura de 2 bits
4 blocos	1024 bits	44 bits	4 bits	cobertura de 4 bits
8 blocos	512 bits	80 bits	8 bits	80 bits é maior que a área reserva de 64 bits

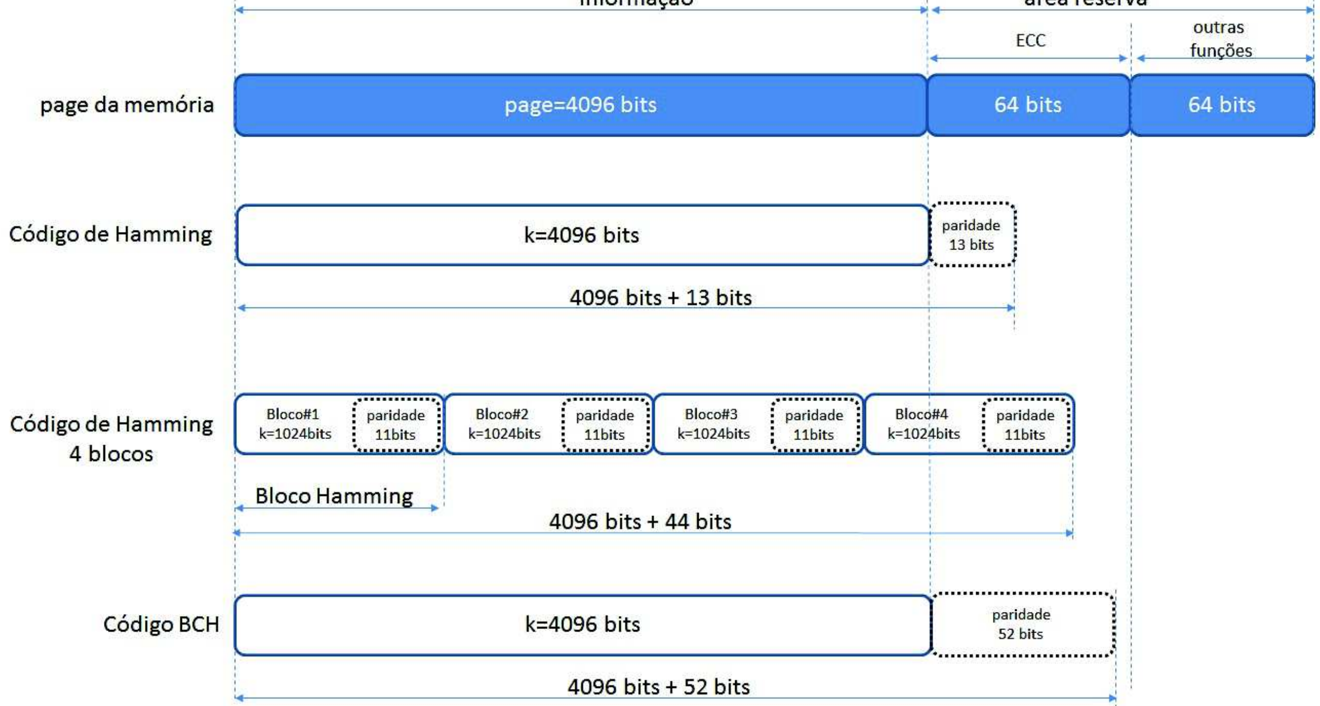
Pela Tabela 12, verifica-se que o bloco com 1024 bits é o menor bloco em que pode ser

aplicado o código de Hamming para a página de 4096 bits. Aplicando o código de Hamming para cada bloco são necessários 11 bits de paridade por bloco e um total de 44 bits de paridade e a cobertura de total de 4 bits para os 4 blocos. Desta maneira é possível fazer a comparação entre os códigos de R/S e Hamming com 4 blocos, ambos com cobertura de 4 bits para a página de 4096 bits.

Para fazer esta comparação, foram descritos em VHDL, circuitos de codificação e decodificação:
- Hamming para 4096 bits
- Hamming para 1024 bits - 4 blocos de 1024 bits = 4096 bits
- BCH para 4096 bits

A Figura 26 mostra as 3 implementações em VHDL para a comparação entre os códigos de Hamming para 4096 bits, 4*Hamming para 1024 bits e o código BCH para 4096 bits.

Figura 26 – Comparação dos Códigos BCH vs 4xHamming para NAND Flash



Fonte: autor

A Figura 26 mostra a comparação dos códigos de Hamming para 4096 bits, 4 x Hamming para 1024 bits e BCH para 4096 bits que foi implementado em VHDL.

5.3 Ferramentas

Para implementar os códigos, primeiramente foi feita a implementação utilizando-se planilhas Excel, para melhorar o entendimento dos algoritmos e para posterior validação com as simulações das descrições em HDL.

As descrições dos circuitos foram feitas em linguagem VHSIC (*Very High Speed Integrated Circuit*) utilizando o software ISE versão 14.6 do fabricante Xilinx e simulado em uma placa Xilinx família Virtex 6 e componente XC6VLX75T. Após a descrição e *debug* dos códigos, o software fornece a quantidade de gates utilizados e a área de silício necessária para a implementação.

5.4 Metodologia

Foi feita a análise da viabilidade entre os 4 algoritmos propostos utilizando-se a mesma informação com 9 bits, realizou-se primeiramente as implementações e simulações dos algoritmos em planilhas e indentificou-se de forma clara quais códigos eram ECC e que a paridade não é um código ECC.

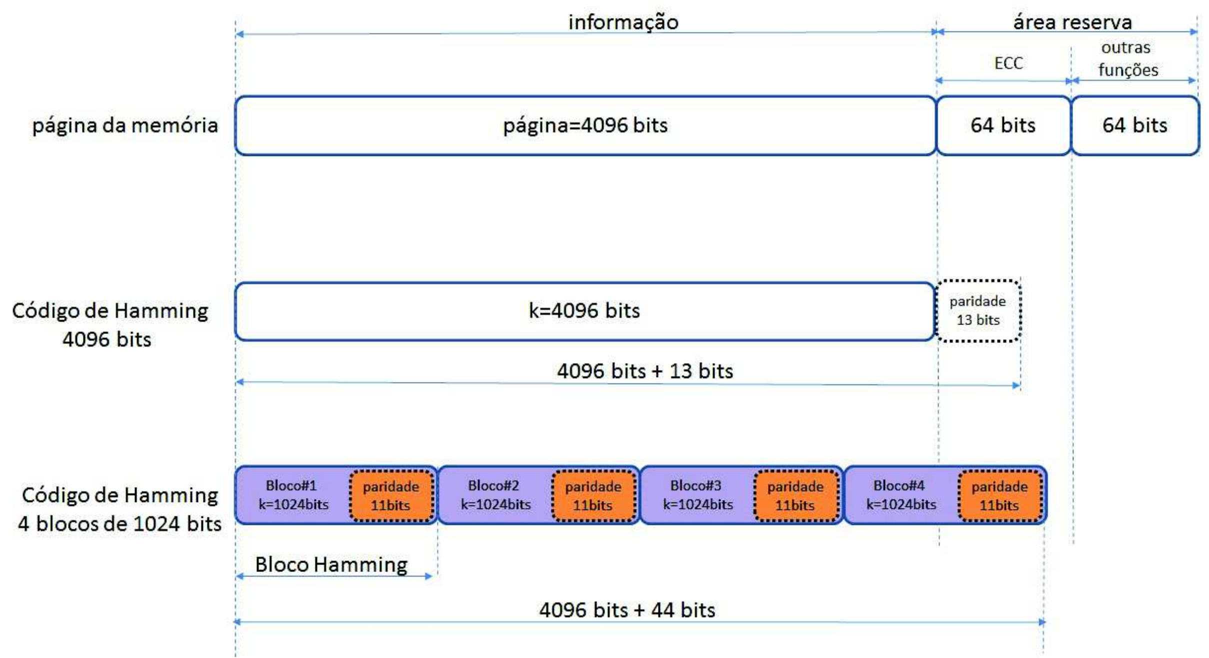
Em seguida foi feita a análise dos 3 algoritmos de ECC, quando aplicados à memória NAND Flash de forma a identificar qual seria o algoritmo de ECC com mais vantagens comerciais. A análise foi feita focando nos 2 códigos que tem a mesma cobertura, BCH e R/S. Identificou-se que o código BCH é mais adequado que o R/S, porém a comparação do código BCH não podia ser feita devido a diferença entre coberturas. Para fazer a comparação entre os códigos BCH e Hamming, foi proposto dividir a página da memória de 4096 bits em 4 blocos de 1024 bits e aplicar o código de Hamming em cada um deles, obtendo assim uma cobertura de 4 bits.

Em seguida foi iniciada a descrição dos circuitos com os códigos de Hamming com 4 blocos de 1024 bits divididos em blocos de codificação e decodificação. A descrição do código BCH para 4096 bits foi feita em 4 blocos separados para melhor controle, pois os algoritmos são longos. Primeiro implementou-se o bloco de codificação, depois o bloco de cálculo de síndromes, o bloco de cálculo dos sigmas e por ultimo o bloco de cálculo das ráizes do polinomio de erros. Após cada implementação em VHDL os resultados dos circuitos eram comparados com os valores obtidos através das planilhas de forma a validar a implementação em VHDL. Para a comparação e análise dos dados, foram utilizados os relatórios gerados pelo programa ISE e feitas as comparações.

5.4.1 Implementação em VHDL código de Hamming com 4 blocos de 1024 bits

Implementação em VHDL do código de Hamming em 4 blocos de 1024 bits de uma Memória NAND Flash com página de 4096 bits e 64 bits de área reserva.

Figura 27 – Comparação Hamming 4096 bits vs Hamming 4 Blocos 1024 bits



Fonte: autor

A Figura 27 mostra a distribuição dos 4 blocos de 1024 bits na página de 4096 bits + 64 bits da área reserva. Comparando-se com o código de Hamming para 4096 bits utiliza 13 bits da área reserva, enquanto o código de Hamming com 4 blocos de 1024 bits utiliza 44 bits.

Na Figura 28 observa-se o diagrama de blocos do algoritmo descrito em VHDL para a codificação e decodificação do Código de Hamming para 1024 bits no controlador de memória da *NAND Flash*.

Figura 28 – Diagrama de Blocos - Código Hamming para 1024 bits

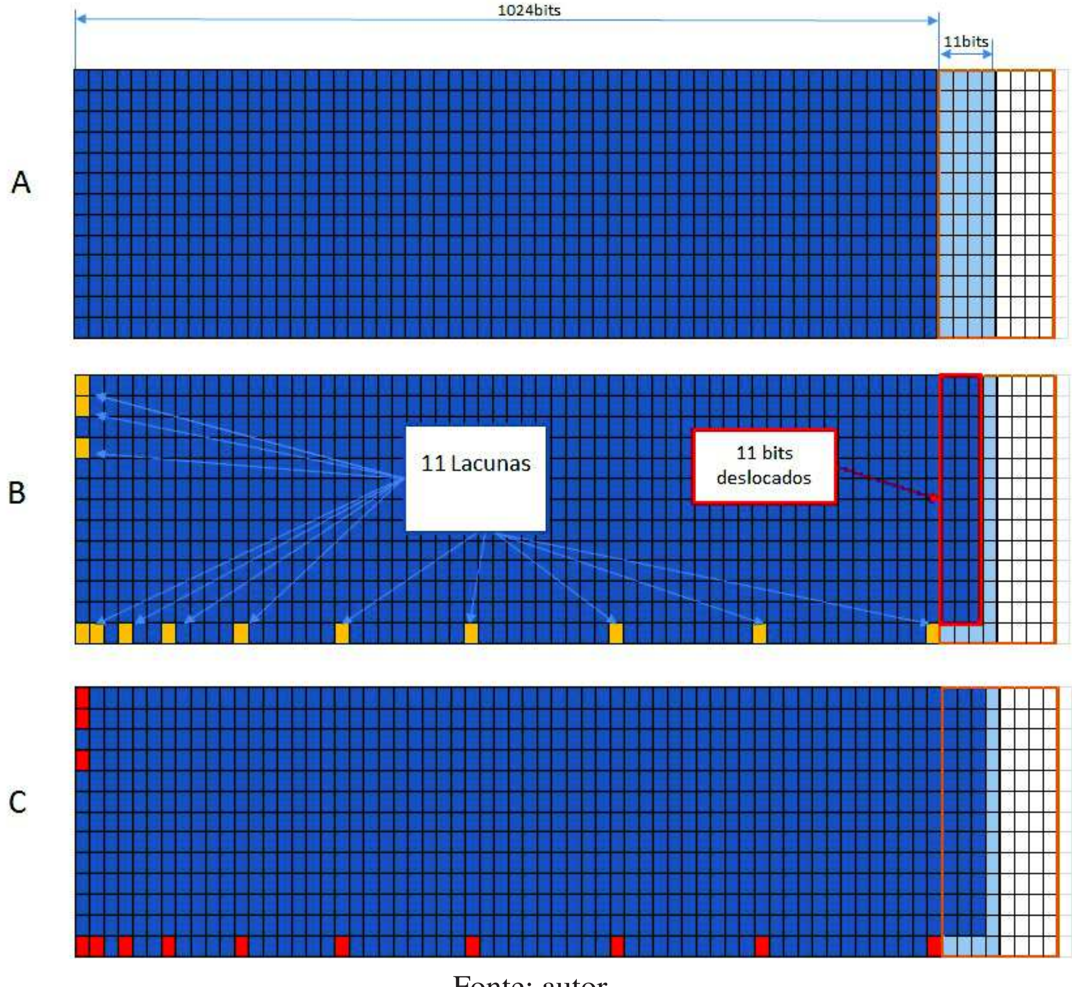
Fonte: autor

A Figura 28 mostra o diagrama de blocos do circuito em VHDL descrito para a codifição e decodificação de Hamming para 1024 bits.

O codificador é composto pelo bloco da memória que guarda a informação para que o bloco do deslocador deixe os endereços das paridades vazios criando lacunas para que sejam preenchidos pelos valores das paridades calculados pelo bloco de cálculo de paridade.

A Figura 29 mostra a sequência de como a a memoria do bloco do codificador é preenchida, o deslocamento da informação dentro da memória e o preenchimento pelos bits de paridade.

Figura 29 – Distribuição da Página - Código Hamming 1024 bits



Fonte: autor

Na Figura 29 mostra a sequência de como a memória do bloco do codificador é preenchida:

a) informação dentro da memória NAND Flash, não há informação na área reserva.

b) informação é deslocada para a criação de lacunas que são espaços vazios na memória, a área reserva é utilizada com 13 bits.

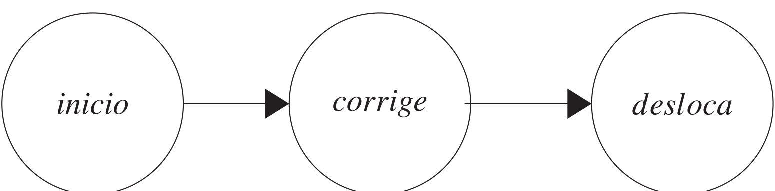
c) as lacunas são preenchidas com o valores de paridade calculados pelo bloco cálculo de paridade.

O decodificador tem os mesmos blocos da codificação acrescido do bloco que verifica e corrige o bit errado. A informação recebida da memória NAND Flash é armazenada na memória, as síndromes são calculadas pelo bloco de cálculo das síndromes e, se todas as síndromes forem iguais a zero, não há erro, e a informação não necessita de correção e o bloco do deslocacor retorna a informação para os endereços originais removendo os bits de paridade. Se alguma das

síndromes não for igual a zero, indica que há uma falha no código e o bloco que verifica e corrige o bit errado calcula a posição do erro através das síndromes, inverte o valor do bit com erro e o bloco do deslocador retorna a informação para a posição original e retira os bits de paridade.

Para a decodificação é utilizada a sequência inversa, são recebidos 1024 bits + 11 bits, calculadas as síndromes, corrigido o bit com erro e as lacunas retiradas. Se não houver erro não há correção do bit de erro.

Diagrama de Estados utilizado dos blocos Desloca e Verifica e Corrige bit Errado do decodificador:



No estado início os registradores auxiliares e contadores são zerados, se o XOR das síndromes for igual a 1 então há erro e vai para estado corrige, se for 0 não há erro e vai para o estado denominado de desloca. No estado corrige, a posição do bit indicado pelas síndromes é corrigido com inversão e depois vai para o estado desloca. O estado desloca retira os bits de paridade e retorna a informação para os endereços originais.

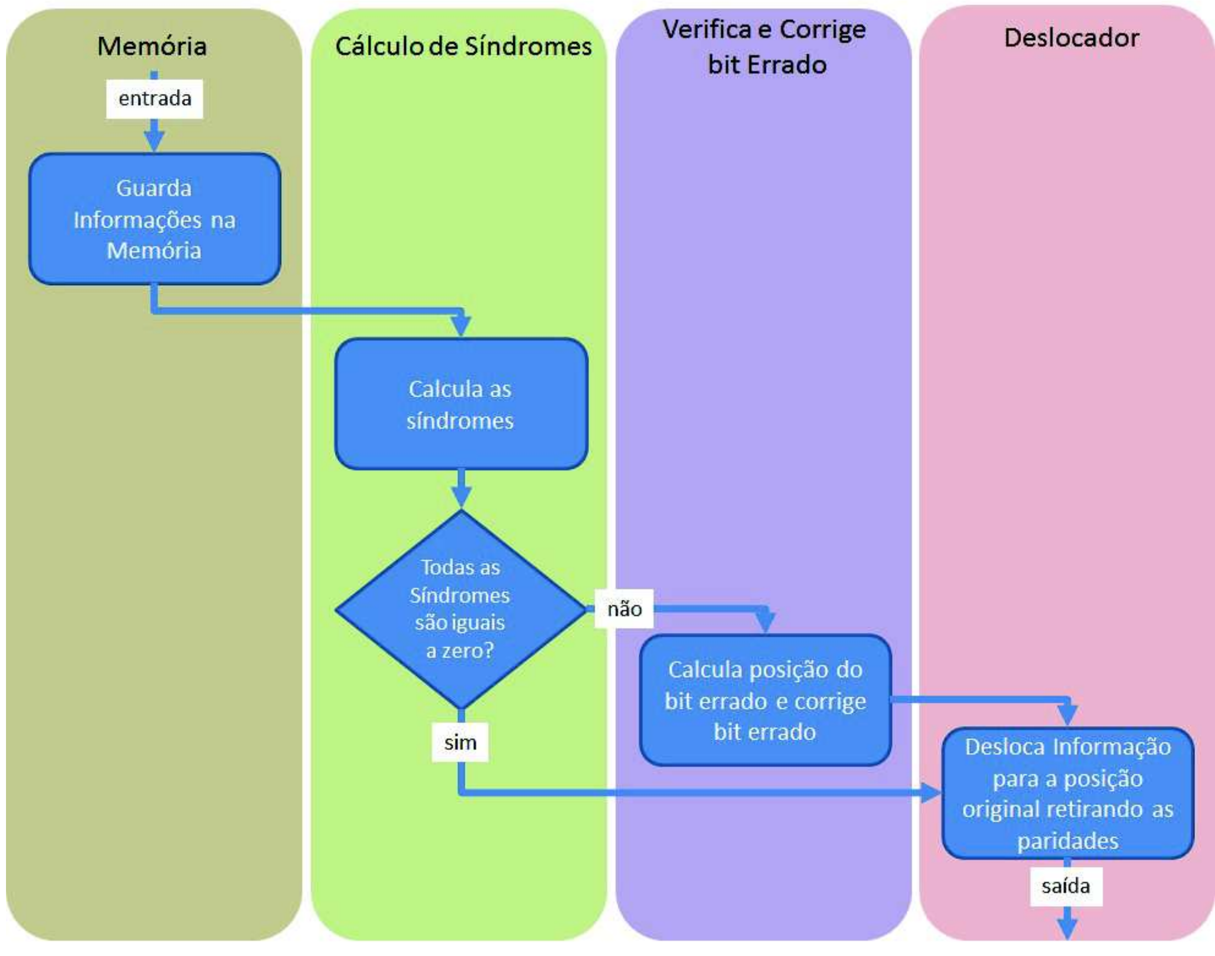
Figura 30 – Fluxograma - Bloco Codificador - Código Hamming para 1024 bits

Memória
Deslocador
Cálculo de Paridades
entrada
Armazena informação na memória interna
Desloca os dados na memória para criar lacunas
Calcula as paridades e guarda nas lacunas da memória
saída

Fonte: autor

Figura 30 mostra o fluxograma do Bloco Codificador para o código de Hamming para 1024 bits. A informação é guardada na memória interna e depois os bits internamente são deslocados para formar as lacunas onde serão armazenados os bits de paridade. O Bloco de Cálculo das Paridades calcula as paridades e armazena nas lacunas.

Figura 31 – Fluxograma - Bloco Decodificador - Código Hamming para 1024 bits



Fonte: autor

A Figura 31 mostra o Fluxograma do Bloco Decodificador do Código de Hamming para 1024 bits. Neste algoritmo a informação que vem da memória NAND Flash é guardada na memória interna. Em seguida são calculadas as síndromes pelo Bloco Cálculo das Síndromes e, se todas as síndromes são iguais a zero então não há erro e a informação é enviada para o bloco Deslocador para retirar as lacunas com as paridades. Se houver síndromes diferentes de zero então há erro na informação e o Bloco Verifica e Corrige o bit errado calcula a posição do bit errado, utilizando as síndromes, localiza o bit e inverte o seu valor e no final o Bloco Deslocador retira as paridades da informação.

5.4.2 Implementação em VHDL do Código BCH para página de 4096 bits

Para a implementação em VHDL do código BCH para uma memória NAND Flash com página de 4096 bits e 64 bits de área reserva, foi-se dividido em vários blocos que serão descritos em detalhes mais adiante. Bloco de Codificação, Bloco de Decodificação, composto de vários blocos menores com memória interna, cálculo de síndromes, cálculo do polinômio de erros e cálculo da localização dos erros.

Para utilizar o código em uma memória NAND Flash com página de 4096 bits e 64 bits de área reserva totalizando 4160 bits, é necessário utilizar um polinômio de grau m = 13. Onde $2^m = 2^{13} = 8192$. A informação tem o tamanho de 4096 bits (512Bytes) que é o tamanho da página e metade de 128 bits ou seja 64 bits (8Bytes) na área reserva. Para a codificação da informação de k=4096 bits, são necessário 13 bits, pois com m=12 temos somente 4096 bits (2^{12}=4096) e necessitamos codificar 4096+64=4160 bits, então são necessários 13 bits. Para a correção de 4 bits, serão necessários 52 bits de paridade.

Figura 32 – Diagrama de Blocos - Código BCH para 4096 bits

Fonte: autor

A Figura 32 mostra o diagrama de blocos do código BCH divididos em codificação e

decodificação. O bloco de decodificação é composto pelos blocos de: Cálculo das Síndromes - onde são calculadas as síndromes, Cálculo Polinômio de Erros - onde são calculados os índices σ do polinômio localizador de erros e Cálculo da Localização de Erros - onde são calculadas as raízes do polinômio localizador de erros que indicam a localização do bit que tem erro.

Para melhor explicação cada bloco do diagrama de blocos da Figura 32 foi dividido em 4 itens:

a) bloco de codificação,

b) bloco de Cálculo das Síndromes,

c) bloco de Cálculo do Polinômio de Erros,

d) bloco de Cálculo da Localização dos Erros.

Explicação dos blocos:

a) Bloco de Codificação
Para codificar a informação com m=13, utiliza-se o Corpo de Galois GF(2^{13}). A tabela de GF(2^{13}) foi gerada através de LSFR com o polinômio mínimo para m=13 $f_1(x) = x^{13}+x^4+x^3+x+1$.

Os polinômios mínimos para GF(2^{13}) segundo (Macronix, 2014):

$$f_1(x) = x^{13}+x^4+x^3+x+1 \tag{36}$$

$$f_3(x) = x^{13}+x^{10}+x^9+x^7+x^5+x^4+x+1 \tag{37}$$

$$f_5(x) = x^{13}+x^{11}+x^8+x^7+x^4+x+x+1 \tag{38}$$

$$f_7(x) = x^{13}+x^{10}+x^9+x^8+x^6+x^3+x^2+x+1 \tag{39}$$

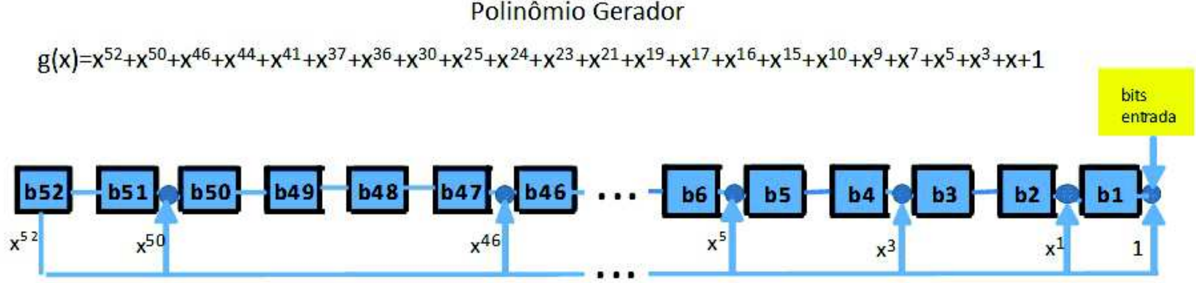
Fazendo g(x) = MMC [f_1(x), f_3(x), f_5(x), f_7(x)] tem-se:

$$\begin{aligned} g(x) = x^{52}+x^{50}+x^{46}+x^{44}+x^{41}+x^{37}+x^{36}+x^{30}+x^{25}+x^{24}+x^{23} \\ +x^{21}+x^{19}+x^{17}+x^{16}+x^{15}+x^{10}+x^9+x^7+x^5+x^3+x^2+x+1 \end{aligned} \tag{40}$$

sendo a informação m(x) e seguindo os passos já descritos no item 3.2.8 tem-se: x^{n-k} * m(x) que é equivalente a acrescentar n-k zeros aos bits menos significativos da informação m(x). Como m(x) tem 4096 bits acrescenta-se 52 zeros após os bits menos significativos dos 4096 bits de m(x).

Para fazer a divisão polinomial de x^{n-k} * m(x) por g(x) foi utilizando o circuito LFSR conforme circuito mostrado na Figura 33.

Figura 33 – Circuito Codificador - Código BCH para 4096 bits



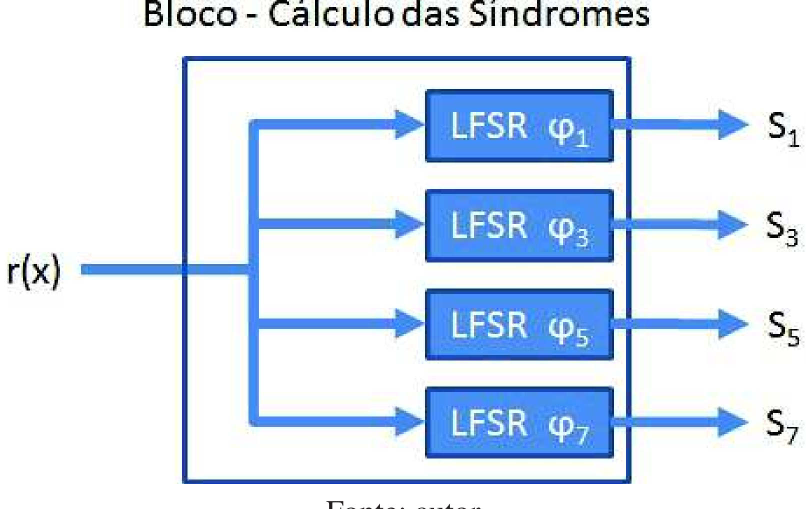
Fonte: autor

A Figura 33 mostra o circuito LFSR de g(x) para fazer o cálculo das paridades da informação m(x) através de divisão polinomial. A figura completa do circuito codificador é mostrada no apêndice A. A descrição do circuito em VHDL utilizando o comando For Generate, é possível criar circuitos de LFSR de qualquer tamanho, através da informação do polinômio 40.

O Bloco de Decodificação conforme já visto, é composto por outros 3 blocos, o primeiro bloco é o bloco que calcula as síndromes, descrito a seguir:

b) Bloco Cálculo das Síndromes:
Para calcular as síndromes, foi utilizado o método da divisão polinomial com o LFSR em que foi utilizada a mesma descrição VHDL do bloco codificador, porém configurado para as equações φ_1, φ_3, φ_5 e φ_7.
Os cálculo de r(x) mod φ_1, r(x) mod φ_3, r(x) mod φ_5 e r(x) mod φ_7

Figura 34 – Diagrama de Blocos - Cálculo das Síndromes



Fonte: autor

c) Bloco de Cálculo do Polinômio de Erros

A seguir são apresentadas as equações para cálculo dos coeficientes do polinômio localizador de erros em função das síndromes conforme Tabela 6 para t = 4:

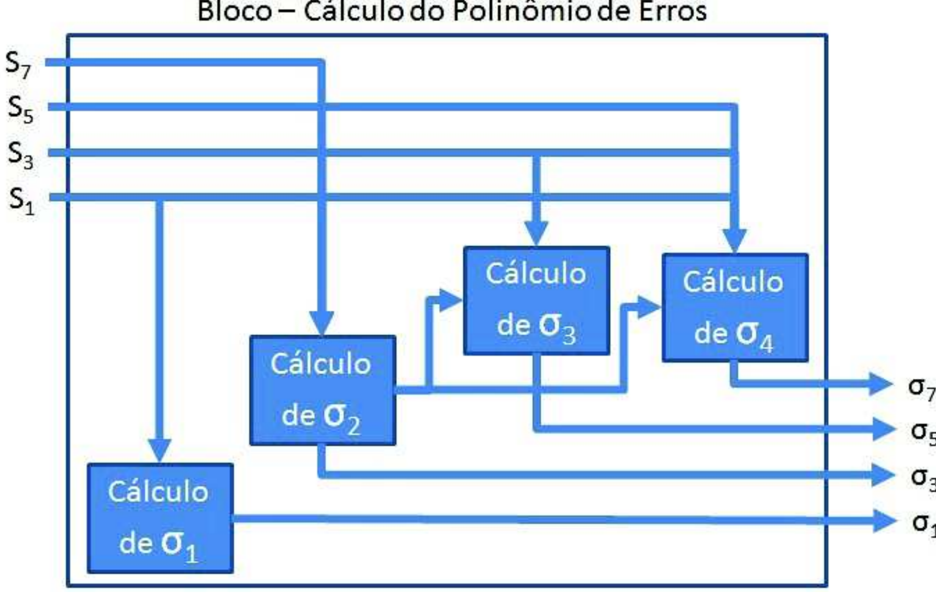
$$\sigma_1 = S_1 \tag{41}$$

$$\sigma_2 = \frac{S_1\left(S_7 + S_1^7\right) + S_3\left(S_1^5 + S_5\right)}{S_3\left(S_1^3 + S_3\right) + S_1\left(S_1^5 + S_5\right)} \tag{42}$$

$$\sigma_3 = \left(S_1^3 + S_3\right) + S_1\sigma_2 \tag{43}$$

$$\sigma_4 = \frac{\left(S_5 + S_1^2 S_3\right) + \left(S_1^3 + S_3\right)\sigma_2}{S_1} \tag{44}$$

Figura 35 – Diagrama de Blocos - Cálculo do Polinômio de Erros



Fonte: autor

Com os valores de $\sigma_1, \sigma_2, \sigma_3$ e σ_4, substituí-se no polinômio localizador de erros.

Polinômio localizador de erros:

$$\sigma(x) = 1 + x\sigma_1 + x^2\sigma_2 + x^3\sigma_3 + x^4\sigma_4$$

d) Bloco de Cálculo da Localização dos Erros

A localização dos erros é realizado pelo bloco Cálculo da Localização de Erros e o algoritmo descrito em VHDL foi realizado através de máquina de estados. A adoção de máquinas de estado para controlar o reuso de *hardware* proporciona uma economia de

hardware em troca de mais tempo de computação.

$$x = \alpha^0 \rightarrow 1 + 1\sigma_1 + 1^2\sigma_2 + 1^3\sigma_3 + 1^4\sigma_4$$

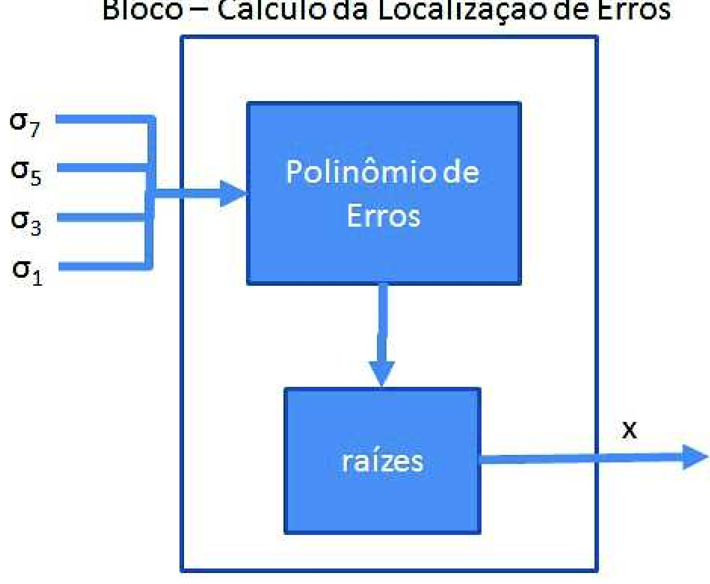
$$\vdots$$

$$x = \alpha^{52} \rightarrow 1 + \alpha^{52}\sigma_1 + \alpha^{52\cdot 2}\sigma_2 + \alpha^{52\cdot 3}\sigma_3 + \alpha^{52\cdot 4}\sigma_4$$

Quando x=0, α^n é raiz.
Fazendo $\frac{1}{\alpha^n} = \alpha^{-n} = \alpha^{52-n} \rightarrow 52 - n \rightarrow$, indica a localização do erro.

Diagrama de Blocos do Cálculo da Localização de Erros é mostrado na Figura 36, no qual os erros são calculados baseados nas informações dos σ calculados pelo bloco anterior.

Figura 36 – Diagrama de Blocos - Cálculo da Localização dos Erros



Fonte: autor

Na Figura 36 observa-se o bloco de Cálculo da Localização dos Erros, no qual os valores de σ calculados pelo bloco anterior são recebidos e utilizados para o cálculo das raízes do polinômio de erros. As raízes são calculadas e indicam a posição dos bits com erro dentro da informação recebida. Os bits errados são invertidos e as paridades retiradas restando ao final a mensagem corrigida.

6 RESULTADOS

São apresentados a seguir os resultados obtidos pela implementação em VHDL dos códigos de 4xHamming1024 e BCH4096. Os arquivos com os resultados das sínteses podem ser encontrados no apêndice C - Resultados - Síntese ISE Xilinx.

Através dos dados dos relatórios das sínteses dos circuitos, foram utilizados 2 métodos de comparação entre os códigos 4xHamming1024 e BCH 4096: o primeiro comparando as primitivas utilizadas pela síntese e o segundo comparando a quantidade de transistores equivalentes utilizados pela descrição de cada código.

6.1 Comparação entre os Recursos Utilizados - Primitivas

Para a comparação dos recursos utilizados pela síntese dos códigos 4xHamming1024 e BCH4096, utilizou-se os dados dos relatórios de síntese gerados pelo ISE.

A Tabela 13 mostra a quantidade de primitivas utilizadas por cada código.

Tabela 13 – Primitivas Utilizadas Pela Implementação em VHDL dos Códigos 4xHamming 1024 e BCH 4096

Primitivas	4 x Hamming 1024				BCH Page 4096 bits			
# BELS	135.716				4.231			
# GND	4				7			
# INV	8				29			
# LUT1	120				60			
# LUT2	28.016				327			
# LUT3	7.808				84			
# LUT4	8.484				89			
# LUT5	21.956				20			
# LUT6	68.760				1.873			
# MUXF7	4				884			
# MUXCY	284				209			
# MUXF8	0				416			
# VCC	4				7			
# XORCY	268				226			
# FlipFlops/Latches	33.004				630			
# FD	0				104			
# FDE	16.492				357			
# FDR	16.456				6			
# FDRE	56				163			
# RAMS	0				85			
# RAMB18E1	0				25			
# RAMB36E1	0				60			
# Shift Registers	0				16			
# SRLC16E	0				16			
# Clock Buffers	8				6			
# BUFG	4				0			
# BUFGP	4				6			
# IO Buffers	60				125			
# IBUF	8				63			
# OBUF	52				62			
Slice Logic Utilization:								
Number of Slice Registers:	33.004	out of	372.480	9%	630	out of	760.800	0%
Number of Slice LUTs:	135.152	out of	186.240	73%	2.498	out of	380.400	1%
Number used as Logic:	135.152	out of	186.240	73%	2.482	out of	380.400	1%
Number used as Memory:	0	out of	0		16	out of	38.000	0%
Number used as SRL:	0				16			
Slice Logic Distribution:								
Number of LUT Flip Flop pairs used	135.152				2.662			
Number with an unused Flip Flop	102.148	out of	135.152	76%	2.032	out of	2.662	76%
Number with an unused LUT:	0	out of	135.152	0%	164	out of	2.662	6%
Number of fully used LUT-FF pairs	33.004	out of	135.152	24%	466	out of	2.662	18%
Number of unique control sets:	24				43			
IO Utilization:								
Number of IOs:	64				237			
Number of bonded IOBs:	64	out of	960	7%	178	out of	1.470	12%
Specific Feature Utilization:								
Number of Block RAM/FIFO:	0	out of	0	0%	75	out of	810	9%
Number using Block RAM only:	0				75			
Number of BUFG/BUFGCTRLs:	8	out of	128	6%	6	out of	192	3%

Observa-se que há primitivas iguais a zero na síntese do código 4xHamming1024 e maiores que zero na síntese do código BCH4096, como por exemplo:

Tabela 14 – Comparação Entre Primitivas

Primitiva	**4xHamming1024**	**BCH4096**
FD (Flip Flop D)	0	104
Shift Register (Registrador de Deslocamento)	0	85

A síntese do circuito do código 4xHamming1024 não utiliza Flip Flops D, enquanto o circuito do código BCH4096 utiliza 104. O mesmo para *Shift Register*: 0 para o circuito 4xHamming1024 e 85 para BCH4096. A comparação entre essas duas primitivas não é possível, pois não há dados suficentes para concluir se o Flip Flop D é equivalente a alguma outra primitiva que seja usada pelas duas sínteses, o mesmo ocorre para o *Shift Register*. Conclui-se que a comparação dos códigos através de primitivas não é possível, pois não há uma estrutura de referência comum entre as primitivas.

6.2 Comparação entre os Recursos Utilizados - Transistores

Para fazer a comparação entre os códigos utilizando uma mesma base foi feita a comparação utilizando a quantidade de transistores equivalentes utilizados para a descrição de cada código. A descrição dos circuitos dos códigos foi feita em linguagem VHDL com a ferramenta ISE que gera um relatório da síntese do circuito. Este relatório mostra as quantidades de primitivas necessárias para descrever o circuito. Primitivas são circuitos digitais básicos dedicados como por exemplo: memória, registrador, contador, comparador, somador, multiplexador, portas lógicas XOR, entre outros.

Para exemplificar a transcrição de primitiva para transistores, utilizamos a primitiva Registrador que é um Flip-Flop, o qual é composto de 16 transistores conforme circuito mostrado no anexo B, esse procedimento foi feito para cada primitiva.

Tabela 15 – Transistores Utilizados Pela Implementação em VHDL dos Códigos 4x Hamming 1024 e BCH 4096

1	2	3	4	5	6	7
Primitivas	**Recursos Utilizados - Primitivas**	**transistores**	Qtde Primtivas		Qtde Transistores	
			TOTAL Ham4	TOTAL BCH	TOTAL Ham4	TOTAL BCH
# RAMs	4096x1-bit single-port block Read Only RAM	4.096	0	5	0	20.480
	8192x13-bit single-port block Read Only RAM	106.496	0	12	0	1.277.952
	8192x13-bit single-port distributed Read Only RAM	106.496	0	1	0	106.496
	8192x14-bit single-port block Read Only RAM	114.688	0	8	0	917.504
	8192x13-bit dual-port block Read Only RAM	106.496	0	4	0	425.984
# Multipliers	13x2-bit registered multiplier	476	0	1	0	476
	13x3-bit registered multiplier	952	0	2	0	1.904
# MACs	13x2-to-13-bit MAC	1.334	0	1	0	1.334
# Adders/ Subtractors	13-bit adder	650	4	10	2.600	6.500
	13-bit subtractor	650	4	2	2.600	1.300
	32-bit subtractor	1.600	4	0	6.400	0
	5-bit adder	250	4	0	1.000	0
# Counters	11-bit up counter	550	4	0	2.200	0
	13-bit up counter	650	0	6	0	3.900
	4-bit up counter	200	4	1	800	200
# Registers	Flip-Flops	16	8288	936	132.608	14.976
# Comparators	11-bit comparator greater	104	4	0	416	0
	13-bit comparator equal	124	0	1	0	124
	13-bit comparator lessequal	124	0	5	0	620
	32-bit comparator equal	314	4	0	1.256	0
	4-bit comparator greater	34	4	0	136	0
# Multiplexers	1-bit 2-to-1 multiplexer	4	4100	0	16.400	0
	1-bit 1036-to-1 multiplexer	4.144	4	0	16.576	0
	1-bit 2-to-1 multiplexer	4	4144	0	16.576	0
	13-bit 2-to-1 multiplexer	52	0	16	0	832
# Logic shifters	32-bit shifter logical left	512	4	0	2.048	0
# Xors	1-bit xor2	16	45080	104	721.280	1.664
	1-bit xor3	32	8	0	256	0
	1-bit xor4	48	24	0	1.152	0
	1-bit xor5	64	8	0	512	0
	1-bit xor6	80	4	0	320	0
	13-bit xor2	208	0	9	0	1.872
	13-bit xor4	416	0	1	0	416
					Total Transistores	
					925.136	2.784.534

Fonte: autor

Da esquerda para a direita tem-se numeradas as colunas. Os dados das colunas 1, 2, 4 e 5 foram gerados pelo relátorio de síntese da ferramenta ISE. A primeira coluna contém as primitivas utilizadas pelos circuitos sintetizados, a segunda coluna contém a descrição das primitivas, a terceira coluna mostra a quantidade de transistores utilizados para sintetizar a primitiva cujas descrições estão no Apêndice B, a quarta e quinta coluna mostram as quantidades de primitivas utilizadas para descrever os circuitos do códigos 4xHamming1024bits e BCH 4096 bits respectivamente. A sexta e sétima mostram as quantidades de transistores para cada primitiva utilizada, e foram obtidas multiplicando-se a coluna 3 pela coluna 4 e 5 para obter as colunas 6 e 7.

Ao final das colunas 6 e 7 tem-se as somas totais de transistores equivalentes utilizados pelos circuitos dos códigos 4xHamming 1024 bits e BCH 4096. Verifica-se que a menor quantidade de transistores utilizados é da coluna 6 do código 4xHamming1024, enquanto o código BCH utiliza aproximadamente 3 vezes mais transistores.

Para uma melhor análise, a Tabela 15 foi rearranjada de forma a mostrar percentualmente a distribuição das primitivas utilizada por cada código, conforme Tabela 16.

Tabela 16 – Distribuição de Transistores

Primitivas	4xHam1024		BCH 4096	
# RAMs	0	0,00%	2.748.416	**98,70%**
# Multipliers	0	0,00%	2.380	0,09%
# MACs	0	0,00%	1.334	0,05%
# Adders/Subtractors	12.600	1,36%	7.800	0,28%
# Counters	3.000	0,32%	4.100	0,15%
# Registers	132.608	**14,33%**	14.976	**0,54%**
# Comparators	1.808	0,20%	744	0,03%
# Multiplexers	49.552	5,36%	832	0,03%
# Logic shifters	2.048	0,22%	0	0,00%
# Xors	723.520	**78,21%**	3.952	0,14%
Total Transistores	**925.136**	100,00%	**2.784.534**	100,00%

Fonte: autor

Pelos dados da Tabela 16 observa-se que para o código de 4xHamming1024bits as primitivas que mais utilizam recursos na forma de transistores são as portas XOR com 78,21% e os registradores com 14,33% e que juntos somam 92,54%. As XORs são as primitivas mais utilizadas pois o algoritmo do código de Hamming é baseado em XORs entre os seus componentes para o cálculo das paridades. Os registradores são a segunda maior demanda de recursos, pois os dados de entrada, os 1024 bits são guardados em registradores tanto na codificação como na decodificação.

No caso do código BCH 4096 bits as memórias RAM foram as primitivas que mais consumiram recursos, mesmo considerando apenas 1 transistor por bit, elas são responsáveis por 98,7% dos transistores utilizados pelo código BCH. A explicação para isso é devido a utilização de tabelas internas com os valores do Corpo de Galois com 8k posições cada uma com 13 bits, portanto cada tabela consome mais de 106k bits, além das duas memórias que guardam a informação na entrada e saída com 4.148 bits. O segundo maior consumo de primitivas são os

registradores com 0,54% que podem ser explicados pelos registradores de deslocamento (LFSR) utilizados na codificação e cálculos das síndromes.

6.2.1 Comparação entre os Tempos de Computação

Os tempos totais de computação são os tempos utilizados pelos blocos para fazer a codificação e decodificação. Portanto o código que utiliza menos tempo de computação é o código mais rápido.

Para o cálculo dos tempos de computação, seguiram-se os seguintes passos: através do relatório implementação da síntese gerado pela ferramenta VHDL determina-se o valor do período mínimo ou a máxima frequência que o circuito pode processar. Com os dados do Test Bench da simulação do programa ISE foi possível determinar qual o tempo total utilizado por cada código. A Tabela 17 mostra os tempos de computação para a implementação em VHDL do código 4xHamming1024 e o código BCH.

Tabela 17 – Tempos de Computação para Código 4xHamming1024 e Código BCH4096

Descrição	TOTAL Ham4		TOTAL BCH			
	bloco codificação	bloco decodificação	bloco codificação	cálculo das síndromes	cálculo dos Sigma	Cálculo das raízes
Período Mínimo:	**4,135ns**	**7,212ns**	**2,246ns**	**2,661ns**	**3,456ns**	**2,919ns**
Frequencia Máxima:	241,844Mhz	138,658MHz	445,177MHz	375,770MHz	289,320MHz	342,630Mhz
tempo de computação da ferramenta ISE	10.225ns	20ns	414.900.000ps	41,39us	455ns	1.474,585us
período do clock	10ns	10ns	10ns	10ns	10ns	10ns
número de ciclos de clocks	**1.024**	**2**	**41.490**	**4.139**	**46**	**147.459**
tempo de computação (período mínimo x número de ciclos de clocks)	4,23us	14,42ns	93,18us	44,05us	157,24us	430,43us
total	**16,97us**		**724,90us**			

Fonte: autor

O programa em seu relatório de síntese mostra o período mínimo e a frequência máxima como mostrados na tabela. Os dados dos tempos de computação do programa ISE foram obtidos através da ferramenta Test Bench e foram medidos os tempos totais de processamento de cada bloco implementado e dividos pelo seu período de clock, com isso obtém-se o número de ciclos de clocks necessários para a execução da tarefa pelo circuito. Com os tempos do período mínimo de cada bloco e o número de ciclos de clocks utilizado por cada bloco para executar a tarefa foi calculado o tempo de computação de cada bloco implentado de acordo com a fórmula abaixo:

$$TC = N_{ciclos} x Periodo_{Ciclo}$$

$$TC(tempodecomputacao)$$

Fazendo a comparação dos tempos de processamento, o código de Hamming com 4 blocos de 1024 bits é o que utiliza menos tempo aproximadamente 46 vezes menos tempo que o código BCH 4096 bits para realizar a codificação e decodificação dos mesmos 4096 bits. O código Hamming com 4 blocos de 1024 bits é o mais rápido porque utiliza menos recursos (transistores) e utiliza circuitos combinacionais com portas XORs. O circuito do código BCH consome mais tempo que o circuito de Hamming por que utiliza algoritmos que demandam mais tempo com operações matemáticas com somas e multiplicações e consulta a tabelas. No bloco cálculo das raízes foi descrito através de uma máquina de estados para fazer a verificação dos 8.192 valores de α para determinar quais deles são as raízes da equação de erros. Novamente a adoção de máquinas de estado foi utilizada para otimizar o uso dos recursos de *hardware* do dispositivo alvo (Artix 7).

O uso da máquina de estados foi devido ao fato de que a placa Artix 7 não possuir recursos para realizar a sintetização do circuito do bloco Cálculo das Ráizes da forma combinacional.

Mesmo que fosse possível a implementação do bloco Cálculo das Raízes através de lógica combinacional, o tempo de computação do circuito com o código BCH foi maior que o do código de Hamming com 4 blocos de 1024 bits. Todos os blocos do circuito do código BCH 4096 bits têm tempos de computação maiores do que 16,97us que é o tempo de computação total do código de Hamming de 4 blocos de 1024 bits, conforme observa-se na Tabela 17.

7 CONCLUSÃO

Este trabalho teve como objetivo a comparação entre os códigos de ECC aplicados à memória NAND Flash e a identificação de qual é o melhor código do ponto de vista comercial.

Na comparação entre os códigos de 4xHamming 1024 bits e BCH 4096 bits, foram tiradas as seguintes conclusões:

a) Sobre a quantidade de bits de paridade necessários.

   Como visto anteriormente, o código de Hamming quando aplicado à página de 4096 bits de uma memória NAND Flash comercial de 4GB da fabricante Micron não pôde ser comparada ao código BCH por não possuir a mesma cobertura já que o código de Hamming detecta e corrige 1 bit para 4096 bits enquanto o código BCH detecta e corrige 4 bits para a mesma página de 4096 bits. Em termos de bits de paridade necessários, o código de Hamming necessita de 13 bits para cobrir os 4096 bits enquanto o código BCH necessita de 52 bits de paridade, porém o código de Hamming, quando aplicado em blocos de 1024 bits, necessita de 11 bits para detectar e corrigir 1 bit. Então dividindo-se 4096 bits em 4 blocos de 1024 bits tem-se 11 bits de paridade para cada bloco totalizando 44 bits de paridade e uma cobertura de 1 bit por bloco, totalizando 4 bits de detecção e correção para 4096 bits. Desta forma, foi possível comparar o código de Hamming com o código BCH. O código de Hamming com 4 blocos de 1024 bits necessita de 44 bits de paridade enquanto que o código BCH para 4096 bits necessita 52 bits de paridade.

b) Sobre a quantidade de recursos necessários para implementar o circuito do código de Hamming para 4 blocos de 1024 bits e o código BCH para 4096 bits.

   Os dois códigos foram implementados na linguagem VHDL e o programa ISE gera relatórios com os recursos de hardware utilizados pelos circuitos implementados. Para uma comparação mais justa dos recursos utilizados pelas duas implentações, os recursos foram transcritos em transistores e feita a comparação. Observando-se a Figura da Tabela 15 verificamos que a implementação do código de Hamming com 4 blocos de 1024 bits utiliza menos transistores do que a implementação do código BCH. As quantidades de transitores utilizados para o circuito com o código BCH são aproximadamente três vezes maiores do que os utilizados pelo código de Hamming com 4 blocos de 1024 bits. A quantidade de transistores equivalentes pode ser utilizada como métrica para indicar também a área de silício utilizada pelo circuito. Logo pela área ocupada pelo circuito e pelos resultados apresentados, pode-se dizer que o código de Hamming de 4 blocos de 1024 bits é 3 vezes menor do que a área ocupada pelo circuito do código BCH para 4096 bits.

c) Sobre os Tempo de Computação.

O relatório do programa ISE mostra o período mínimo de cada implementação, assim foi feita uma análise dos períodos mínimos para cada bloco implementado conforme mostra a Tabela da Figura 15. Para cada implementação foi medido o número de ciclos de clock necessários para se realizar a tarefa de cada bloco. Com o período mínimo e o número ciclos de clock conseguiu-se saber o tempo de computação de cada bloco e assim chegar-se ao tempo total (tempo total = tempo de codificação + tempo de decodificação) utilizado pelo circuito de cada código. Novamente o código de Hamming com 4 blocos de 1024 bits é mais vantajoso que o código BCH pois utiliza menos tempo. O código de Hamming com 4 blocos de 1024 bits necessita de 16,96us enquanto o código BCH para 4096 bits necessita de 724,94us.

A conclusão final é de que o código de Hamming com 4 blocos de 1024 bits é o mais indicado para uso comercial, pois é o código que utiliza menos bits de paridade, consome menos recursos de hardware e é o mais rápido.

7.1 Trabalhos futuros

Para trabalhos futuros visualiza-se a implementação dos códigos já descritos em VHDL em conjunto com uma descrição de um controlador de NAND Flash e utilizar uma placa de FPGA para emular um controlador de NAND Flash para então ligar a uma memória NAND Flash física para assim, validar o código e possibilitar comprovar os resultados obtidos neste trabalho em um circuito real, assim realizando a análise de aplicabilidade do mesmo.

REFERÊNCIAS

ALMEIDA, G. m. *Códigos Corrretores de Erros em Hardware para Sistemas de Telecomando e Telemetria em Aplicações Espaciais*. 96 p. Tese (Doutorado) — Pontificia Universidade Católica do Rio Grande do Sul, 2011. Disponível em: <http://hdl.handle.net/10923/1462>. Citado 2 vezes nas páginas 41 e 43.

BARROS, F. Ferri-de. ECC in Micron Single-Level Cell (SLC) NAND. *Micron*, v. 31, n. 4, p. e37, 2011. ISSN 1539-2570. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/21584672>. Citado na página 26.

BOSE, R. *Information Theory, Coding and Cryptography*. Tata McGraw-Hill Education, 2008. 326 p. ISBN 0070669015. Disponível em: <https://books.google.com/books?id=AizEjrRIEHYC&pgis=1>. Citado na página 49.

BOSE, R.; RAY-CHAUDHURI, D. On a class of error correcting binary group codes. *Information and Control*, v. 3, n. 1, p. 68–79, 1960. ISSN 00199958. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0019995860902874>. Citado na página 40.

CHEN, S. What Types of ECC Should Be Used on Flash Memory ? Application Note. *ReVision*, p. 1–5, 2007. Citado 2 vezes nas páginas 25 e 29.

FREGNI, E.; SARAIVA, A. M. *Engenharia do Projeto Lógico Digital - Conceitos e Prática*. [S.l.]: Edgard Blucher, 1995. 498 p. ISBN 8521201125. Citado 4 vezes nas páginas 35, 37, 38 e 39.

HAMMING, R. W. *Error Detecting and Error Correcting Codes*. 1950. 147–160 p. Citado 5 vezes nas páginas 56, 57, 58, 60 e 61.

JIANG, Y. *A practical guide to error-control coding using Matlab*. Artech House, 2010. 281 p. ISBN 9781608070886. Disponível em: <http://books.google.com/books?hl=en&lr=&id=uZRz-1PXMb8C&oi=fnd&pg=PP6&dq=a+practical+guide+to+error+control+coding+using+MATLAB&ots=5Jbp3af4Iv&sig=OQUk2oh_y3IjHuuwhBx--wo1iAs>. Citado 3 vezes nas páginas 41, 42 e 57.

JUSTESEN, J.; HOHOLDT, T. *A Course in Error-Correcting Codes*. [s.n.], 2004. 192 p. ISBN 9783037190012. Disponível em: <http://www.amazon.com/Course-Error-Correcting-Codes-Textbooks-Mathematics/dp/3037190019/ref=sr_1_1?s=books&ie=UTF8&qid=1399217103&sr=1-1>. Citado na página 45.

LIN, C.-s.; CHEN, K.-y.; WANG, Y.-h. A NAND Flash Memory Controller for SD / MMC Flash Memory Card. *Memory*, p. 1284–1287, 2006. Citado 2 vezes nas páginas 26 e 56.

LOPES, G. F. *ANÁLISE DE IMPACTO DE ALGORITMOS DO CONTROLADOR DE MEMÓRIA NA PERFORMANCE DE MEMÓRIA NAND FLASH*. 44 p. Tese (Doutorado) — Unisinos, 2015. Citado na página 32.

Macronix. *NAND Error Correction Codes Introduction Error Correct Code : The Fundamental Operation*. 2014. 1–17 p. Citado 2 vezes nas páginas 79 e 90.

MASUOKA, F. et al. New ultra high density EPROM and flash EEPROM with NAND structure cell. *1987 International Electron Devices Meeting*, v. 33, p. 552–555, 1987. ISSN 01631918. Citado 2 vezes nas páginas 30 e 73.

MICHELONI, R.; MARELLI, A.; RAVASIO, R. *Error correction codes for non-volatile memories*. [S.l.: s.n.], 2008. 1–337 p. ISBN 9781402083907. Citado 6 vezes nas páginas 31, 32, 36, 46, 63 e 79.

Micron. Hamming Codes for NAND Flash Memory Devices. *Micron*, p. 1–7, 2005. Citado na página 79.

Micron. Numonyx ® NAND SLC small page 70 nm Discrete. n. October, p. 1–51, 2012. Citado 4 vezes nas páginas 31, 32, 40 e 79.

Micron. *Ddr4 Sdram Rdimm*. 2013. 1–19 p. Citado na página 40.

MOZHIARASI, P.; GAYATHRI, M. C. A Mathematical Derivation for Error Correction and Detection in Communication using BCH Codes. v. 2, n. 10, p. 6–12, 2014. Citado 2 vezes nas páginas 44 e 55.

MUTLU, O. Error Analysis and Management for MLC NAND Flash Memory. 2014. Citado na página 25.

PANDA, A. K.; SARIK, S.; AWASTHI, A. FPGA implementation of encoder for (15, k) binary BCH code using VHDL and performance comparison for multiple error correction control. In: *Proceedings - International Conference on Communication Systems and Network Technologies, CSNT 2012*. [S.l.: s.n.], 2012. p. 780–784. ISBN 9780769546926. Citado 2 vezes nas páginas 73 e 74.

PETERSON, W. Encoding and error-correction procedures for the Bose-Chaudhuri codes. *IEEE Transactions on Information Theory*, v. 6, n. 4, p. 459–470, 1960. ISSN 0018-9448. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1057586>. Citado 2 vezes nas páginas 51 e 52.

PETERSON, W.; BROWN, D. Cyclic Codes for Error Detection. *Proceedings of the IRE*, v. 49, n. 1, p. 228–235, 1961. ISSN 0096-8390. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4066263>. Citado 4 vezes nas páginas 43, 47, 48 e 54.

REGULAPATI, V. *Error Correction Codes in NAND Flash Memory*. 66 p. Tese (Doutorado) — University of Texas, 2015. Citado 5 vezes nas páginas 33, 73, 79, 80 e 81.

RINO MICHELONI LUCA, C. A. M. *Inside NAND Flash Memories*. [S.l.]: springer, 2010. 1–582 p. ISBN 978-90-481-9430-8. Citado 11 vezes nas páginas 25, 26, 29, 30, 31, 32, 56, 62, 63, 65 e 80.

RUMSEY, F.; WATKINSON, J. *Digital interface handbook*. [S.l.: s.n.], 2004. v. 2004. 388 p. ISBN ISBN-10: 0-240-51909-4. Citado 2 vezes nas páginas 64 e 65.

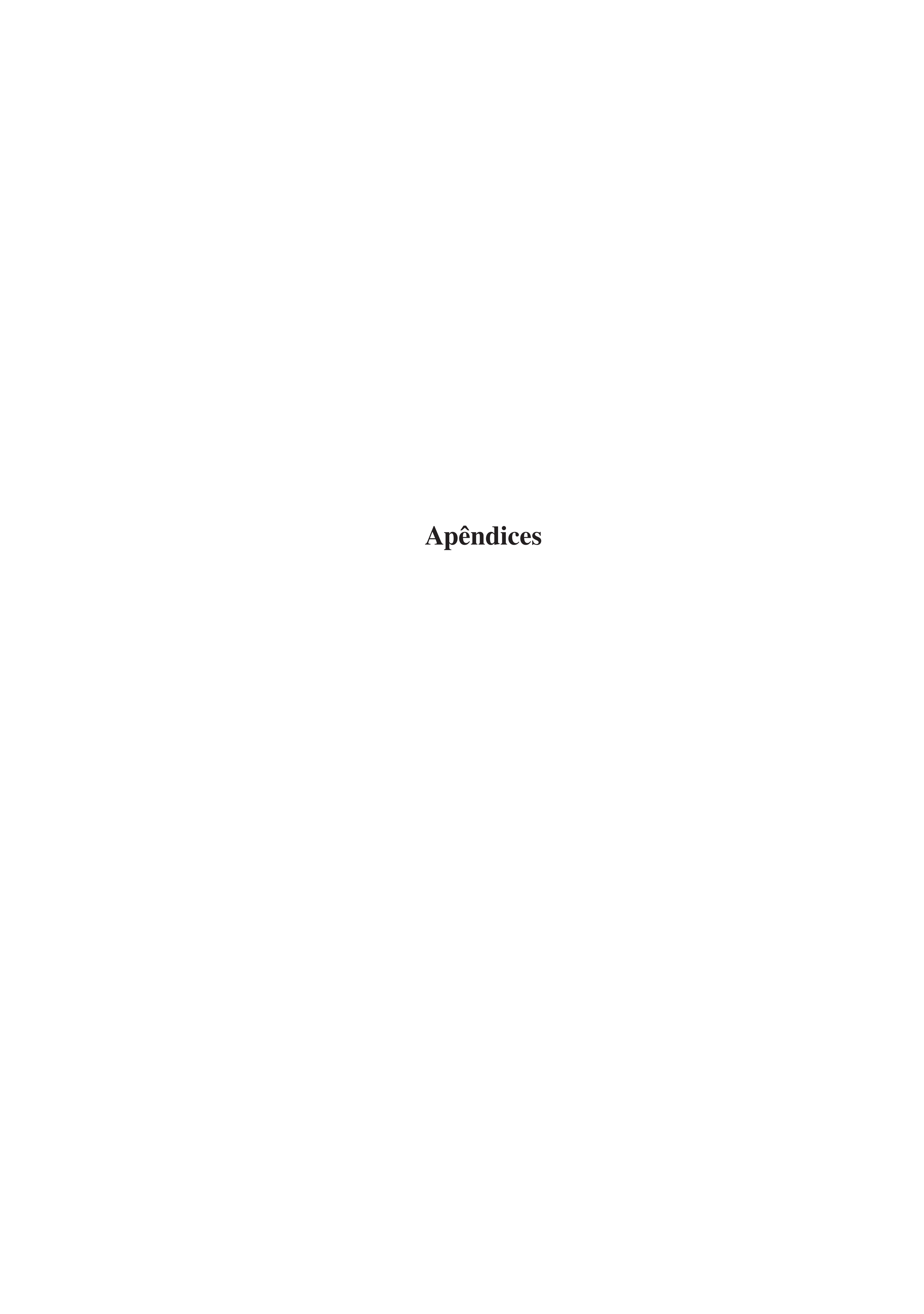
SKLAR, B. Reed-Solomon Codes by. *Journal of the Society for Industrial and Applied Mathematics*, v. 1, n. 3, p. 646–56, 1960. Disponível em: <http://hscc.cs.nthu.edu.tw/~sheujp/lecture_note/rs.pdf>. Citado 6 vezes nas páginas 62, 64, 65, 66, 68 e 69.

SWEENEY, P. *Error Control Coding: From Theory to Practice*. [S.l.: s.n.], 2002. ISBN 047084356X. Citado na página 56.

TANAKAMARU, S.; YANAGIHARA, Y.; TAKEUCHI, K. Error-prediction LDPC and error-recovery schemes for highly reliable solid-state drives (SSDs). *IEEE Journal of Solid-State Circuits*, v. 48, n. 11, p. 2920–2933, 2013. ISSN 00189200. Citado na página 35.

TANZAWA, T. et al. A compact on-chip ECC for low cost flash memories. *IEEE Journal of Solid-State Circuits*, v. 32, n. 5, p. 662–668, 1997. ISSN 00189200. Citado 2 vezes nas páginas 37 e 73.

YAAKOBI, E. et al. Error Correction Coding For Memories Flash. *CMRR Report Number 31*, n. 31, 2009. Citado na página 73.

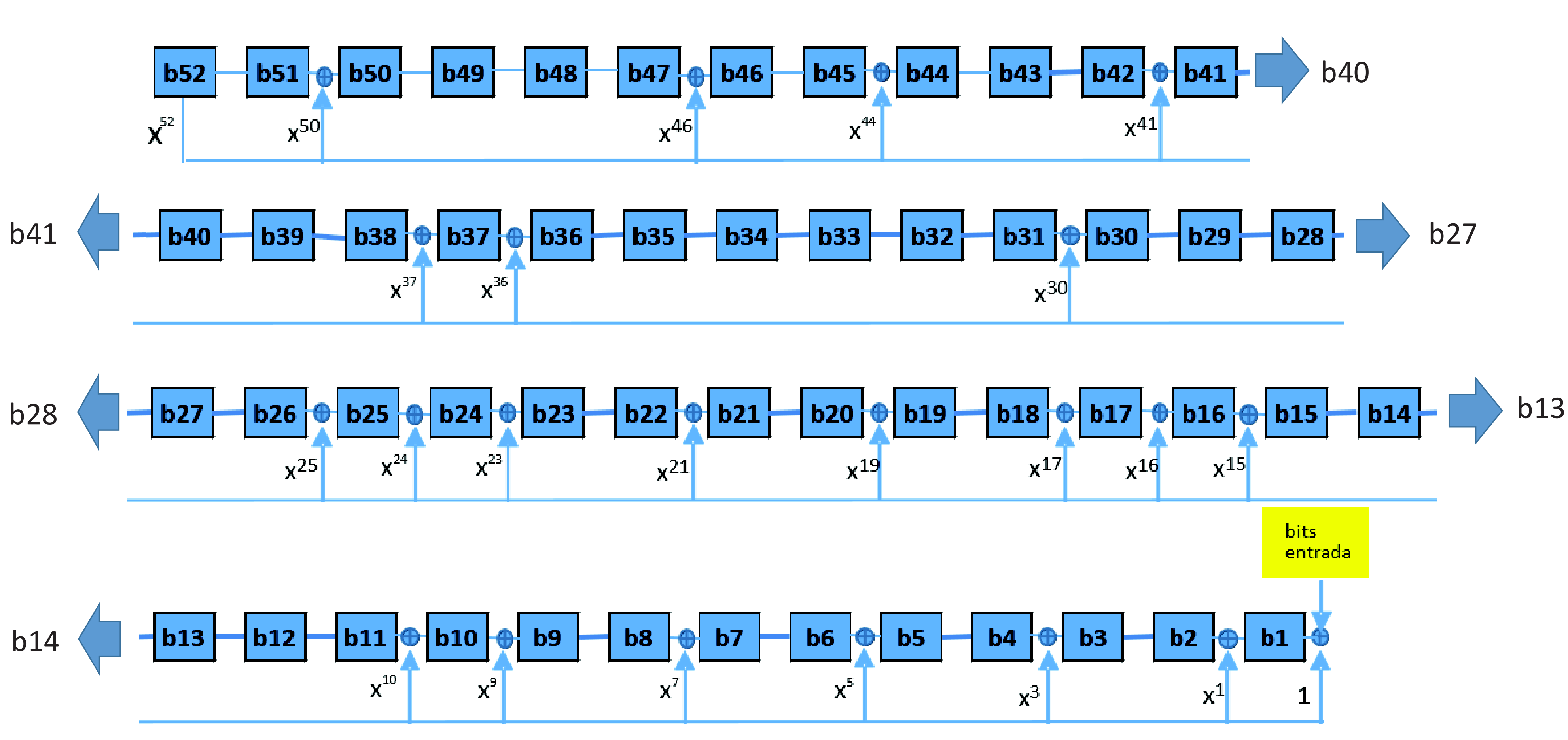


Apêndices

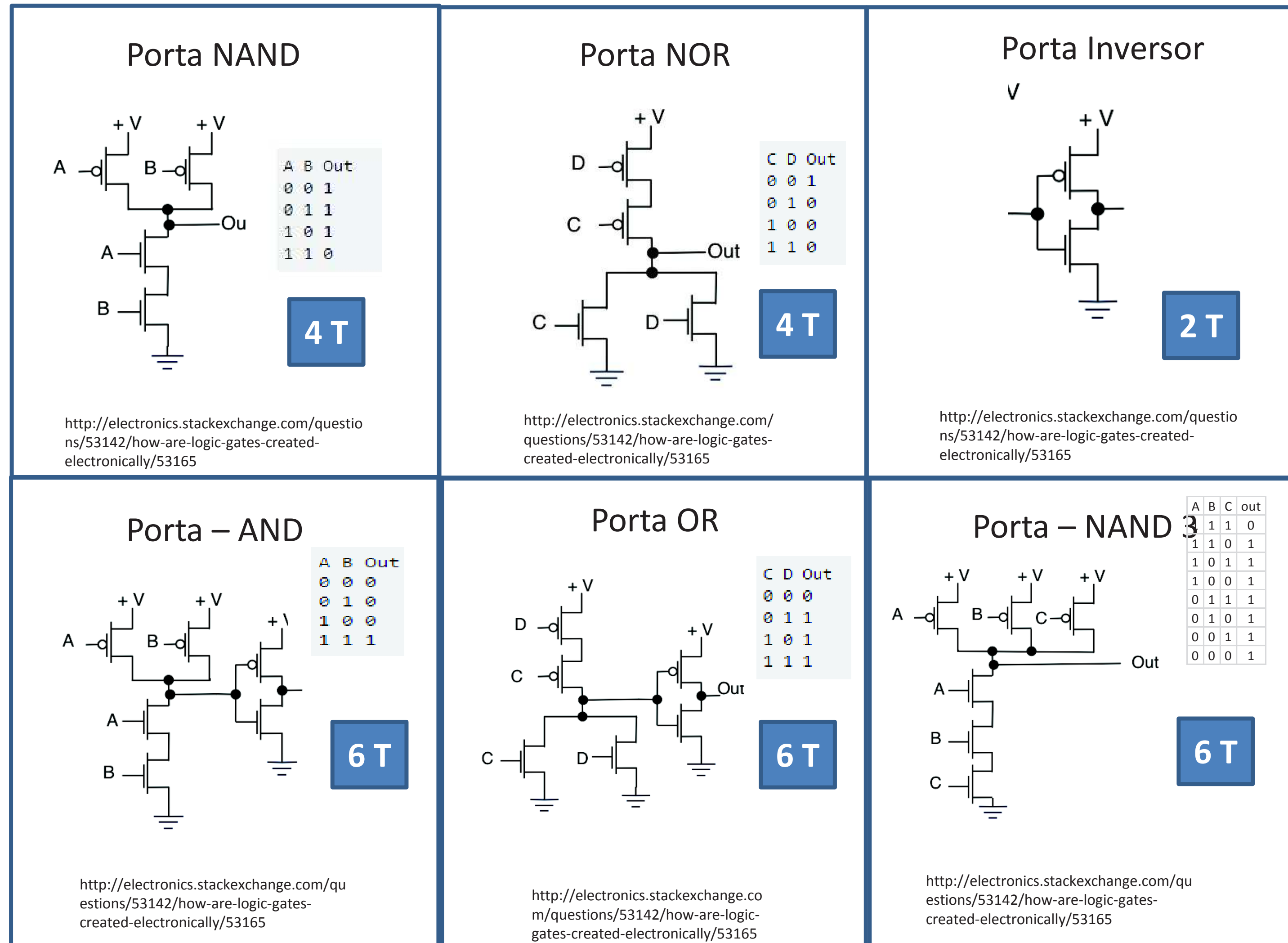
APÊNDICE A – CIRCUITO DE DIVISÃO POLINOMIAL - BCH 4096 BITS

Circuito LSFR para g(x)

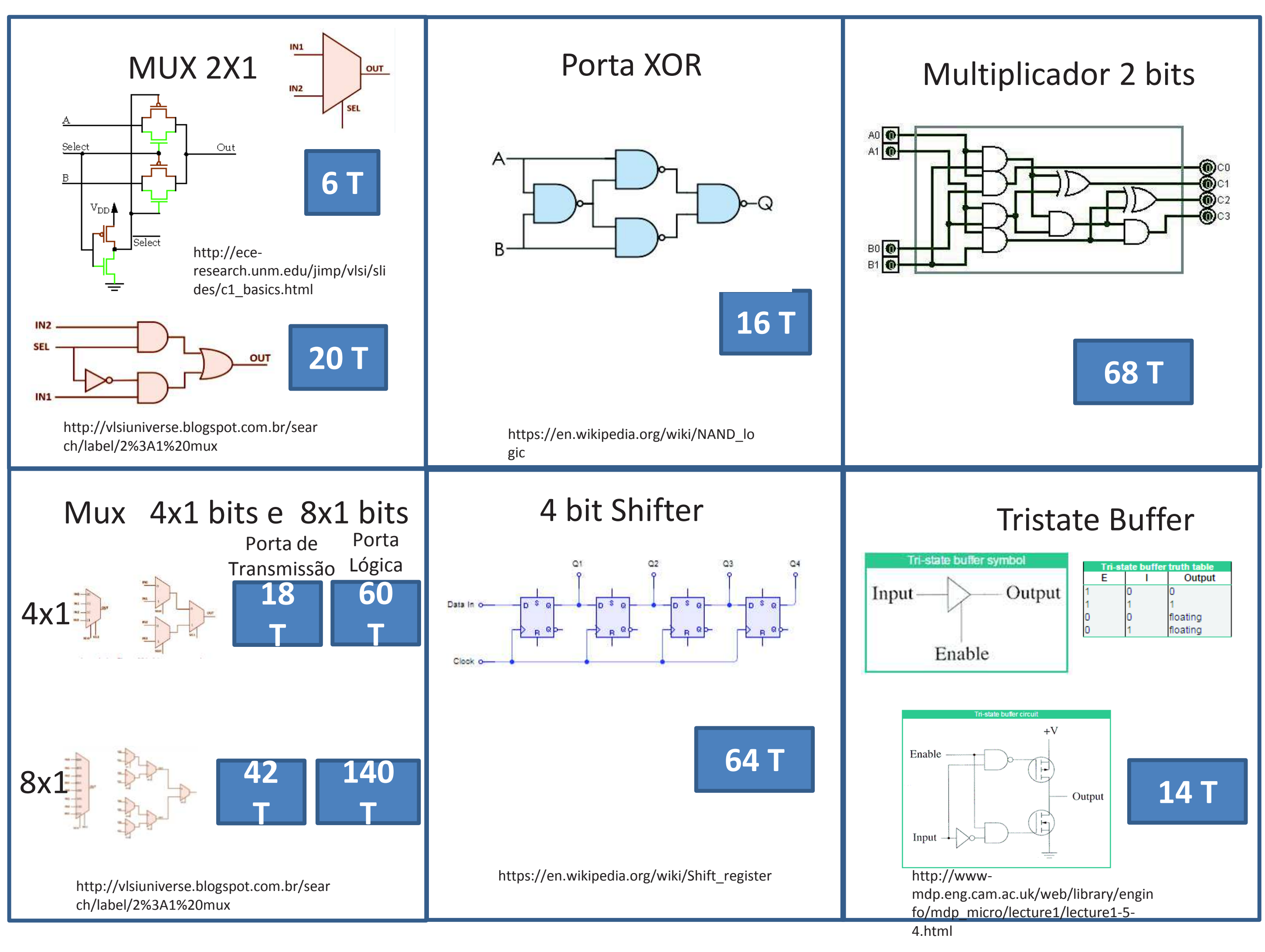
$$g(x)=x^{52}+x^{50}+x^{46}+x^{44}+x^{41}+x^{37}+x^{36}+x^{30}+x^{25}+x^{24}+x^{23}+x^{21}+x^{19}+x^{17}+x^{16}+x^{15}+x^{10}+x^{9}+x^{7}+x^{5}+x^{3}+x+1$$

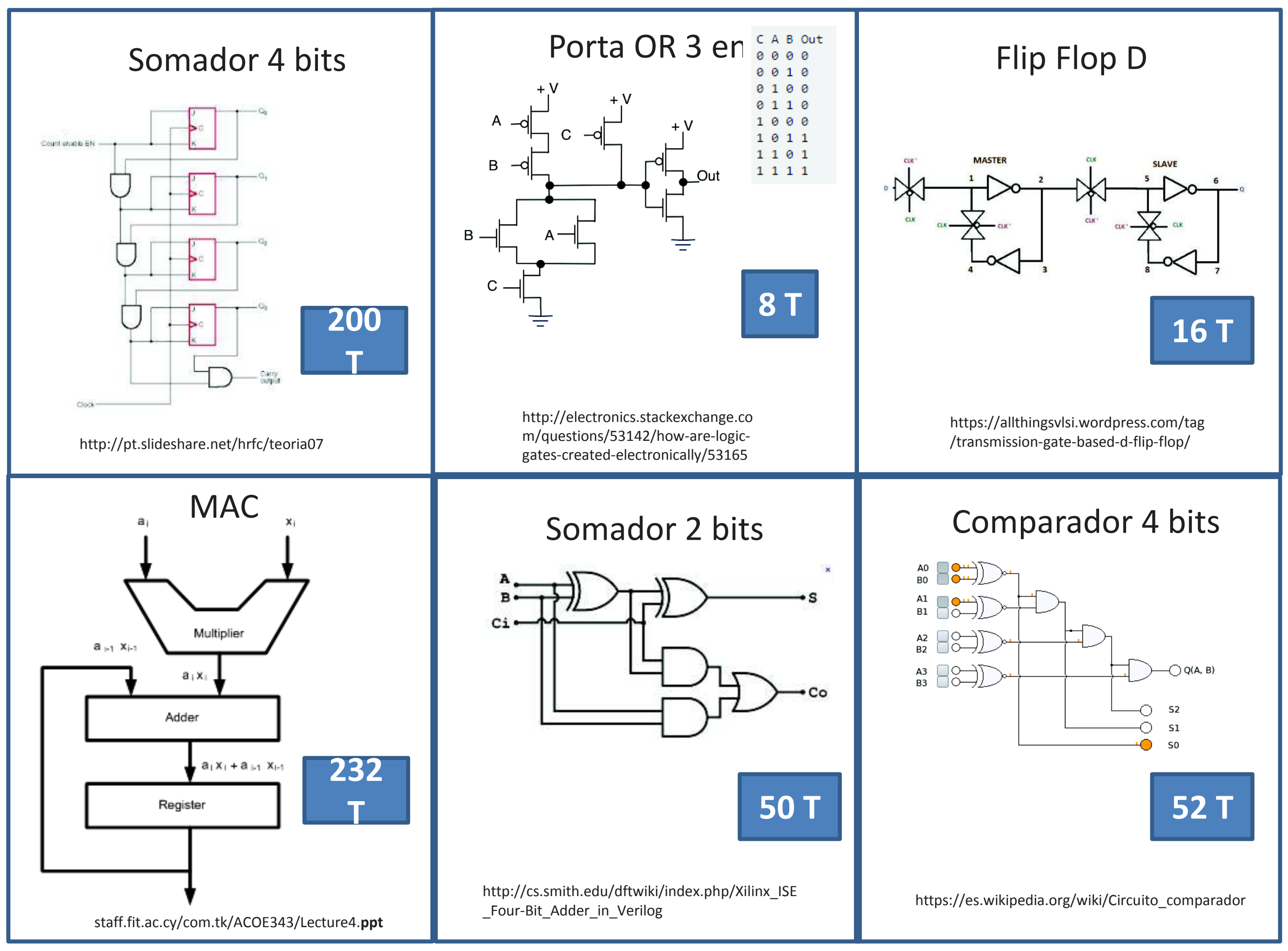


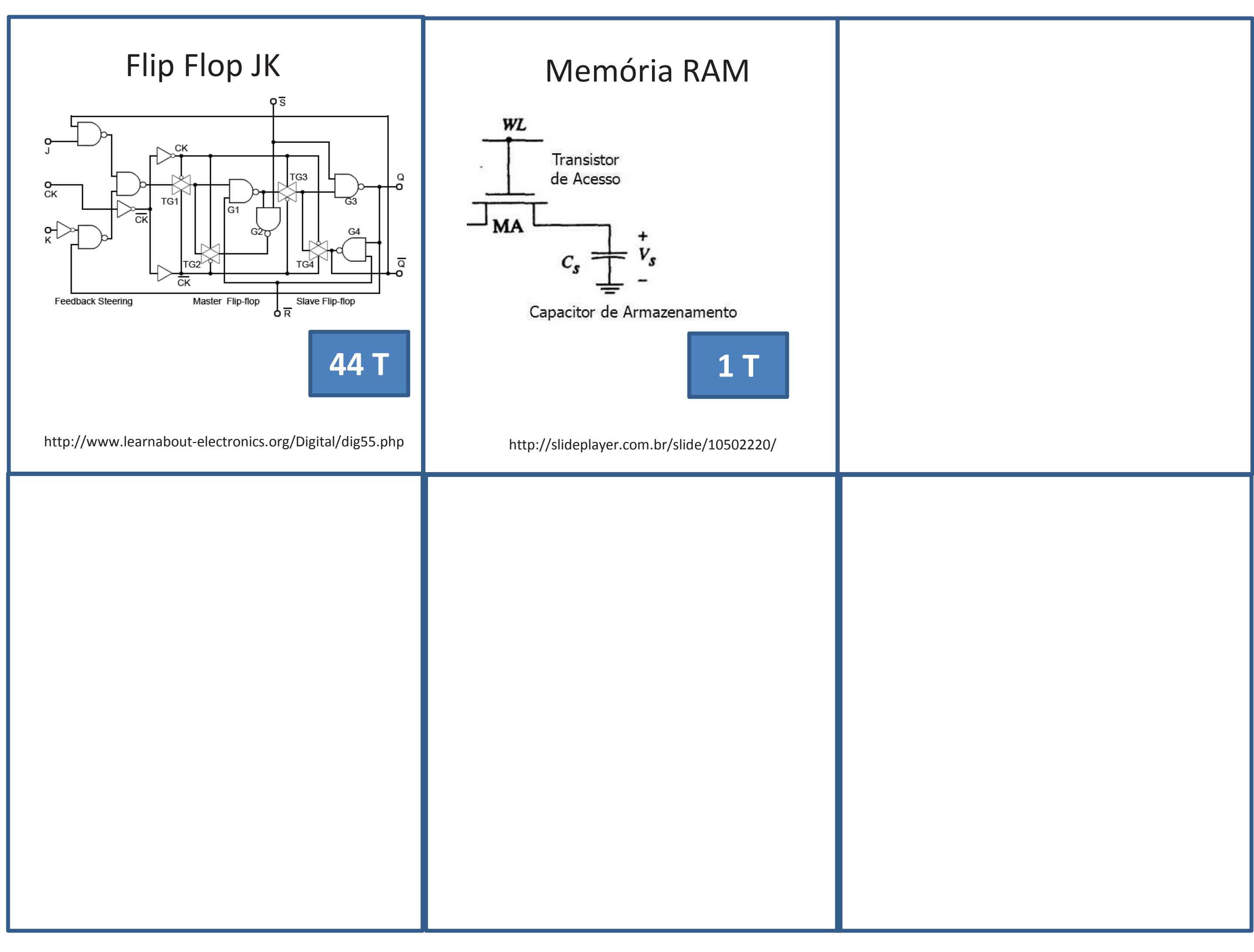
APÊNDICE B – QTDE DE TRANSISTORES EM CIRCUITOS PRIMITIVOS



Porta NAND
+ V
+ V
A
B
Ou
A
B
A B Out
0 0 1
0 1 1
1 0 1
1 1 0
4 T
http://electronics.stackexchange.com/questions/53142/how-are-logic-gates-created-electronically/53165
Porta NOR
+ V
D
C
Out
C
D
C D Out
0 0 1
0 1 0
1 0 0
1 1 0
4 T
http://electronics.stackexchange.com/questions/53142/how-are-logic-gates-created-electronically/53165
Porta Inversor
V
+ V
2 T
http://electronics.stackexchange.com/questions/53142/how-are-logic-gates-created-electronically/53165
Porta – AND
A B Out
0 0 0
0 1 0
1 0 0
1 1 1
+ V
+ V
A
B
A
B
6 T
http://electronics.stackexchange.com/questions/53142/how-are-logic-gates-created-electronically/53165
Porta OR
C D Out
0 0 0
0 1 1
1 0 1
1 1 1
+ V
D
C
+ V
Out
C
D
6 T
http://electronics.stackexchange.com/questions/53142/how-are-logic-gates-created-electronically/53165
Porta – NAND 3
A B C out
1 1 1 0
1 1 0 1
1 0 1 1
1 0 0 1
0 1 1 1
0 1 0 1
0 0 1 1
0 0 0 1
+ V
+ V
+ V
A
B
C
Out
A
B
C
6 T
http://electronics.stackexchange.com/questions/53142/how-are-logic-gates-created-electronically/53165


MUX 2X1
IN1
IN2
OUT
SEL
A
Select
Out
B
Select
6 T
http://ece-research.unm.edu/jimp/vlsi/slides/c1_basics.html
IN2
SEL
OUT
IN1
20 T
http://vlsiuniverse.blogspot.com.br/search/label/2%3A1%20mux
Porta XOR
A
B
Q
16 T
https://en.wikipedia.org/wiki/NAND_logic
Multiplicador 2 bits
A0
A1
B0
B1
C0
C1
C2
C3
68 T
Mux 4x1 bits e 8x1 bits
Porta de Transmissão
Porta Lógica
4x1
18 T
60 T
8x1
42 T
140 T
http://vlsiuniverse.blogspot.com.br/search/label/2%3A1%20mux
4 bit Shifter
Q1
Q2
Q3
Q4
Data In
Clock
64 T
https://en.wikipedia.org/wiki/Shift_register
Tristate Buffer
Tri-state buffer symbol
Input
Output
Enable
Tri-state buffer truth table
E
I
Output
1
0
0
1
1
1
0
0
floating
0
1
floating
Tri-state buffer circuit
+V
Enable
Output
Input
14 T
http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture1/lecture1-5-4.html


Somador 4 bits
200 T
http://pt.slideshare.net/hrfc/teoria07
Porta OR 3 en
C A B Out
0 0 0 0
0 0 1 0
0 1 0 0
0 1 1 0
1 0 0 0
1 0 1 1
1 1 0 1
1 1 1 1
+ V
A
B
C
Out
8 T
http://electronics.stackexchange.com/questions/53142/how-are-logic-gates-created-electronically/53165
Flip Flop D
MASTER
SLAVE
16 T
https://allthingsvlsi.wordpress.com/tag/transmission-gate-based-d-flip-flop/
MAC
Multiplier
Adder
Register
232 T
staff.fit.ac.cy/com.tk/ACOE343/Lecture4.ppt
Somador 2 bits
A
B
Ci
S
Co
50 T
http://cs.smith.edu/dftwiki/index.php/Xilinx_ISE_Four-Bit_Adder_in_Verilog
Comparador 4 bits
A0
B0
A1
B1
A2
B2
A3
B3
Q(A, B)
S2
S1
S0
52 T
https://es.wikipedia.org/wiki/Circuito_comparador


Flip Flop JK
J
CK
K
TG1
TG2
TG3
TG4
G1
G2
G3
G4
Q
Feedback Steering
Master Flip-flop
Slave Flip-flop
44 T
http://www.learnabout-electronics.org/Digital/dig55.php
Memória RAM
WL
Transistor
de Acesso
MA
Capacitor de Armazenamento
1 T
http://slideplayer.com.br/slide/10502220/

APÊNDICE C – RESULTADOS - SÍNTESE ISE XILINX

```
===================HammingEncDec.prj========================

Release 14.6 - xst P.68d (nt)
Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp


Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs

--> Parameter xsthdpdir set to xst


Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs

--> Reading design: HammingEncDec.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Parsing
  3) HDL Elaboration
  4) HDL Synthesis
       4.1) HDL Synthesis Report
  5) Advanced HDL Synthesis
       5.1) Advanced HDL Synthesis Report
  6) Low Level Synthesis
  7) Partition Report
  8) Design Summary
       8.1) Primitive and Black Box Usage
       8.2) Device utilization summary
       8.3) Partition Resource Summary
       8.4) Timing Report
            8.4.1) Clock Information
            8.4.2) Asynchronous Control Signals Information
            8.4.3) Timing Summary
            8.4.4) Timing Details
            8.4.5) Cross Clock Domains Report


=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "HammingEncDec.prj"
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "HammingEncDec"
Output Format                      : NGC
Target Device                      : xc6vlx75t-3-ff484

---- Source Options
Top Module Name                    : HammingEncDec
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : LUT
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Shift Register Extraction          : YES
ROM Style                          : Auto
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Shift Register Minimum Size        : 2
Use DSP Block                      : Auto
Automatic Register Balancing       : No

---- Target Options
```

```
LUT Combining                         : Auto
Reduce Control Sets                   : Auto
Add IO Buffers                        : YES
Global Maximum Fanout                 : 100000
Add Generic Clock Buffer(BUFG)        : 32
Register Duplication                  : YES
Optimize Instantiated Primitives      : NO
Use Clock Enable                      : Auto
Use Synchronous Set                   : Auto
Use Synchronous Reset                 : Auto
Pack IO Registers into IOBs           : Auto
Equivalent register Removal           : YES

---- General Options
Optimization Goal                     : Speed
Optimization Effort                   : 1
Power Reduction                       : No
Keep Hierarchy                        : No
Netlist Hierarchy                     : As_Optimized
RTL Output                            : Yes
Global Optimization                   : AllClockNets
Read Cores                            : YES
Write Timing Constraints              : NO
Cross Clock Analysis                  : NO
Hierarchy Separator                   : /
Bus Delimiter                         : <>
Case Specifier                        : Maintain
Slice Utilization Ratio               : 100
BRAM Utilization Ratio                : 100
DSP48 Utilization Ratio               : 100
Auto BRAM Packing                     : NO
Slice Utilization Ratio Delta         : 5

=========================================================================


=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file "\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\xor2in.vhd" into
library work
Parsing entity <xor2in>.
Parsing architecture <Behavioral> of entity <xor2in>.
Parsing VHDL file
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\Encoding_Hamming.vhd" into
library work
Parsing entity <Encoding_Hamming>.
Parsing architecture <Behavioral> of entity <encoding_hamming>.
Parsing VHDL file
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\decoding_HammingPage.vhd" into
library work
Parsing entity <Decoding_Hamming>.
Parsing architecture <Behavioral> of entity <decoding_hamming>.
Parsing VHDL file
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\HammingEncDec.vhd" into library
work
Parsing entity <HammingEncDec>.
Parsing architecture <Behavioral> of entity <hammingencdec>.

=========================================================================
*                          HDL Elaboration                              *
=========================================================================

Elaborating entity <HammingEncDec> (architecture <Behavioral>) from library <work>.

Elaborating entity <Encoding_Hamming> (architecture <Behavioral>) from library <work>.

Elaborating entity <xor2in> (architecture <Behavioral>) from library <work>.

Elaborating entity <Decoding_Hamming> (architecture <Behavioral>) from library <work>.
```

```
INFO:HDLCompiler:679 -
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\decoding_HammingPage.vhd" Line
1538. Case statement is complete. others clause is never selected

=========================================================================
*                          HDL Synthesis                                *
=========================================================================

Synthesizing Unit <HammingEncDec>.
    Related source file is
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\HammingEncDec.vhd".
    Summary:
        no macro.
Unit <HammingEncDec> synthesized.

Synthesizing Unit <Encoding_Hamming>.
    Related source file is
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\Encoding_Hamming.vhd".
    Found 4-bit register for signal <shifter>.
    Found 4110-bit register for signal <nova_tab>.
    Found 13-bit register for signal <count_i>.
    Found 13-bit adder for signal <count_i[12]_GND_4_o_add_2_OUT> created at line 231.
    Found 14-bit adder for signal <n4168> created at line 239.
    Found 14-bit adder for signal <n4189> created at line 257.
    Found 5-bit adder for signal <n4184> created at line 257.
    Found 4-bit adder for signal <shifter[3]_GND_4_o_add_17_OUT> created at line 258.
    Found 32-bit subtractor for signal <GND_4_o_GND_4_o_sub_15_OUT<31:0>> created at line 257.
    Found 32-bit shifter logical left for signal <GND_4_o_BUS_0005_shift_left_13_OUT> created at line 257
    Found 13-bit comparator greater for signal <count_i[12]_GND_4_o_LessThan_2_o> created at line 230
    Found 32-bit comparator equal for signal <GND_4_o_GND_4_o_equal_16_o> created at line 257
    Found 4-bit comparator greater for signal <shifter[3]_PWR_4_o_LessThan_17_o> created at line 257
    Summary:
        inferred   6 Adder/Subtractor(s).
        inferred 4127 D-type flip-flop(s).
        inferred   3 Comparator(s).
        inferred 4096 Multiplexer(s).
        inferred   1 Combinational logic shifter(s).
Unit <Encoding_Hamming> synthesized.

Synthesizing Unit <xor2in>.
    Related source file is
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\xor2in.vhd".
    Summary:
Unit <xor2in> synthesized.

Synthesizing Unit <Decoding_Hamming>.
    Related source file is
"\\vboxsrv\pasta_compartilhada\Hamming\Hamming_4096page_EncodingDecoding\decoding_HammingPage.vhd".
    Found 2-bit register for signal <state>.
    Found 13-bit register for signal <posicao_erro>.
    Found 4110-bit register for signal <ent_tmp>.
    Found finite state machine <FSM_0> for signal <state>.
    -----------------------------------------------------------------------
    | States             | 3                                              |
    | Transitions        | 4                                              |
    | Inputs             | 1                                              |
    | Outputs            | 2                                              |
    | Clock              | Clk (rising_edge)                              |
    | Reset              | reset (positive)                               |
    | Reset type         | synchronous                                    |
    | Reset State        | inicio                                         |
    | Power Up State     | inicio                                         |
    | Encoding           | auto                                           |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-bit 4110-to-1 multiplexer for signal <novo_vet[12]_X_8_o_Mux_2_o> created at line 1524.
    Summary:
        inferred 4123 D-type flip-flop(s).
        inferred 4111 Multiplexer(s).
        inferred   1 Finite State Machine(s).
```

```
Unit <Decoding_Hamming> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                   : 6
 13-bit adder                          : 1
 14-bit adder                          : 2
 32-bit subtractor                     : 1
 4-bit adder                           : 1
 5-bit adder                           : 1
# Registers                            : 5
 13-bit register                       : 2
 4-bit register                        : 1
 4110-bit register                     : 2
# Comparators                          : 3
 13-bit comparator greater             : 1
 32-bit comparator equal               : 1
 4-bit comparator greater              : 1
# Multiplexers                         : 8207
 1-bit 2-to-1 multiplexer              : 8206
 1-bit 4110-to-1 multiplexer           : 1
# Logic shifters                       : 1
 32-bit shifter logical left           : 1
# FSMs                                 : 1
# Xors                                 : 57354
 1-bit xor2                            : 57335
 1-bit xor3                            : 10
 1-bit xor4                            : 1
 1-bit xor5                            : 2
 1-bit xor6                            : 4
 1-bit xor7                            : 2

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================


Synthesizing (advanced) Unit <Encoding_Hamming>.
The following registers are absorbed into counter <shifter>: 1 register on signal <shifter>.
The following registers are absorbed into counter <count_i>: 1 register on signal <count_i>.
Unit <Encoding_Hamming> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                   : 4
 13-bit adder                          : 1
 14-bit adder                          : 1
 32-bit subtractor                     : 1
 5-bit adder                           : 1
# Counters                             : 2
 13-bit up counter                     : 1
 4-bit up counter                      : 1
# Registers                            : 8233
 Flip-Flops                            : 8233
# Comparators                          : 3
 13-bit comparator greater             : 1
 32-bit comparator equal               : 1
 4-bit comparator greater              : 1
# Multiplexers                         : 8207
 1-bit 2-to-1 multiplexer              : 8206
 1-bit 4110-to-1 multiplexer           : 1
# Logic shifters                       : 1
 32-bit shifter logical left           : 1
# FSMs                                 : 1
```

```
# Xors                                  : 57354
 1-bit xor2                             : 57335
 1-bit xor3                             : 10
 1-bit xor4                             : 1
 1-bit xor5                             : 2
 1-bit xor6                             : 4
 1-bit xor7                             : 2

=========================================================================

=========================================================================
*                      Low Level Synthesis                             *
=========================================================================

Analyzing FSM <MFsm> for best encoding.
Optimizing FSM <decHamm/FSM_0> on signal <state[1:2]> with gray encoding.
-----------------------
 State     | Encoding
-----------------------
 inicio    | 00
 corrige   | 11
 joga_fora | 01
-----------------------

Optimizing unit <HammingEncDec> ...

Optimizing unit <Encoding_Hamming> ...

Optimizing unit <Decoding_Hamming> ...
WARNING:Xst:1293 - FF/Latch <encHamm/count_i_12> has a constant value of 0 in block <HammingEncDec>. This
FF/Latch will be trimmed during the optimization process.

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block HammingEncDec, actual ratio is 81.

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers                             : 8251
 Flip-Flops                             : 8251

=========================================================================

=========================================================================
*                      Partition Report                                *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                      Design Summary                                  *
=========================================================================

Top Level Output File Name         : HammingEncDec.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 33929
#      GND                         : 1
#      INV                         : 2
#      LUT1                        : 30
```

```
#      LUT2                       : 7004
#      LUT3                       : 1952
#      LUT4                       : 2121
#      LUT5                       : 5489
#      LUT6                       : 17190
#      MUXCY                      : 71
#      MUXF7                      : 1
#      VCC                        : 1
#      XORCY                      : 67
# FlipFlops/Latches               : 8251
#      FDE                        : 4123
#      FDR                        : 4114
#      FDRE                       : 14
# Clock Buffers                   : 2
#      BUFG                       : 1
#      BUFGP                      : 1
# IO Buffers                      : 15
#      IBUF                       : 2
#      OBUF                       : 13

Device utilization summary:
---------------------------

Selected Device : 6vlx75tff484-3


Slice Logic Utilization:
 Number of Slice Registers:            8251  out of  93120     8%
 Number of Slice LUTs:                33788  out of  46560    72%
    Number used as Logic:             33788  out of  46560    72%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  33788
   Number with an unused Flip Flop:   25537  out of  33788    75%
   Number with an unused LUT:             0  out of  33788     0%
   Number of fully used LUT-FF pairs:  8251  out of  33788    24%
   Number of unique control sets:         6

IO Utilization:
 Number of IOs:                          16
 Number of bonded IOBs:                  16  out of    240     6%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                2  out of     32     6%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clock                              | BUFGP                  | 8251  |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
```

```
---------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 7.204ns (Maximum Frequency: 138.821MHz)
   Minimum input arrival time before clock: 1.359ns
   Maximum output required time after clock: 0.562ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clock'
  Clock period: 7.204ns (frequency: 138.821MHz)
  Total number of paths / destination ports: 23272174438 / 12402
-------------------------------------------------------------------------
Delay:               7.204ns (Levels of Logic = 13)
  Source:            decHamm/ent_tmp_2230 (FF)
  Destination:       decHamm/ent_tmp_4109 (FF)
  Source Clock:      clock rising
  Destination Clock: clock rising

  Data Path: decHamm/ent_tmp_2230 to decHamm/ent_tmp_4109
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q              7   0.280   0.622  decHamm/ent_tmp_2230 (decHamm/ent_tmp_2230)
     LUT6:I0->O            1   0.053   0.481  decHamm/Mxor_k0_tmp11<4>_xo<0>53
(decHamm/Mxor_k0_tmp11<4>_xo<0>52)
     LUT4:I0->O            1   0.053   0.296  decHamm/Mxor_k0_tmp11<4>_xo<0>57
(decHamm/Mxor_k0_tmp11<4>_xo<0>56)
     LUT5:I4->O            1   0.053   0.592  decHamm/Mxor_k0_tmp11<4>_xo<0>62
(decHamm/Mxor_k0_tmp11<4>_xo<0>61)
     LUT6:I1->O            1   0.053   0.594  decHamm/Mxor_k0_tmp11<4>_xo<0>83
(decHamm/Mxor_k0_tmp11<4>_xo<0>82)
     LUT6:I0->O         2075   0.053   0.523  decHamm/Mxor_k0_tmp11<4>_xo<0>445 (decHamm/k0_tmp11<4>)
     LUT6:I5->O            1   0.053   0.481  decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_29175
(decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_29175)
     LUT6:I2->O            1   0.053   0.481  decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_2450
(decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_2450)
     LUT6:I2->O            1   0.053   0.481  decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_1914
(decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_1914)
     LUT6:I2->O            1   0.053   0.481  decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_144
(decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_144)
     LUT6:I2->O            1   0.053   0.481  decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_92
(decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_92)
     LUT6:I2->O            1   0.053   0.000  decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_4
(decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_4)
     MUXF7:I0->O        4110   0.186   0.585  decHamm/Mmux_novo_vet[12]_X_8_o_Mux_2_o_2_f7
(decHamm/novo_vet[12]_X_8_o_Mux_2_o)
     LUT5:I4->O            1   0.053   0.000  decHamm/state[1]_entrada[4109]_mux_7_OUT<4100>1
(decHamm/state[1]_entrada[4109]_mux_7_OUT<9>)
     FDE:D                    -0.012          decHamm/ent_tmp_9
    ----------------------------------------
    Total                      7.204ns (1.102ns logic, 6.102ns route)
                                       (15.3% logic, 84.7% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clock'
  Total number of paths / destination ports: 12346 / 12346
-------------------------------------------------------------------------
Offset:              1.359ns (Levels of Logic = 2)
  Source:            reset (PAD)
  Destination:       decHamm/ent_tmp_4109 (FF)
  Destination Clock: clock rising
```

```
  Data Path: reset to decHamm/ent_tmp_4109
                          Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O         4117   0.003   0.586  reset_IBUF (reset_IBUF)
     LUT2:I1->O        4110   0.053   0.568  decHamm/_n4237_inv1 (decHamm/_n4237_inv)
     FDE:CE            0.149                 decHamm/ent_tmp_0
    ----------------------------------------
    Total                      1.359ns (0.205ns logic, 1.154ns route)
                                   (15.1% logic, 84.9% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clock'
  Total number of paths / destination ports: 13 / 13
-------------------------------------------------------------------------
Offset:              0.562ns (Levels of Logic = 1)
  Source:            decHamm/posicao_erro_12 (FF)
  Destination:       posicao_erro<12> (PAD)
  Source Clock:      clock rising

  Data Path: decHamm/posicao_erro_12 to posicao_erro<12>
                          Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q              1   0.280   0.279  decHamm/posicao_erro_12 (decHamm/posicao_erro_12)
     OBUF:I->O                 0.003          posicao_erro_12_OBUF (posicao_erro<12>)
    ----------------------------------------
    Total                      0.562ns (0.283ns logic, 0.279ns route)
                                   (50.3% logic, 49.7% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

Clock to Setup on destination clock clock
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clock          |    7.204|         |         |         |
---------------+---------+---------+---------+---------+

=========================================================================


Total REAL time to Xst completion: 936.00 secs
Total CPU time to Xst completion: 935.72 secs

-->

Total memory usage is 655600 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings : 15913 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

===================== bch_page_encoding.prj===================

Release 14.6 - xst P.68d (nt)
Copyright (c) 1995-2013 Xilinx, Inc. All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp

Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.14 secs

--> Parameter xsthdpdir set to xst

Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.14 secs

--> Reading design: bch_page_encoding.prj

TABLE OF CONTENTS

1) Synthesis Options Summary
2) HDL Parsing
3) HDL Elaboration
4) HDL Synthesis
    4.1) HDL Synthesis Report
5) Advanced HDL Synthesis
    5.1) Advanced HDL Synthesis Report
6) Low Level Synthesis
7) Partition Report
8) Design Summary
    8.1) Primitive and Black Box Usage
    8.2) Device utilization summary
    8.3) Partition Resource Summary
    8.4) Timing Report
        8.4.1) Clock Information
        8.4.2) Asynchronous Control Signals Information
        8.4.3) Timing Summary
        8.4.4) Timing Details
        8.4.5) Cross Clock Domains Report

===
* Synthesis Options Summary *
===
---- Source Parameters
Input File Name : "bch_page_encoding.prj"
Ignore Synthesis Constraint File : NO

---- Target Parameters
Output File Name : "bch_page_encoding"
Output Format : NGC
Target Device : xc7a100t-3-csg324

---- Source Options
Top Module Name : bch_page_encoding
Automatic FSM Extraction : YES
FSM Encoding Algorithm : Auto
Safe Implementation : No
FSM Style : LUT
RAM Extraction : Yes
RAM Style : Auto
ROM Extraction : Yes
Shift Register Extraction : YES
ROM Style : Auto
Resource Sharing : YES
Asynchronous To Synchronous : NO
Shift Register Minimum Size : 2
Use DSP Block : Auto
Automatic Register Balancing : No

---- Target Options

```
LUT Combining                          : Auto
Reduce Control Sets                    : Auto
Add IO Buffers                         : YES
Global Maximum Fanout                  : 100000
Add Generic Clock Buffer(BUFG)         : 32
Register Duplication                   : YES
Optimize Instantiated Primitives       : NO
Use Clock Enable                       : Yes
Use Synchronous Set                    : Yes
Use Synchronous Reset                  : Yes
Pack IO Registers into IOBs            : Auto
Equivalent register Removal            : YES

---- General Options
Optimization Goal                      : Speed
Optimization Effort                    : 1
Power Reduction                        : No
Keep Hierarchy                         : No
Netlist Hierarchy                      : As_Optimized
RTL Output                             : Yes
Global Optimization                    : AllClockNets
Read Cores                             : YES
Write Timing Constraints               : NO
Cross Clock Analysis                   : NO
Hierarchy Separator                    : /
Bus Delimiter                          : <>
Case Specifier                         : Maintain
Slice Utilization Ratio                : 100
BRAM Utilization Ratio                 : 100
DSP48 Utilization Ratio                : 100
Auto BRAM Packing                      : NO
Slice Utilization Ratio Delta          : 5

=========================================================================


=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file "\\vboxsrv\pasta_compartilhada\BCH_PAGE\bch_page_encoding\xor2in.vhd" into library work
Parsing entity <basicBLK>.
Parsing architecture <Behavioral> of entity <basicblk>.
Parsing VHDL file "\\vboxsrv\pasta_compartilhada\BCH_PAGE\bch_page_encoding\bch_page_encoding.vhd" into
library work
Parsing entity <bch_page_encoding>.
Parsing architecture <codigo> of entity <bch_page_encoding>.

=========================================================================
*                            HDL Elaboration                            *
=========================================================================

Elaborating entity <bch_page_encoding> (architecture <codigo>) with generics from library <work>.

Elaborating entity <basicBLK> (architecture <Behavioral>) from library <work>.

=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <bch_page_encoding>.
    Related source file is "\\vboxsrv\pasta_compartilhada\BCH_PAGE\bch_page_encoding\bch_page_encoding.vhd".
        n = 52
WARNING:Xst:647 - Input <SI> is never used. This port will be preserved and left unconnected if it belongs to a top-
level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.
WARNING:Xst:2999 - Signal 'data', unconnected in block 'bch_page_encoding', is tied to its initial value.
    Found 4096x1-bit single-port Read Only RAM <Mram_data> for signal <data>.
    Found 1-bit register for signal <fio<0>>.
    Found 13-bit register for signal <cnt>.
    Found 13-bit adder for signal <cnt[12]_GND_3_o_add_1_OUT> created at line 72.
    Found 13-bit comparator lessequal for signal <cnt[12]_PWR_3_o_LessThan_1_o> created at line 71
```

```
    Summary:
        inferred   1 RAM(s).
        inferred   1 Adder/Subtractor(s).
        inferred  14 D-type flip-flop(s).
        inferred   1 Comparator(s).
Unit <bch_page_encoding> synthesized.

Synthesizing Unit <basicBLK>.
    Related source file is "\\vboxsrv\pasta_compartilhada\BCH_PAGE\bch_page_encoding\xor2in.vhd".
    Found 1-bit register for signal <c>.
    Summary:
        inferred   1 D-type flip-flop(s).
Unit <basicBLK> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs                                      : 1
 4096x1-bit single-port Read Only RAM       : 1
# Adders/Subtractors                        : 1
 13-bit adder                               : 1
# Registers                                 : 54
 1-bit register                             : 53
 13-bit register                            : 1
# Comparators                               : 1
 13-bit comparator lessequal                : 1
# Xors                                      : 52
 1-bit xor2                                 : 52

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================


Synthesizing (advanced) Unit <bch_page_encoding>.
The following registers are absorbed into counter <cnt>: 1 register on signal <cnt>.
INFO:Xst:3226 - The RAM <Mram_data> will be implemented as a BLOCK RAM, absorbing the following register(s): <fio_0>
    -----------------------------------------------------------------------
    | ram_type           | Block                               |          |
    -----------------------------------------------------------------------
    | Port A                                                   |
    |     aspect ratio   | 4096-word x 1-bit                   |          |
    |     mode           | write-first                         |          |
    |     clkA           | connected to signal <CLK>           | rise     |
    |     enA            | connected to signal <ce>            | high     |
    |     weA            | connected to signal <GND>           | high     |
    |     addrA          | connected to signal <cnt<11:0>>     |          |
    |     diA            | connected to signal <GND>           |          |
    |     doA            | connected to signal <fio>           |          |
    |     dorstA         | connected to signal <reset_0>       | high     |
    | reset value        | 0                                   |
    -----------------------------------------------------------------------
    | optimization       | speed                               |          |
    -----------------------------------------------------------------------
Unit <bch_page_encoding> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                                  : 1
 4096x1-bit single-port block Read Only RAM             : 1
# Counters                                              : 1
 13-bit up counter                                      : 1
# Registers                                             : 52
 Flip-Flops                                             : 52
```

```
# Comparators                                          : 1
 13-bit comparator lessequal                           : 1
# Xors                                                 : 52
 1-bit xor2                                            : 52

===========================================================================

===========================================================================
*                       Low Level Synthesis                           *
===========================================================================

Optimizing unit <bch_page_encoding> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block bch_page_encoding, actual ratio is 0.

Final Macro Processing ...

Processing Unit <bch_page_encoding> :
        Found 5-bit shift register for signal <forgen1[35].ifgen2.celln/c>.
        Found 4-bit shift register for signal <forgen1[29].ifgen2.celln/c>.
        Found 4-bit shift register for signal <forgen1[14].ifgen2.celln/c>.
Unit <bch_page_encoding> processed.

===========================================================================
Final Register Report

Macro Statistics
# Registers                                 : 52
 Flip-Flops                                 : 52
# Shift Registers                           : 3
 4-bit shift register                       : 2
 5-bit shift register                       : 1

===========================================================================

===========================================================================
*                        Partition Report                             *
===========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

===========================================================================
*                         Design Summary                              *
===========================================================================

Top Level Output File Name         : bch_page_encoding.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 68
#      GND                         : 1
#      INV                         : 1
#      LUT1                        : 12
#      LUT2                        : 27
#      LUT3                        : 1
#      MUXCY                       : 12
#      VCC                         : 1
#      XORCY                       : 13
# FlipFlops/Latches                : 62
#      FDE                         : 3
#      FDRE                        : 59
# RAMS                             : 1
#      RAMB18E1                    : 1
```

```
# Shift Registers                  : 3
#      SRLC16E                     : 3
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 3
#      IBUF                        : 2
#      OBUF                        : 1

Device utilization summary:
---------------------------

Selected Device : 7a100tcsg324-3


Slice Logic Utilization:
 Number of Slice Registers:              62  out of  126800     0%
 Number of Slice LUTs:                   44  out of   63400     0%
    Number used as Logic:                41  out of   63400     0%
    Number used as Memory:                3  out of   19000     0%
       Number used as SRL:                3

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     64
   Number with an unused Flip Flop:       2  out of     64     3%
   Number with an unused LUT:            20  out of     64    31%
   Number of fully used LUT-FF pairs:    42  out of     64    65%
   Number of unique control sets:         3

IO Utilization:
 Number of IOs:                           5
 Number of bonded IOBs:                   4  out of    210     1%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                1  out of    135     0%
    Number using Block RAM only:          1
 Number of BUFG/BUFGCTRLs:                1  out of     32     3%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
CLK                                | BUFGP                  | 66    |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
-----------------------------------+------------------------+-------+
Control Signal                     | Buffer(FF name)        | Load  |
-----------------------------------+------------------------+-------+
N1(XST_GND:G)                      | NONE(Mram_data)        | 2     |
-----------------------------------+------------------------+-------+

Timing Summary:
```

```
---------------
Speed Grade: -3

   Minimum period: 2.246ns (Maximum Frequency: 445.177MHz)
   Minimum input arrival time before clock: 1.313ns
   Maximum output required time after clock: 0.738ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'CLK'
  Clock period: 2.246ns (frequency: 445.177MHz)
  Total number of paths / destination ports: 193 / 90
-------------------------------------------------------------------------
Delay:               2.246ns (Levels of Logic = 1)
  Source:            Mram_data (RAM)
  Destination:       forgen1[0].ifgen1.celln/c (FF)
  Source Clock:      CLK rising
  Destination Clock: CLK rising

  Data Path: Mram_data to forgen1[0].ifgen1.celln/c
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     RAMB18E1:CLKARDCLK->DOADO0    1   1.846   0.295  Mram_data (fio<0>)
     LUT2:I1->O            1   0.097   0.000  forgen1[0].ifgen1.celln/Mxor_a_b_XOR_1_o_xo<0>1
(forgen1[0].ifgen1.celln/a_b_XOR_1_o)
     FDRE:D                    0.008          forgen1[0].ifgen1.celln/c
    ----------------------------------------
    Total                      2.246ns (1.951ns logic, 0.295ns route)
                                       (86.9% logic, 13.1% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'CLK'
  Total number of paths / destination ports: 140 / 126
-------------------------------------------------------------------------
Offset:              1.313ns (Levels of Logic = 2)
  Source:            ce (PAD)
  Destination:       Mram_data (RAM)
  Destination Clock: CLK rising

  Data Path: ce to Mram_data
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            56   0.001   0.622  ce_IBUF (ce_IBUF)
     LUT3:I0->O            1   0.097   0.279  reset1 (reset_0)
     RAMB18E1:RSTRAMARSTRAM        0.314          Mram_data
    ----------------------------------------
    Total                      1.313ns (0.412ns logic, 0.901ns route)
                                       (31.4% logic, 68.6% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'CLK'
  Total number of paths / destination ports: 1 / 1
-------------------------------------------------------------------------
Offset:                   0.738ns (Levels of Logic = 1)
  Source:                 forgen1[51].ifgen2.celln/c (FF)
  Destination:            SO (PAD)
  Source Clock:           CLK rising

  Data Path: forgen1[51].ifgen2.celln/c to SO
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q            23   0.361   0.377  forgen1[51].ifgen2.celln/c (forgen1[51].ifgen2.celln/c)
     OBUF:I->O                 0.000          SO_OBUF (SO)
```

```
    ----------------------------------------
    Total                      0.738ns (0.361ns logic, 0.377ns route)
                                     (48.9% logic, 51.1% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

Clock to Setup on destination clock CLK
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
CLK            |    2.246|         |         |         |
---------------+---------+---------+---------+---------+

=========================================================================


Total REAL time to Xst completion: 8.00 secs
Total CPU time to Xst completion: 8.01 secs

-->

Total memory usage is 318640 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    2 (   0 filtered)
Number of infos    :    1 (   0 filtered)
```

================== bch_page_decodingSindrome.prj================

```
Release 14.6 - xst P.68d (nt)
Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp


Total REAL time to Xst completion: 2.00 secs
Total CPU time to Xst completion: 1.37 secs

--> Parameter xsthdpdir set to xst


Total REAL time to Xst completion: 3.00 secs
Total CPU time to Xst completion: 2.15 secs

--> Reading design: bch_page_decodingSindrome.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Parsing
  3) HDL Elaboration
  4) HDL Synthesis
       4.1) HDL Synthesis Report
  5) Advanced HDL Synthesis
       5.1) Advanced HDL Synthesis Report
  6) Low Level Synthesis
  7) Partition Report
  8) Design Summary
       8.1) Primitive and Black Box Usage
       8.2) Device utilization summary
       8.3) Partition Resource Summary
       8.4) Timing Report
            8.4.1) Clock Information
            8.4.2) Asynchronous Control Signals Information
            8.4.3) Timing Summary
            8.4.4) Timing Details
            8.4.5) Cross Clock Domains Report


=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "bch_page_decodingSindrome.prj"
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "bch_page_decodingSindrome"
Output Format                      : NGC
Target Device                      : xc7a100t-3-csg324

---- Source Options
Top Module Name                    : bch_page_decodingSindrome
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : LUT
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Shift Register Extraction          : YES
ROM Style                          : Auto
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Shift Register Minimum Size        : 2
Use DSP Block                      : Auto
Automatic Register Balancing       : No
```

```
---- Target Options
LUT Combining                      : Auto
Reduce Control Sets                : Auto
Add IO Buffers                     : YES
Global Maximum Fanout              : 100000
Add Generic Clock Buffer(BUFG)     : 16
Register Duplication               : YES
Optimize Instantiated Primitives   : NO
Use Clock Enable                   : Auto
Use Synchronous Set                : Auto
Use Synchronous Reset              : Auto
Pack IO Registers into IOBs        : Auto
Equivalent register Removal        : YES

---- General Options
Optimization Goal                  : Speed
Optimization Effort                : 1
Power Reduction                    : No
Keep Hierarchy                     : No
Netlist Hierarchy                  : As_Optimized
RTL Output                         : Yes
Global Optimization                : AllClockNets
Read Cores                         : YES
Write Timing Constraints           : NO
Cross Clock Analysis               : NO
Hierarchy Separator                : /
Bus Delimiter                      : <>
Case Specifier                     : Maintain
Slice Utilization Ratio            : 100
BRAM Utilization Ratio             : 100
DSP48 Utilization Ratio            : 100
Auto BRAM Packing                  : No
Slice Utilization Ratio Delta      : 5

=========================================================================


=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file "\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Decoding_Sindromes\xor2in.vhd" into library work
Parsing entity <basicBLK>.
Parsing architecture <Behavioral> of entity <basicblk>.
Parsing VHDL file "\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Decoding_Sindromes\bch_page_encoding.vhd"
into library work
Parsing entity <bch_page_decodingSindrome>.
Parsing architecture <codigo> of entity <bch_page_decodingsindrome>.

=========================================================================
*                            HDL Elaboration                            *
=========================================================================

Elaborating entity <bch_page_decodingSindrome> (architecture <codigo>) with generics from library <work>.

Elaborating entity <basicBLK> (architecture <Behavioral>) from library <work>.
WARNING:HDLCompiler:1127 -
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Decoding_Sindromes\bch_page_encoding.vhd" Line 90: Assignment
to code ignored, since the identifier is never used

=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <bch_page_decodingSindrome>.
    Related source file is
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Decoding_Sindromes\bch_page_encoding.vhd".
        n = 52
WARNING:Xst:647 - Input <SI> is never used. This port will be preserved and left unconnected if it belongs to a top-
level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.
WARNING:Xst:2999 - Signal 'data', unconnected in block 'bch_page_decodingSindrome', is tied to its initial value.
```

```
WARNING:Xst:3035 - Index value(s) does not match array range for signal <data>, simulation mismatch.
    Found 4148x1-bit single-port Read Only RAM <Mram_data> for signal <data>.
    Found 1-bit register for signal <fio<0>>.
    Found 13-bit register for signal <cnt>.
    Found 13-bit adder for signal <cnt[12]_GND_3_o_add_1_OUT> created at line 76.
    Found 13-bit comparator lessequal for signal <cnt[12]_PWR_3_o_LessThan_1_o> created at line 75
    Summary:
        inferred   1 RAM(s).
        inferred   1 Adder/Subtractor(s).
        inferred  14 D-type flip-flop(s).
        inferred   1 Comparator(s).
Unit <bch_page_decodingSindrome> synthesized.

Synthesizing Unit <basicBLK>.
    Related source file is "\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Decoding_Sindromes\xor2in.vhd".
    Found 1-bit register for signal <c>.
    Summary:
        inferred   1 D-type flip-flop(s).
Unit <basicBLK> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs                                                 : 1
 4148x1-bit single-port Read Only RAM                  : 1
# Adders/Subtractors                                   : 1
 13-bit adder                                          : 1
# Registers                                            : 15
 1-bit register                                        : 14
 13-bit register                                       : 1
# Comparators                                          : 1
 13-bit comparator lessequal                           : 1
# Xors                                                 : 13
 1-bit xor2                                            : 13

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================


Synthesizing (advanced) Unit <bch_page_decodingSindrome>.
The following registers are absorbed into counter <cnt>: 1 register on signal <cnt>.
INFO:Xst:3226 - The RAM <Mram_data> will be implemented as a BLOCK RAM, absorbing the following register(s): <fio_0>
    -----------------------------------------------------------------------
    | ram_type           | Block                               |          |
    -----------------------------------------------------------------------
    | Port A                                                   |
    |     aspect ratio   | 4148-word x 1-bit                   |          |
    |     mode           | write-first                         |          |
    |     clkA           | connected to signal <CLK>           | rise     |
    |     enA            | connected to internal node          | low      |
    |     weA            | connected to signal <GND>           | high     |
    |     addrA          | connected to signal <cnt>           |          |
    |     diA            | connected to signal <GND>           |          |
    |     doA            | connected to signal <fio>           |          |
    |     dorstA         | connected to internal node          | high     |
    | reset value        | 0                                   |
    -----------------------------------------------------------------------
    | optimization       | speed                               |          |
    -----------------------------------------------------------------------
Unit <bch_page_decodingSindrome> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
```

```
# RAMs                                              : 1
 4148x1-bit single-port block Read Only RAM         : 1
# Counters                                          : 1
 13-bit up counter                                  : 1
# Registers                                         : 13
 Flip-Flops                                         : 13
# Comparators                                       : 1
 13-bit comparator lessequal                        : 1
# Xors                                              : 13
 1-bit xor2                                         : 13

=========================================================================

=========================================================================
*                         Low Level Synthesis                           *
=========================================================================

Optimizing unit <bch_page_decodingSindrome> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block bch_page_decodingSindrome, actual ratio is 0.

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers                                 : 26
 Flip-Flops                                 : 26

=========================================================================

=========================================================================
*                           Partition Report                            *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                            Design Summary                             *
=========================================================================

Top Level Output File Name         : bch_page_decodingSindrome.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 54
#      GND                         : 1
#      INV                         : 2
#      LUT1                        : 12
#      LUT2                        : 9
#      LUT4                        : 1
#      LUT5                        : 1
#      LUT6                        : 2
#      MUXCY                       : 12
#      VCC                         : 1
#      XORCY                       : 13
# FlipFlops/Latches                : 26
#      FDRE                        : 26
# RAMS                             : 1
#      RAMB18E1                    : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
```

```
# IO Buffers                     : 3
#      IBUF                      : 2
#      OBUF                      : 1

Device utilization summary:
---------------------------

Selected Device : 7a100tcsg324-3


Slice Logic Utilization:
 Number of Slice Registers:              26  out of  126800     0%
 Number of Slice LUTs:                   27  out of   63400     0%
    Number used as Logic:                27  out of   63400     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     32
   Number with an unused Flip Flop:       6  out of     32    18%
   Number with an unused LUT:             5  out of     32    15%
   Number of fully used LUT-FF pairs:    21  out of     32    65%
   Number of unique control sets:         2

IO Utilization:
 Number of IOs:                           5
 Number of bonded IOBs:                   4  out of    210     1%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                1  out of    135     0%
    Number using Block RAM only:          1
 Number of BUFG/BUFGCTRLs:                1  out of     32     3%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
CLK                                | BUFGP                  | 27    |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
-----------------------------------+------------------------+-------+
Control Signal                     | Buffer(FF name)        | Load  |
-----------------------------------+------------------------+-------+
N1(XST_GND:G)                      | NONE(Mram_data)        | 2     |
-----------------------------------+------------------------+-------+

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 2.661ns (Maximum Frequency: 375.770MHz)
   Minimum input arrival time before clock: 1.869ns
   Maximum output required time after clock: 0.677ns
```

```
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'CLK'
  Clock period: 2.661ns (frequency: 375.770MHz)
  Total number of paths / destination ports: 307 / 53
-------------------------------------------------------------------------
Delay:               2.661ns (Levels of Logic = 3)
  Source:            cnt_4 (FF)
  Destination:       Mram_data (RAM)
  Source Clock:      CLK rising
  Destination Clock: CLK rising

  Data Path: cnt_4 to Mram_data
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             3   0.361   0.703  cnt_4 (cnt_4)
     LUT6:I0->O            2   0.097   0.383  _n00282 (_n00282)
     LUT5:I3->O            1   0.097   0.279  _n002811 (_n0028_0)
     INV:I->O              1   0.113   0.279  _n0028_0_inv_INV_0 (_n0028_0_inv)
     RAMB18E1:ENARDEN          0.348          Mram_data
    ----------------------------------------
    Total                      2.661ns (1.016ns logic, 1.645ns route)
                                       (38.2% logic, 61.8% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'CLK'
  Total number of paths / destination ports: 69 / 54
-------------------------------------------------------------------------
Offset:              1.869ns (Levels of Logic = 3)
  Source:            ce (PAD)
  Destination:       Mram_data (RAM)
  Destination Clock: CLK rising

  Data Path: ce to Mram_data
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            16   0.001   0.752  ce_IBUF (ce_IBUF)
     LUT5:I0->O            1   0.097   0.279  _n002811 (_n0028_0)
     INV:I->O              1   0.113   0.279  _n0028_0_inv_INV_0 (_n0028_0_inv)
     RAMB18E1:ENARDEN          0.348          Mram_data
    ----------------------------------------
    Total                      1.869ns (0.559ns logic, 1.311ns route)
                                       (29.9% logic, 70.1% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'CLK'
  Total number of paths / destination ports: 1 / 1
-------------------------------------------------------------------------
Offset:              0.677ns (Levels of Logic = 1)
  Source:            forgen1[12].ifgen2.celln/c (FF)
  Destination:       SO (PAD)
  Source Clock:      CLK rising

  Data Path: forgen1[12].ifgen2.celln/c to SO
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             9   0.361   0.316  forgen1[12].ifgen2.celln/c (forgen1[12].ifgen2.celln/c)
     OBUF:I->O                 0.000          SO_OBUF (SO)
    ----------------------------------------
    Total                      0.677ns (0.361ns logic, 0.316ns route)
                                       (53.3% logic, 46.7% route)
```

```
=========================================================================

Cross Clock Domains Report:
--------------------------

Clock to Setup on destination clock CLK
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
CLK            |    2.661|         |         |         |
---------------+---------+---------+---------+---------+

=========================================================================


Total REAL time to Xst completion: 14.00 secs
Total CPU time to Xst completion: 13.91 secs

-->

Total memory usage is 239600 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    4 (   0 filtered)
Number of infos    :    1 (   0 filtered)
```

==============BCH_PageDecodingSIGMA.prj======================

```
Release 14.6 - xst P.68d (nt)
Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp


Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.14 secs

--> Parameter xsthdpdir set to xst


Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.15 secs

--> Reading design: BCH_PageDecodingSIGMA.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Parsing
  3) HDL Elaboration
  4) HDL Synthesis
       4.1) HDL Synthesis Report
  5) Advanced HDL Synthesis
       5.1) Advanced HDL Synthesis Report
  6) Low Level Synthesis
  7) Partition Report
  8) Design Summary
       8.1) Primitive and Black Box Usage
       8.2) Device utilization summary
       8.3) Partition Resource Summary
       8.4) Timing Report
            8.4.1) Clock Information
            8.4.2) Asynchronous Control Signals Information
            8.4.3) Timing Summary
            8.4.4) Timing Details
            8.4.5) Cross Clock Domains Report


=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "BCH_PageDecodingSIGMA.prj"
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "BCH_PageDecodingSIGMA"
Output Format                      : NGC
Target Device                      : xc7a100t-3-csg324

---- Source Options
Top Module Name                    : BCH_PageDecodingSIGMA
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : LUT
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Shift Register Extraction          : YES
ROM Style                          : Auto
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Shift Register Minimum Size        : 2
Use DSP Block                      : Auto
Automatic Register Balancing       : No
```

```
---- Target Options
LUT Combining                          : Auto
Reduce Control Sets                    : Auto
Add IO Buffers                         : YES
Global Maximum Fanout                  : 100000
Add Generic Clock Buffer(BUFG)   32
Register Duplication                   : YES
Optimize Instantiated Primitives       : NO
Use Clock Enable                       : Auto
Use Synchronous Set                    : Auto
Use Synchronous Reset                  : Auto
Pack IO Registers into IOBs            : Auto
Equivalent register Removal            : YES

---- General Options
Optimization Goal                      : Speed
Optimization Effort                    : 1
Power Reduction                        : No
Keep Hierarchy                         : No
Netlist Hierarchy                      : As_Optimized
RTL Output                             : Yes
Global Optimization                    : AllClockNets
Read Cores                             : YES
Write Timing Constraints               : No
Cross Clock Analysis                   : No
Hierarchy Separator                    : /
Bus Delimiter                          : <>
Case Specifier                         : Maintain
Slice Utilization Ratio                : 100
BRAM Utilization Ratio                 : 100
DSP48 Utilization Ratio                : 100
Auto BRAM Packing                      : No
Slice Utilization Ratio Delta          : 5

=========================================================================


=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\copiaBCH_Page_decoding_SIGMA\BCH_PageDecodingSIGMA_copia.vh
d" into library work
Parsing entity <BCH_PageDecodingSIGMA>.
Parsing architecture <Behavioral> of entity <bch_pagedecodingsigma>.

=========================================================================
*                            HDL Elaboration                            *
=========================================================================

Elaborating entity <BCH_PageDecodingSIGMA> (architecture <Behavioral>) from library <work>.

=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <BCH_PageDecodingSIGMA>.
    Related source file is
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\copiaBCH_Page_decoding_SIGMA\BCH_PageDecodingSIGMA_copia.vh
d".
    Found 13-bit register for signal <auxA>.
    Found 13-bit register for signal <auxB>.
    Found 13-bit register for signal <auxC>.
    Found 13-bit register for signal <auxD>.
    Found 13-bit register for signal <Sigma1>.
    Found 13-bit register for signal <E1>.
    Found 13-bit register for signal <E2>.
    Found 13-bit register for signal <auxE2>.
    Found 13-bit register for signal <estado[5]_dff_172_OUT>.
    Found 13-bit register for signal <F1>.
```

Found 13-bit register for signal <auxF1>.
Found 13-bit register for signal <F2>.
Found 13-bit register for signal <G1>.
Found 13-bit register for signal <auxG1>.
Found 13-bit register for signal <G2>.
Found 13-bit register for signal <G3>.
Found 13-bit register for signal <auxG2>.
Found 13-bit register for signal <estado[5]_dff_181_OUT>.
Found 13-bit register for signal <H1>.
Found 13-bit register for signal <H2>.
Found 13-bit register for signal <auxI1>.
Found 13-bit register for signal <I2>.
Found 13-bit register for signal <J1>.
Found 13-bit register for signal <auxJ1>.
Found 13-bit register for signal <J2>.
Found 13-bit register for signal <auxJ2>.
Found 13-bit register for signal <estado[5]_dff_190_OUT>.
Found 13-bit register for signal <auxK1>.
Found 13-bit register for signal <K2>.
Found 13-bit register for signal <L1>.
Found 13-bit register for signal <auxL1>.
Found 13-bit register for signal <estado[5]_dff_195_OUT>.
Found 13-bit register for signal <auxM1>.
Found 13-bit register for signal <M2>.
Found 13-bit register for signal <auxN1>.
Found 13-bit register for signal <N2>.
Found 13-bit register for signal <Sigma2>.
Found 13-bit register for signal <O1>.
Found 13-bit register for signal <P1>.
Found 13-bit register for signal <auxP1>.
Found 13-bit register for signal <P2>.
Found 13-bit register for signal <J3>.
Found 13-bit register for signal <Q1>.
Found 13-bit register for signal <auxQ1>.
Found 13-bit register for signal <estado[5]_dff_208_OUT>.
Found 13-bit register for signal <Sigma3>.
Found 13-bit register for signal <auxR1>.
Found 13-bit register for signal <auxR4>.
Found 13-bit register for signal <auxR2>.
Found 13-bit register for signal <auxR3>.
Found 13-bit register for signal <R2>.
Found 13-bit register for signal <auxU1>.
Found 13-bit register for signal <T1>.
Found 13-bit register for signal <U2>.
Found 13-bit register for signal <auxV1>.
Found 13-bit register for signal <V2>.
Found 13-bit register for signal <X1>.
Found 13-bit register for signal <Sigma4>.
Found 1-bit register for signal <estado[5]_clock_DFF_20>.
Found 1-bit register for signal <estado[5]_clock_DFF_21>.
Found 1-bit register for signal <estado[5]_clock_DFF_22>.
Found 1-bit register for signal <estado[5]_clock_DFF_23>.
Found 1-bit register for signal <estado[5]_clock_DFF_24>.
Found 1-bit register for signal <estado[5]_clock_DFF_25>.
Found 1-bit register for signal <estado[5]_clock_DFF_26>.
Found 1-bit register for signal <estado[5]_clock_DFF_27>.
Found 1-bit register for signal <estado[5]_clock_DFF_28>.
Found 1-bit register for signal <estado[5]_clock_DFF_29>.
Found 1-bit register for signal <estado[5]_clock_DFF_30>.
Found 1-bit register for signal <estado[5]_clock_DFF_31>.
Found 1-bit register for signal <estado[5]_clock_DFF_32>.
Found 1-bit register for signal <estado[5]_clock_DFF_33>.
Found 1-bit register for signal <estado[5]_clock_DFF_34>.
Found 1-bit register for signal <estado[5]_clock_DFF_35>.
Found 1-bit register for signal <estado[5]_clock_DFF_36>.
Found 1-bit register for signal <estado[5]_clock_DFF_37>.
Found 1-bit register for signal <estado[5]_clock_DFF_38>.
Found 1-bit register for signal <estado[5]_clock_DFF_39>.
Found 1-bit register for signal <estado[5]_clock_DFF_40>.
Found 1-bit register for signal <estado[5]_clock_DFF_41>.

Found 1-bit register for signal <estado[5]_clock_DFF_42>.
Found 1-bit register for signal <estado[5]_clock_DFF_43>.
Found 1-bit register for signal <estado[5]_clock_DFF_44>.
Found 1-bit register for signal <estado[5]_clock_DFF_45>.
Found 1-bit register for signal <estado[5]_clock_DFF_46>.
Found 1-bit register for signal <estado[5]_clock_DFF_47>.
Found 1-bit register for signal <estado[5]_clock_DFF_48>.
Found 1-bit register for signal <estado[5]_clock_DFF_49>.
Found 1-bit register for signal <estado[5]_clock_DFF_50>.
Found 1-bit register for signal <estado[5]_clock_DFF_51>.
Found 1-bit register for signal <estado[5]_clock_DFF_52>.
Found 1-bit register for signal <estado[5]_clock_DFF_53>.
Found 1-bit register for signal <estado[5]_clock_DFF_54>.
Found 1-bit register for signal <estado[5]_clock_DFF_55>.
Found 1-bit register for signal <estado[5]_clock_DFF_56>.
Found 1-bit register for signal <estado[5]_clock_DFF_57>.
Found 1-bit register for signal <estado[5]_clock_DFF_58>.
Found 1-bit register for signal <estado[5]_clock_DFF_59>.
Found 1-bit register for signal <estado[5]_clock_DFF_60>.
Found 1-bit register for signal <estado[5]_clock_DFF_61>.
Found 1-bit register for signal <estado[5]_clock_DFF_62>.
Found 1-bit register for signal <estado[5]_clock_DFF_63>.
Found 1-bit register for signal <estado[5]_clock_DFF_64>.
Found 1-bit register for signal <estado[5]_clock_DFF_65>.
Found 1-bit register for signal <estado[5]_clock_DFF_66>.
Found 1-bit register for signal <estado[5]_clock_DFF_67>.
Found 1-bit register for signal <estado[5]_clock_DFF_68>.
Found 1-bit register for signal <estado[5]_clock_DFF_69>.
Found 1-bit register for signal <estado[5]_clock_DFF_70>.
Found 1-bit register for signal <estado[5]_clock_DFF_71>.
Found 1-bit register for signal <estado[5]_clock_DFF_72>.
Found 1-bit register for signal <estado[5]_clock_DFF_73>.
Found 1-bit register for signal <estado[5]_clock_DFF_74>.
Found 1-bit register for signal <estado[5]_clock_DFF_75>.
Found 1-bit register for signal <estado[5]_clock_DFF_76>.
Found 1-bit register for signal <estado[5]_clock_DFF_77>.
Found 1-bit register for signal <estado[5]_clock_DFF_78>.
Found 1-bit register for signal <estado[5]_clock_DFF_79>.
Found 1-bit register for signal <estado[5]_clock_DFF_80>.
Found 1-bit register for signal <estado[5]_clock_DFF_81>.
Found 1-bit register for signal <estado[5]_clock_DFF_82>.
Found 1-bit register for signal <estado[5]_clock_DFF_83>.
Found 1-bit register for signal <estado[5]_clock_DFF_84>.
Found 1-bit register for signal <estado[5]_clock_DFF_85>.
Found 1-bit register for signal <estado[5]_clock_DFF_86>.
Found 1-bit register for signal <estado[5]_clock_DFF_87>.
Found 1-bit register for signal <estado[5]_clock_DFF_88>.
Found 1-bit register for signal <estado[5]_clock_DFF_89>.
Found 1-bit register for signal <estado[5]_clock_DFF_90>.
Found 1-bit register for signal <estado[5]_clock_DFF_91>.
Found 1-bit register for signal <estado[5]_clock_DFF_92>.
Found 1-bit register for signal <estado[5]_clock_DFF_93>.
Found 1-bit register for signal <estado[5]_clock_DFF_94>.
Found 1-bit register for signal <estado[5]_clock_DFF_95>.
Found 1-bit register for signal <estado[5]_clock_DFF_96>.
Found 1-bit register for signal <estado[5]_clock_DFF_97>.
Found 1-bit register for signal <estado[5]_clock_DFF_98>.
Found 1-bit register for signal <estado[5]_clock_DFF_99>.
Found 1-bit register for signal <estado[5]_clock_DFF_100>.
Found 1-bit register for signal <estado[5]_clock_DFF_101>.
Found 1-bit register for signal <estado[5]_clock_DFF_102>.
Found 1-bit register for signal <estado[5]_clock_DFF_103>.
Found 1-bit register for signal <estado[5]_clock_DFF_104>.
Found 1-bit register for signal <estado[5]_clock_DFF_105>.
Found 1-bit register for signal <estado[5]_clock_DFF_106>.
Found 1-bit register for signal <estado[5]_clock_DFF_107>.
Found 1-bit register for signal <estado[5]_clock_DFF_108>.
Found 1-bit register for signal <estado[5]_clock_DFF_109>.
Found 1-bit register for signal <estado[5]_clock_DFF_110>.
Found 1-bit register for signal <estado[5]_clock_DFF_111>.

```
Found 1-bit register for signal <estado[5]_clock_DFF_112>.
Found 1-bit register for signal <estado[5]_clock_DFF_113>.
Found 1-bit register for signal <estado[5]_clock_DFF_114>.
Found 1-bit register for signal <estado[5]_clock_DFF_115>.
Found 1-bit register for signal <estado[5]_clock_DFF_116>.
Found 1-bit register for signal <estado[5]_clock_DFF_117>.
Found 1-bit register for signal <estado[5]_clock_DFF_118>.
Found 1-bit register for signal <estado[5]_clock_DFF_119>.
Found 1-bit register for signal <estado[5]_clock_DFF_120>.
Found 1-bit register for signal <estado[5]_clock_DFF_121>.
Found 1-bit register for signal <estado[5]_clock_DFF_122>.
Found 1-bit register for signal <estado[5]_clock_DFF_123>.
Found 6-bit register for signal <estado>.
Found finite state machine <FSM_0> for signal <estado>.
-----------------------------------------------------------------------
| States            | 45                                    |
| Transitions       | 45                                    |
| Inputs            | 0                                     |
| Outputs           | 14                                    |
| Clock             | clock (rising_edge)                   |
| Reset             | reset (positive)                      |
| Reset type        | synchronous                           |
| Reset State       | inicio                                |
| Power Up State    | inicio                                |
| Encoding          | auto                                  |
| Implementation    | LUT                                   |
-----------------------------------------------------------------------
Found 13-bit adder for signal <S1[12]_E4[12]_add_7_OUT> created at line 203.
Found 13-bit adder for signal <S3[12]_G4[12]_add_13_OUT> created at line 235.
Found 13-bit adder for signal <S3[12]_J4[12]_add_21_OUT> created at line 269.
Found 13-bit adder for signal <S1[12]_L3[12]_add_26_OUT> created at line 289.
Found 13-bit adder for signal <S1[12]_O1[12]_add_32_OUT> created at line 314.
Found 13-bit adder for signal <auxA[12]_auxR4[12]_add_37_OUT> created at line 356.
Found 13-bit adder for signal <J4[12]_O1[12]_add_39_OUT> created at line 367.
Found 13-bit subtractor for signal <GND_5_o_GND_5_o_sub_31_OUT<12:0>> created at line 306.
Found 13-bit subtractor for signal <GND_5_o_GND_5_o_sub_45_OUT<12:0>> created at line 386.
Found 13x2-bit multiplier for signal <n0907> created at line 169.
Found 13x3-bit multiplier for signal <n0908> created at line 170.
Found 13x3-bit multiplier for signal <n0909> created at line 171.
Found 8192x13-bit Read Only RAM for signal <S7[12]_GND_5_o_wide_mux_3_OUT>
Found 8192x13-bit Read Only RAM for signal <auxD[12]_GND_5_o_wide_mux_4_OUT>
Found 8192x13-bit Read Only RAM for signal <auxF1[12]_GND_5_o_wide_mux_8_OUT>
Found 8192x13-bit Read Only RAM for signal <auxC[12]_GND_5_o_wide_mux_9_OUT>
Found 8192x13-bit Read Only RAM for signal <auxG1[12]_GND_5_o_wide_mux_10_OUT>
Found 8192x13-bit Read Only RAM for signal <H1[12]_GND_5_o_wide_mux_14_OUT>
Found 8192x13-bit Read Only RAM for signal <auxB[12]_GND_5_o_wide_mux_17_OUT>
Found 8192x13-bit Read Only RAM for signal <auxJ1[12]_GND_5_o_wide_mux_18_OUT>
Found 8192x13-bit Read Only RAM for signal <auxK1[12]_GND_5_o_wide_mux_22_OUT>
Found 8192x13-bit Read Only RAM for signal <auxM1[12]_GND_5_o_wide_mux_27_OUT>
Found 8192x13-bit Read Only RAM for signal <auxP1[12]_GND_5_o_wide_mux_33_OUT>
Found 8192x13-bit Read Only RAM for signal <auxR3[12]_GND_5_o_wide_mux_38_OUT>
Found 8192x13-bit Read Only RAM for signal <auxU1[12]_GND_5_o_wide_mux_41_OUT>
Found 8192x14-bit Read Only RAM for signal <_n11413>
Found 8192x14-bit Read Only RAM for signal <_n19606>
Found 8192x14-bit Read Only RAM for signal <_n27799>
Found 8192x14-bit Read Only RAM for signal <_n35992>
Found 8192x14-bit Read Only RAM for signal <_n44185>
Found 8192x14-bit Read Only RAM for signal <_n52378>
Found 8192x14-bit Read Only RAM for signal <_n60571>
Found 8192x14-bit Read Only RAM for signal <_n68764>
Found 1-bit tristate buffer for signal <E4<12>> created at line 150
Found 1-bit tristate buffer for signal <E4<11>> created at line 150
Found 1-bit tristate buffer for signal <E4<10>> created at line 150
Found 1-bit tristate buffer for signal <E4<9>> created at line 150
Found 1-bit tristate buffer for signal <E4<8>> created at line 150
Found 1-bit tristate buffer for signal <E4<7>> created at line 150
Found 1-bit tristate buffer for signal <E4<6>> created at line 150
Found 1-bit tristate buffer for signal <E4<5>> created at line 150
Found 1-bit tristate buffer for signal <E4<4>> created at line 150
Found 1-bit tristate buffer for signal <E4<3>> created at line 150
```

Found 1-bit tristate buffer for signal <E4<2>> created at line 150
Found 1-bit tristate buffer for signal <E4<1>> created at line 150
Found 1-bit tristate buffer for signal <E4<0>> created at line 150
Found 1-bit tristate buffer for signal <G4<12>> created at line 150
Found 1-bit tristate buffer for signal <G4<11>> created at line 150
Found 1-bit tristate buffer for signal <G4<10>> created at line 150
Found 1-bit tristate buffer for signal <G4<9>> created at line 150
Found 1-bit tristate buffer for signal <G4<8>> created at line 150
Found 1-bit tristate buffer for signal <G4<7>> created at line 150
Found 1-bit tristate buffer for signal <G4<6>> created at line 150
Found 1-bit tristate buffer for signal <G4<5>> created at line 150
Found 1-bit tristate buffer for signal <G4<4>> created at line 150
Found 1-bit tristate buffer for signal <G4<3>> created at line 150
Found 1-bit tristate buffer for signal <G4<2>> created at line 150
Found 1-bit tristate buffer for signal <G4<1>> created at line 150
Found 1-bit tristate buffer for signal <G4<0>> created at line 150
Found 1-bit tristate buffer for signal <I2<12>> created at line 150
Found 1-bit tristate buffer for signal <I2<11>> created at line 150
Found 1-bit tristate buffer for signal <I2<10>> created at line 150
Found 1-bit tristate buffer for signal <I2<9>> created at line 150
Found 1-bit tristate buffer for signal <I2<8>> created at line 150
Found 1-bit tristate buffer for signal <I2<7>> created at line 150
Found 1-bit tristate buffer for signal <I2<6>> created at line 150
Found 1-bit tristate buffer for signal <I2<5>> created at line 150
Found 1-bit tristate buffer for signal <I2<4>> created at line 150
Found 1-bit tristate buffer for signal <I2<3>> created at line 150
Found 1-bit tristate buffer for signal <I2<2>> created at line 150
Found 1-bit tristate buffer for signal <I2<1>> created at line 150
Found 1-bit tristate buffer for signal <I2<0>> created at line 150
Found 1-bit tristate buffer for signal <J4<12>> created at line 150
Found 1-bit tristate buffer for signal <J4<11>> created at line 150
Found 1-bit tristate buffer for signal <J4<10>> created at line 150
Found 1-bit tristate buffer for signal <J4<9>> created at line 150
Found 1-bit tristate buffer for signal <J4<8>> created at line 150
Found 1-bit tristate buffer for signal <J4<7>> created at line 150
Found 1-bit tristate buffer for signal <J4<6>> created at line 150
Found 1-bit tristate buffer for signal <J4<5>> created at line 150
Found 1-bit tristate buffer for signal <J4<4>> created at line 150
Found 1-bit tristate buffer for signal <J4<3>> created at line 150
Found 1-bit tristate buffer for signal <J4<2>> created at line 150
Found 1-bit tristate buffer for signal <J4<1>> created at line 150
Found 1-bit tristate buffer for signal <J4<0>> created at line 150
Found 1-bit tristate buffer for signal <L3<12>> created at line 150
Found 1-bit tristate buffer for signal <L3<11>> created at line 150
Found 1-bit tristate buffer for signal <L3<10>> created at line 150
Found 1-bit tristate buffer for signal <L3<9>> created at line 150
Found 1-bit tristate buffer for signal <L3<8>> created at line 150
Found 1-bit tristate buffer for signal <L3<7>> created at line 150
Found 1-bit tristate buffer for signal <L3<6>> created at line 150
Found 1-bit tristate buffer for signal <L3<5>> created at line 150
Found 1-bit tristate buffer for signal <L3<4>> created at line 150
Found 1-bit tristate buffer for signal <L3<3>> created at line 150
Found 1-bit tristate buffer for signal <L3<2>> created at line 150
Found 1-bit tristate buffer for signal <L3<1>> created at line 150
Found 1-bit tristate buffer for signal <L3<0>> created at line 150
Found 1-bit tristate buffer for signal <N2<12>> created at line 150
Found 1-bit tristate buffer for signal <N2<11>> created at line 150
Found 1-bit tristate buffer for signal <N2<10>> created at line 150
Found 1-bit tristate buffer for signal <N2<9>> created at line 150
Found 1-bit tristate buffer for signal <N2<8>> created at line 150
Found 1-bit tristate buffer for signal <N2<7>> created at line 150
Found 1-bit tristate buffer for signal <N2<6>> created at line 150
Found 1-bit tristate buffer for signal <N2<5>> created at line 150
Found 1-bit tristate buffer for signal <N2<4>> created at line 150
Found 1-bit tristate buffer for signal <N2<3>> created at line 150
Found 1-bit tristate buffer for signal <N2<2>> created at line 150
Found 1-bit tristate buffer for signal <N2<1>> created at line 150
Found 1-bit tristate buffer for signal <N2<0>> created at line 150
Found 1-bit tristate buffer for signal <Q2<12>> created at line 150
Found 1-bit tristate buffer for signal <Q2<11>> created at line 150

```
    Found 1-bit tristate buffer for signal <Q2<10>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<9>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<8>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<7>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<6>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<5>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<4>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<3>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<2>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<1>> created at line 150
    Found 1-bit tristate buffer for signal <Q2<0>> created at line 150
    Found 1-bit tristate buffer for signal <V2<12>> created at line 150
    Found 1-bit tristate buffer for signal <V2<11>> created at line 150
    Found 1-bit tristate buffer for signal <V2<10>> created at line 150
    Found 1-bit tristate buffer for signal <V2<9>> created at line 150
    Found 1-bit tristate buffer for signal <V2<8>> created at line 150
    Found 1-bit tristate buffer for signal <V2<7>> created at line 150
    Found 1-bit tristate buffer for signal <V2<6>> created at line 150
    Found 1-bit tristate buffer for signal <V2<5>> created at line 150
    Found 1-bit tristate buffer for signal <V2<4>> created at line 150
    Found 1-bit tristate buffer for signal <V2<3>> created at line 150
    Found 1-bit tristate buffer for signal <V2<2>> created at line 150
    Found 1-bit tristate buffer for signal <V2<1>> created at line 150
    Found 1-bit tristate buffer for signal <V2<0>> created at line 150
    Summary:
            inferred  21 RAM(s).
            inferred   3 Multiplier(s).
            inferred   9 Adder/Subtractor(s).
            inferred 858 D-type flip-flop(s).
            inferred  16 Multiplexer(s).
            inferred 104 Tristate(s).
            inferred   1 Finite State Machine(s).
Unit <BCH_PageDecodingSIGMA> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs                                                 : 21
 8192x13-bit single-port Read Only RAM                 : 13
 8192x14-bit single-port Read Only RAM                 : 8
# Multipliers                                          : 3
 13x2-bit multiplier                                   : 1
 13x3-bit multiplier                                   : 2
# Adders/Subtractors                                   : 9
 13-bit adder                                          : 7
 13-bit subtractor                                     : 2
# Registers                                            : 160
 1-bit register                                        : 104
 13-bit register                                       : 54
 26-bit register                                       : 2
# Multiplexers                                         : 16
 1-bit 2-to-1 multiplexer                              : 8
 13-bit 2-to-1 multiplexer                             : 8
# Tristates                                            : 104
 1-bit tristate buffer                                 : 104
# FSMs                                                 : 1
# Xors                                                 : 9
 13-bit xor2                                           : 9

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================

WARNING:Xst:1293 - FF/Latch <auxA_0> has a constant value of 0 in block <BCH_PageDecodingSIGMA>. This
FF/Latch will be trimmed during the optimization process.

Synthesizing (advanced) Unit <BCH_PageDecodingSIGMA>.
```

```
        Found pipelined multiplier on signal <n0908>:
                - 1 pipeline level(s) found in a register connected to the multiplier macro output.
                Pushing register(s) into the multiplier macro.
        Found pipelined multiplier on signal <n0907>:
                - 1 pipeline level(s) found in a register connected to the multiplier macro output.
                Pushing register(s) into the multiplier macro.
        Found pipelined multiplier on signal <n0909>:
                - 1 pipeline level(s) found in a register connected to the multiplier macro output.
                Pushing register(s) into the multiplier macro.
INFO:Xst:3226 - The RAM <Mram__n52378> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxQ1>
    -----------------------------------------------------------------------
    | ram_type          | Block                              |      |
    -----------------------------------------------------------------------
    | Port A                                                 |
    |     aspect ratio  | 8192-word x 14-bit                 |      |
    |     mode          | write-first                        |      |
    |     clkA          | connected to signal <clock>        | rise |
    |     enA           | connected to internal node         | low  |
    |     weA           | connected to signal <GND>          | high |
    |     addrA         | connected to signal <Q1>           |      |
    |     diA           | connected to signal <GND>          |      |
    |     doA           | connected to internal node         |      |
    -----------------------------------------------------------------------
    | optimization      | speed                              |      |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram__n60571> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxV1>
    -----------------------------------------------------------------------------------------
    | ram_type          | Block                                              |       |
    -----------------------------------------------------------------------------------------
    | Port A                                                                 |
    |     aspect ratio  | 8192-word x 14-bit                                 |       |
    |     mode          | write-first                                        |       |
    |     clkA          | connected to signal <clock>                        | rise  |
    |     enA           | connected to internal node                         | low   |
    |     weA           | connected to signal <GND>                          | high  |
    |     addrA         | connected to signal <T1[12]_U2[12]_xor_42_OUT>     |       |
    |     diA           | connected to signal <GND>                          |       |
    |     doA           | connected to internal node                         |       |
    -----------------------------------------------------------------------------------------
    | optimization      | speed                                              |       |
    -----------------------------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxP1[12]_GND_5_o_wide_mux_33_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <auxP1>
    -----------------------------------------------------------------
    | ram_type          | Block                          |      |
    -----------------------------------------------------------------
    | Port A                                             |
    |     aspect ratio  | 8192-word x 13-bit             |      |
    |     mode          | write-first                    |      |
    |     clkA          | connected to signal <clock>    | rise |
    |     enA           | connected to internal node     | low  |
    |     weA           | connected to signal <GND>      | high |
    |     addrA         | connected to signal <P1>       |      |
    |     diA           | connected to signal <GND>      |      |
    |     doA           | connected to internal node     |      |
    -----------------------------------------------------------------
    | optimization      | speed                          |      |
    -----------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxU1[12]_GND_5_o_wide_mux_41_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <T1,U2>
    -----------------------------------------------------------------
    | ram_type          | Block                          |      |
    -----------------------------------------------------------------
    | Port A                                                    |
    |     aspect ratio  | 8192-word x 13-bit             |      |
    |     mode          | write-first                    |      |
    |     clkA          | connected to signal <clock>    | rise |
    |     enA           | connected to internal node     | low  |
```

```
    |     weA            | connected to signal <GND>       | high     |
    |     addrA          | connected to signal <auxU1>     |          |
    |     diA            | connected to signal <GND>       |          |
    |     doA            | connected to internal node      |          |
    -----------------------------------------------------------------------
    | optimization       | speed                           |          |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram__n44185> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxI1>
    -----------------------------------------------------------------------------------------------
    | ram_type           | Block                                                  |        |
    -----------------------------------------------------------------------------------------------
    | Port A                                                                               |
    |     aspect ratio   | 8192-word x 14-bit                                     |        |
    |     mode           | write-first                                            |        |
    |     clkA           | connected to signal <clock>                            | rise   |
    |     enA            | connected to internal node                             | low    |
    |     weA            | connected to signal <GND>                              | high   |
    |     addrA          | connected to signal <H2[12]_F2[12]_xor_15_OUT>         |        |
    |     diA            | connected to signal <GND>                              |        |
    |     doA            | connected to internal node                             |        |
    -------------------------------------------------------------------------------------------------
    | optimization       | speed                                                  |        |
    -------------------------------------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram__n68764> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxN1>
    --------------------------------------------------------------------------------------------------
    | ram_type           | Block                                                  |        |
    --------------------------------------------------------------------------------------------------
    | Port A                                                                               |
    |     aspect ratio   | 8192-word x 14-bit                                     |        |
    |     mode           | write-first                                            |        |
    |     clkA           | connected to signal <clock>                            | rise   |
    |     enA            | connected to internal node                             | low    |
    |     weA            | connected to signal <GND>                              | high   |
    |     addrA          | connected to signal <K2[12]_M2[12]_xor_28_OUT>         |        |
    |     diA            | connected to signal <GND>                              |        |
    |     doA            | connected to internal node                             |        |
    ---------------------------------------------------------------------------------------------------
    | optimization       | speed                                                  |        |
    ---------------------------------------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxF1[12]_GND_5_o_wide_mux_8_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <F2>
    -------------------------------------------------------------------------
    | ram_type           | Block                           |        |
    -------------------------------------------------------------------------
    | Port A                                               |
    |     aspect ratio   | 8192-word x 13-bit              |        |
    |     mode           | write-first                     |        |
    |     clkA           | connected to signal <clock>     | rise   |
    |     enA            | connected to internal node      | low    |
    |     weA            | connected to signal <GND>       | high   |
    |     addrA          | connected to signal <auxF1>     |        |
    |     diA            | connected to signal <GND>       |        |
    |     doA            | connected to signal <F2>        |        |
    ---------------------------------------------------------------------------
    | optimization       | speed                           |        |
    ----------------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_H1[12]_GND_5_o_wide_mux_14_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <H2>
    --------------------------------------------------------------------------
    | ram_type           | Block                           |        |
    --------------------------------------------------------------------------
    | Port A                                               |
    |     aspect ratio   | 8192-word x 13-bit              |        |
    |     mode           | write-first                     |        |
    |     clkA           | connected to signal <clock>     | rise   |
    |     enA            | connected to internal node      | low    |
    |     weA            | connected to signal <GND>       | high   |
    |     addrA          | connected to signal <H1>        |        |
```

```
    |     diA           | connected to signal <GND>           |        |
    |     doA           | connected to signal <H2>            |        |
    -----------------------------------------------------------------------
    | optimization      | speed                               |        |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxK1[12]_GND_5_o_wide_mux_22_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <K2>
    -----------------------------------------------------------------------
    | ram_type          | Block                               |        |
    -----------------------------------------------------------------------
    | Port A                                                  |
    |     aspect ratio  | 8192-word x 13-bit                  |        |
    |     mode          | write-first                         |        |
    |     clkA          | connected to signal <clock>         | rise   |
    |     enA           | connected to internal node          | low    |
    |     weA           | connected to signal <GND>           | high   |
    |     addrA         | connected to signal <auxK1>         |        |
    |     diA           | connected to signal <GND>           |        |
    |     doA           | connected to signal <K2>            |        |
    -----------------------------------------------------------------------
    | optimization      | speed                               |        |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxM1[12]_GND_5_o_wide_mux_27_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <M2>
    -----------------------------------------------------------------------
    | ram_type          | Block                               |        |
    -----------------------------------------------------------------------
    | Port A                                                  |
    |     aspect ratio  | 8192-word x 13-bit                  |        |
    |     mode          | write-first                         |        |
    |     clkA          | connected to signal <clock>         | rise   |
    |     enA           | connected to internal node          | low    |
    |     weA           | connected to signal <GND>           | high   |
    |     addrA         | connected to signal <auxM1>         |        |
    |     diA           | connected to signal <GND>           |        |
    |     doA           | connected to signal <M2>            |        |
    -----------------------------------------------------------------------
    | optimization      | speed                               |        |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram__n35992> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxG2>
    -----------------------------------------------------------------------
    | ram_type          | Block                               |        |
    -----------------------------------------------------------------------
    | Port A                                                  |
    |     aspect ratio  | 8192-word x 14-bit                  |        |
    |     mode          | write-first                         |        |
    |     clkA          | connected to signal <clock>         | rise   |
    |     enA           | connected to internal node          | low    |
    |     weA           | connected to signal <GND>           | high   |
    |     addrA         | connected to signal <G3>            |        |
    |     diA           | connected to signal <GND>           |        |
    |     doA           | connected to internal node          |        |
    -----------------------------------------------------------------------
    | optimization      | speed                               |        |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram__n27799> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxE2>
    ---------------------------------------------------------------------------------------
    | ram_type          | Block                                             |        |
    ---------------------------------------------------------------------------------------
    | Port A                                                                |
    |     aspect ratio  | 8192-word x 14-bit                                |        |
    |     mode          | write-first                                       |        |
    |     clkA          | connected to signal <clock>                       | rise   |
    |     enA           | connected to internal node                        | low    |
    |     weA           | connected to signal <GND>                         | high   |
    |     addrA         | connected to signal <E1[12]_E2[12]_xor_5_OUT>     |        |
    |     diA           | connected to signal <GND>                         |        |
    |     doA           | connected to internal node                        |        |
```

```
    -----------------------------------------------------------------------
    | optimization       | speed                                    |          |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram__n19606> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxJ2>
    -----------------------------------------------------------------------
    | ram_type           | Block                                    |          |
    -----------------------------------------------------------------------
    | Port A                                                        |
    |     aspect ratio   | 8192-word x 14-bit                       |          |
    |     mode           | write-first                              |          |
    |     clkA           | connected to signal <clock>              | rise     |
    |     enA            | connected to internal node               | low      |
    |     weA            | connected to signal <GND>                | high     |
    |     addrA          | connected to signal <J2[12]_J1[12]_xor_19_OUT> |    |
    |     diA            | connected to signal <GND>                |          |
    |     doA            | connected to internal node               |          |
    -----------------------------------------------------------------------
    | optimization       | speed                                    |          |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram__n11413> will be implemented as a BLOCK RAM, absorbing the following
register(s): <auxL1>
    -----------------------------------------------------------------------
    | ram_type           | Block                                    |          |
    -----------------------------------------------------------------------
    | Port A                                                                   |
    |     aspect ratio   | 8192-word x 14-bit                       |          |
    |     mode           | write-first                              |          |
    |     clkA           | connected to signal <clock>              | rise     |
    |     enA            | connected to internal node               | low      |
    |     weA            | connected to signal <GND>                | high     |
    |     addrA          | connected to signal <L1[12]_G2[12]_xor_24_OUT> |    |
    |     diA            | connected to signal <GND>                |          |
    |     doA            | connected to internal node               |          |
    -----------------------------------------------------------------------
    | optimization       | speed                                    |          |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxR3[12]_GND_5_o_wide_mux_38_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <R2>
    -------------------------------------------------------------
    | ram_type           | Block                        |          |
    -------------------------------------------------------------
    | Port A                                            |
    |     aspect ratio   | 8192-word x 13-bit           |          |
    |     mode           | write-first                  |          |
    |     clkA           | connected to signal <clock>  | rise     |
    |     enA            | connected to internal node   | low      |
    |     weA            | connected to signal <GND>    | high     |
    |     addrA          | connected to signal <auxR3>  |          |
    |     diA            | connected to signal <GND>    |          |
    |     doA            | connected to signal <R2>     |          |
    -------------------------------------------------------------
    | optimization       | speed                        |          |
    -------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxJ1[12]_GND_5_o_wide_mux_18_OUT> will be implemented as a BLOCK RAM,
absorbing the following register(s): <J2>
    -------------------------------------------------------------
    | ram_type           | Block                        |          |
    -------------------------------------------------------------
    | Port A                                            |
    |     aspect ratio   | 8192-word x 13-bit           |          |
    |     mode           | write-first                  |          |
    |     clkA           | connected to signal <clock>  | rise     |
    |     enA            | connected to internal node   | low      |
    |     weA            | connected to signal <GND>    | high     |
    |     addrA          | connected to signal <auxJ1>  |          |
    |     diA            | connected to signal <GND>    |          |
    |     doA            | connected to signal <J2>     |          |
    -------------------------------------------------------------
    | optimization       | speed                        |          |
```

```
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxG1[12]_GND_5_o_wide_mux_10_OUT> will be implemented as a BLOCK RAM, absorbing the following register(s): <G2>
    -----------------------------------------------------------------------
    | ram_type           | Block                                  |          |
    -----------------------------------------------------------------------
    | Port A                                                      |
    |     aspect ratio   | 8192-word x 13-bit                     |          |
    |     mode           | write-first                            |          |
    |     clkA           | connected to signal <clock>            | rise     |
    |     enA            | connected to internal node             | low      |
    |     weA            | connected to signal <GND>              | high     |
    |     addrA          | connected to signal <auxG1>            |          |
    |     diA            | connected to signal <GND>              |          |
    |     doA            | connected to signal <G2>               |          |
    -----------------------------------------------------------------------
    | optimization       | speed                                  |          |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxB[12]_GND_5_o_wide_mux_17_OUT> will be implemented as a BLOCK RAM, absorbing the following register(s): <J1>
    -----------------------------------------------------------------------
    | ram_type           | Block                                  |          |
    -----------------------------------------------------------------------
    | Port A                                                      |
    |     aspect ratio   | 8192-word x 13-bit                     |          |
    |     mode           | write-first                            |          |
    |     clkA           | connected to signal <clock>            | rise     |
    |     enA            | connected to internal node             | low      |
    |     weA            | connected to signal <GND>              | high     |
    |     addrA          | connected to signal <auxB>             |          |
    |     diA            | connected to signal <GND>              |          |
    |     doA            | connected to signal <J1>               |          |
    -----------------------------------------------------------------------
    | optimization       | speed                                  |          |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_auxD[12]_GND_5_o_wide_mux_4_OUT> will be implemented as a BLOCK RAM, absorbing the following register(s): <E2>
    -----------------------------------------------------------------------
    | ram_type           | Block                                  |          |
    -----------------------------------------------------------------------
    | Port A                                                      |
    |     aspect ratio   | 8192-word x 13-bit                     |          |
    |     mode           | write-first                            |          |
    |     clkA           | connected to signal <clock>            | rise     |
    |     enA            | connected to internal node             | low      |
    |     weA            | connected to signal <GND>              | high     |
    |     addrA          | connected to signal <auxD>             |          |
    |     diA            | connected to signal <GND>              |          |
    |     doA            | connected to signal <E2>               |          |
    -----------------------------------------------------------------------
    | optimization       | speed                                  |          |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_S7[12]_GND_5_o_wide_mux_3_OUT> will be implemented as a BLOCK RAM, absorbing the following register(s): <E1>
    -----------------------------------------------------------------------
    | ram_type           | Block                                  |          |
    -----------------------------------------------------------------------
    | Port A                                                      |
    |     aspect ratio   | 8192-word x 13-bit                     |          |
    |     mode           | write-first                            |          |
    |     clkA           | connected to signal <clock>            | rise     |
    |     enA            | connected to internal node             | low      |
    |     weA            | connected to signal <GND>              | high     |
    |     addrA          | connected to signal <S7>               |          |
    |     diA            | connected to signal <GND>              |          |
    |     doA            | connected to signal <E1>               |          |
    -----------------------------------------------------------------------
    | optimization       | speed                                  |          |
    -----------------------------------------------------------------------
```

```
INFO:Xst:3218 - HDL ADVISOR - The RAM <Mram_auxC[12]_GND_5_o_wide_mux_9_OUT> will be implemented on
LUTs either because you have described an asynchronous read or because of currently unsupported block RAM
features. If you have described an asynchronous read, making it synchronous would allow you to take advantage of
available block RAM resources, for optimized device usage and improved timings. Please refer to your documentation
for coding guidelines.
    -----------------------------------------------------------------------
    | ram_type           | Distributed                         |      |
    -----------------------------------------------------------------------
    | Port A                                                   |
    |     aspect ratio   | 8192-word x 13-bit                  |      |
    |     weA            | connected to signal <GND>           | high |
    |     addrA          | connected to signal <auxC>          |      |
    |     diA            | connected to signal <GND>           |      |
    |     doA            | connected to internal node          |      |
    -----------------------------------------------------------------------
Unit <BCH_PageDecodingSIGMA> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                                 : 21
 8192x13-bit single-port block Read Only RAM           : 12
 8192x13-bit single-port distributed Read Only RAM     : 1
 8192x14-bit single-port block Read Only RAM           : 8
# Multipliers                                          : 3
 13x2-bit registered multiplier                        : 1
 13x3-bit registered multiplier                        : 2
# Adders/Subtractors                                   : 9
 13-bit adder                                          : 7
 13-bit subtractor                                     : 2
# Registers                                            : 559
 Flip-Flops                                            : 559
# Multiplexers                                         : 8
 13-bit 2-to-1 multiplexer                             : 8
# FSMs                                                 : 1
# Xors                                                 : 9
 13-bit xor2                                           : 9

=========================================================================

=========================================================================
*                          Low Level Synthesis                          *
=========================================================================

Analyzing FSM <MFsm> for best encoding.
Optimizing FSM <FSM_0> on signal <estado[1:6]> with user encoding.
-------------------------
 State       | Encoding
-------------------------
 inicio      | 000000
 e_sigma1    | 000001
 e_sigma2    | 000010
 e_sigma2_1  | 000011
 e_sigma2_2  | 000100
 e_sigma2_3  | 000101
 e_sigma2_4  | 000110
 e_sigma2_5  | 000111
 e_sigma2_6  | 001000
 e_sigma2_7  | 001001
 e_sigma2_8  | 001010
 e_sigma2_9  | 001011
 e_sigma2_10 | 001100
 e_sigma2_11 | 001101
 e_sigma2_12 | 001110
 e_sigma2_13 | 001111
 e_sigma2_14 | 010000
 e_sigma2_15 | 010001
 e_sigma2_16 | 010010
 e_sigma2_17 | 010011
```

```
 e_sigma2_18 | 010100
 e_sigma2_19 | 010101
 e_sigma2_20 | 010110
 e_sigma2_21 | 010111
 e_sigma2_22 | 011000
 e_sigma2_23 | 011001
 e_sigma2_24 | 011010
 e_sigma2_25 | 011011
 e_sigma2_26 | 011100
 e_sigma3_1  | 011101
 e_sigma3_2  | 011110
 e_sigma3_3  | 011111
 e_sigma3_4  | 100000
 e_sigma3_5  | 100001
 e_sigma3_6  | 100010
 e_sigma3_7  | 100011
 e_sigma4_1  | 100100
 e_sigma4_2  | 100101
 e_sigma4_3  | 100110
 e_sigma4_4  | 100111
 e_sigma4_5  | 101000
 e_sigma4_6  | 101001
 e_sigma4_7  | 101010
 e_sigma4_8  | 101011
 e_sigma4_9  | 101100
-------------------------


Optimizing unit <BCH_PageDecodingSIGMA> ...
INFO:Xst:2261 - The FF/Latch <Mmult_n0909_14> in Unit <BCH_PageDecodingSIGMA> is equivalent to the following
FF/Latch, which will be removed : <Mmult_n0907_13>

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block BCH_PageDecodingSIGMA, actual ratio is 4.

Final Macro Processing ...

Processing Unit <BCH_PageDecodingSIGMA> :
	Found 2-bit shift register for signal <Sigma4_0>.
	Found 2-bit shift register for signal <Sigma4_1>.
	Found 2-bit shift register for signal <Sigma4_2>.
	Found 2-bit shift register for signal <Sigma4_3>.
	Found 2-bit shift register for signal <Sigma4_4>.
	Found 2-bit shift register for signal <Sigma4_5>.
	Found 2-bit shift register for signal <Sigma4_6>.
	Found 2-bit shift register for signal <Sigma4_7>.
	Found 2-bit shift register for signal <Sigma4_8>.
	Found 2-bit shift register for signal <Sigma4_9>.
	Found 2-bit shift register for signal <Sigma4_10>.
	Found 2-bit shift register for signal <Sigma4_11>.
	Found 2-bit shift register for signal <Sigma4_12>.
Unit <BCH_PageDecodingSIGMA> processed.

=========================================================================
Final Register Report

Macro Statistics
# Registers                                            : 451
 Flip-Flops                                            : 451
# Shift Registers                                      : 13
 2-bit shift register                                  : 13

=========================================================================

=========================================================================
*                          Partition Report                             *
=========================================================================

Partition Implementation Status
```

No Partitions were found in this design.

```
=========================================================================
*                        Design Summary                               *
=========================================================================

Top Level Output File Name         : BCH_PageDecodingSIGMA.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 3945
#      GND                         : 1
#      INV                         : 20
#      LUT2                        : 264
#      LUT3                        : 83
#      LUT4                        : 85
#      LUT5                        : 16
#      LUT6                        : 1865
#      MUXCY                       : 149
#      MUXF7                       : 884
#      MUXF8                       : 416
#      VCC                         : 1
#      XORCY                       : 161
# FlipFlops/Latches                : 464
#      FD                          : 104
#      FDE                         : 354
#      FDR                         : 6
# RAMS                             : 80
#      RAMB18E1                    : 20
#      RAMB36E1                    : 60
# Shift Registers                  : 13
#      SRLC16E                     : 13
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 105
#      IBUF                        : 53
#      OBUF                        : 52

Device utilization summary:
---------------------------

Selected Device : 7a100tcsg324-3


Slice Logic Utilization:
 Number of Slice Registers:             464  out of  126800     0%
 Number of Slice LUTs:                 2346  out of   63400     3%
    Number used as Logic:              2333  out of   63400     3%
    Number used as Memory:               13  out of   19000     0%
       Number used as SRL:               13

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:   2470
   Number with an unused Flip Flop:    2006  out of    2470    81%
   Number with an unused LUT:           124  out of    2470     5%
   Number of fully used LUT-FF pairs:   340  out of    2470    13%
   Number of unique control sets:        32

IO Utilization:
 Number of IOs:                         106
 Number of bonded IOBs:                 106  out of     210    50%

Specific Feature Utilization:
 Number of Block RAM/FIFO:               70  out of     135    51%
    Number using Block RAM only:         70
 Number of BUFG/BUFGCTRL                  1  out of      32     3%
```

```
---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clock                              | BUFGP                  | 557   |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
-----------------------------------+------------------------------------------------+--------+
Control Signal                     | Buffer(FF name)                                | Load   |
-----------------------------------+------------------------------------------------+--------+
auxA<0>(XST_GND:G)                 | NONE(Mram_auxJ1[12]_GND_5_o_wide_mux_18_OUT4)| 40     |
-----------------------------------+------------------------------------------------+--------+

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 3.456ns (Maximum Frequency: 289.320MHz)
   Minimum input arrival time before clock: 2.642ns
   Maximum output required time after clock: 0.855ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clock'
  Clock period: 3.456ns (frequency: 289.320MHz)
  Total number of paths / destination ports: 28714 / 941
-------------------------------------------------------------------------
  Delay:               3.456ns (Levels of Logic = 6)
  Source:              auxA_1 (FF)
  Destination:         G1_0 (FF)
  Source Clock:        clock rising
  Destination Clock: clock rising

  Data Path: auxA_1 to G1_0
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q           1674   0.361   0.908  auxA_1 (auxA_1)
     LUT6:I0->O            1   0.097   0.000  Mram_auxC[12]_GND_5_o_wide_mux_9_OUT192
(Mram_auxC[12]_GND_5_o_wide_mux_9_OUT192)
     MUXF7:I1->O           1   0.279   0.000  Mram_auxC[12]_GND_5_o_wide_mux_9_OUT19_f7_0
(Mram_auxC[12]_GND_5_o_wide_mux_9_OUT19_f71)
     MUXF8:I0->O           1   0.218   0.556  Mram_auxC[12]_GND_5_o_wide_mux_9_OUT19_f8
(Mram_auxC[12]_GND_5_o_wide_mux_9_OUT19_f8)
     LUT6:I2->O            1   0.097   0.000  Mram_auxC[12]_GND_5_o_wide_mux_9_OUT341
(Mram_auxC[12]_GND_5_o_wide_mux_9_OUT341)
```

```
    MUXF7:I1->O           1   0.279   0.556  Mram_auxC[12]_GND_5_o_wide_mux_9_OUT34_f7
(Mram_auxC[12]_GND_5_o_wide_mux_9_OUT34_f7)
    LUT6:I2->O            2   0.097   0.000  Mram_auxC[12]_GND_5_o_wide_mux_9_OUT3610
(auxC[12]_GND_5_o_wide_mux_9_OUT<0>)
    FDE:D                     0.008          G1_0
    ----------------------------------------
    Total                      3.456ns (1.436ns logic, 2.020ns route)
                                       (41.5% logic, 58.5% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clock'
  Total number of paths / destination ports: 3403 / 674
-------------------------------------------------------------------------
Offset:              2.642ns (Levels of Logic = 15)
  Source:            S1<2> (PAD)
  Destination:       Mmult_n0909_3 (FF)
  Destination Clock: clock rising

  Data Path: S1<2> to Mmult_n0909_3
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            12   0.001   0.430  S1_2_IBUF (S1_2_IBUF)
     LUT2:I0->O            1   0.097   0.000  Mmult_n0909_Madd_lut<2> (Mmult_n0909_Madd_lut<2>)
     MUXCY:S->O            1   0.353   0.000  Mmult_n0909_Madd_cy<2> (Mmult_n0909_Madd_cy<2>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<3> (Mmult_n0909_Madd_cy<3>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<4> (Mmult_n0909_Madd_cy<4>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<5> (Mmult_n0909_Madd_cy<5>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<6> (Mmult_n0909_Madd_cy<6>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<7> (Mmult_n0909_Madd_cy<7>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<8> (Mmult_n0909_Madd_cy<8>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<9> (Mmult_n0909_Madd_cy<9>)
     MUXCY:CI->O           1   0.023   0.000  Mmult_n0909_Madd_cy<10> (Mmult_n0909_Madd_cy<10>)
     XORCY:CI->O           1   0.370   0.379  Mmult_n0909_Madd_xor<11> (Mmult_n0909_Madd_11)
     LUT2:I0->O            1   0.097   0.000  Mmult_n0909_Madd1_lut<11> (Mmult_n0909_Madd1_lut<11>)
     MUXCY:S->O            0   0.353   0.000  Mmult_n0909_Madd1_cy<11> (Mmult_n0909_Madd1_cy<11>)
     XORCY:CI->O           1   0.370   0.000  Mmult_n0909_Madd1_xor<12> (Mmult_n0909_Madd_121)
     FDE:D                     0.008          Mmult_n0909_3
    ----------------------------------------
    Total                      2.642ns (1.833ns logic, 0.809ns route)
                                       (69.4% logic, 30.6% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clock'
  Total number of paths / destination ports: 52 / 52
-------------------------------------------------------------------------
Offset:              0.855ns (Levels of Logic = 1)
  Source:            auxA_1 (FF)
  Destination:       Sigma1<0> (PAD)
  Source Clock:      clock rising

  Data Path: auxA_1 to Sigma1<0>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q           1674   0.361   0.494  auxA_1 (auxA_1)
     OBUF:I->O                 0.000          Sigma1_0_OBUF (Sigma1<0>)
    ----------------------------------------
    Total                      0.855ns (0.361ns logic, 0.494ns route)
                                       (42.2% logic, 57.8% route)

=========================================================================

Cross Clock Domains Report:
--------------------------

Clock to Setup on destination clock clock
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
```

```
---------------+---------+---------+---------+---------+
clock          |    3.456|         |         |         |
---------------+---------+---------+---------+---------+

=========================================================================


Total REAL time to Xst completion: 495.00 secs
Total CPU time to Xst completion: 494.83 secs

-->

Total memory usage is 514736 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :   11 (   0 filtered)
Number of infos    :   57 (   0 filtered)
```

============== BCH_Page_decoding_ErrorEquation.prj===============

Release 14.6 - xst P.68d (nt)
Copyright (c) 1995-2013 Xilinx, Inc. All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp

Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.18 secs

--> Parameter xsthdpdir set to xst

Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.18 secs

--> Reading design: BCH_Page_decoding_ErrorEquation.prj

TABLE OF CONTENTS

1) Synthesis Options Summary
2) HDL Parsing
3) HDL Elaboration
4) HDL Synthesis
    4.1) HDL Synthesis Report
5) Advanced HDL Synthesis
    5.1) Advanced HDL Synthesis Report
6) Low Level Synthesis
7) Partition Report
8) Design Summary
    8.1) Primitive and Black Box Usage
    8.2) Device utilization summary
    8.3) Partition Resource Summary
    8.4) Timing Report
        8.4.1) Clock Information
        8.4.2) Asynchronous Control Signals Information
        8.4.3) Timing Summary
        8.4.4) Timing Details
        8.4.5) Cross Clock Domains Report

===
*   Synthesis Options Summary   *
===
---- Source Parameters
Input File Name   : "BCH_Page_decoding_ErrorEquation.prj"
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name   : "BCH_Page_decoding_ErrorEquation"
Output Format   : NGC
Target Device   : xc7a100t-3-csg324

---- Source Options
Top Module Name   : BCH_Page_decoding_ErrorEquation
Automatic FSM Extraction   : YES
FSM Encoding Algorithm   : Auto
Safe Implementation   : No
FSM Style   : LUT
RAM Extraction   : Yes
RAM Style   : Auto
ROM Extraction   : Yes
Shift Register Extraction   : YES
ROM Style   : Auto
Resource Sharing   : YES
Asynchronous To Synchronous   : NO
Shift Register Minimum Size   : 2

```
Use DSP Block                        : Auto
Automatic Register Balancing         : No

---- Target Options
LUT Combining                        : Auto
Reduce Control Sets                  : Auto
Add IO Buffers                       : YES
Global Maximum Fanout                : 100000
Add Generic Clock Buffer(BUFG)       : 32
Register Duplication                 : YES
Optimize Instantiated Primitives     : NO
Use Clock Enable                     : Auto
Use Synchronous Set                  : Auto
Use Synchronous Reset                : Auto
Pack IO Registers into IOBs          : Auto
Equivalent register Removal          : YES

---- General Options
Optimization Goal                    : Speed
Optimization Effort                  : 1
Power Reduction                      : No
Keep Hierarchy                       : No
Netlist Hierarchy                    : As_Optimized
RTL Output                           : Yes
Global Optimization                  : AllClockNets
Read Cores                           : YES
Write Timing Constraints             : NO
Cross Clock Analysis                 : NO
Hierarchy Separator                  : /
Bus Delimiter                        : <>
Case Specifier                       : Maintain
Slice Utilization Ratio              : 100
BRAM Utilization Ratio               : 100
DSP48 Utilization Ratio              : 100
Auto BRAM Packing                    : NO
Slice Utilization Ratio Delta        : 5

=========================================================================


=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Page_decoding_ErrorEquation\BCH_Page_decoding_ErrorEquation
.vhd" into library work
Parsing entity <BCH_Page_decoding_ErrorEquation>.
Parsing architecture <Behavioral> of entity <bch_page_decoding_errorequation>.

=========================================================================
*                            HDL Elaboration                            *
=========================================================================

Elaborating entity <BCH_Page_decoding_ErrorEquation> (architecture <Behavioral>) from library <work>.
INFO:HDLCompiler:679 -
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Page_decoding_ErrorEquation\BCH_Page_decoding_ErrorEquation
.vhd" Line 230. Case statement is complete. others clause is never selected
WARNING:HDLCompiler:1127 -
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Page_decoding_ErrorEquation\BCH_Page_decoding_ErrorEquation
.vhd" Line 115: Assignment to temp_aux2 ignored, since the identifier is never used

=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <BCH_Page_decoding_ErrorEquation>.
    Related source file is
"\\vboxsrv\pasta_compartilhada\BCH_PAGE\BCH_Page_decoding_ErrorEquation\BCH_Page_decoding_ErrorEquation
.vhd".
```

```
WARNING:Xst:2999 - Signal 'tabGFexpbin', unconnected in block 'BCH_Page_decoding_ErrorEquation', is tied to its
initial value.
    Found 13-bit register for signal <i>.
    Found 3-bit register for signal <conta_raiz>.
    Found 2-bit register for signal <root<0>>.
    Found 2-bit register for signal <root<1>>.
    Found 2-bit register for signal <root<2>>.
    Found 2-bit register for signal <root<3>>.
    Found 13-bit register for signal <TermoA1>.
    Found 13-bit register for signal <TermoA2>.
    Found 13-bit register for signal <TermoA3>.
    Found 13-bit register for signal <TermoA_aux1>.
    Found 13-bit register for signal <TermoB1>.
    Found 13-bit register for signal <TermoB2>.
    Found 13-bit register for signal <TermoB3>.
    Found 13-bit register for signal <TermoB_aux1>.
    Found 13-bit register for signal <TermoC1>.
    Found 13-bit register for signal <TermoC2>.
    Found 13-bit register for signal <TermoC3>.
    Found 13-bit register for signal <TermoC_aux1>.
    Found 13-bit register for signal <TermoD1>.
    Found 13-bit register for signal <TermoD2>.
    Found 13-bit register for signal <TermoD3>.
    Found 13-bit register for signal <TermoD_aux1>.
    Found 13-bit register for signal <Temp_aux1>.
    Found 13-bit register for signal <Raiz1>.
    Found 13-bit register for signal <Raiz2>.
    Found 13-bit register for signal <Raiz3>.
    Found 13-bit register for signal <Raiz4>.
    Found 5-bit register for signal <state>.
INFO:Xst:1799 - State final is never reached in FSM <state>.
    Found finite state machine <FSM_0> for signal <state>.
    -----------------------------------------------------------------------
    | States             | 21                    |
    | Transitions        | 20                    |
    | Inputs             | 0                     |
    | Outputs            | 5                     |
    | Clock              | clk (rising_edge)     |
    | Reset              | reset (positive)      |
    | Reset type         | synchronous           |
    | Reset State        | inicio                |
    | Power Up State     | inicio                |
    | Encoding           | auto                  |
    | Implementation     | LUT                   |
    -----------------------------------------------------------------------
    Found 13-bit adder for signal <TermoA1[12]_i[12]_add_0_OUT> created at line 141.
    Found 14-bit adder for signal <n0143> created at line 161.
    Found 15-bit adder for signal <n0146> created at line 178.
    Found 15-bit adder for signal <n0148> created at line 195.
    Found 3-bit adder for signal <conta_raiz[2]_GND_5_o_add_14_OUT> created at line 217.
    Found 13-bit adder for signal <i[12]_GND_5_o_add_27_OUT> created at line 220.
    Found 13x2-bit multiplier for signal <n0325> created at line 178.
    Found 8192x13-bit dual-port Read Only RAM <Mram_tabGFexpbin> for signal <tabGFexpbin>.
    Found 13-bit comparator equal for signal <Temp_aux1[12]_TermoD_aux1[12]_equal_14_o> created at line 216
    Summary:
        inferred   2 RAM(s).
        inferred   1 Multiplier(s).
        inferred   6 Adder/Subtractor(s).
        inferred 297 D-type flip-flop(s).
        inferred   1 Comparator(s).
        inferred   6 Multiplexer(s).
        inferred   1 Finite State Machine(s).
Unit <BCH_Page_decoding_ErrorEquation> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs                                                 : 2
 8192x13-bit dual-port Read Only RAM                   : 2
```

```
# Multipliers                                  : 1
 13x2-bit multiplier                           : 1
# Adders/Subtractors                           : 6
 13-bit adder                                  : 2
 14-bit adder                                  : 1
 15-bit adder                                  : 2
 3-bit adder                                   : 1
# Registers                                    : 27
 13-bit register                               : 22
 2-bit register                                : 4
 3-bit register                                : 1
# Comparators                                  : 1
 13-bit comparator equal                       : 1
# Multiplexers                                 : 6
 13-bit 2-to-1 multiplexer                     : 5
 3-bit 2-to-1 multiplexer                      : 1
# FSMs                                         : 1
# Xors                                         : 1
 13-bit xor4                                   : 1

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================


Synthesizing (advanced) Unit <BCH_Page_decoding_ErrorEquation>.
The following registers are absorbed into counter <i>: 1 register on signal <i>.
The following registers are absorbed into counter <conta_raiz>: 1 register on signal <conta_raiz>.
	Multiplier <Mmult_n0325> in block <BCH_Page_decoding_ErrorEquation> and adder/subtractor <Madd_n0146_Madd> in block <BCH_Page_decoding_ErrorEquation> are combined into a MAC<Maddsub_n0325>.
INFO:Xst:3226 - The RAM <Mram_tabGFexpbin1> will be implemented as a BLOCK RAM, absorbing the following register(s): <TermoC_aux1> <TermoD_aux1>
    -----------------------------------------------------------------------
    | ram_type           | Block                                  |      |
    -----------------------------------------------------------------------
    | Port A                                                             |
    |     aspect ratio   | 8192-word x 13-bit                     |      |
    |     mode           | write-first                            |      |
    |     clkA           | connected to signal <clk>              | rise |
    |     enA            | connected to internal node             | low  |
    |     weA            | connected to signal <GND>              | high |
    |     addrA          | connected to signal <TermoC3>          |      |
    |     diA            | connected to signal <GND>              |      |
    |     doA            | connected to signal <TermoC_aux1>      |      |
    -----------------------------------------------------------------------
    | optimization       | speed                                  |      |
    -----------------------------------------------------------------------
    | Port B                                                             |
    |     aspect ratio   | 8192-word x 13-bit                     |      |
    |     mode           | write-first                            |      |
    |     clkB           | connected to signal <clk>              | rise |
    |     enB            | connected to internal node             | low  |
    |     addrB          | connected to signal <TermoD3>          |      |
    |     doB            | connected to signal <TermoD_aux1>      |      |
    -----------------------------------------------------------------------
    | optimization       | speed                                  |      |
    -----------------------------------------------------------------------
INFO:Xst:3226 - The RAM <Mram_tabGFexpbin> will be implemented as a BLOCK RAM, absorbing the following register(s): <TermoA_aux1> <TermoB_aux1>
    -----------------------------------------------------------------------
    | ram_type           | Block                                  |      |
    -----------------------------------------------------------------------
    | Port A                                                             |
    |     aspect ratio   | 8192-word x 13-bit                     |      |
    |     mode           | write-first                            |      |
    |     clkA           | connected to signal <clk>              | rise |
    |     enA            | connected to internal node             | low  |
```

```
    |     weA              | connected to signal <GND>                 | high  |
    |     addrA            | connected to signal <TermoA3>             |       |
    |     diA              | connected to signal <GND>                 |       |
    |     doA              | connected to signal <TermoA_aux1>         |       |
    -----------------------------------------------------------------------------------------
    | optimization         | speed                                     |       |
    -----------------------------------------------------------------------------------------
    | Port B                                                                   |
    |     aspect ratio     | 8192-word x 13-bit                        |       |
    |     mode             | write-first                               |       |
    |     clkB             | connected to signal <clk>                 | rise  |
    |     enB              | connected to internal node                | low   |
    |     addrB            | connected to signal <TermoB3>             |       |
    |     doB              | connected to signal <TermoB_aux1>         |       |
    -----------------------------------------------------------------------------------------
    | optimization         | speed                                     |       |
    -----------------------------------------------------------------------------------------
Unit <BCH_Page_decoding_ErrorEquation> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                                 : 2
 8192x13-bit dual-port block Read Only RAM             : 2
# MACs                                                 : 1
 13x2-to-13-bit MAC                                    : 1
# Adders/Subtractors                                   : 3
 13-bit adder                                          : 3
# Counters                                             : 2
 13-bit up counter                                     : 1
 3-bit up counter                                      : 1
# Registers                                            : 229
 Flip-Flops                                            : 229
# Comparators                                          : 1
 13-bit comparator equal                               : 1
# Multiplexers                                         : 4
 13-bit 2-to-1 multiplexer                             : 4
# FSMs                                                 : 1
# Xors                                                 : 1
 13-bit xor4                                           : 1

=========================================================================

=========================================================================
*                          Low Level Synthesis                          *
=========================================================================

-------------------
 State  | Encoding
-------------------
 inicio | 00000
 ta_1   | 00001
 ta_11  | 00010
 ta_12  | 00011
 ta_13  | 00100
 tb_1   | 00101
 tb_11  | 00110
 tb_12  | 00111
 tb_13  | 01000
 tc_1   | 01001
 tc_11  | 01010
 tc_12  | 01011
 tc_13  | 01100
 td_1   | 01101
 td_11  | 01110
 td_12  | 01111
 td_13  | 10000
 te_1   | 10001
 te_11  | 10010
```

```
 te_12  | 10011
 final  | unreached
--------------------



Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block BCH_Page_decoding_ErrorEquation, actual ratio is 0.

Final Macro Processing ...

=========================================================================
Final Register Report

Found no macro
=========================================================================

=========================================================================
*                          Partition Report                          *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                            Design Summary                             *
=========================================================================

Top Level Output File Name         : BCH_Page_decoding_ErrorEquation.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 2
#      GND                         : 1
#      VCC                         : 1
# IO Buffers                       : 52
#      OBUF                        : 52

Device utilization summary:
---------------------------

Selected Device : 7a100tcsg324-3


Slice Logic Utilization:

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:      0
   Number with an unused Flip Flop:       0  out of      0
   Number with an unused LUT:             0  out of      0
   Number of fully used LUT-FF pairs:     0  out of      0
   Number of unique control sets:         0

IO Utilization:
 Number of IOs:                         106
 Number of bonded IOBs:                  52  out of    210    24%

Specific Feature Utilization:

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.
```

```
---------------------------

================================================================================
Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
No clock signals found in this design

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

================================================================================

Cross Clock Domains Report:
--------------------------

================================================================================


Total REAL time to Xst completion: 17.00 secs
Total CPU time to Xst completion: 17.12 secs

-->

Total memory usage is 248368 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :  297 (   0 filtered)
Number of infos    :    4 (   0 filtered)
```