VOL. 10, NO. 8, MAY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

PIPELINE ARCHITECTURE FOR FAST DECODING OF BCH CODES FOR NOR FLASH MEMORY

Sunita M.S.[1,2], ChiranthV.[2], Akash H.C.[2] and Kanchana Bhaaskaran V.S.[1]
[1]VIT University, Chennai Campus, India
[2]PES Institute of Technology, Bangalore, India
E-Mail: sunitha@pes.edu

ABSTRACT

The Bose-Chaudhuri-Hocquenghem (BCH) codes form a class of random error correcting cyclic codes capable of multiple error correction. This paper develops a new high throughput error correction mechanism for NOR flashe memories employing BCH codes. The high throughput is achieved by using pipeline architecture for decoding. The decoding of BCH codes is a complex process with multiple decoding stages and hence incurs a large decoding time. The pipeline mechanism enables multiple decoding stages to run concurrently rather than sequentially, which can in effect, significantly increase the throughput. Thus, this paper proposes a novel 2-stage pipeline circuit for the decoder. For validating the circuit, this has been compared with the conventional 3-stage pipeline and also with the non-pipeline decoding. The decoder area and power are found to be about 30% less than that of the 3-stage pipeline architecture. The throughput of the decoder is found to increase from 200Mb/s to 437Mb/s while operating for a clock frequency of 1GHz, which is a sweeping increase of about 118%. This significantly improves the system performance and hence, this architecture is depicted ideal for the high speed NOR flash memory.

Keywords: Memory testing, BCH codes, pipeline decoder, double error correction, NOR flash memory.

1. INTRODUCTION

Embedded memories play an important role in the semiconductor market primarily because of the fact that the system-on-chip market is booming and almost every system chip contains some type of embedded memory. The impact of technology scaling for high-density, low voltage levels, small feature size and small noise margins has made the memory chips increasingly susceptible to soft errors, which can change the logical value of a memory cell without damaging it. Memory cells are, therefore, affected not only at extreme radiation environments but also at normal terrestrial conditions [1]. There are various self-diagnostic techniques to test embedded memories [2]. Error Correcting Codes (ECC) is one of the commonly used methods of mitigating errors in memories. Various ECC schemes have been proposed in the literature to correct single, double and multiple errors [3] - [6]. For low soft error rates, at normal terrestrial conditions, the single error correction codes such as the Hamming codes are excellent due to their low encoding and decoding complexity. The Cyclic codes, with their algebraic structure, are highly efficient for single error correction, since their encoding and syndrome computation circuits can be implemented easily using shift registers with feedback connections. The decoder area and the decoding latency are found to be much lesser than a few other single error correction codes [3]. For superior error correction capabilities, more powerful error correction codes are needed. Many codes such as the Orthogonal Latin Square Code (OLSC) proposed by Hsiao M.Y *et al*. [4], Reed-Solomon (RS) Codes [5] and other advanced codes such as Low Density Parity Check (LDPC) Codes [6] have been proposed which can correct larger number of errors, however, either at the cost of high decoding complexity and large overhead or slow decoding time and reduced system performance.

This paper focuses mainly on correction of double errors in memory using (15, 7) BCH code. One of the main applications of double-error correcting BCH codes (DEC-BCH) is in flash memory, particularly the NOR flash. Traditional NOR flash memory products use Hamming code with only 1-bit error correction capability due to its simple decoding algorithm, small circuit area and less decoding time. However, as the bit error rate (BER) increases, the 2-bit error correcting BCH code becomes the preferable ECC.

Normally, the NOR flash is used for code storage and acts as execute in place (XIP) memory where CPU fetches instructions directly from memory. The code storage requires an exceedingly reliable NOR flash memory, since any code error will cause a system fault. In addition, the NOR flash memory has fast read access. This imposes stringent requirement on the latency of the ECC decoder that is inserted between the flash memory and the data bus [7]. Hence, the primary concern in using the DEC BCH code in NOR flash memory is the decoding latency.

The decoding of BCH codes is normally done in multiple stages. Hence, it is not only complex, but it also incurs a longer decoding time in the process. Various decoding algorithms and decoding architectures have been proposed to reduce the decoding time and the hardware complexity of the decoder. The use of a 3-stage pipeline structure for the decoder with the 3 stages consisting of syndrome computation stage, the Berlekamp-Massey (BM) decoding stage and the Chien search stage has been discussed in [8]. To reduce the hardware complexity of the various decoder stages, group matching scheme has been proposed in [9]. A parallel BCH Encoder - Decoder architecture using the simplified inverse-free BM

VOL. 10, NO. 8, MAY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

algorithm has also been proposed in [10], which focussed on reducing the hardware complexity and hence reduced decoder power.

In this paper, we have developed a 2-stage pipelined architecture for the DEC BCH (15, 7) code. Furthermore, a high speed decoder is designed for the NOR flash memory aiming at reduction of the decoding latency, such that the faster read access ability of the NOR flash memory remains unaffected.

The remainder of this paper is organized as follows. Section 2 describes the encoding of data using (15, 7) BCH code for double error correction. Section 3 describes the decoding algorithm of the BCH code. Section 4 deals with the 3-stage pipeline architecture for the decoder. The newly developed 2-stage pipeline architecture is described in Section 5. The implementation method is explained in Section 6. Section 7 presents the results and discussion. Section 8 concludes the paper.

2. BCH ENCODING FOR DOUBLE ERROR CORRECTION

Binary BCH codes belong to the class of cyclic codes with the following parameters

Block length: $n = 2m - 1$
Number of parity - check bits: $n - k \leq mt$
Minimum distance: $dmin \geq 2t + 1$

This code is capable of correcting any combination of *t* or fewer errors in a block of *n* bits. Here, *k* represents the number of data bits and *m* is a positive integer ($m \geq 3$). Double error correction is achieved by using a code whose (n, k) = (15, 7). Thus m = 4 and t = 2 for this code. Therefore, this method is capable of correcting a maximum of 2 errors in a code of length of 15 bits and has a minimum distance of exactly 5 [5].

Encoding involves multiplying the data polynomial $d(x) = d_0 + d_1x + d_2x^2 + \ldots\ldots + d_6x^6$ with the generator polynomial $g(x) = g_0 + g_1x + g_2x^2 + \ldots\ldots + g_8x^8$ to obtain the code polynomial $c(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{14}x^{14}$. The encoding and decoding of the binary BCH code is based on binary Galois field represented by GF(2^m). The generator polynomial for the (15, 7) BCH code is specified in terms of its roots from the Galois field GF(2^4). If α is a primitive element in GF (2^4), then g(x) is the lowest degree polynomial over the binary Galois field GF(2) that has $\alpha, \alpha^2, \alpha^3, \ldots\ldots \alpha^{2t}$ as its roots. For t = 2, the roots are $\alpha, \alpha^2, \alpha^3$and α^4. However, since α^2 and α^4 are conjugates of α, they are the roots of the same minimal polynomial given by equation (1).

$$\Phi_1(x) = \Phi_2(x) = \Phi_4(x) = X^4 + X + 1 \quad (1)$$

The minimal polynomial $\Phi_3(x)$ of α^3 is given by equation (2).

$$\Phi_3(x) = X^4 + X^3 + X^2 + X + 1 \quad (2)$$

Hence, $g(x) = \Phi_1(x)\ \Phi3(x) = X^8 + X^7 + X^6 + X^4 + 1 \quad (3)$

Perl script was used to generate the Verilog source code for the encoder. The values of *n, k, t* and *m* along with the exponents of the primitive polynomial were given as an input to the Perl script. The Perl code was written to calculate the minimal polynomials and hence obtain the generator polynomial. The generator polynomial was multiplied by the data polynomial to obtain the Verilog source code for the encoder. The code obtained is that of a combinational logic consisting of XOR gates and buffers.

This implementation has a time complexity of O(1). A simpler implementation would have required the use of a sequential multiplier with a time complexity of O(n). Thus, the delay of the encoder is significantly reduced with this implementation. The code obtained using this encoder is in the non-systematic form.

3. DECODING ALGORITHM OF BCH DECODER

The 15 - bit code word, consisting of the data bits and the check bits, stored in memory is susceptible to soft errors. Decoding is the process of detecting and correcting the errors present in the stored code word and finally, extracting the data from the error-free code word. Upon receiving the read signal, the memory starts the decoding process.

Let the erroneous code word r(x) and the error pattern e(x) be represented by equations (4) and (5) respectively.

$$r(x) = r_0 + r_1x + r_2x^2 + \ldots\ldots + r_{14}x^{14} \quad (4)$$

$$e(x) = e_0 + e_1x + e_2x^2 + \ldots\ldots + e_{14}x^{14} \quad (5)$$

Then, $r(x) = c(x) + e(x) \quad (6)$

Decoding the code involves the following steps:

a) Determine the syndrome vectors S_1, S_2 and S_3 from the 15-bit code vector read from memory.

The syndrome is a 2t-tuple for a t-error-correcting code. Thus, for a double-error-correcting code, the syndrome is a 4-tuple represented by $S = (S_1, S_2, S_3, S_4)$. The syndrome component S_i is obtained by dividing the erroneous code word r(x) by the corresponding minimal polynomial $\Phi_i(x)$ of α^i and obtaining the remainder polynomial $b_i(x)$. The syndrome component is then given by $S_i = b_i(\alpha^i)$. Syndrome computation can be implemented using simple linear feedback shift register.

b) Determine the coefficients σ_1 and σ_2 of the error polynomial from the syndrome vectors.

The error location polynomial is given by equation (7).

$$\sigma(x) = \sigma_0 + \sigma_1x + \sigma_2x^2 \quad (7)$$

The error co-efficient $\sigma_0 = 1$ and the other error coefficients σ_1 and σ_2 are determined using the

Berlekamp's iterative decoding algorithm. They are given by equations (8) and (9).

$$\sigma_1 = S_1 \quad (8)$$

$$\sigma_2 = S_3 S_1^{-1} + S_2 \quad (9)$$

c) Run an exhaustive search for the roots of the error polynomial equation.

On obtaining σ_1 and σ_2, the roots of the error location polynomial are determined by running an exhaustive search. This is done by checking if any of the 4-tuples of the Galois field from 0001 to 1111 satisfy the error polynomial equation.

d) Determine the error location positions from the inverse of the roots.

e) Obtain the error vector and thus the corrected code vector.

For example, if the 4-tuples 0101 and 1011 satisfy the equation, then the roots of the equation are α^{9} and α^{13} which are the elements of GF (2^4). The inverse of the roots are α^{15-9} and α^{15-13}. This gives the error locations as 6 and 2. The error vector can then be identified to be 000000001000100. This error vector is then added in modulo-2 addition (XOR) with the erroneous code vector to produce the correct code vector c'(x). Thus c'(x) = r(x) + e(x).

f) Extract the 7-bit data from the corrected code

The 7-bit data vector is extracted from the corrected code vector by dividing it by the generator polynomial. Thus, $d(x) = c'(x) / g(x)$. The division is carried out by using a circuit similar to the syndrome calculator circuit, in which the shift register circuit is built using the generator polynomial.

4. 3-STAGE PIPELINE ARCHITECTURE OF THE DECODER

This section presents the pipeline architecture of the decoder. In order to implement the pipeline architecture, the decoding process has been divided into 3 stages as shown below:

a) Stage 1 consists of determining the syndrome vectors from the 15-bit erroneous code vector followed by the determination of the error polynomial coefficients σ_1 and σ_2 from the syndrome vectors obtained.
b) Stage 2 consists of determining the roots of the error location polynomial, determining the error vector from the inverse of the roots and finally obtaining the corrected vector.
c) Stage 3 consists of extracting the data from the corrected code vector.

The first step in implementing the pipeline effectively was to ensure that every decoding stage involves the equal time interval. It was found that the time required in determining $\sigma_2$was large since the Galois field multiplier used in multiplying S_3 and S_1^{-1} was found consuming more time. Hence, the original circuit consisting of the shift registers was replaced by a combinational circuit, which does the same multiplication within one clock cycle. It was found that each stage then required n+1 clock cycles. Thus, for the (15, 7) code, 16 clock cycles were required. The division of time interval for each stage is shown in Table-1.

Table-1. Time division in each stage.

Decoder stage	**16 clock cycles**	
	15 clock cycles	**1 clock cycle**
Stage 1	Determination of syndrome vectors	Determination of σ_1 and σ_2 from the syndrome vectors
Stage 2	Determination of the roots of the error location polynomial	Determination of the inverse of roots and obtaining the corrected vector
Stage 3	Extraction of data	

The pipeline mechanism is as illustrated in the block diagram shown in Figure-1.

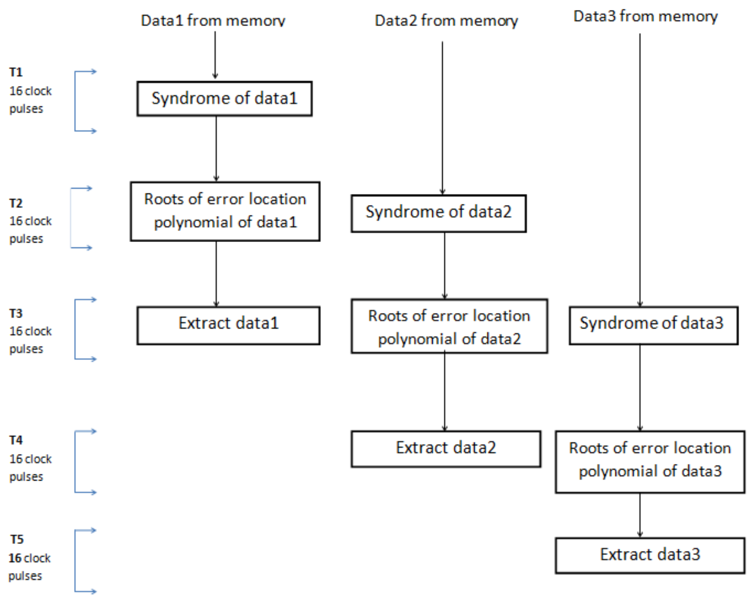


Figure-1. Block diagram illustrating the 3-stage pipeline.

5. 2-STAGE PIPELINE ARCHITECTURE OF THE DECODER

In the 3-stage pipeline architecture, the 3rd stage involved extracting the data from the corrected code word. This stage was required for the reason that the code generated by the encoder is in a non-systematic form. This stage can be eliminated, if the code generated by the encoder is made available in systematic form. In this direction, the systematic encoding of the data vector for the generator polynomial $g(x) = X^8 + X^7 + X^6 + X^4 + 1$ has been implemented using the shift register circuit depicted in Figure-2.

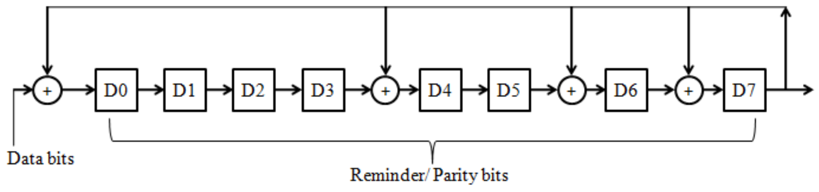


Figure-2. Encoder for the (15, 7) BCH code.

The 7 data bits are appended with 8 zeroes to its right to form a 15-bit vector. This 15-bit vector is passed through the shift register starting from the left. After 15 clock cycles, when all bits are shifted in, the contents of the shift register present the parity bits. These 8 parity bits are appended to the data bits, forming the 15-bit code vector.

This can be expressed in polynomial form by equation (10) as given by

$$c(x) = d(x)x^{n-k} + \text{Rem}[(d(x)x^{n-k}) / g(x)] \qquad (10)$$

where Rem[a(x) / b(x)] is the remainder obtained on dividing a(x) by b(x)[11]. Since the code obtained with this encoder is systematic, the last stage of the decoder shown in Figure-2 can be eliminated to obtain a 2-stage pipeline structure consisting of the following 2 stages:

a) **Stage 1 -** Determination of the syndrome vectors from the 15-bit erroneous code vector followed by the determination of the error polynomial coefficients σ_1 and σ_2 from these syndrome vectors obtained.
b) **Stage 2 -**Determination of the roots of the error location polynomial, determining the error vector from the inverse of the roots and in the process, obtaining the corrected code vector. The first 7 bits of the code vector from the left denote the data vector.

6. IMPLEMENTATION

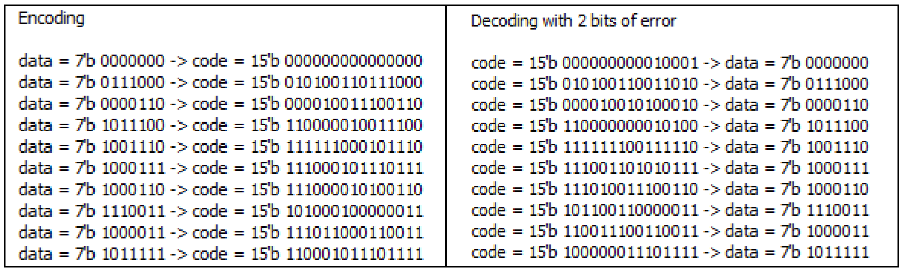
The encoding and the decoding algorithms as described in Sections 2, 3, 4 and 5 were coded in Verilog® HDL. The functional simulation of the design was carried out using Xilinx®ISim Simulator. It was tested for its correct functionality by providing various random inputs through the test benches. The architectures were synthesized using the tool Encounter from Cadence® RTL Compiler using 180nm technology libraries. Area and power estimation were done using the Encounter tool of Cadence.

7. RESULTS AND DISCUSSIONS

This section presents the results and related inferences. An authentic memory environment in the form of a memory block consisting of an array of 7-bit data has been created. To simulate the actual behavior, random data is generated using a random () function and stored in the data array. During the memory write operation, when the write signal *wr* goes high, the data vectors are encoded individually and stored in another memory block which forms the code array. The code vectors are subjected to errors by deliberately introducing error at two bit positions. The two error locations are different for each code vector. During a memory read operation, when the read signal *rd* goes high, these code vectors are passed through a decoder to obtain the corrected data vectors. The results obtained for the 3-stage and 2-stage pipeline architectures are discussed in the following sub-sections.

7.1 3-Stagespipeline architecture

A sample of the simulated output of the encoding and decoding stages of the 3-stage pipeline architecture as seen on the console window of the simulator is shown in Figure-3.



```
Encoding

data = 7'b 0000000 -> code = 15'b 000000000000000
data = 7'b 0111000 -> code = 15'b 010100110111000
data = 7'b 0000110 -> code = 15'b 000010011100110
data = 7'b 1011100 -> code = 15'b 110000010011100
data = 7'b 1001110 -> code = 15'b 111111000101110
data = 7'b 1000111 -> code = 15'b 111000101110111
data = 7'b 1000110 -> code = 15'b 111000010100110
data = 7'b 1110011 -> code = 15'b 101000100000011
data = 7'b 1000011 -> code = 15'b 111011000110011
data = 7'b 1011111 -> code = 15'b 110001011101111
```

```
Decoding with 2 bits of error

code = 15'b 000000000010001 -> data = 7'b 0000000
code = 15'b 010100110011010 -> data = 7'b 0111000
code = 15'b 000010010100010 -> data = 7'b 0000110
code = 15'b 110000000010100 -> data = 7'b 1011100
code = 15'b 111111100111110 -> data = 7'b 1001110
code = 15'b 111001101010111 -> data = 7'b 1000111
code = 15'b 111010011100110 -> data = 7'b 1000110
code = 15'b 101100110000011 -> data = 7'b 1110011
code = 15'b 110011100110011 -> data = 7'b 1000011
code = 15'b 100000011101111 -> data = 7'b 1011111
```

Figure-3. Simulation output for the 3-stage pipeline architecture.

The simulated output of the encoder for the 3-stage pipeline architecture is shown in Figure-4.

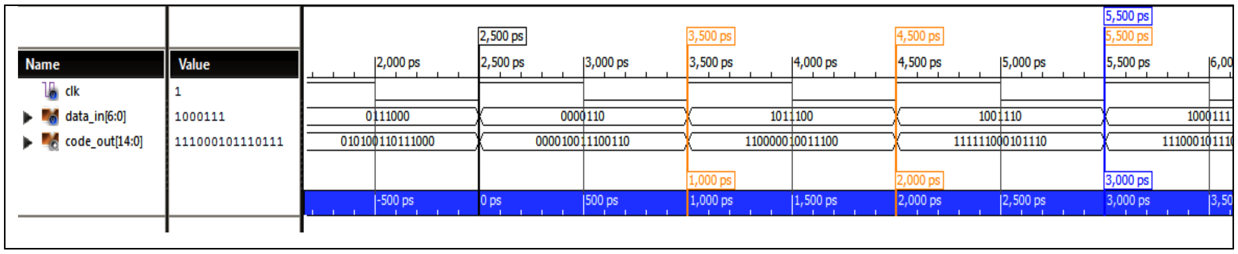


Figure-4. Simulation output of the encoder for the 3-stage pipeline architecture.

As seen from the markers in Figure-4, encoding of data occurs instantaneously, since the encoder used is a combinational circuit. Therefore, a new data is encoded during every clock cycle. The *clk* signal employed has a time period of 1ns and hence, a frequency of 1GHz.

Figure-5 shows the simulated output of the decoder for the 3-stage pipeline structure.

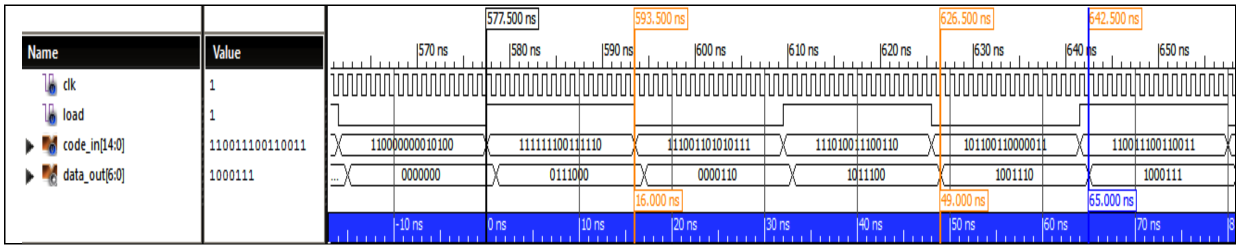
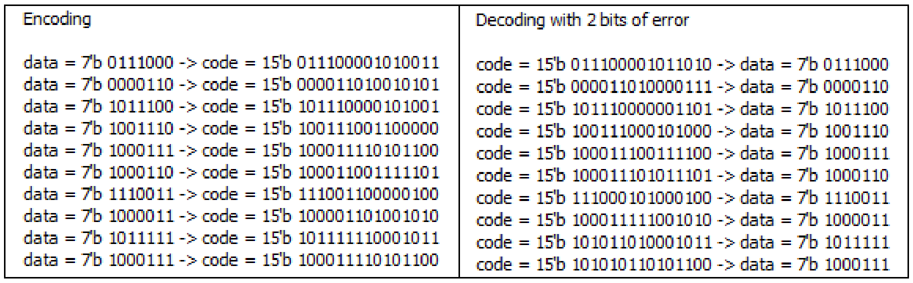


Figure-5. Simulation output of the decoder for the 3-stage pipeline architecture.

The 15-bit erroneous code is loaded into the decoder during every transition of the *load* signal. The *load* signal makes a transition once in 16 clock cycles as seen from the 1st two markers, which can be observed 16ns apart. The data corresponding to the code at the 1st marker which is at a time instant of 577.5ns is decoded and available at the output 49ns later, at a time instant of 626.5ns. Since each decoder stage requires 16 clock cycles and there are 3 stages, a total of 48 clock cycles are required for decoding and extracting data. The 49th clock cycle is required to read the data from the buffer register. Similarly, the data corresponding to the code at the 2nd marker, which is at the time instant of 593.5ns is decoded and available at a time instant of 642.5ns, 49 clock cycles after the code is loaded into the decoder. The decoded data are available at an interval of 16 clock cycles as seen from the 3rd and the 4th markers. Thus, once the pipeline is full, the data are made available at a rate of 16 clock cycles. The decoder throughput is therefore found to be 437Mb/s for a clock frequency of 1GHz. This is an increase of about 118% as compared to the non-pipeline decoding.

7.2 2-Stages pipeline architecture

A sample of the simulated output of encoding and decoding for the 2-stage pipeline architecture as seen on the console window of the simulator is shown in Figure-6.



```
Encoding

data = 7'b 0111000 -> code = 15'b 011100001010011
data = 7'b 0000110 -> code = 15'b 000011010010101
data = 7'b 1011100 -> code = 15'b 101110000101001
data = 7'b 1001110 -> code = 15'b 100111001100000
data = 7'b 1000111 -> code = 15'b 100011110101100
data = 7'b 1000110 -> code = 15'b 100011001111101
data = 7'b 1110011 -> code = 15'b 111001100000100
data = 7'b 1000011 -> code = 15'b 100001101001010
data = 7'b 1011111 -> code = 15'b 101111110001011
data = 7'b 1000111 -> code = 15'b 100011110101100
```

```
Decoding with 2 bits of error

code = 15'b 011100001011010 -> data = 7'b 0111000
code = 15'b 000011010000111 -> data = 7'b 0000110
code = 15'b 101110000001101 -> data = 7'b 1011100
code = 15'b 100111000101000 -> data = 7'b 1001110
code = 15'b 100011100111100 -> data = 7'b 1000111
code = 15'b 100011101011101 -> data = 7'b 1000110
code = 15'b 111000101000100 -> data = 7'b 1110011
code = 15'b 100011111001010 -> data = 7'b 1000011
code = 15'b 101011010001011 -> data = 7'b 1011111
code = 15'b 101010110101100 -> data = 7'b 1000111
```

Figure-6. Simulation output of the 2-stage pipeline architecture.

The simulated output of the encoder for the 2-stage pipeline architecture is shown in Figure-7.

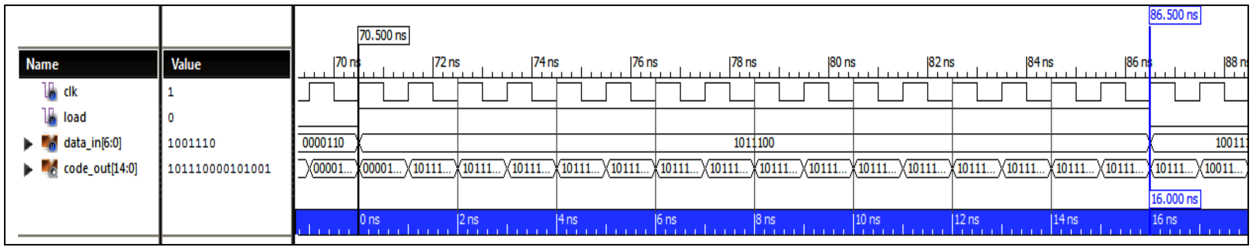


Figure-7. Simulation output of the encoder for the 2-stage pipeline architecture.

Data is loaded into the encoder at every transition of the load signal, which is once in every 16 clock cycles. As depicted in Figure-7, the data input 1011100 is loaded into the encoder at time 70.5ns, when load signal goes *high*. The corresponding code vector 101110000101001 in the systematic form is available at the output at 86.5ns, at which point of time, the load signal goes *low* and new data is loaded into the encoder. The value of the code at this time instant is seen in the '*value*' window seen on the left hand side. It can hence be concluded that encoding of data requires 16 clock cycles.

Figure-8 shows the simulated output of the decoder for the 2-stage pipeline structure.

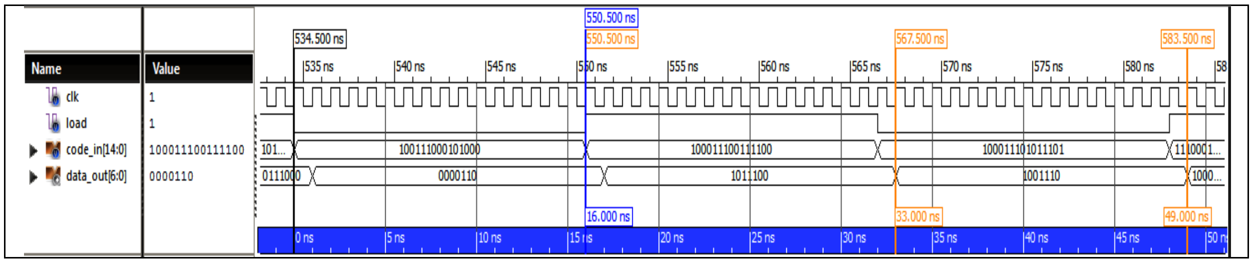


Figure-8. Simulation output of the decoder for the 2-stage pipeline architecture.

The 15-bit erroneous code is loaded into the decoder during every transition of the *load* signal which is once in 16 clock cycles as seen from the 1st two markers which are 16ns apart. The data corresponding to the code at the 1st marker which is at the time instant of 534.5ns is decoded and made available at the output 33ns later, that is, at the time instant of 567.5ns. Since each decoder stage requires 16 clock cycles and there are only 2 stages, a total of 32 clock cycles are required for the decoding and extracting data processes. The 33rd clock cycle is required to *read* the data from the buffer register. Similarly, the data corresponding to the code at the 2nd marker, which is at a time instant of 550.5ns is decoded and available at a time instant of 583.5ns, 33 clock cycles after the code is loaded into the decoder. The decoded data are available at an interval of 16 clock cycles as seen from the 3rd and the 4th markers. Thus, once the pipeline is full, data are available at the rate of 16 clock cycles.

Table-2 shows a comparison of the different architectures in terms of area, power and memory access time. The power estimation obtained is dependent on the synthesized architectures.

Table-2. Comparison of the different architectures.

Architecture	**Decoder**					
	Area μm^2	**Power** (μW)			**Decoding time of individual codes** (No. of clock cycles)	**Decoding time of a code in an array** (No. of clock cycles)
		Leakage power	**Dynamic power**	**Total power**		
2-stage pipeline	9,121	0.046	486.46	486.52	32	16
3-stage pipeline	13,588	0.07	705.40	705.47	48	16
Non-pipeline	21,871	0.10	522.15	522.25	20-120	variable

It may be noted from the Table that the decoding time is not a constant in the case of non-pipelined architecture, as it depends on the number of errors and their positions. If there is no error in the code word, the syndrome vector is zero. The decoder then skips the additional steps and extracts the data from the code vector. Hence, the decoding time becomes less. When there are errors in the code word, the decoding time can be more. On the other hand, in the case of the two pipelined architectures, though the decoding time of individual code varies, once the pipeline is full, the decoding time of each code is the same, or equal to 16 clock cycles for both the architectures. Further, the decoding time of an individual code is a constant and is independent of the number of errors and their positions.

Additionally, as seen from Table-2, the decoder area of the 2-stage pipeline architecture is found to be the least of the three architectures, which is about 33% less than the 3-stage architecture and 58% less than the non-pipeline architecture.

Furthermore, it may be noted that the power consumed by the 2-stage pipeline decoder is the least, since only two decoder stages are running concurrently at any given time. This power is found to be 31% less than the 3-stage pipeline decoder, which consumes the maximum power since three decoder stages are running concurrently at any given time.

8. CONCLUSIONS

In this paper, a novel 2-stage pipeline architecture has been proposed for the BCH decoder with the two stages being the Syndrome generation stage and the Berlekamp-Chien stage. With its low power, small area and high decoding speed the 2-stage pipeline decoder is ideally suited for high speed NOR flash memory.

Though this work is limited to double error detection and correction using (15, 7) BCH code, it may be extended for higher (n, k) values. Thus for a (31, 21) BCH code with double error correction capability, 32 clock cycles would be required to decode each code which gives a decoder throughput of 656Mb/s. For a (63,51) BCH code, 64 clock cycles would be required to decode each code to give a throughput of 796Mb/s for a clock of frequency 1GHz. Thus, higher the (n, k) values, more is the throughput.

REFERENCES

[1] R. C. Baumann. 2005. Radiation-induced soft errors in advanced semiconductor technologies. IEEE Trans. Device Mater. Reliabil. 5(3): 301-316.

[2] Sunita M.S, Kanchana Bhaaskaran V.S. 2013. Matrix Code based multiple error correction technique for n-bit memory data. Intl. Journal of VLSI Design and Communication Systems (VLSICS). 4(1):29-37.

[3] S.M. Sunita, V. S. KanchanaBhaaskaran, DeepakakumarHegde and PavanDhareshwar. 2013. Error Detection and Correction in Embedded Memories using Cyclic Code. Proceedings of International Conference on VLSI, Communication, Advanced Devices, Signals and Systems and Networking (VCASAN-2013), Bangalore, India, July, 2013, Lecture Notes in Electrical Engineering. 258: 109-116.

[4] Hsiao M.Y, Bossen D.C, Chien R.T. 1970. Orthogonal Latin Square Codes. IBM Journal of Research and Development. 14: 390-394.

[5] S. Lin and D. J. Costello. 2011. Error Control Coding. 2ndedition, Pearson Education.

[6] S. Ghosh and P. D. Lincoln. 2007. Dynamic low-density parity check codes for fault-tolerant nano-scale memory.Presented at the Foundations Nanosci. (FNANO), Snowbird, Utah.

[7] Xueqiang Wang, Guiqiang Dong, Liyang Pan and Runde Zhou. 2011. Error Correction Codes and Signal Processing in Flash Memory, Flash Memories, Prof. Igor Stievano (Ed.), ISBN: 978-953-307-272-2, InTech, DOI: 10.5772/19083.

[8] Kijun Lee, Sejin Lim, Jaehong Kim. 2012. Low-cost, low-power and high-throughput BCH decoder for NAND Flash Memory. 2012 IEEE International Symposium on Circuits and Systems (ISCAS), May 20-23, Seoul, South Korea.

[9] Y. Chen and K. K. Parhi. 2004. Area efficient parallel decoder architecture for long BCH codes. Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing. pp. V-73-V-76.

[10] Wei Liu, Junrye Rho, and Wonyong Sung. 2006. Low-Power High-Throughput BCH Error Correction VLSI Design for Multi-Level Cell NAND Flash Memories. Proc. IEEE Workshop Signal Processing Systems Design and Implementation. pp. 303-308.

[11] Xinmiao Zhang and Keshab K. Parhi. 2004. High-speed Architectures for Parallel Long BCH Encoders. GLSVLSI'04, April 26-28, Boston, Massachusetts, USA.