Second

Binary BCH

Preview

Final Document

(Customized to Meet Customer's Requirements)

Version 1.0

By

ECC Technologies, Inc. ("ECC Tek")
4750 Coventry Road East
Minnetonka, MN 55345-3909
www.ecctek.com
Phone: 952-935-2885
Fax: 952-935-2491
Email: phil.white@ecctek.com

© 2008 ECC Technologies, Inc. All Rights Reserved.

Notice

This document is a "Preview" document and does not contain material which ECC Tek considers confidential so everyone is free to copy and distribute this document without any obligations whatsoever to ECC Tek.

The Binary BCH (BBCH) designs described in this document, the copyright to this document, the copyright to the C source code and Verilog source code which simulate and implement the BBCH encoder and decoder designs, and US Patent Number 5,754,563 entitled "*Byte-Parallel System For Implementing Reed-Solomon Error-Correcting Codes*" (the "PRS Patent") protecting these designs are owned by

ECC Technologies, Inc. (ECC Tek)
4750 Coventry Road East
Minnetonka, MN 55345-3909

Phone: 952-935-2885
Fax: 952-935-2491
E-mail: phil.white@ecctek.com
Website: www.ecctek.com.

The BBCH encoder and decoder designs described in this document must be licensed from ECC Tek in order to legally implement them.

Revision History

02-06-07	pew	Created initial version of this document based on previous documents and drawings.
02-26-07	pew	Edited and added material.
02-27-07	pew	Edited and added material.
04-04-07	pew	Updated the EVAL block diagram.
04-05-07	pew	Edited the material.
04-06-07	pew	Updated the SYNB block diagram.
12-12-07	pew	Added Encoder and Decoder Signal Definition Tables
12-21-07	pew	Added a column to the Signal Definition Tables to show what the original signal names are.
02-16-08	pew	Corrected the INIT block diagram.
03-21-08	pew	Changed some of the heading levels.
03-21-08	pew	Corrected the K values selectable.
03-21-08	pew	Added material about when DECODERE will and will not work well.
03-23-08	pew	Added first two sections.
09-11-08	pew	Edited the document and released a preliminary version.
09-12-08	pew	Edited the document. Corrected grammar.
09-16-08	pew	Edited the document.
09-16-08	pew	Created Preview Final Document.

© 2008 ECC Technologies, Inc. All Rights Reserved.

Sections

1. Overview of Second Binary BCH Encoder and Decoder Designs 8
2. Issues Involved in Licensing Synthesizable Verilog Code 9
3. Different Ways to Synthesize the Verilog Code 9
4. (t, K) Options 9
5. Generator Polynomials 11
6. Versions of the C Code and Verilog Code 11
7. What This Document Does 12
8. What This Document Does Not Do 12
9. References 13
10. Notation 13
11. C Code 18
  11.1. C Functions for Generating Verilog Input Data and Expected Results Files 20
  11.2. C Functions for Testing the Encoder, Decoder and System 20
12. Introduction to the Hardware Designs Described in Verilog 22
13. Verilog Coding Style and Signal Naming Conventions 25
14. General Hardware Design Development Methodology 26
15. Basic Logic Design Concepts Used by ECC Tek 26
16. Drawing Conventions 27
17. Implementing Finite Field Operations in Digital Logic 28
18. Encoder Versions 30
19. Encoder Signal Definitions 32
20. Decoder Latency 34
21. Decoder Versions 35
  21.1. DECODERB1 35
  21.2. DECODERB2 37
  21.3. DECODERB3 38
  21.4. DECODERC1 39
  21.5. DECODERC2 40
22. Decoder Clocking Options 41
  22.1. Decoder Clocks 41
  22.2. Gating the Clocks 41
23. Decoder Signal Definitions 41
24. Syndrome Generator Mathematics 43
25. Syndrome Generator Circuits 43
  25.1. SYNB1 43
  25.2. SYN1D1 44
  25.3. SYN2C1 44
26. DFIFO 44
27. PPU 45
  27.1. PPU Versions 47
    27.1.1. PPUC1 47
    27.1.2. PPUF1 48
28. INIT 48
29. EVAL 48

© 2008 ECC Technologies, Inc. All Rights Reserved.

30. XOR 48

Figures

Figure 1 Registers 23
Figure 2 Multiplexers (Muxes) 23
Figure 3 Generic State Machine 24
Figure 4 Generic Control Unit with a State Machine 25
Figure 5 General Form for all Logic Blocks 27
Figure 6 Full Multiplier (top) and Partial Multiplier (bottom) 29
Figure 7 Multiplier that Multiplies the Variable “A” by the Constant “B” 30
Figure 8 ENCODERA1 31
Figure 9 Encoder Mathematics 31
Figure 10 Iterative Serial Encoder Circuit 32
Figure 11 Parallelized Encoder Circuit 32
Figure 12 DECODERB1 36
Figure 13 DECODERB2 37
Figure 14 DECODERB3 38
Figure 15 DECODERC1 39
Figure 16 DECODERC2 40
Figure 17 Syndrome Generator Mathematics 43
Figure 18 Dividing the Received Word Polynomial by g(x) to find the Remainder 43
Figure 19 Evaluating the Remainder at $\alpha^1, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$ 43
Figure 20 SYNB1 44
Figure 21 Iterative Serial Circuit for SYN1 44
Figure 22 SYN1D1 44
Figure 23 SYN2C1 44
Figure 24 Final EBM Algorithm (V5) 45
Figure 25 P=2 PPU Top Level Block Diagram 46
Figure 26 P=2 PPU L Section 46
Figure 27 P=2 PPU Branch Section 46
Figure 28 P=2 PPU V Section 46

Tables

Table 1 Selectable Values for t and r before Adjustment 8
Table 2 Selectable Values for t and R 8
Table 3 (t, K) Options 10
Table 4 Comparison of Notation used by ECC Tek, Eastman and Berlekamp 14
Table 5 Definition of Symbols, Abbreviations and Terms 15
Table 6 C Functions Included in the C Code and What They Do 21
Table 7 Encoder Signal Definitions 33
Table 8 Decoder Signal Definitions 42

© 2008 ECC Technologies, Inc. All Rights Reserved.

1. Overview of Second Binary BCH Encoder and Decoder Designs

This document describes digital logic designs for encoding and decoding Binary BCH (BBCH) codes which can correct t = 16, 24, 25, 28, 29, or 30 bits in error for use with NAND Flash memory chips with data field sizes of from K = 1024, 1025, …, 1053 bytes. The encoder adds from 28 to 53 bytes of redundancy onto each data field. The decoder handles 81 (t, K) options where the total number of bytes in a codeword, N, is limited to 1081 or less as shown in Table 3. The limitation on N could easily be changed if needed.

The BCH codes being used are binary and usually encoders for binary codes input and output 1 bit at a time and syndrome calculators input 1 bit at a time. For this design the encoder and decoder input and output 8 bits at a time and appear, from the outside, to be a Reed-Solomon (RS) encoder and decoder operating on 8-bit symbols.

Capital letters in the following Tables are variables measured in bytes and lower case letters are variables measured in bits.

Table 1 shows selectable values for t and the number of redundant bits, r, needed before adjustment is made to make codewords an even number of bytes.

t (bits)	16	24	25	28	29	30
r (bits)	224	336	350	392	406	420

Table 1 Selectable Values for t and r before Adjustment

The number of bits of redundancy, r, added to a message for each value of t as shown in Table 1 was adjusted by multiplying the binary BCH generator polynomials by irreducible binary polynomials of low degree so that the adjusted amount of redundancy is a multiple of 8 and therefore is an even number of bytes which simplifies the design of the encoder and decoder. Selectable values t and R are shown in Table 2.

t (bits)	16	24	25	28	29	30
R (bytes)	28	42	44	49	51	53

Table 2 Selectable Values for t and R

© 2008 ECC Technologies, Inc. All Rights Reserved.

2. Issues Involved in Licensing Synthesizable Verilog Code

ECC Tek is in the business of licensing synthesizable Verilog code which describes the design of ECC encoders and decoders at a very high level. Licensing synthesizable Verilog code presents a unique set of challenges.

Since our customers do synthesis, ECC Tek has no control over what synthesis tools are used or how they are used. ECC Tek does not know what its customer's synthesis tools will do in optimizing logic or if one customer’s tools will do the same things or different things than another customer’s tools.

For example, suppose ECC Tek creates a number of Verilog modules so that the output of one module feeds the input of another. Assume we have "upstream" and "downstream" modules. If a downstream module forces a signal to a constant value, will the synthesis software, in all cases, be intelligent enough to know that certain upstream logic can be deleted because it is unneeded? ECC Tek has no way of knowing the answer to that question.

Another example is a Galois Field multiplier. If ECC Tek designs a GF multiplier to multiply two variables, will all of the synthesis software tools reduce and minimize the logic needed if one of the inputs is a constant or should ECC Tek design specific multipliers to multiply by specific constants? ECC Tek has no way of knowing the answer to this question either. Most likely, most synthesis software will properly reduce the logic, but, because of that uncertainty, ECC Tek has created many different multipliers that multiply by a constant such as the ones delivered to Customer called MULT_BY_XXXX.

Gate counts and chip surface area required by these designs depend upon the circuit technology each licensee uses and the synthesis software they use.

3. Different Ways to Synthesize the Verilog Code

The programmable binary BCH encoder and decoder Verilog code that can be synthesized with one set of fixed (t, K) values so that specific circuits will be synthesized for that one case.

The Verilog code can also be synthesized into programmable or configurable binary BCH encoder and decoder circuits and the sel_t and delta_k inputs can be used to configure the encoder and decoder for one (t, K) set of values.

4. (t, K) Options

There are 81 possible (t, K) options as shown in Table 3 on the following page.

© 2008 ECC Technologies, Inc. All Rights Reserved.

Table 3 (t, K) Options

t (bits)	**16**	**24**	**25**	**28**	**29**	**30**	
R (bytes)	**28**	**42**	**44**	**49**	**51**	**53**	
N (bytes)	1052	1066	1068	1073	1075	1077	K=1024 (bytes)
N (bytes)	1053	1067	1069	1074	1076	1078	K=1025 (bytes)
N (bytes)	1054	1068	1070	1075	1077	1079	K=1026 (bytes)
N (bytes)	1055	1069	1071	1076	1078	1080	K=1027 (bytes)
N (bytes)	1056	1070	1072	1077	1079	1081	K=1028 (bytes)
N (bytes)	1057	1071	1073	1078	1080	-	K=1029 (bytes)
N (bytes)	1058	1072	1074	1079	1081	-	K=1030 (bytes)
N (bytes)	1059	1073	1075	1080	-	-	K=1031 (bytes)
N (bytes)	1060	1074	1076	1081	-	-	K=1032 (bytes)
N (bytes)	1061	1075	1077	-	-	-	K=1033 (bytes)
N (bytes)	1062	1076	1078	-	-	-	K=1034 (bytes)
N (bytes)	1063	1077	1079	-	-	-	K=1035 (bytes)
N (bytes)	1064	1078	1080	-	-	-	K=1036 (bytes)
N (bytes)	1065	1079	1081	-	-	-	K=1037 (bytes)
N (bytes)	1066	1080	-	-	-	-	K=1038 (bytes)
N (bytes)	1067	1081	-	-	-	-	K=1039 (bytes)
N (bytes)	1068	-	-	-	-	-	K=1040 (bytes)
N (bytes)	1069	-	-	-	-	-	K=1041 (bytes)
N (bytes)	1070	-	-	-	-	-	K=1042 (bytes)
N (bytes)	1071	-	-	-	-	-	K=1043 (bytes)
N (bytes)	1072	-	-	-	-	-	K=1044 (bytes)
N (bytes)	1073	-	-	-	-	-	K=1045 (bytes)
N (bytes)	1074	-	-	-	-	-	K=1046 (bytes)
N (bytes)	1075	-	-	-	-	-	K=1047 (bytes)
N (bytes)	1076	-	-	-	-	-	K=1048 (bytes)
N (bytes)	1077	-	-	-	-	-	K=1049 (bytes)
N (bytes)	1078	-	-	-	-	-	K=1050 (bytes)
N (bytes)	1079	-	-	-	-	-	K=1051 (bytes)
N (bytes)	1080	-	-	-	-	-	K=1052 (bytes)
N (bytes)	1081	-	-	-	-	-	K=1053 (bytes)

© 2008 ECC Technologies, Inc. All Rights Reserved.

5. Generator Polynomials

Generator polynomials used after adjustment are shown in the C code.

Generator polynomials for binary BCH codes are products of Minimal Polynomials, Mi(x), whose roots are (α^i), $(\alpha^i)^2$, $(\alpha^i)^4$, $(\alpha^i)^{16}$, $(\alpha^i)^{32}$, ... , $(\alpha^i)^n$ where $n = 2^j$ for j=0,1,2 …J. This sequence of powers of (α^i) will repeat for some J and for that J, $(\alpha^i)^n = 1$. Sometimes this sequence will repeat sooner and the degree of g(x) will be less than wt, but in this case the degrees of g(x) are always wt where w is the width of the finite field elements used to locate the position of each bit. In this case w=14 and the degrees of g(x) before adjustment are 14t as can be seen in Table 1.

The irreducible binary polynomial used to generate a finite field with 2^{14} elements is

p(x) = 44443 octal = 100 100 100 100 011 = $x^{14} + x^{11} + x^8 + x^5 + x + 1$.

Any primitive element can be used as α. However, it is conventional to use "x" or 0002 hex as the primitive element, so that is what is used in the BBCH designs which is the same as for the RS designs.

In order for an error-correcting code to correct t errors, 2t consecutive powers of α must be roots of g(x). For binary BCH codes to correct t errors, the minimal polynomials M1, M3, …, M2t-1 must be factors of g(x). That is, g(x) = M1*M3*M5*…*M2t-1. For example, to correct 30 bits, M1, M3, …, M60 must be factors of g(x).

Minimal polynomials Mi were computed using a C program by multiplying factors $(x-(\alpha^i))*(x-(\alpha^i)^2) \ldots *(x-(\alpha^i)^n)$.

The coefficients of Minimal Polynomials are binary. That is, all of the coefficients are either 0 or 1 so that g(x) is also binary and all of the coefficients of g(x) are also either 0 or 1 which necessarily must be the case for binary codes.

Much of the decoder logic operates on 14-bit finite field elements internally, but data is input and output as 8-bit bytes.

6. Versions of the C Code and Verilog Code

The simulation and testing C code and the synthesizable Verilog code have been developed in versions.

Versions of the C software have suffixes V1, V2, V3, etc. The names of the C file code files are Customer binary bBCH V2 All cases V1.c, Customer binary BCH V2 All cases V2.c, etc.

Some of the Verilog modules have module names with suffixes of 1, 2, 3, etc., to distinguish the second version of the BBCH Verilog modules from the first version of the BBCH modules which use suffixes of A, B, C, etc. For example, the first version of SYN1A is called SYN1A1.

© 2008 ECC Technologies, Inc. All Rights Reserved.

Verilog modules used for multiplying two finite field elements have the same names as those modules for the first BCH designs.

7. What This Document Does

This document describes BBCH encoder and decoder versions which correct all error patterns with t or fewer bits in error and is intended to help the reader understand the BBCH encoder and decoder hardware designs which are described in synthesizable Verilog and the C modeling software so that the designs can be easily modified and maintained.

8. What This Document Does Not Do

This document does not often repeat data that can be found in the C or Verilog code. For example, readers are referred to the C code to view the generator polynomials rather than repeating all of them in this document.

This document is not intended to be a substitute for a textbook on coding theory. It takes a lot of time, effort and study to understand algebraic coding theory. There are many good books on coding theory which should be consulted to gain an in-depth understanding of the math and theory associated with Binary BCH codes. It would be impossible for this document to cover all of the topics that a good book on coding theory covers so no attempt is made to do that.

This document does not provide mathematical proofs of mathematical results that have already been proven in textbooks or papers on coding theory. However, the C and Verilog models of the BBCH encoder and decoder designs can be used to verify that the BBCH system does, in fact, correctly correct all correctable error patterns. The C and Verilog models can also be used to determine the frequency of miscorrection.

This document presents and uses a version of the well-known Berlekamp-Massey ("BM") algorithm to solve the key decoding equation without presenting a proof of its validity. The BM algorithm has been proven to be valid in various textbooks and papers on coding theory. A version of the BM algorithm developed by Willard Eastman is used by ECC Tek in all of ECC Tek's designs and is referred to as the "EBM" algorithm to stand for Eastman's version of the Berlekamp-Massey algorithm.

The term "key equation" is more correctly called the "key congruency" but the term "key equation" is used in most of the textbooks on coding theory so it will be used in this document also. For binary BCH codes, the key equation is $S(x)L(x) \equiv V(x) \bmod x^{2t}$ or, in words, S(x) times L(x) is congruent to V(x) modulo x^{2t} where S(x) is the syndrome polynomial, L(x) is the error locator polynomial, V(x) is the error evaluator polynomial and t is the number of bits the code can correct.

When operations are performed on polynomials modulo x^{14}, all of the terms resulting from a polynomial operation, such as multiplication of two polynomials, of the form $c_i x^i$ are 0 if i is greater than or equal to 14 because x^{14} is a factor of all terms of the form $c_i x^i$ when i is greater than or equal to 14 and x^{14} is congruent to 0.

© 2008 ECC Technologies, Inc. All Rights Reserved.

It is well-known by coding theorists and stated in some textbooks on coding theory that the integer variable used in the EBM algorithm which ECC Tek refers to as "degLx" will be the actual degree of L(x) when decoding is successful. This fact is used to determine when the decoder has failed, but a mathematical proof of this fact not provided.

It has been proven in many textbooks that the degree of V(x) must be strictly less than the degree of L(x) when decoding is successful, and, if L(x) = 1, then V(x) must be 0. These facts are used to simplify the decoder design but, again, are not mathematically proven.

It has been proven that calculating the syndrome in one step by evaluating the received polynomial at α, α^2, α^3, …, α^{2t} is equivalent to calculating the syndrome in two steps where the first step is to find the remainder of dividing r(x) by g(x) and the second step is evaluating the remainder at α, α^2, α^3, …, α^{2t}, but this fact is not proven in this document.

This document does not provide a detailed description of the C code because most of the C code is self-explanatory and also because the primary purpose of this document is to describe the hardware design – not the C software.

9. References

This document refers to other documents which are listed below:

1. Customer's "*BBCH Design Requirements Document*" referred to in this document as "**Customer's Requirements Document**"

2. A paper entitled *"Euclideanization of the Berlekamp-Massey Algorithm"* by Willard L. Eastman from Proceedings of the 1988 Tactical Communications Conference, Vol. 1 (1988) pp. 295-303. Referred to in this document as **"Eastman's Paper"**.

3. United States Patent Number 5,754,563 entitled *"Byte-Parallel System for Implementing Binary BCH Error-correcting Codes"*, May 19, 1998. Referred to in this document as **"the PRS Patent"**.

4. Reuse Methodology Manual, Third printing 1999. Referred to in this document as "**the RMM**".

5. The book entitled *"Algebraic Coding Theory"*, written in 1968 by Elwyn Berlekamp. Referred to in this document as **"Berlekamp's book"**.

If there is a conflict or contradiction between what this documents states, indicates or implies and what the Verilog code is, the Verilog code should be taken to be the final authority.

10. Notation

To simplify the notation used in this document, in the C code and in the Verilog code, English letters are used as symbols instead of Greek letters as much as possible.

© 2008 ECC Technologies, Inc. All Rights Reserved.

A summary of the notation used by ECC Tek, Eastman and Berlekamp for the polynomials involved in solving the key decoding equation is given in Table 4.

Table 4 Comparison of Notation used by ECC Tek, Eastman and Berlekamp

	ECC Tek's Notation	Eastman's Notation (and Notation in the PRS Patent)	Berlekamp's Notation
Error Locator Polynomial	L(x)	$b^T(x)$, $\lambda(x)$ and $\Pi(x)$	$\sigma(z)$
Auxiliary Locator Polynomial	aL(x)	$b^O(x)$	$\tau(z)$
Error Evaluator Polynomial	V(x)	$p^T(x)$	$\omega(z)$
Auxiliary Evaluator Polynomial	aV(x)	$p^O(x)$	$\gamma(z)$
Last L(x) for Software Program	last_L(x)	$b^T(x)_{j-1}$	NA
Last V(x) for Software Program	last_V(x)	$p^T(x)_{j-1}$	NA
Number of Erasures	ne	μ	NA
Erasure Location i	eli	Wi	Xi
Current Discrepancy	dj	dj	$\Delta_1^{(k)}$
Last Nonzero Discrepancy	lnd	δ	NA
Number of Erasures Index	nei	λ	NA
Degree L(x) Variable	degLx	l	D(k)

© 2008 ECC Technologies, Inc. All Rights Reserved.

Table 5 defines symbols, abbreviations and terms used in this document, in ECC Tek's C and Verilog code, in Eastman's paper, in Berlekamp's book and in other papers and books on coding theory.

Table 5 Definition of Symbols, Abbreviations and Terms

aL(x)	In ECC Tek's notation, the auxiliary error Locator polynomial.
aV(x)	In ECC Tek's notation, auxiliary error eValuator polynomial.
BM	Berlekamp-Massey – Elwyn Berlekamp and James Massey who are credited with discovering or inventing an efficient algorithm for solving the key decoding equation for BBCH codes. There are numerous versions of the BM algorithm. The version presented in Eastman's paper is used in this document and denoted as the EBM algorithm.
c[i]	In ECC Tek's C code, the codeword symbol in position i.
ci	In ECC Tek's Verilog code, the codeword symbol in position i.
dj	The current discrepancy. In this document, the current discrepancy = Vj where j is the EBM algorithm iteration number. If the current L(x) being calculated in the decoder is the correct L(x), the discrepancy will be 0 from this point until the end of the EBM algorithm is reached. Sometimes dj is denoted as just d.
degLx	The degree of L(x). The variable degLx is the maximum degree – or an upper bound on the degree of L(x) – while L(x) is being formed from S(x) using the EBM algorithm. Once L(x) has been formed from S(x), the degLx variable is the actual degree of L(x). The degLx variable is used by the final stages of the decoder to determine if the error pattern calculated by the decoder is legitimate.
el[i]	In the C code, erasure location i.
eli	In the Verilog code, erasure location i.
g(x)	The generator polynomial for an algebraic block code such as a BBCH code. $g(x) = g_0 + g_1x+g_2x^2+\ldots+g_Rx^R$ where R is the number of redundant symbols. g_R is always 1.
gLx[]	In the C code, a global variable for storing the coefficients of L(x).

© 2008 ECC Technologies, Inc. All Rights Reserved.

K	The number of symbols in each message or data field.
L[i]	In the C code, the coefficient of L(x) in position i.
Li	In the Verilog code, the coefficient of L(x) in position i.
Li-1	In the Verilog code, the coefficient of L(x) in position (i-1).
lnd	In the Verilog and C code, the last nonzero discrepancy calculated while executing the BM algorithm. The lnd variable is initialized to 1.
Lp[i]	In the C code, the coefficient in position i of the polynomial which is the formal derivative of the error locator polynomial. The formal derivative of L(x) is represented as L'(x) or Lpx or Lp(x).
Lpx_eval	The evaluation of Lp(x) at some specific value for x. An evaluation is the value obtained by replacing the indeterminate "x" in a polynomial with an actual finite field element and evaluating the resulting expression. The evaluation Lp(x) at "α^{-i}" is denoted as Lpx_eval or Lp(α^{-i}).
Lpxi	In the Verilog code, the coefficient in the polynomial which is the formal derivative of the error locator polynomial which is in the Th position.
Lx_eval	The evaluation of L(x) at some specific value for x. An evaluation is the value obtained by replacing the indeterminate "x" in a polynomial with an actual finite field element and evaluating the resulting expression. The evaluation L(x) at "α^{-i}" is denoted as Lx_eval or L(α^{-i}).
mlci	most likely codeword symbol in the ith position in the Verilog code.
mle[i]	In the C code, the most likely error symbol (symbol) in position i.
mlei	In the Verilog code, the most likely error pattern symbol in position i.
mlmi	In the Verilog code, the most likely message or data symbol i.
N	The number of symbols (symbols) in a codeword.
nei	The "number of erasures initiated" index.

© 2008 ECC Technologies, Inc. All Rights Reserved.

R	The number of redundant symbols in a codeword.
r[i]	In the C code, the received symbol in position i.
ri	In the Verilog code, the received symbol in position i.
RS	Binary BCH – Dr. Irving Reed and Dr. Gustave Solomon who are credited with discovering rs codes in 1960.
s	The number of erasures the code can correct.
S(x)	The syndrome polynomial.
S[i]	In the C code, the coefficient of S(x) in position i.
Si	In the Verilog code, the coefficient of S(x) in position i.
t	In most books on coding theory, the number of errors a block code can correct.
Vi	In the Verilog code, the coefficient of V(x) in position i.
Vi-1	In the Verilog code, the coefficient of V(x) in position (i-1).
Vx_eval	The evaluation of V(x) at some specific value for x. An evaluation is the value obtained by replacing the indeterminate "x" in a polynomial with an actual finite field element and evaluating the resulting expression. The evaluation V(x) at "α^{-i}" is denoted as Vx_eval or $V(\alpha^{-i})$.
Wj	In Eastman's paper and in some versions of the C code, the jth erasure location. The W can be thought of as standing for the W in "Where".
Π(x)	In Eastman's paper, the error locator polynomial. Π probably stands for "position".
Ω(x)	In Eastman's paper, the error evaluator or error magnitude polynomial. Omega – where "m" stands for magnitude.
α	A primitive element of a finite field. In this document $\alpha = 2$ which is usually one of many primitive elements in a finite field.

© 2008 ECC Technologies, Inc. All Rights Reserved.

γ(z)	In Berlekamp's book, the auxiliary error evaluator or auxiliary error magnitude polynomial which ECC Tek refers to as aV(x). Berlekamp uses the indeterminate "z" instead of "x" for polynomials with nonbinary coefficients I believe so that the reader can quickly distinguish between binary and nonbinary polynomials.
μ	In Eastman's paper, the number of erasures.
τ(z)	In Berlekamp's book, the auxiliary error locator polynomial which ECC Tek refers to as aL(x). Berlekamp uses the indeterminate "z" instead of "x" for polynomials with nonbinary coefficients I believe so that the reader can quickly distinguish binary and nonbinary polynomials.
ω(z)	In Berlekamp's book, the error evaluator or error magnitude polynomial which ECC Tek refers to as V(x). Berlekamp uses the indeterminate "z" instead of "x" for polynomials with nonbinary coefficients I believe so that the reader can quickly distinguish between binary and nonbinary polynomials.

11. C Code

The C code is used to test the encoder and decoder algorithms and to generate input data and expected results files for debugging the Verilog code. The C code can be used to see if the decoder correctly corrects correctable error patterns and to determine the probability of miscorrection when the number of errors in a received word exceeds the capability of the code to correct.

The WriteTestbenchFiles() function in the C program writes testbench files. The other functions in the C program are used to test the encoder and decoder.

After being compiled with a standard ANSI C compiler, the C program should run on any PC or Mac with no alteration.

The C program is very basic. Printf() statements can be inserted at various points to view variables as the program executes.

What coding theorists call "symbols" are one short integer in the C code. For binary BCH codes, the symbols are "bits" and therefore each bit is a separate short integer variable in the C code. The bits are not packed into words so each bit is in a separate word.

A 14-bit finite field element is used to identify each bit position in the binary BCH codewords. These 14-bit quantities are also one short integer in the C code.

Many of the variables in the C program are global variables which simplifies modification of the functions. The message array, m[i], the codeword array, c[i], and the most likely error and codeword arrays, mle[i] and mlc[i], are some of the primary global variables. Normally global variables are undesirable in software, but, in this case, they are helpful because different functions can operate on the same data without having to pass arguments from function to function. Different encoding, decoding and testing functions can be selected by calls from main() to perform various operations. Each possible encoding function assigns values to the global codeword array, c[i]. Errors can be added to the codeword c[i] array to create the received word r[i] array. Then any selected decoding function can be used to decode the r[i] array and generate the most likely error and codeword arrays, mle[i] and mlc[i].

The primary functions in the C program are the Encode() and Decode() functions. Other testing and multiplication functions allow the Encode() and Decode() functions to be tested individually or tied together and tested as a system.

There are three read-only tables in the programs. Each of the tables is initialized by main(). Although algorithms could be used instead of tables to find reciprocals and do multiplications, tables are much faster and allow many more error patterns to be tested in the same amount of time than what could be tested if algorithms were used.

Since the binary BCH codes described in this document use 14-bit bit location symbols, a multiplication table would be inordinately large so log and antilog tables are used for multiplying.

The first table created by the C code is a list of finite field reciprocals. When a reciprocal of a finite field element is needed, this table is accessed.

The second table is a table of powers of a primitive element, α^i, where i is the index or address of a one-dimensional array and α^i is the value stored in the table or array at that address. This table is used by the decoder to quickly determine α^i given i where i = 0x0000, 0x0001, ... , 0x3fff.

The third table is a table of negative powers of a primitive element, α^{-i}, where i is the index or address of a one-dimensional array and α^{-i} is the value stored at that address or location. This table is used by the decoder to quickly determine α^{-i} given i. Received word and codeword locations are labeled as α^{-i} where i runs from 0x0000 to 0x03fff.

The Decode() function determines the most likely error word, mle[i], and the most likely codeword, mlc[i], for a particular received word, r[i].

When the Decode() function detects an abnormal condition, it sets the decode_Fail flag to TRUE (or 1) indicating that the decoding operation has failed. The decode Fail flag is used to detect error patterns that exceed the capability of the decoder to correct.

Some error patterns that exceed the correction capability of the code will either not be detected or will be miscorrected. There is no way around this fact. All decoders for algebraic block codes have this property. If a severe error pattern happens to also be a codeword, no decoder can detect it or correctly correct the received word because the error pattern will map one legitimate codeword into another legitimate codeword and the syndrome will be all zeros indicating no errors.

© 2008 ECC Technologies, Inc. All Rights Reserved. September 16, 2008

The probability of miscorrection can be measured and quantified by using the C testing code. Error patterns can be generated that exceed the correction capability of the code and then the Decode() function can be executed to determine the frequency of miscorrection.

11.1. C Functions for Generating Verilog Input Data and Expected Results Files

The WriteTestbenchFiles() function generates Verilog input data and expected results files for use with testbenches.

Actual results generated by running Verilog simulations are compared with the contents of the expected results files generated by the WriteTestbenchFiles() function. If the Verilog simulation results are correct, they will match the results generated by the WriteTestbenchFiles() function.

11.2. C Functions for Testing the Encoder, Decoder and System

The testing functions are useful for verifying that the decoder can correctly correct all correctable error patterns and for determining how frequently the decoder will declare a decoding failure when error patterns exceed the correction capability of the code.

There is an error pattern generator in the C code that can be used to generate error patterns with any number of errors.

Theoretically, it is possible to exhaustively test most of the decoder by assuming that an all-zero message has been encoded into an all-zero codeword because, if the syndrome generator is working correctly, the syndrome does not depend upon what message or codeword has been sent. The syndrome only depends upon the error pattern so, once it has been determined that the syndrome generator is working correctly, the remainder of the decoder can be tested by assuming an all-zero codeword was sent. Most of the C functions that generate error patterns use the error pattern generated to be the received word and assume that an all-zero codeword was sent. If the user does not feel comfortable assuming an all-zero codeword was sent, then random messages can be generated, encoded into codewords and the error pattern can be added to the codeword to form the received word.

The C functions included in the C program are listed in Table 6 with a brief description of what they do.

© 2008 ECC Technologies, Inc. All Rights Reserved.

Table 6 C Functions Included in the C Code and What They Do

Decode()	Determines the most likely code word, mlc[], for a particular received word, r[].
Encode()	Generates a code word, c[], from a message, m[].
Build_Log_Table()	Builds log to the base α table.
Build_Antilog_Table()	Builds antilog table.
Initialize_alphaToThe()	Initializes the α^i table.
Initialize_alphaToTheMinus()	Initializes the α^{-i} table.
Initialize_Mult()	Initializes the finite field multiplication table.
Initialize_Recip()	Initializes the table of reciprocals.
Manually_Test_System()	Manually tests the encoder and decoder system.
Multiply()	Multiplies a_of_x by b_of_x to form c_of_x. a_of_x, b_of_x and c_of_x are global variables.
Test_Encoder()	Tests the encoder.
TestR_Decode()	Test decoder's ability to correct equally probable random error patterns.
TestRn_Decode(i)	Test decoder's ability to correct i bit random error patterns.
WriteTestbenchFiles()	Writes Verilog input data and expected results testbench files.

It is not possible to exhaustive test decoders which correct many symbols/bits when codewords are long because there are too many possible error patterns.

The purpose of the TestR() functions is to determine the probability of miscorrection when the error pattern contains a large number of random errors.

© 2008 ECC Technologies, Inc. All Rights Reserved.

12. Introduction to the Hardware Designs Described in Verilog

ECC Tek's hardware design strategy is to use only a very small number of basic Verilog language constructs as building blocks in developing BBCH encoder and decoder designs to ensure that the designs will be synthesizable, will achieve a high level of performance, will be easy to understand and maintain and can be quickly developed.

The basic Verilog constructs used by ECC Tek are as follows:

- Constructs that ECC Tek knows will synthesize into Registers
- Constructs that ECC Tek knows will synthesize into Multiplexers
- Constructs that ECC Tek knows will synthesize into State Machines
- Constructs that ECC Tek knows will synthesize into Simple Combinatorial Logic

ECC Tek does not view Verilog as a programming language such as C, but as a way to easily simulate and synthesize circuits that have been previously laid out in block diagrams as shown in the Figures of this document.

The way ECC Tek designs digital logic is by first creating block diagrams (pictures) of the data flow required for a particular design to achieve a desired level of performance. For example, most of the block diagrams shown in the Figures in this document were created before any of the Verilog code was written, and the resulting performance of the encoder and decoder was predicted based upon previous experience in designing encoders and decoders.

Writing and debugging the Verilog code was the last step in the design process.

By adhering to this discipline, ECC Tek is able to quickly create and debug designs that can be customized to each customer's unique requirements.

It is ECC Tek's belief that, if ECC Tek did not develop designs using only a few well-proven constructs, then the time to finish a design and the risk of failure would be much higher than it is using this design methodology.

In addition to the benefits of reducing development time and development risk, this design strategy also results in designs that are easy-to-understand, easy-to-modify and easy-to-maintain by the customer.

The symbol used in the drawings for a register, R, the associated Verilog code and the circuit ECC Tek knows will be synthesized from the Verilog code are illustrated in Figure 1.

© 2008 ECC Technologies, Inc. All Rights Reserved.

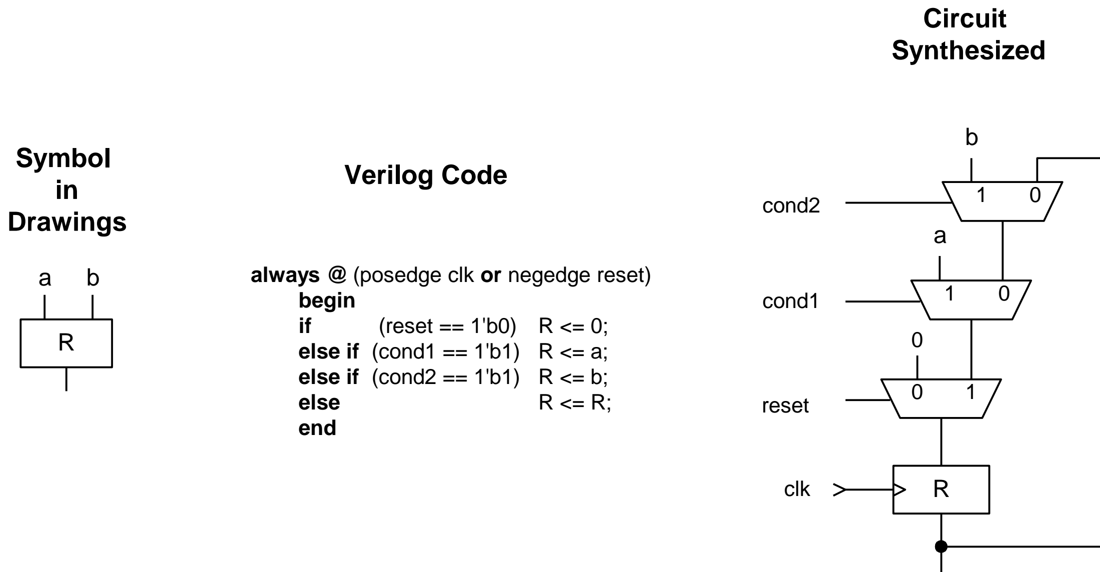

Figure 1 Registers

The symbol used in the drawings to represent a multiplexer (mux), the associated Verilog code and the circuit ECC Tek knows the Verilog code will be synthesized into are shown in Figure 2.

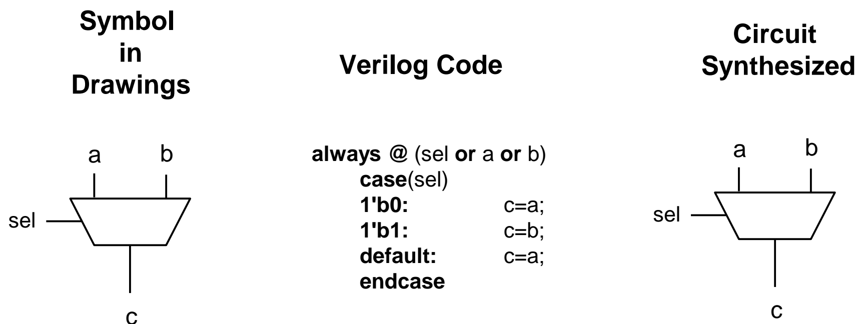

Figure 2 Multiplexers (Muxes)

State Machines combine a state register with a set of muxes and ECC Tek knows the Verilog code will synthesize into a circuit of the form shown in Figure 3. A sample of the Verilog code for implementing a state machine can be seen by looking at the SM Verilog code in the PPU module, for example.

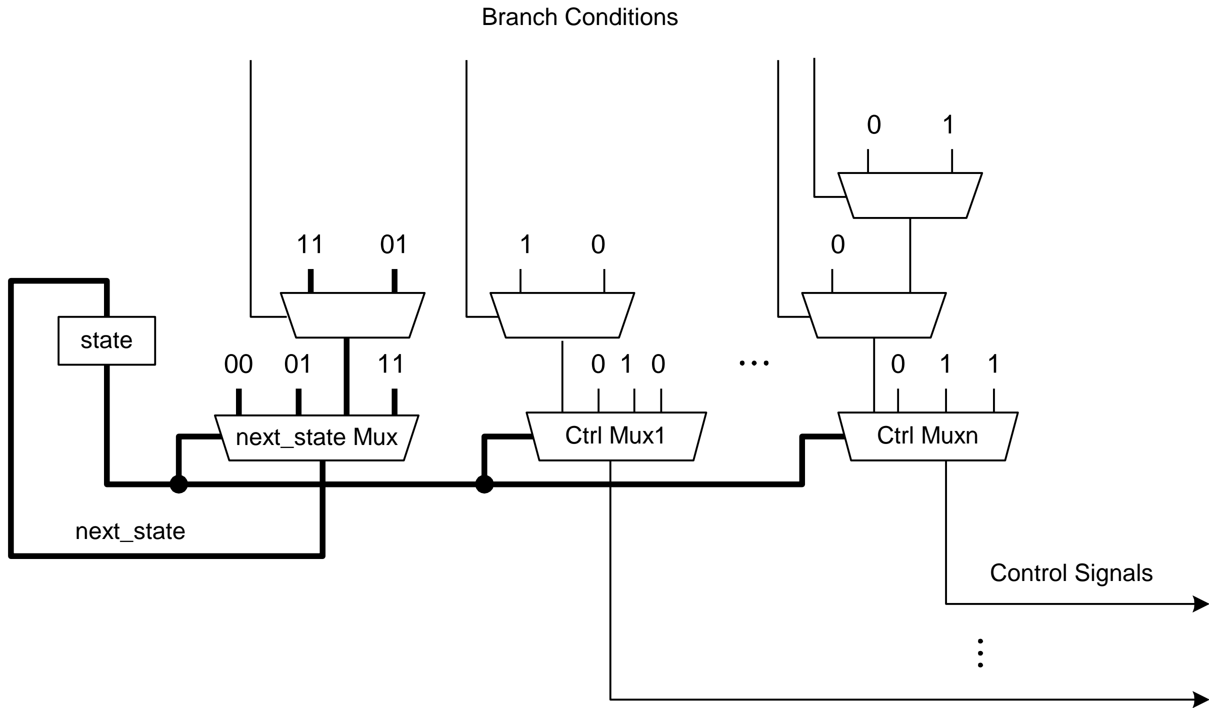

Figure 3 Generic State Machine

A generic control unit is shown in Figure 4. It consists of a number of registers (often up and down counters) used for control purposes and, usually, a state machine ("SM") to implement the control logic. In come cases the control is done completely by combinatorial logic and there is no state machine.

© 2008 ECC Technologies, Inc. All Rights Reserved.

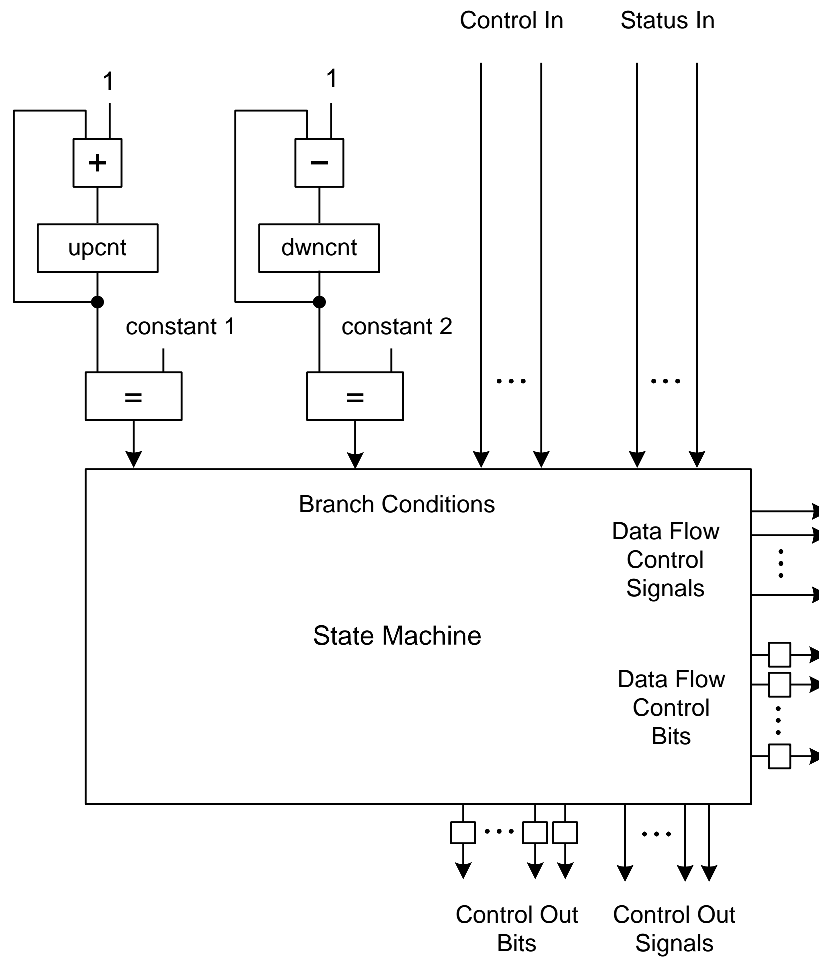

Figure 4 Generic Control Unit with a State Machine

13. Verilog Coding Style and Signal Naming Conventions

The encoder and decoder Verilog code has been written to, for the most part, comply with the Verilog coding style guidelines recommended in the RMM.

The Verilog code is written in a synthesizable form so that synthesizer software can be used to synthesize the actual circuits. This makes the BBCH designs independent of IC vendor.

The Verilog code can be synthesized to be implemented in an ASIC, a structured ASIC or in an FPGA.

© 2008 ECC Technologies, Inc. All Rights Reserved.

Each Verilog module is in a separate file. The names of the files are the same as the names of the modules they contain with the addition of the ".v" filename extension. All filenames use primarily capital letters. For example, the file named "MULTA.v" contains the Verilog module named MULTA.

All signal or variable names in Verilog use lowercase letters.

All Verilog module inputs are prefixed with "i_" to indicate "input" and all Verilog module outputs are prefixed with "o_" for "output". For example, i_clk could be the name of a clock input and o_clk could be the name of a clock output.

The Verilog code is written in a style that makes it easy to spot errors and make changes. All input and output ports to all of the modules are on separate lines and all register and wire type variable declarations are on separate lines.

14. General Hardware Design Development Methodology

The design methodology used to develop the BBCH encoder and decoder hardware designs was to develop them in "steps" as follows:

1. Use a variation of the well-proven and efficient Berlekamp-Massey ("BM") algorithm for decoding. ECC Tek uses one of many variations of the BM algorithm for decoding as described by Willard Eastman in Eastman's paper and refers to it as the "EBM" algorithm.

2. Implement encoder and decoder algorithms in C to prove they work correctly and to generate input data and expected results files for debugging the Verilog code.

3. Thoroughly test the C models.

4. Create an initial "skeleton" version of the decoder that works, but does not have all of the final features required.

5. Add features to the "skeleton" version of the decoder to create a series of decoder versions until all of the required features have been successfully implemented. Each version is one "step". If we are unsuccessful in implementing a "next step", then we can always go back to the previously successful last step and try again.

6. Repeat the above development process for the encoder.

7. Use the C models and Verilog testbenches to debug and validate the designs.

15. Basic Logic Design Concepts Used by ECC Tek

All of the blocks of logic in the encoder and decoder can be divided into two parts – one part is the "control structure", "control unit" or just "control" and the other part is the "data flow structure", "data flow unit" or just "data flow" as illustrated in Figure 5. The control unit receives status signals from the

© 2008 ECC Technologies, Inc. All Rights Reserved.

data flow unit and sends control signals to the data flow unit to control the flow of data. Control signals are generally fed into and out of the control unit and data is fed into and out of the data flow unit.

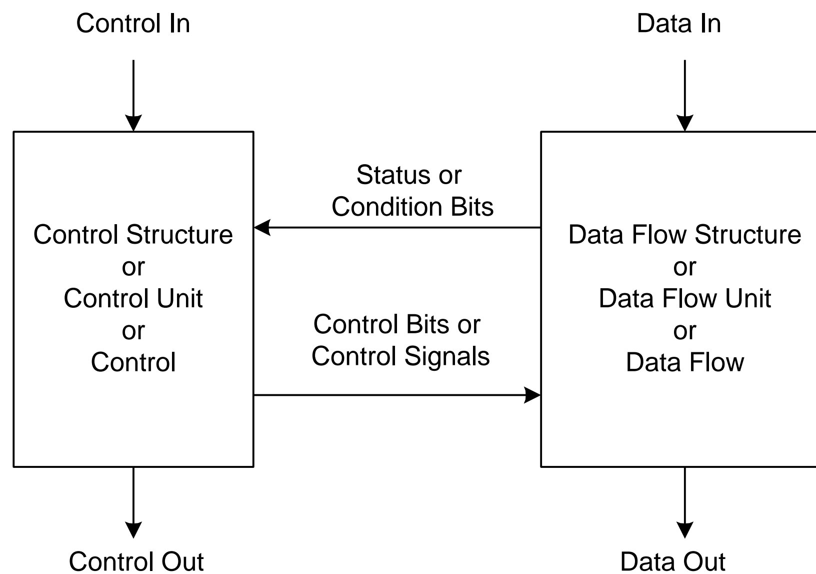

Figure 5 General Form for all Logic Blocks

State machines are described in Verilog in a standard format as described in the RMM so that the Verilog code for a SM will synthesize into a logic circuit of the form shown in Figure 3.

Probably the best way to understand the SM logic is to look at the Verilog code. State diagrams can also be drawn from the Verilog code, but viewing the Verilog code is how ECC Tek creates the SM. The Verilog statements are usually simple, easy to follow and easy to understand. All of the state machines used in the encoder and decoder are very simple and usually only have a few states.

16. Drawing Conventions

In the drawings, the data flow is usually top down.

The control unit for a block of logic is usually drawn to the left of the data flow unit.

All Galois Field full multipliers are drawn with a dot (".") or "x" inside a box. Partial Galois Field multipliers are drawn with a star ("*") inside a box.

17. Implementing Finite Field Operations in Digital Logic

The encoder performs operation in a finite (or Galois) field with 2 1-bit elements or "bits".

The decoder performs operations in a finite field with 2 1-bit elements "bits" and also in another finite field with 16,384 14-bit elements.

The finite field operations used for encoding and decoding BBCH codes are as follows:

- addition of two variable finite field elements,
- multiplication of a variable finite field element by a constant finite field element,
- multiplication of two variable finite field elements, and
- finding the reciprocal (or inverse) of a finite field element.

Addition of finite field elements is a bit-wise exclusive-or (XOR) operation.

The finite field multiplier that multiplies two variable finite field elements is a Verilog module called MULTA. MULTA is a highly structured full multiplier as described in the PRS Patent and illustrated in the top part of Figure 6.

If n multipliers are used and one of the inputs is common to all n multipliers, then the common multiplier logic that would normally be implemented inside each multiplier can be pulled out and implemented outside the multiplier only one time rather than n times. ECC Tek calls the resulting multiplier with the missing piece a "partial" multiplier. Partial multipliers are illustrated in bottom part of Figure 6.

© 2008 ECC Technologies, Inc. All Rights Reserved.

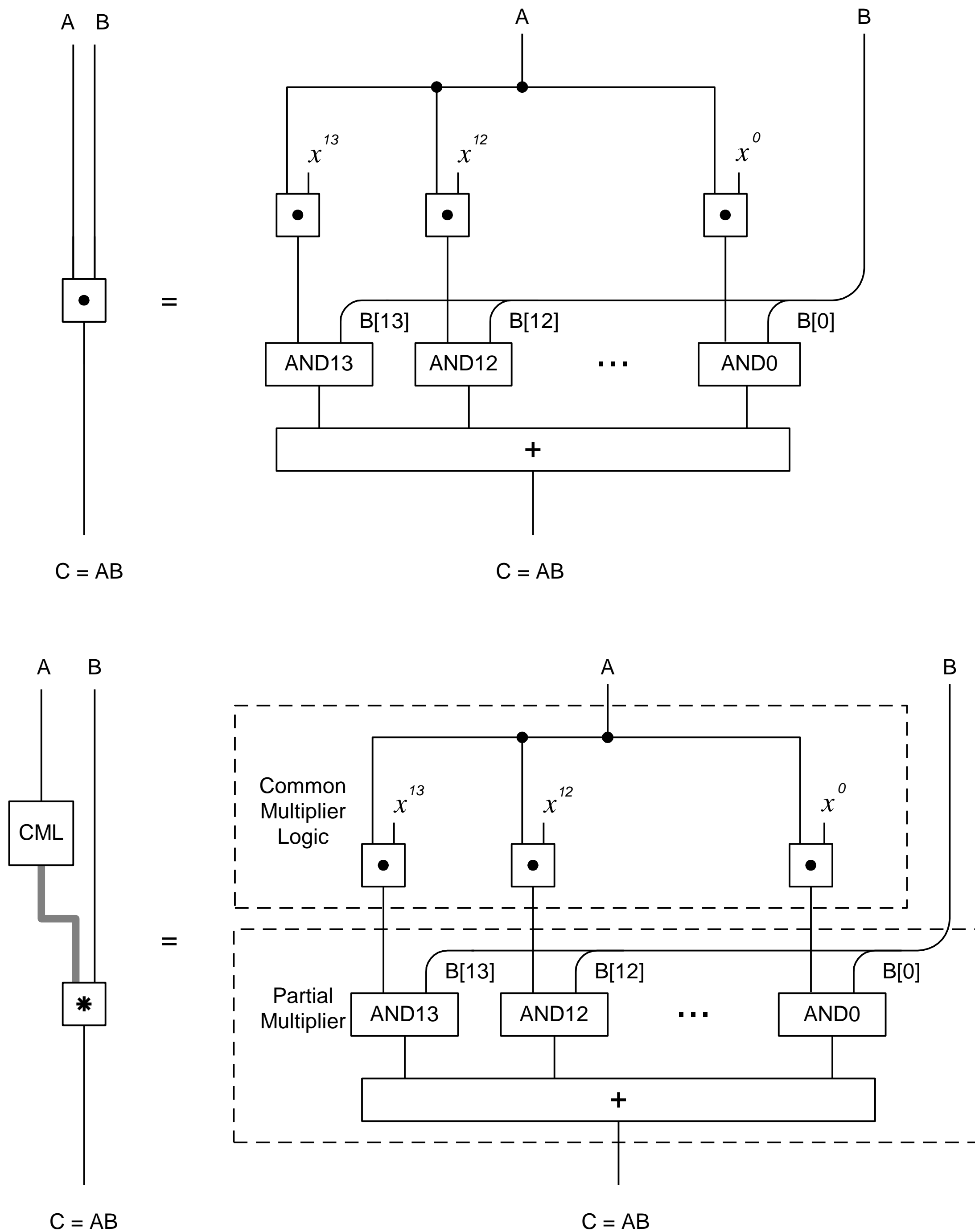

Figure 6 Full Multiplier (top) and Partial Multiplier (bottom)

If an intelligent synthesizer is used, the synthesizer should also recognize that part of the multiplier is common and the result of synthesizing full multipliers and partial multipliers should be the same. Since the synthesis result is synthesizer-dependent, ECC Tek provides both full and partial multiplier versions of the Verilog modules.

Multipliers that multiply a variable finite field element by a constant field element are implemented as shown in Figure 7. The constant, B, determines the connections in Figure 7.

Multipliers that multiply a variable by a constant are separate Verilog modules. The Verilog module that multiplies by 0ABC is called MULT_BY_0ABC.

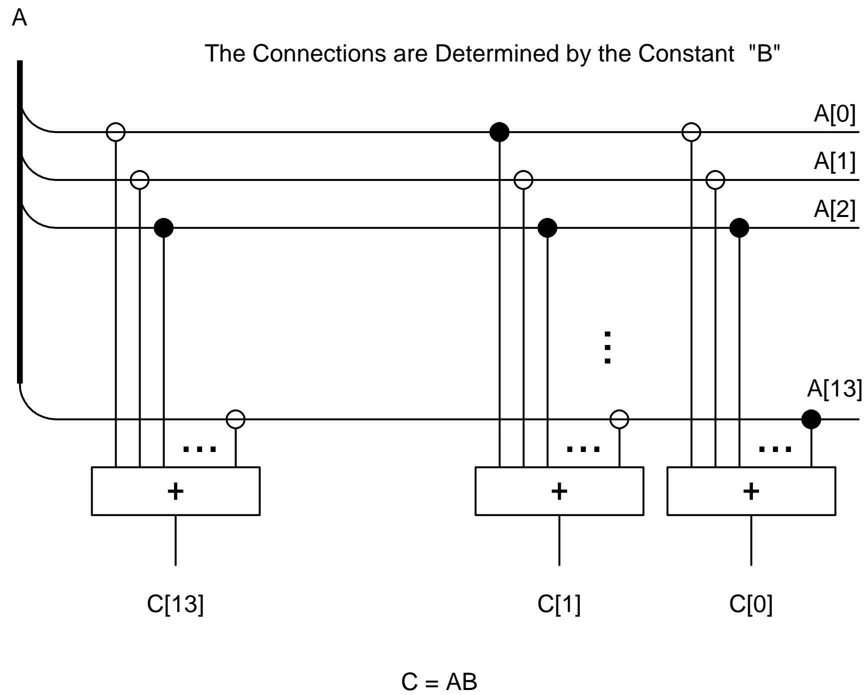

Figure 7 Multiplier that Multiplies the Variable "A" by the Constant "B"

If reciprocals are needed, they are stored in a ROM. Reciprocals are not needed in binary BCH decoders, but are needed for RS decoders.

18. Encoder Versions

Only one version of the encoder was developed and called ENCODERA1. A block diagram of ENCODERA1 is shown in Figure 8.

© 2008 ECC Technologies, Inc. All Rights Reserved.

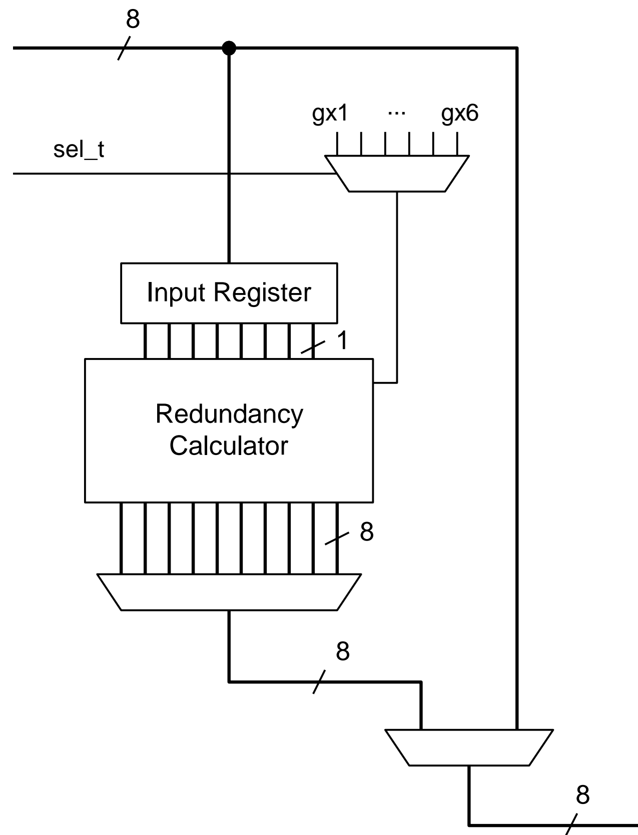

Figure 8 ENCODERA1

The encoder multiplies m(x) by x^r, divides x^r m(x) by g(x), finds the remainder, and appends the remainder onto m(x) as the redundancy. The mathematics involved in encoding is shown in Figure 9.

Figure 9 Encoder Mathematics

The message symbols/bits are shown as a row vector that is multiplied by a generator matrix. Multiplication of the row vector by the generator matrix results in generating the required redundancy which is m(x) x^r mod g(x).

An iterative circuit for implementing the operations shown in Figure 9 is shown in Figure 10.

Figure 10 Iterative Serial Encoder Circuit

The input message bits, m_i, are multiplied by x^r mod g(x). The "&X" symbol indicates an AND multiplication operation. One input to the "&X" box is a "bit". The other input is a 14-bit finite field element. The multiplication operation consists of 14 AND gates which is referred to as an "AND MULTIPLY" operation or an "&X" operation.

Multiplication by "x" as shown in Figure 10 is a left shift with zero enter.

The iterative circuit shown in Figure 10 is a bit-serial circuit and is applicable to situations where input bits are received serially.

To handle 8-bits at a time, the serial circuit of Figure 10 is "parallelized" as shown in Figure 11 to receive 8 bits simultaneously. There is still only one accumulator/register in Figure 11 as in Figure 10, but the rest of the logic is duplicated 8 times to operate on all 8 input bits simultaneously.

The circuit shown in Figure 11 is implemented as the BBCH Encoder circuit with the addition of a set of AND gates inserted into the feedback loop which enables the BBCH Encoder control logic to zero the feedback.

Figure 11 Parallelized Encoder Circuit

19. Encoder Signal Definitions

Input signals have a prefix of "i_". Output signals have a prefix of "o_". For example, i_symbol_valid is an input signal and o_symbol_valid is an output signal.

Signal Name	Description
i_rst	Asynchronous reset, active low.
i_clk	Synchronous clock, active on rising edge.

© 2008 ECC Technologies, Inc. All Rights Reserved.

i_abort	Abort encoding .
i_k_delta[5:0]	Value added to 512 to form K.
i_sel_t[2:0]	t value selected.
i_byte[7:0]	Input byte.
i_byte_valid	Input byte valid.
i_enable_parity	Enable parity bytes to be output. Parity bytes will be outputted when this signal is asserted.
o_byte[7:0]	Output byte.
o_byte_valid	Output byte valid.
o_first_byte	First output byte valid.
o_last_byte	Last output byte valid.

Table 7 Encoder Signal Definitions

© 2008 ECC Technologies, Inc. All Rights Reserved.

20. Decoder Latency

Decoder latency is defined as the number of clock cycles from the last input byte received by the decoder to the first byte outputted.

All of the decoder versions described in this document have a variable latency.

Let T be the number of errors that actually occur and t be the maximum number of errors the decoder is configured to correct.

The decoder latency is S + P + I + O where

S = the latency of the Syndrome Generator
P = the latency of the PPU
I = the latency of the Initialization section
O = the number of overhead cycles

S = 0 or r depending upon which syndrome generator is used where r is the number of "bits" of redundancy. S = 0 if SYNB is used. S = r if SYN1+SYN2 is used. r can vary from 224 to 424 so it takes a significant amount of time to calculate the syndrome if SYN1+SYN2 is used instead of SYNB. However, SYNB will probably require more chip surface area than SYN1+SYN2.

The latency of the PPU depends upon the number of errors the decoder is configured to correct, t, and the number of errors that actually occur, T. P = tT if PPUB1 or PPUC1 is used. P = T if PPUF1 is used. If T = 0, the latencies are equal, but if T = 30, for example, then the PPUF latency will be 30 while the PPUB1 or C latency will be 900 cycles. PPUF1 most likely will take much more area than PPUB1 or C1. In general, the PPU latency is 2tT/P where P is the amount of parallelism in the PPU. For PPUB1 and C1, P=2.

I=1081-N where N is the number of "bytes" in a codeword. I could be reduced to 0, but it would take many more gates.

O is the overhead which is 12 cycles.

The above formula for decoder latency agrees with what was measured except for the end cases when T=t. The latency is slightly different when T=t because they are end cases. The case when t = T = 30 is slightly different from the other end cases because it is the last end case. These minor variations are due to the way the state machine controller is implemented in the PPU.

21. Decoder Versions

ECC Tek has delivered the following 5 decoder versions to Customer: DECODERB1, DECODERB2, DECODERB3, DECODERC1 and DECODERC2.

Decoders B1, B2 and B3 retain the pause signal as it was implemented in the first binary BCH designs but with improvements to increase the effectiveness of the pause signal.

The pause function was improved from previous versions by observing that the only modules which must be clocked with the output clock and therefore need to be paused are EVAL, XOR and the reading of the DFIFO. In earlier versions of the decoder, INIT was also paused.

To improve the pause behavior, the pause signal was removed from INIT and changes were made so the PPU and INIT modules hold their outputs after setting their output go bits and stay in a WAIT state until the next section of logic clears the go bits.

The pause signal should only be used by logic at the output of the DECODER after the first output byte valid is asserted and before the last output byte valid signal is asserted.

If the DFIFO is implemented by two RAM buffers in a ping-pong fashion, the pause signal can be used to temporarily pause the decoder. With that type of design the input would normally be disabled until at least one of the RAM buffers is empty. Incoming received data would be written to the two buffers in a ping-pong fashion – first one and then the other.

All of the decoders use a DFIFO which correctly models Customer's dual-port RAM FIFO, contains changes to improve the way the pause signal is handled from previous versions, and uses a k_delta input signal instead of sel_k so that 30 K values (1024 to 1053) are selectable.

21.1. DECODERB1

A block diagram of DECODERB1 is shown in Figure 12. DECODERB1 is implemented in the same fashion as the first Binary BCH decoder with a pause signal that freezes all of the registers when the pause is asserted.

The decoder latency for DECODERB1 = r + tT + 1081-N + 12 as defined above. When t = T, the formula is not precise, but very close to the actual latency.

© 2008 ECC Technologies, Inc. All Rights Reserved.

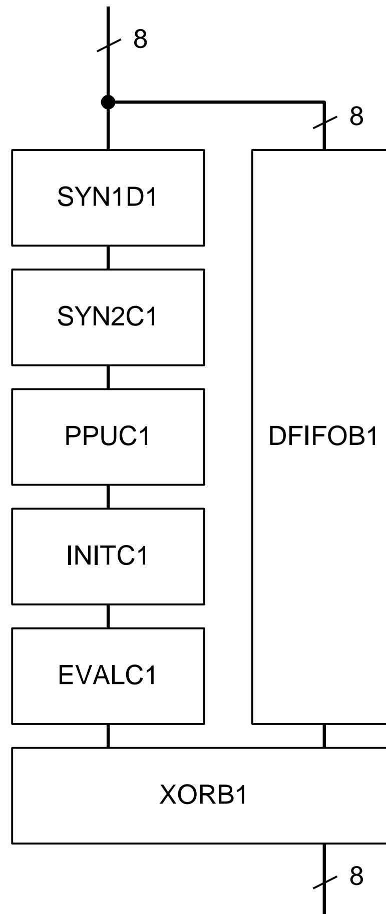

Figure 12 DECODERB1

21.2. DECODERB2

A block diagram of DECODERB2 is shown in Figure 12. DECODERB2 contains the pause signal.

DECODERB2 has a significantly lower latency than DECOBERB1 because of the use of SYNB to replace SYN1+SYN2, but the gate count is also higher.

The decoder latency for DECODERB2 = tT + 1081-N + 12 as defined above. When t = T, the formula is not precise, but very close to the actual latency.

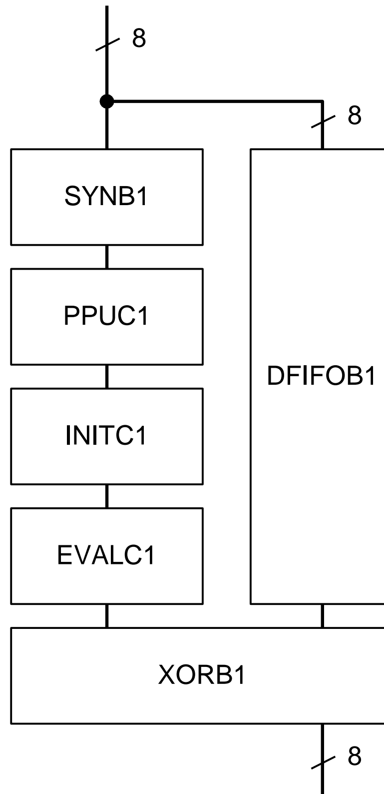

Figure 13 DECODERB2

21.3. DECODERB3

DECODERB3 has close to the lowest possible latency. The latency could be reduced further only if INITC1 were replaced by a very high gate count INIT module.

The decoder latency for DECODERB2 = T + 1081-N + 12 as defined above.

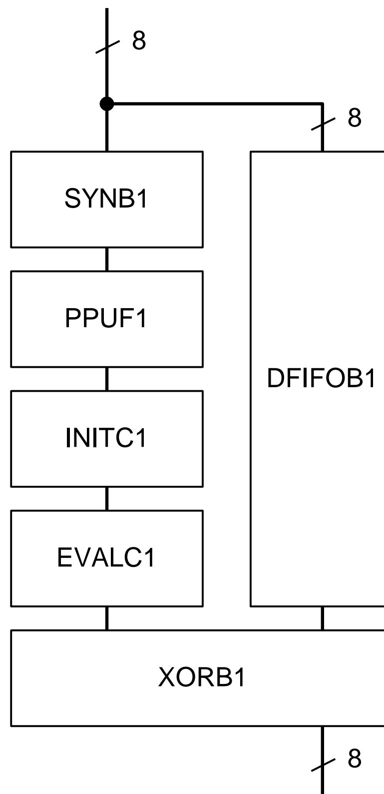

Figure 14 DECODERB3

21.4. DECODERC1

DECODERC1 has the pause signal completely removed from the Verilog modules. The same effect can be achieved through synthesis by fixing the pause signal at 0.

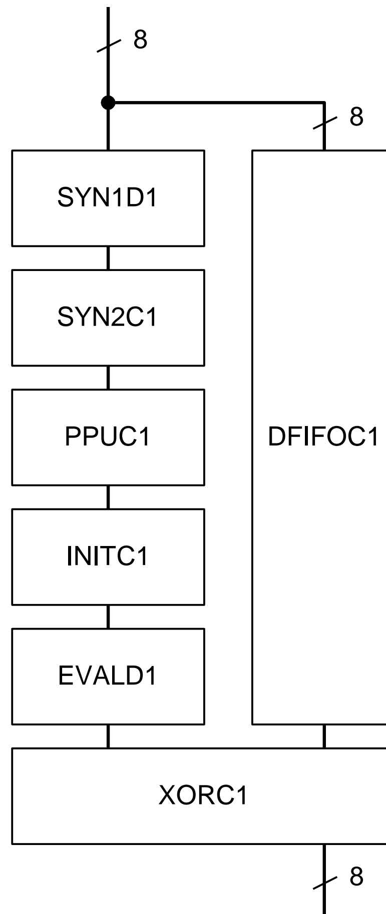

Figure 15 DECODERC1

21.5. DECODERC2

DECODERC2 has the pause signal completely removed from the Verilog modules except for the PFIFO1 module.

DECODERC2 implements a pause FIFO (PFIFO1) at the output of the decoder to implement the pause function without pausing the actual decoding logic.

The decoder latency for DECODERC2 = r + tT + 1081-N + 12 + L where L is the latency of PFIFO1. When t = T, the formula is not precise, but very close to the actual latency.

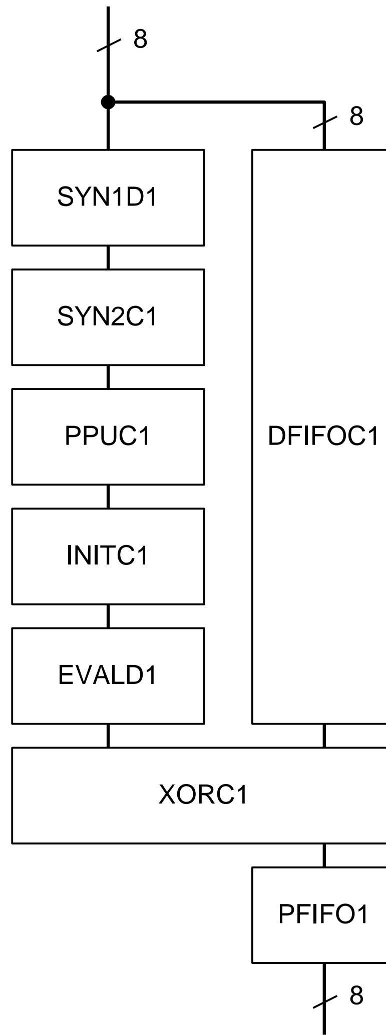

Figure 16 DECODERC2

© 2008 ECC Technologies, Inc. All Rights Reserved.

22. Decoder Clocking Options

22.1. Decoder Clocks

Most of ECC Tek's previous decoder designs have used only one clock, but it is possible to have separate and independent clocks for the input logic, the PPU and the output logic. If the clocks are not synchronized, then logic would have to be added to resynchronize signals that crossed clock domains.

Clocking the PPU at a higher rate would reduce the decoder delay and reduce the DFIFO size needed.

Clocking the output with a separate clock allows the output to be paused by gating the clock.

22.2. Gating the Clocks

ECC Tek normally operates under the assumption that gating the clock is not acceptable. If gating the clock is acceptable, a number of options are opened up such as not clocking either the encoder or decoder if they are not being used and pausing the output by gating the output clock.

23. Decoder Signal Definitions

Input signals have a prefix of "i_". Output signals have a prefix of "o_". For example, i_symbol_valid is an input signal and o_symbol_valid is an output signal.

© 2008 ECC Technologies, Inc. All Rights Reserved.

Signal Name	Description
i_rst	Asynchronous reset, active low
i_clk	Synchronous clock, active on rising edge
i_abort	Abort decoding
i_pause	Pause decoding. i_pause should only to be asserted after o_first_byte has been asserted and before o_last_byte is asserted.
i_k_delta[5:0]	Value added to 512 to form K
i_sel_t[2:0]	t value selected
i_byte[7:0]	Input byte
i_byte_valid	Input byte valid
o_byte_valid	Output byte valid
o_byte[7:0]	Corrected output byte
o_decode_fail	Decoder failed
o_error_count[4:0]	Number of errors corrected in a page. When o_last_byte is asserted, the o_error_count[4:0] is updated for the current page
o_error_cnt_valid	Error count valid
o_first_byte	Asserted when the first output byte is valid at the output
o_last_byte	Asserted when the last output byte is valid at the output
o_syn_not_0	Syndrome is not 0

Table 8 Decoder Signal Definitions

© 2008 ECC Technologies, Inc. All Rights Reserved.

24. Syndrome Generator Mathematics

The syndrome generator performs the mathematical operations shown in Figure 17. The received word symbols/bits, r_i, as a row vector are multiplied by the transpose of the parity check matrix, H^T, as shown in Figure 17. Another way to view the syndrome generation process is that the received word polynomial, r(x), is evaluated at $\alpha^1, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$.

Figure 17 Syndrome Generator Mathematics

The Syndrome can also be generated by first dividing the incoming received word polynomial by g(x) to find the remainder, r(x), and then evaluating the remainder at $\alpha^1, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$. The two methods are equivalent.

Figure 18 shows the operations involved in dividing the received word polynomial by g(x) to find the remainder.

Figure 19 shows the operations involved in evaluating the remainder at $\alpha^1, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$.

Figure 18 Dividing the Received Word Polynomial by g(x) to find the Remainder

Figure 19 Evaluating the Remainder at $\alpha^1, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$

25. Syndrome Generator Circuits

The SYNB1 module generates the syndrome in one step as shown in Figure 17. The SYN1D1 module divides r(x) by g(x) to find the remainder as shown in Figure 18 and the SYN2C1 module evaluates the remainder of dividing r(x) by g(x) at $\alpha^2, \alpha^3, \ldots, \alpha^{2t}$.

25.1. SYNB1

SYNB1 implements the circuit shown in Figure 20.

Figure 20 SYNB1

25.2. SYN1D1

An iterative serial circuit for implementing the SYN1D1 operations is shown in Figure 21.

Figure 21 Iterative Serial Circuit for SYN1

The circuit shown in Figure 21 is parallelized to form SYN1D1 as shown in Figure 22. The multiplication by x operations are done modulo g(x). g(x) is one of 6 g(x)s selectable by input signals.

Figure 22 SYN1D1

25.3. SYN2C1

SYN2C1 evaluates the remainder from SYN1D1 at α^2, α^3, …, α^{2t} as shown in Figure 23.

Figure 23 SYN2C1

SYN2C1 does not use a parallelized version of the circuit because, if a parallelized version was used, there would be no need to split SYN into SYN1 and SYN2. A parallelized version of SYN2C1 would be the same as SYNB and would take the same number of gates and so it would not make any sense to have a parallelized version of SYN2.

26. DFIFO

The DFIFO is implemented by Customer in an off-chip circuit with interfaces with the BBCH decoder logic.

The DFIFO, as implemented by ECC Tek, is configured to write the first K bytes of data to a FIFO and skip the redundant symbols.

© 2008 ECC Technologies, Inc. All Rights Reserved.

27. PPU

The PPU transfers the syndrome from the Syndrome Generator to the PPU's V(x) registers when it receives a go signal.

The PPU then calculates the error locator polynomial, L(x), and the degree of L(x), degLx, from the syndrome polynomial, S(x) according to the EBM algorithm.

The same EBM algorithm is used in the BBCH decoder as it is used in the RS decoder, but some simplifications have been made for the binary case.

For binary BCH codes, every other discrepancy should be 0. The way the EBM algorithm was being initialized for RS codes had to be changed in order to make every other discrepancy 0.

Several versions of the EBM algorithm were developed to show how the original EBM algorithm needs to be changed for the binary case. The changes made are presented in a Companion Document to this document. This document deals only with the final EBM algorithm as shown in Figure 24.

Figure 24 Final EBM Algorithm (V5)

The key equation is $S(x)L(x) \equiv V(x) \bmod x^{2t}$. For RS codes R=2t so the key equation is $S(x)L(x) \equiv V(x) \bmod x^{R}$.

The EBM algorithm requires the use of two "auxiliary" polynomials which ECC Tek calls aL(x) and aV(x). One auxiliary polynomial is associated with L(x) and the other with V(x).

There is also an auxiliary key equation which is $S(x)aL(x) \equiv aV(x) \bmod x^{2t}$. In order to make every other discrepancy 0, aL(x) and aV(x) must be initialized to the same values as L(x) and V(x) which is different from the way aL(x) and aV(x) were initialized in the RS case. Both initializations are valid and both initializations produce correct results, but only one initialization results in every other discrepancy being 0 which is required in order to simplify the algorithm.

Initially, the EBM algorithm guesses that 0 errors have occurred or, in other words, that the correct L(x) is L(x) = 1. This is a reasonable guess because, in many situations, 0 errors is the most likely error pattern.

If the initial L(x) is 1, then the initial V(x) must be S(x) because, if we substitute L(x) = 1 into the key equation, the key equation becomes $S(x)1 \equiv V(x) \bmod x^{2t}$, and V(x) must be S(x) to satisfy the key equation.

However, we know from coding theory that in order for L(x) = 1 to be the correct L(x), V(x) must be 0, which means the syndrome must be all zeros.

© 2008 ECC Technologies, Inc. All Rights Reserved.

The EBM algorithm examines one coefficient of V(x) in each iteration of the algorithm. The V(x) coefficient being focused on for the jth iteration of the EBM algorithm is called the jth "discrepancy" or dj. Initially, j = 0 and d0 = V0 = S1. Sometimes dj is referred to as d.

Initially, if d0 = V0 = S1 $\neq 0$, the EBM algorithm knows that the correct or valid L(x) is not L(x) = 1, that the initial guess that L(x) = 1 must be modified and that the degree of L(x) must be increased. The EBM algorithm specifies how L(x) must be modified and how this iterative procedure continues until the correct L(x) and the correct degLx are determined. It is difficult to understand why the EBM algorithm works correctly, but it has been proven to be valid in papers and textbooks on coding theory. Fortunately, even though it is difficult to understand why the EBM algorithm works, it is not difficult to implement the EBM algorithm as shown in Figure 24.

The PPU implements the EBM algorithm. A top level block diagram of a PPU with a parallelism level P = 2 is presented in Figure 25.

Figure 25 P=2 PPU Top Level Block Diagram

Details of what is in the boxes in Figure 25 are shown in the following Figures.

Figure 26 P=2 PPU L Section

Figure 27 P=2 PPU Branch Section

Figure 28 P=2 PPU V Section

When operations are performed on polynomials, operations are being performed on each coefficient of each polynomial. For example, if we add two polynomials, we are adding the coefficients. If we multiply a polynomial by a value, we are multiplying each of the coefficients of the polynomial by that value.

A P=2 PPU generates two coefficients of the next L(x) and two coefficients of the next aL(x) in one clock cycle and, simultaneously generates two coefficients of the next V(x) and two coefficients of the next aV(x). Eight new coefficients are generated simultaneously.

© 2008 ECC Technologies, Inc. All Rights Reserved.

When the PPU has finished calculating L(x) and the degLx, it sets an output control bit called "o_PPU_full" which indicates to the next module in the pipeline that next L(x) and degLx values are available.

After the PPU sets the "o_PPU_full" bit, the PPU waits for the next module in the pipeline to transfer the L(x) and degLx values. The following module in the pipeline may be busy, and may not be able to input the new L(x) and degLx values immediately. When the following module is not busy, it transfers the new L(x) and degLx values into its registers from the PPU and clears the "o_PPU_full" bit which triggers the PPU to stop waiting for the got bit to be cleared and start waiting for the next "go" signal again.

The P=2 PPU control unit contains two index registers called "i" and "j". The j index register corresponds to the "j" variable in the BM algorithm shown in Figure 24. The i index register is used to indicate which coefficients of L(x), aL(x), V(x), and aV(x) the PPU is currently generating.

The L section in the P=2 PPU computes two new L(x) and two new aL(x) coefficients in each clock cycle.

The V section in the P=2 PPU computes two new V(x) and two new aV(x) coefficients in each clock cycle.

The degree of L(x) is initially 0. The degree of L(x) increases as needed and, when correction is successful, the degree of L(x) will be equal to the value of the "degLx" variable. The degLx value should be the same as the number of errors corrected when correction takes place in the XOR module. If the degLx is not the same as the number of errors corrected by the XOR module, then the decoder has detected an uncorrectable error pattern.

The Branch Section as shown in Figure 27 contains two more multipliers than the Branch Section for the RS design. The inputs to the branch section are only updated at the end of computing new L and V values so an extra clock cycle could be inserted at the end of each polynomial iteration of the EBM algorithm to allow for the extra delay of the two additional multipliers if necessary without significantly decreasing the performance. This was not done for the current designs.

27.1. PPU Versions

All of the PPU versions described in this document have a variable latency. The latency depends upon the number of errors that have actually occurred.

27.1.1. PPUC1

PPUC1 is a P=2 PPU and operates on two L(x) and two V(x) polynomial coefficients simultaneously and has a variable delay depending upon the number of errors that actually occurred.

The PPUC1 latency is tT + overhead where t is the number of errors the decoder is configured to correct and T is the number of errors that have actually occurred.

27.1.2. PPUF1

PPUF1 is a fully parallelized PPU. The level of parallelism is P=2t.

The PPUF1 latency is T + overhead which is about as fast as ECC Tek can make a PPU. The only way a PPU could be made faster is to implement a very high gate count INIT module which is probably impractical.

PPUF1 instantiates many Coefficient Processing Unit (CPU) modules so that all of the coefficients of L(x), aL(x), V(x) and aV(x) are operated on and updated simultaneously.

Since PPUF1 almost certainly contains many more gates than Customer can accommodate, it is not described in detail in this document.

28. INIT

29. EVAL

30. XOR

© 2008 ECC Technologies, Inc. All Rights Reserved.