Reed-Solomon (RS)

Encoder and Decoder Designs

For Correcting

Five 10-bit Symbol Errors

(Customized to Meet Customer's Requirements)

Document Version 0.1

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

© 2005 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 Reed-Solomon (RS) 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 RS 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 RS encoder and decoder designs described in this document must be licensed from ECC Tek in order to legally implement them.

© 2005 ECC Technologies, Inc. All Rights Reserved.

Revision History

01-21-05	pew	Created initial version of this document based on previous documents and drawings.
05-11-05	pew	Edited and added text and inserted drawings as Figures.
05-13-05	pew	Edited and added text and inserted drawings as Figures.
05-16-05	pew	Edited and added text and inserted drawings as Figures.
05-17-05	pew	Edited and added text and inserted drawings as Figures.

© 2005 ECC Technologies, Inc. All Rights Reserved.

Sections

© 2005 ECC Technologies, Inc. All Rights Reserved.

Figures

© 2005 ECC Technologies, Inc. All Rights Reserved.

1. Overview of RS Encoder and Decoder Designs

This document describes digital logic designs for encoding and decoding a Reed-Solomon (RS) code which can correct five 10-bit symbol errors.

The RS code being implemented uses K = 524 message (or data) symbols, N = 534 codeword symbols and R = 10 redundant parity check symbols.

The RS encoder and decoder logic operate on 10-bit symbols internally, but data is input and output as 8-bit bytes. The bytes are converted from bytes to symbols (B2S) at the input and the symbols are converted from symbols to bytes (S2B) at the output as shown in the encoder and decoder overviews in Figure 1 and Figure 2.

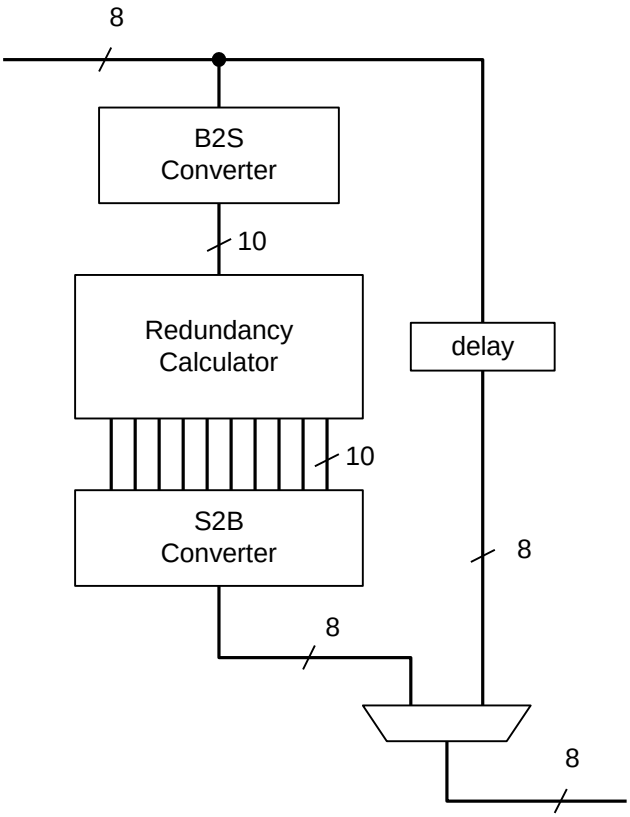

Figure 1 RS Encoder Overview

© 2005 ECC Technologies, Inc. All Rights Reserved.

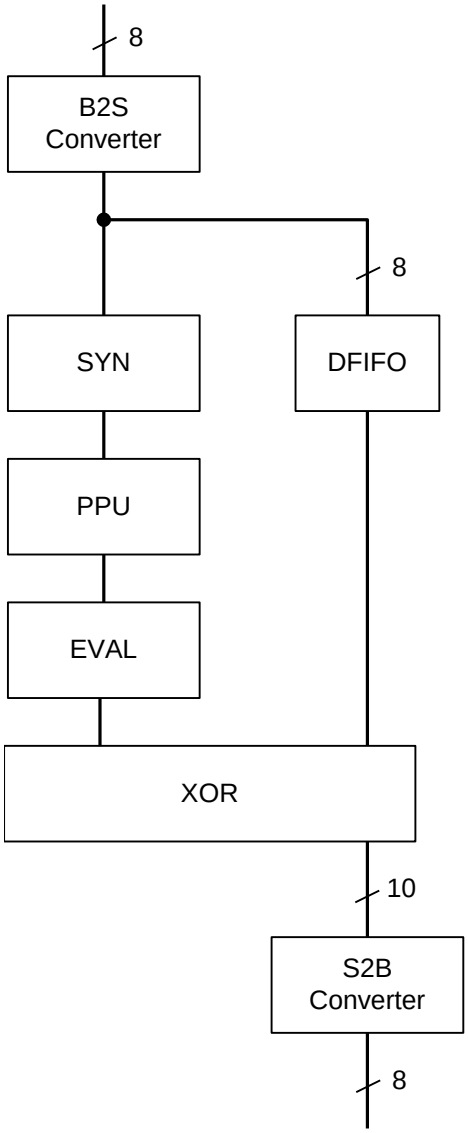

Figure 2 RS Decoder Overview

The encoder maps messages (or data words) which contain 524 symbols into codewords that contain 534 symbols by calculating and appending 10 symbols of redundancy. The decoder corrects up to 5 symbols in error in received words which contain 534 symbols. The decoder outputs a most likely data word and outputs and error count indicating whether 0, 1, 2, 3, 4 or 5 symbols have been corrected in the received word and if the decoder has failed by detected more than 5 symbols in error. If 5 or fewer symbol errors have actually occurred, the most likely data word outputted by the decoder will be the correct original message (or data word). In other words, the decoder will correctly correct all error patterns that can be interpreted as 5 or fewer symbols in error.

The input to the DFIFO can be connected to the output of the B2S Converter as shown in Figure 2 or, alternatively, the DFIFO can be connected as shown in Figure 3. The Decoder configuration as shown

© 2005 ECC Technologies, Inc. All Rights Reserved.

in Figure 2 is probably the preferred configuration because the DFIFO can be slightly smaller than it can be if it is configured as in Figure 3.

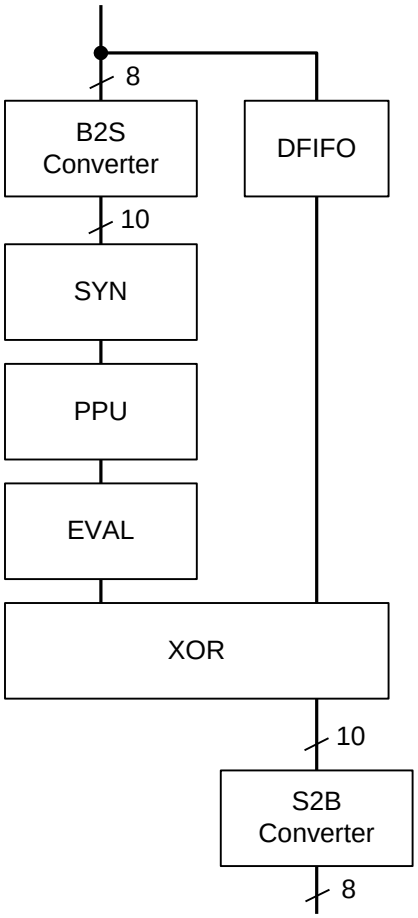

Figure 3 Alternative Decoder Configuration

1.1. What This Document Does

This document describes the final versions of a RS encoder and decoder which correct all error patterns with 5 or fewer symbols in error and is intended to help the reader understand the RS 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.

1.2. What This Document Does Not Do

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 if the reader intends to gain an in-depth understanding of the math and theory

© 2005 ECC Technologies, Inc. All Rights Reserved.

associated with Reed-Solomon 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 RS encoder and decoder designs can be used to verify that the RS 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.

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. The key equation is $S(x)L(x) \equiv V(x) \bmod x^R$ or, in words, S(x) times L(x) is congruent to V(x) modulo x^R where S(x) is the syndrome polynomial, L(x) is the error locator polynomial, V(x) is the error evaluator polynomial and R is the degree of the generator polynomial, g(x). For the RS code described in this document, R = 10.

When operations are performed on polynomials modulo x^{10}, 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 10 because x^{10} is a factor of all terms of the form $c_i x^i$ when i is greater than or equal to 10 and x^{10} is congruent to 0.

It is well-known among coding theorists and stated in some textbooks on coding theory that the integer variable used in the BM 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.

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.

2. References

This document refers to other documents which are listed below:

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

© 2005 ECC Technologies, Inc. All Rights Reserved.

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 Reed-Solomon Error-correcting Codes"*, May 19, 1998. Referred to in this document as **"the RS 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.

3. 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.

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 1.

© 2005 ECC Technologies, Inc. All Rights Reserved.

Table 1 Comparison of Notation used by ECC Tek, Eastman and Berlekamp

	ECC Tek's Notation	Eastman's Notation (and Notation in the RS 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)

© 2005 ECC Technologies, Inc. All Rights Reserved.

Table 2 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 2 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 RS codes. There are numerous versions of the BM algorithm. The version presented in Eastman's paper is used in this document.
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 BM algorithm iteration number. If the current L(x) being calculated in the decoder pipeline is the correct L(x), the discrepancy will be 0 from this point until the end of the Berlekamp-Massey algorithm is reached.
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 BM 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 RS 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).
K	The number of symbols in each message or data field.

© 2005 ECC Technologies, Inc. All Rights Reserved.

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 Th 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.
R	The number of redundant symbols (symbols) in a codeword. For the RS code described in this document, R = 5.

© 2005 ECC Technologies, Inc. All Rights Reserved.

r[i]	In the C code, the received symbol in position i.
ri	In the Verilog code, the received symbol in position i.
RS	Reed-Solomon – 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(α^{-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 one of many primitive elements in a finite field with 1024 elements.

© 2005 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.

4. C Modeling Program

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

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.

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

© 2005 ECC Technologies, Inc. All Rights Reserved.

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, 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.

The first table 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 = 0x000, 0x001,...,ox3ff.

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 0x000 to 0x03ff.

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.

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.

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

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

© 2005 ECC Technologies, Inc. All Rights Reserved.

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

4.2. 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. For example, the testing C code can be used to determine the probability of the decoder declaring a failure conditioned on the fact that a 6-symbol error pattern has occurred.

There are a number of error pattern generator functions in the C code that can be used to generate error patterns with a particular 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 3 with a brief description of what they do.

Table 3 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.
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.

© 2005 ECC Technologies, Inc. All Rights Reserved.

SystemTest2()	Tests the encoder and decoder system.
Test_Encoder()	Tests the encoder.
Test1_Decode()	Test decoder's ability to correct 1 symbol/symbol error patterns.
Test1_System()	Test encoder and decoder system.
Test2_Decode()	Test decoder's ability to correct 2 symbol/symbol error patterns.
Test3_Decode()	Test decoder's ability to correct 3 symbol/symbol error patterns.
Test4_Decode()	Test decoder's ability to correct 4 symbol/symbol error patterns.
TestR_Decode()	Test decoder's ability to correct equally probable random error patterns.
TestR5_Decode()	Test decoder's ability to correct 5 symbol/symbol random error patterns.
TestR8_Decode()	Test decoder's ability to correct 8 symbol/symbol random error patterns.
TestRn_Decode()	Test decoder's ability to correct n symbol/symbol random error patterns.
WriteFiles()	Writes Verilog input data and expected results testbench files.

The SystemTest() function tests both the encoder and decoder as a system. A message is generated at random and then encoded into a codeword. Errors are added to the codeword to form the received word and the received word is decoded to see if the decoder corrects the errors correctly. The decoder test functions are used to test the decoder against error patterns with 1 symbol in error, 2 symbols in error, 3 symbols in error, 4 symbols in error, 5 symbols in error, 8 symbols in error and a random number of bytes in error. For the RS code, all single, double and triple symbol error patterns have been tested.

It is not possible to exhaustive test decoders which correct many symbols 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.

© 2005 ECC Technologies, Inc. All Rights Reserved.

Most of the decoder test functions assume the codeword is the all-zeros codeword. A nonzero received word is created by the test function for the decoder to decode. Each received word, r[i], created by the test function has a nonzero symbol value (Value 1 = V1) in Location 1 (L1). There also may be L2, V2, L3, V3, etc. symbols.

The WriteFiles() function writes input data and expected results files for use with testbenches.

5. 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 RS encoder and decoder designs to ensure that the designs will be synthesizable and will achieve a high level of performance.

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 strategy.

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.

© 2005 ECC Technologies, Inc. All Rights Reserved.

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 4.

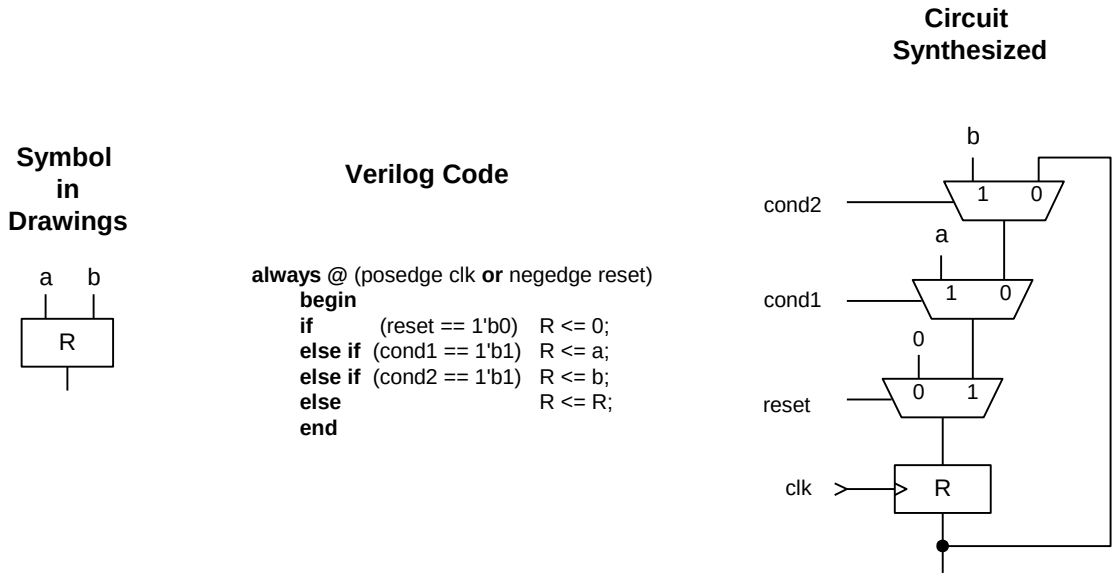

Figure 4 Registers

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 5.

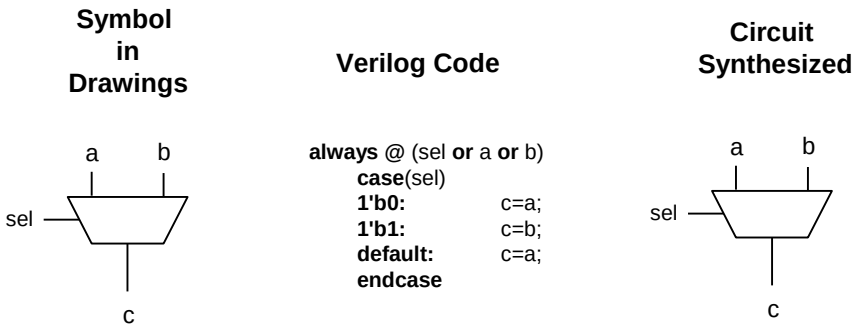

Figure 5 Multiplexers (Muxes)

© 2005 ECC Technologies, Inc. All Rights Reserved.

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 6. 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.

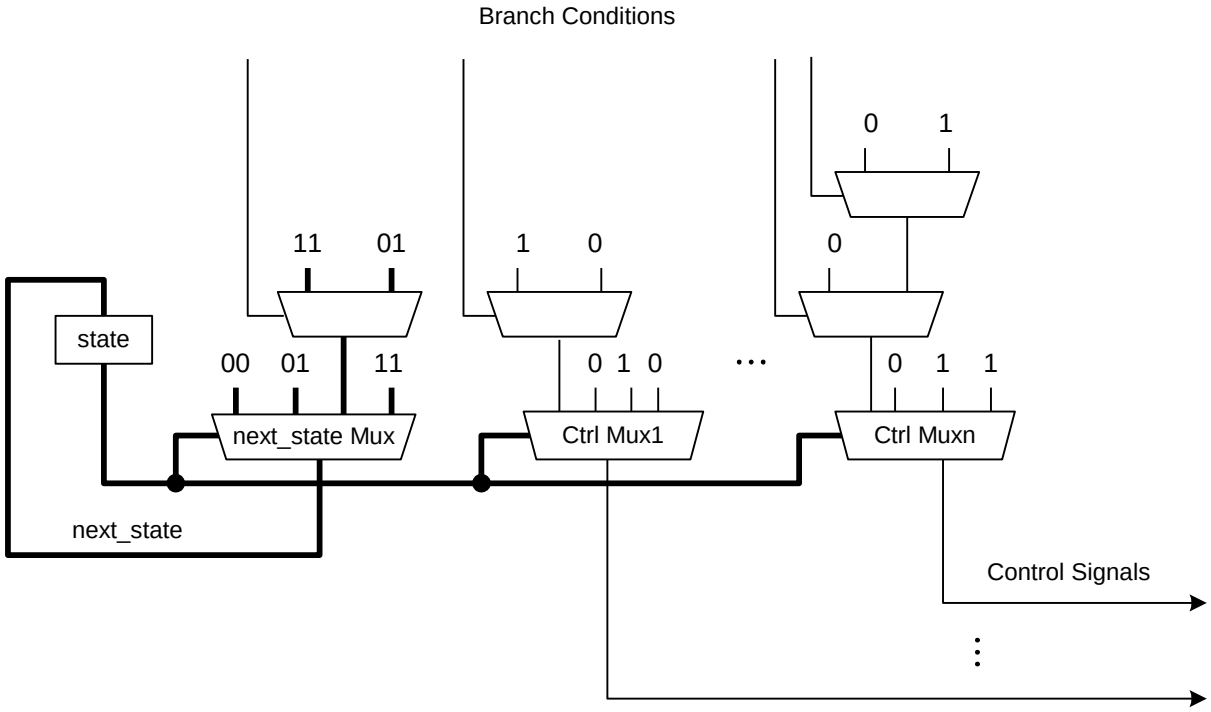

Figure 6 Generic State Machine

A generic control unit is shown in Figure 7. 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. The CONVERT and SYN modules have no state machine – only combinatorial logic to control the flow of data.

© 2005 ECC Technologies, Inc. All Rights Reserved.

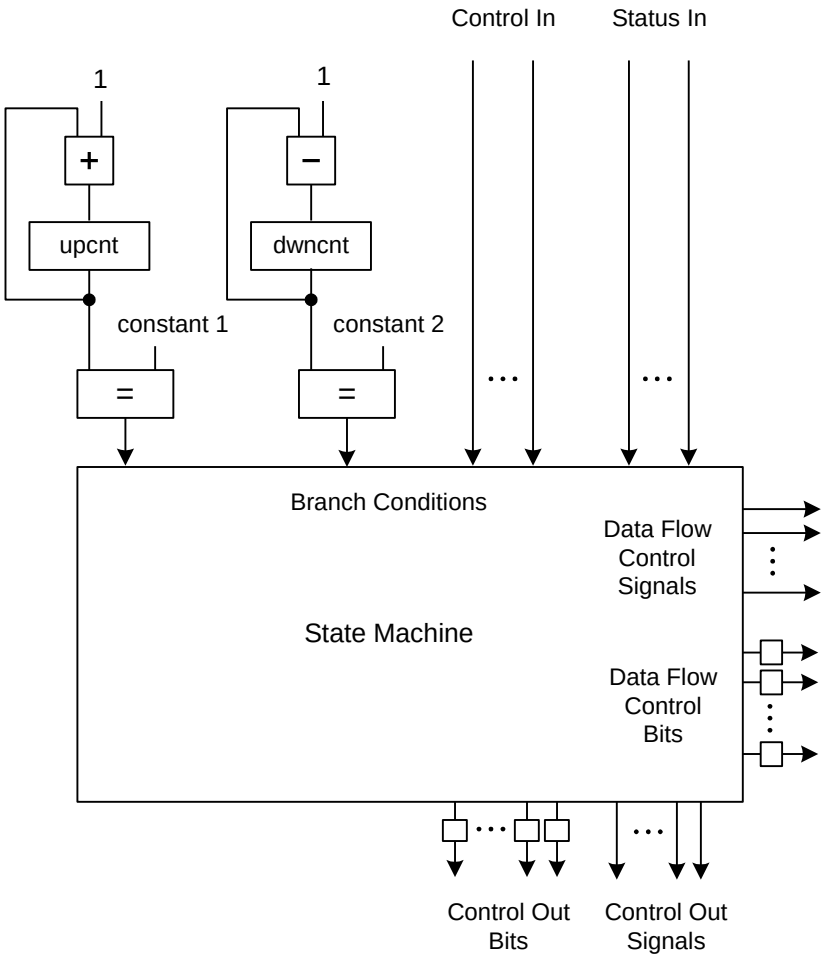

Figure 7 Generic Control Unit with a State Machine

6. 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 RS designs independent of IC vendor.

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

© 2005 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 "MULT2.v" contains the Verilog module named MULT2.

All signal 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".

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.

7. General Hardware Design Development Methodology

The design methodology used to develop the RS 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.

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.

8. Versions of the Verilog Code

As of the date of this document, seven versions of the decoder have been delivered to Customer and are named DECODER1, DECODER2, DECODER3, DECODER4, DECODER4P2, DECODERx and RSD. The RSD version has all of the signal names of the RSD module changed to match Customer's signal names. All of the files for each version are stored in one folder (directory). The focus of this document

© 2005 ECC Technologies, Inc. All Rights Reserved.

is on the latest RSD version since this is the version that Customer will be implementing in their chip. The previous versions probably are of little use to either Customer or ECC Tek once the final version has been developed, so this document will not discuss the initial versions.

Four versions of the encoder have been created and are named ENCODER1, ENCODER2, ENCODERx, and RSE. The RSE version has the signal names changed to match Customer's signal names. All of the files for each version are stored in one folder. Again, the focus of this document is on the final RSE version and not the versions that preceded the RSE version.

In all versions of the encoder and decoder, it was assumed that the synthesizer software will do as good a job of simplifying logic from a general logic description as what a person could do so it was assumed that the synthesizer would do the job of pruning and simplification.

As a result of making the decision to allow the synthesizer to do pruning and simplification of general-purpose or generic logic, an attempt was made to create the fewest number of generic or general-purpose Verilog modules.

9. 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 8. The control unit receives status signals from the 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.

© 2005 ECC Technologies, Inc. All Rights Reserved.

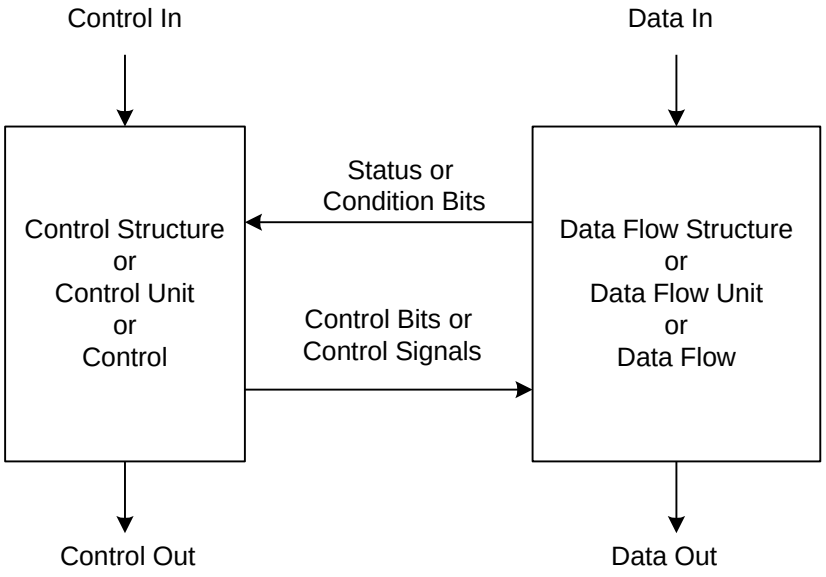

Figure 8 General Form for all Logic Blocks

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 6.

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 very easy to follow and understand. All of the state machines used in the encoder and decoder are very simple and usually only have a few states.

10. 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 (“.”) inside a box. Partial Galois Field multipliers are drawn with a star (“*”) inside a box.

11. Clocking

The encoder and decoder can be clocked at over 120 MHz using worst case maximum logic delays.

© 2005 ECC Technologies, Inc. All Rights Reserved.

12. Implementing Finite Field Operations in Digital Logic

Both the encoder and the decoder perform operations in a finite (or Galois) field with 1024 elements where each element is a 10-bit-wide quantity referred to in this document as a "symbol". The irreducible and primitive binary polynomial used to generate the finite field is p(x)= 0x409.

$p(x) = x^{10} + x^3 + 1 = 0x409.$

The finite field operations used for encoding and decoding RS 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 MULT1. MULT1 is a highly structured full multiplier as described in the RS Patent and illustrated in the top part of Figure 9.

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 9.

© 2005 ECC Technologies, Inc. All Rights Reserved.

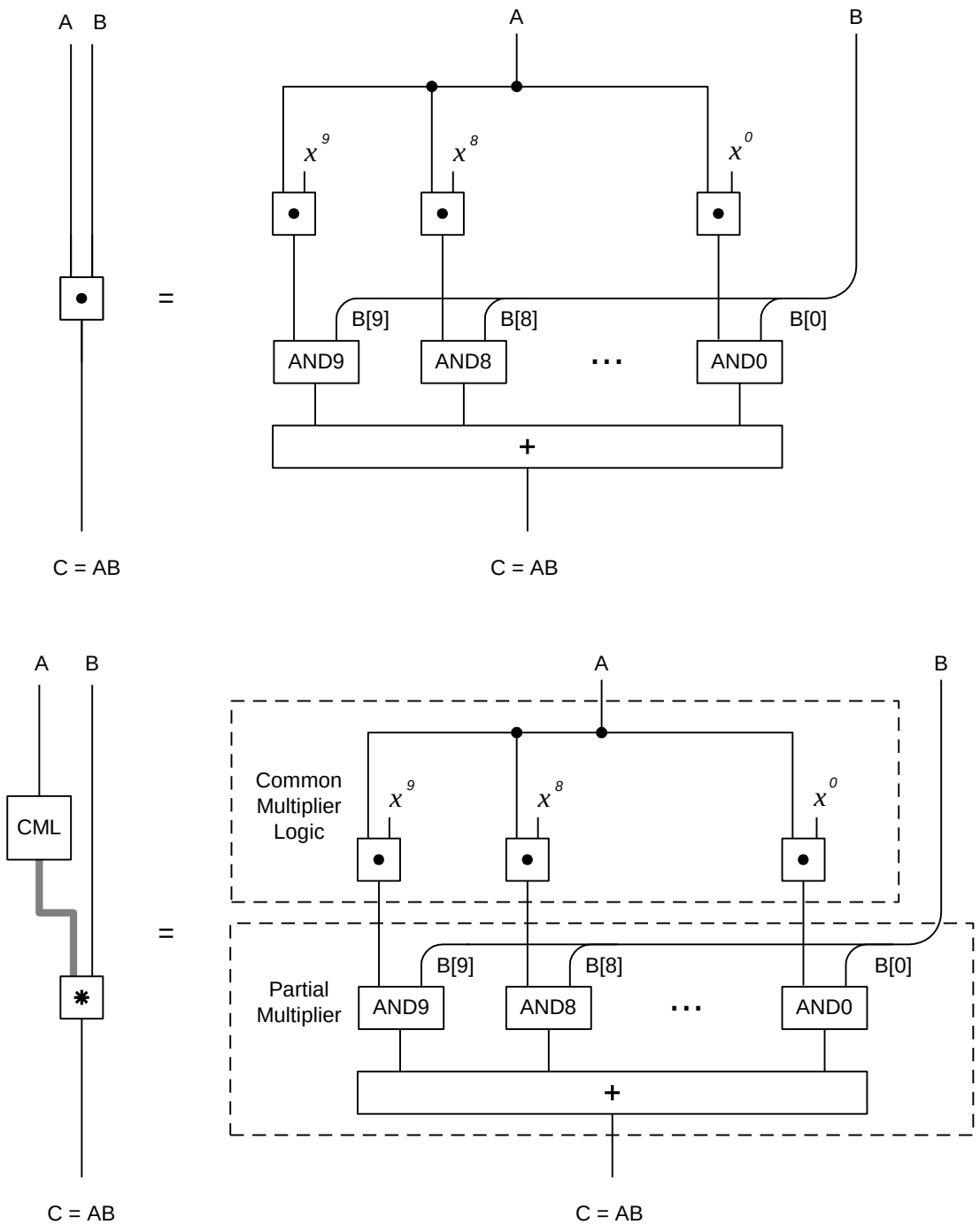

Figure 9 Full Multiplier (top) and Partial Multiplier (bottom)

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 which require multiple multipliers.

© 2005 ECC Technologies, Inc. All Rights Reserved.

Multipliers that multiply a variable finite field element by a constant field element are implemented as shown in Figure 10. The constant, B, determines the connections in Figure 10.

Multipliers that multiply a variable by a constant are separate Verilog modules. The Verilog module that multiplies by 0ab is called MULT_BY_0ab.

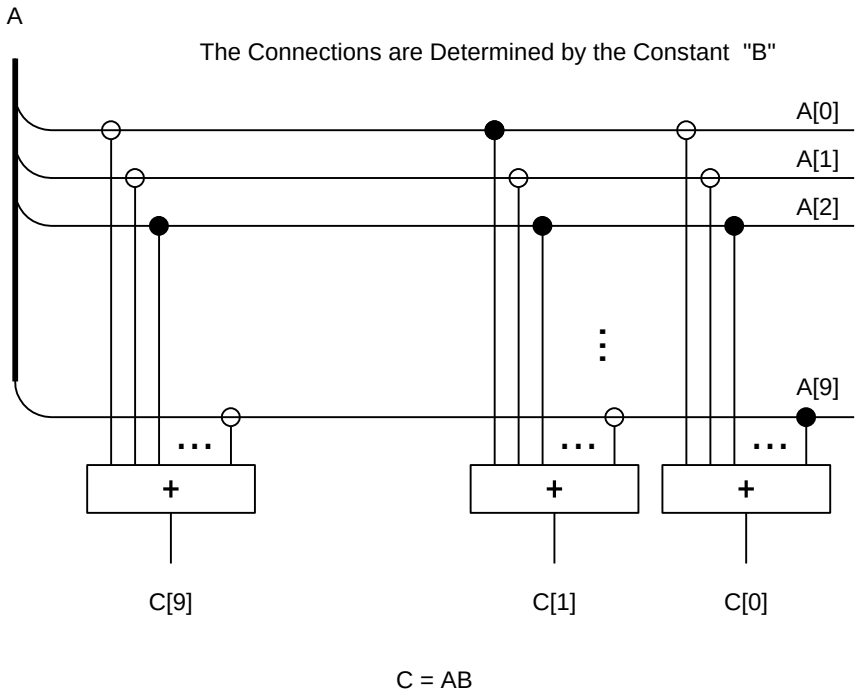

Figure 10 Multiplier that Multiplies the Variable "A" by the Constant "B"

Reciprocals of finite field elements are stored in a ROM. Only one reciprocal ROM is needed.

13. The RS Code Used for the 5 Symbol Error Correcting Designs

Five consecutive powers of the primitive element α = 0x002 are used to form the RS generator polynomial, g(x). g(x) has degree 10.

The generator polynomial, $g(x) = (x-\alpha^0)(x-\alpha^1)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6)(x-\alpha^7)(x-\alpha^8)(x-\alpha^9) = g_0 + g_1x + g_2x^2 + g_3x^3 + g_4x^4 + g_5x^5 + g_6x^6 + g_7x^7 + g_8x^8 + g_9x^9 + g_{10}x^{10}$ where g_0 = 0a9, g_1 = 1f2, g_2 = 3e5, g_3 = 1cf, g_4 = 1d0, g_5 = 396, g_6 = 2a5, g_7 = 0af, g_8 = 0c3, g_9 = 3ff, and g_{10} = 001. The coefficients are shown in hexadecimal notation.

The "first root power" is equal to 0 in this case. The first root power can be any integer, but 0 is used in many standard RS schemes because it simplifies the RS decoder logic.

© 2005 ECC Technologies, Inc. All Rights Reserved.

14. Encoder Design Details

15. Decoder Design Details

15.1. CONVERT

15.2. DFIFO

15.3. SYN

15.4. PPU

15.5. EVAL

15.6. XOR

15.7. Pause Feature

© 2005 ECC Technologies, Inc. All Rights Reserved.