Energy Efficient Digital Baseband Modulator for Cable Terminal Systems Targeted on Field Programmable Gate Array

A thesis presented to

the faculty of the

Fritz J. and Dolores H. Russ

College of Engineering and Technology of

Ohio University

In partial fulfillment

of the requirement for the degree

Master of Science

Feng Wang

June 2004

This thesis entitled

Energy Efficient Digital Baseband Modulator for Cable Terminal Systems Targeted on Field Programmable Gate Array

by Feng Wang

has been approved for

the School of Electric Engineering and Computer Science

and the Russ College of Engineering and Technology by

Janusz A. Starzyk
Professor of School of Electrical Engineering and Computer Science

R. Dennis Irwin
Dean, Fritz J. and Dolores H. Russ
College of Engineering and Technology

Abstract

Feng Wang. M.S. June 2004. Electrical Engineering and Computer Science

Energy Efficient Digital Baseband Modulator for Cable Terminal Systems Targeted on Field Programmable Gate Array (100 pp.)

Director of Thesis: Janusz A. Starzyk

This thesis presents design and research in energy efficient digital baseband modulator for cable terminal systems targeted on field programmable gate array (FPGA). The design specifications of the individual processing blocks of digital baseband modulator are reviewed. Existing low power design techniques at algorithm and architecture levels are examined and their effectiveness for low power design on FPGA is investigated based on the power dissipation characteristics of the FPGA. Low power design strategy for the digital modulator is then derived. Finally, the implementation options for several key modules are investigated and the design space of power and area product is explored. In this design, a new parallel finite field multiplier is proposed, the interleaving algorithm is reformulated and rescheduling is used in the TCM modulator to achieve the low power goal. The results of this research show that most of the low power design techniques, except parallelizing, are very effective for energy efficient design in FPGA.

Approved: Janusz A. Starzyk
Professor, EECS

Acknowledgments

My thesis research could not be accomplished without the help and support from the others. First and foremost, I would like to thank my research advisor Professor Janusz A. Starzyk for his extraordinary help and guidance since I joined the VLSI group. His profound insights into abstract problems and exceptional capabilities to solve them with mathematical model helped me at various time of this thesis work. His high standards of professional conduct and performance serve as a role model for me. Without his financial support, I could not finish my study here. I would also like to thank Professor Jeffrey C. Dill, Professor Xiaoping Shen, and Professor Savas Kaya for reviewing this thesis and providing helpful comments. Also Professor Jeffrey C. Dill‘s lectures built my coding theory background, which is critical for this thesis work.

Next, I would like to thank my friends in VLSI group. I’d like to thank Zhen Zhu. Zhen’s wide knowledge of communication theory and extraordinary skills in the Matlab programming helped me out in various times during the project. I would also like to thank Yinyin Liu for her useful advice in thesis writing. During my two years study in VLSI group, I learned a lot from the other members in the group: Haibo He, Zhineng Zhu, and Mingwei Ding.

Finally, my parents, older brother and my girlfriend, have always been the best support in my life. I would like to express my appreciation to them. My mother’s unconditional love

and continual encouragement have been the source of my strength for years even after she passed away. I dedicate this thesis to you with all my love.

Table of Contents

List of Figures

List of Tables

Symbols and Abbreviations

ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
CAD	Computer Aided Design
CLB	Configurable Logic Block
DFG	Data Flow Graph
DOCSIS	Data Over Cable Service Interface Specification
FEC	Forward Error Correction
FPGA	Field Programmable Gate Array
GF	Galois Field
IP	Intellectual Property
ITU	International Telecommunication Union
LFSR	Linear Feed Shift Register
LSB	Least Significant Bit
LUT	Look Up Table
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MPEG	Moving Picture Experts Group
MSB	Most Significant Bit
PN	Pseudorandom Noise
QAM	Quadrature Amplitude Modulation
RAM	Random Access Memory

RISC	Reduced Instruction Set Computer
RS	Reed-Solomon
TCM	Trellis Coded Modulation
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration

Chapter 1 Introduction

1.1 Motivation

Cable communications system has become the major technology for the broadband internet access. Data Over Cable Service Interface Specification (DOCSIS) compatible cable modem chips from Broadcom and Texas Instruments dominate this competitive market. Xilinx has recently released Intellectual Property (IP) core for the key module, International Telecommunication Union (ITU) J.83 modulator, targeted on FPGA device to allow the other vendors to integrate this broadband access module to their products.

This thesis approaches the cable terminal system design on FPGA from the perspective that low power is most desirable feature. A number of choices, including full custom, standard cell, and FPGA, exist when implementing a digital integrated circuit design. Full custom design is time consuming and can achieve highest performance with fixed functionalities. The standard cell library approaches shorten the design cycle by scarificing performance. The FPGA based design has same critical advantages over these two methods by offering higher flexibility and shorter design time. With reprogrammable features, the design bugs can be fixed with minimal effort and cost. This greatly reduces the design cost and time, especially for low volume design.

Moore's law, which is often stated as the doubling of transistor performance and quadrupling of the number of devices on a chip every three years, has been achieved through scaling of the MOSFET [Doyle02]. Design density and speed are gained through the advances in the semiconductor process technology. Though the performance of the programmable logic lags behind the full custom design [Kusse98], the high performance and high density FPGA embedded with RISC processor is available, for example, in Xilinx Virtex II. Area and speed may then not be the first design priority for the FPGA in many applications, such as cable communication terminal system, and the low-power design becomes a dominant cost factor. As a result, new applications emerge by applying the low power design technology to FPGA based design. For instance, in cable telephone, a desirable feature is obtaining the power from the cable.

FPGAs are traditionally considered as power-hungry devices since their programmability is provided by a large amount of the routing resources and switches that consume a lot of dynamic switching power. Low power design for VLSI has received a lot of attention due to the proliferation of mobile and portable devices. Nevertheless, research in low power FPGA design has only recently gained its popularity due to its unique features in terms of flexibility and signal processing capabilities [Kusse98, George99]. However, its energy inefficiency is a limiting factor for the wide acceptance in mobile and portable applications. In addition, it is difficult to estimate the power consumption during the early phase of the design due to the lack of detailed information about the FPGA circuits. Thus the low power design of FPGA remains a challenging problem.

1.2 Research goals

The goal of this research is to apply the existing low design techniques to digital modulator design on FPGA and to gain a better understanding of the effectiveness of these design methods. The research is focused on the design of several key modules of the digital modulator and optimizing the design for low power purpose at the algorithm and arithmetic levels.

1.3 Thesis organization

This thesis is organized into six chapters. Chapter 2 presents the system specifications of the digital baseband modulator and describes the components for this digital baseband modulator according to the ITU J.83 standard. In this chapter, the framing structure, channel coding, and channel modulation for a digital multi-service television distribution system that is specific to a cable channel is presented. The chapter also covers the features of both 64- and 256-QAM modulation schemes.

Chapter 3 discusses the sources of power dissipation, reviews the existing low power design techniques at the algorithmic and the architectural levels, examines their applicability to FPGA technology and finally derives the design strategy for this digital modulator design.

Chapter 4 applies the low power design techniques to the design of the several key modules including MPEG framer, Reed Solomon Encoder, interleaver, and TCM modulator. Design space is explored and several new structures are proposed.

Chapter 5 provides the simulation results of designed circuits, verifying their performance according to specifications.

Chapter 6 gives a conclusion of the work done, summarizes major results obtained for the thesis and discusses the future work.

Chapter 2 Specification for Digital Baseband Modulator Design

2.1 Introduction

The specification of the ITU Recommendation J.83 Annex B is presented in this chapter. This specification describes the framing structure, channel coding, and channel modulation for a digital multi-service television distribution system that is specific to a cable channel.

Two types of the modulation schemes are used, Quadrature Amplitude Modulation (QAM) with 64-point signal constellation (64QAM) and with 256-point signal constellation. In this chapter, the common features and differences of both modulation schemes are presented.

The design of the modulation, interleaving and coding is based upon testing and characterization of cable systems in North America using 6MHz channeling [ITU J.83]. Concatenated coding schemes, including two encoders and an interleaver are used in Forward Error Correction (FEC) due to their high bit error rate performances with low complexity and low overhead.

MPEG-2 transport layer data stream is the input to this digital modulator. But the synchronization of the MPEG stream is independent from the FEC synchronization,

which can avoid the interference with ATM synchronization. There are two modes supported: Mode 1 has a symbol rate of 5.057 Msymbols/s and it is used for 64-QAM; Mode 2 has a symbol rate of 5.361 Msymbols/s and is typically employed for 256-QAM [ITU J.83]. This system can be easily upgraded to higher order QAM extensions.

The appropriate error correction and demodulation are determined by the expected channel error statistics and distortion characteristics. The cable channel is primarily regarded as a bandwidth-limited linear channel, with a balanced combination of white noise, interference, and multi-path distortion and the QAM technique used, together with adaptive equalization and concatenated coding is well suited to this application and channel [ITU J.83].

The block diagram of the whole digital modulator is shown in Fig.2.1. The specification of each processing block is presented as follows.

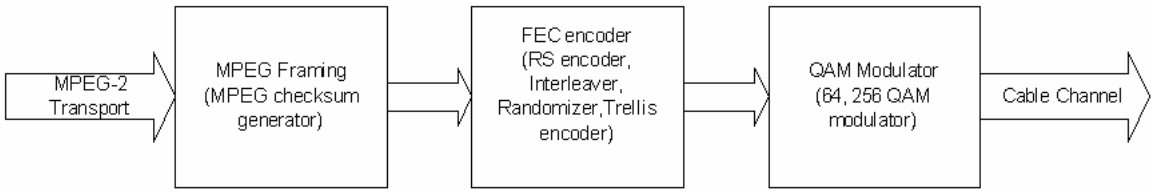


Fig. 2.1 Cable transmitter block diagram

2.2 MPEG-2 transport layer

The MPEG-2 transport layer is defined in Reference [ITU H.222]. Each frame in the transport layer for MPEG-2 has 4 bytes header, which contains one synchronization byte

$47_{HEX,}$ three bytes service identification, scrambling and control information, and 184 bytes of data.

2.3 MPEG-2 transport framing

The MPEG transport framing provided a synchronization mechanism for the robust communication in this layer. Out of the 188-byte data steame in serial format, most significant bit (MSB) is first processed by this block. In addition to the sync byte, a parity checksum is used to improve the delineation functionality and add the additional error detection capability. This checksum is calculated over the 187 bytes following the sync byte. Linear Feedback Shift Register (LFSR) $f(x)$ is used to generate this checksum. According to specifications provided in [ITU J.83], the LFSR is based on the following equations:

$$f(x) = [1 + x^{1496} b(x)] / g(x) \qquad (2.1)$$

Where $g(x) = 1 + x + x^5 + x^6 + x^8$ and $b(x) = 1 + x + x^3 + x^7$

The structure of this checksum generation function is shown in Fig. 2.2. All the additional operations are modulo 2 and the LFSR is reset to all zeros before the processing of a new frame. The 187-byte data is fed into the LFSR in a serial fashion, the MSB first at the bit rate. The feedback network implements the g(x). An offset of 67_{HEX} is added to this checksum result for improved autocorrelation properties, and causes a 47_{HEX} result to be produced during a syndrome decode operation when a valid code word is presented [ITU J.83]. The 8-bit checksum is appended to the 187-byte data and transmitted in MSB first.

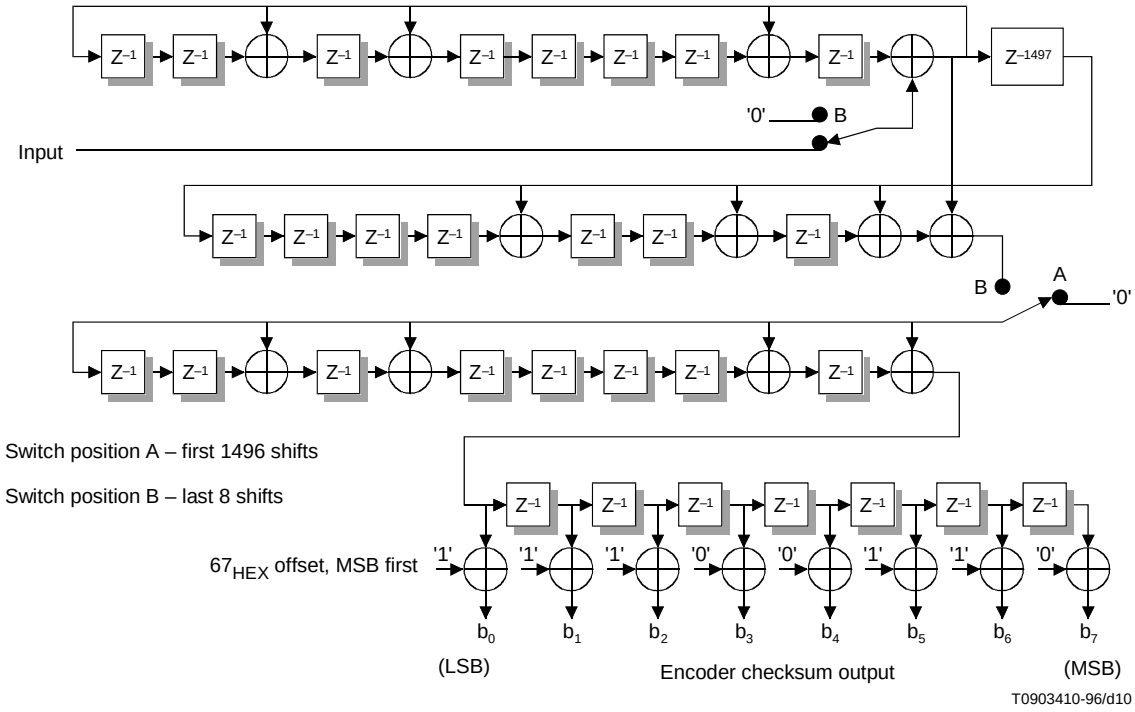


Fig. 2.2 Checksum generator for the MPEG-2 sync byte encoder [ITU J.83]

2.4 Forward error correction

Two levels of coding, an inner coder and outer coder, are employed in the Forward Error Correction (FEC) to achieve high error correcting performance. The inner coder interfaced with the modulator and the channel is configured to correct most of the channel errors, and the outer coder with a high code rate reduced the probability of error to the specified level [Sklar01]. Interleaving and randomization are used between the inner and outer coder to prevent the degradation of error correcting performance caused by correlated errors among successive symbols. Then the overall implementation

complexity is reduced to achieve a low error rate by using the concatenated coding scheme. As shown in Fig.2.3, the FEC consists of four processing blocks: Reed Solomon encoding, Convolutional interleaving, Randomizer, and Trellis encoder. FEC has its own synchronization mechanism and is independent from the input data protocols.

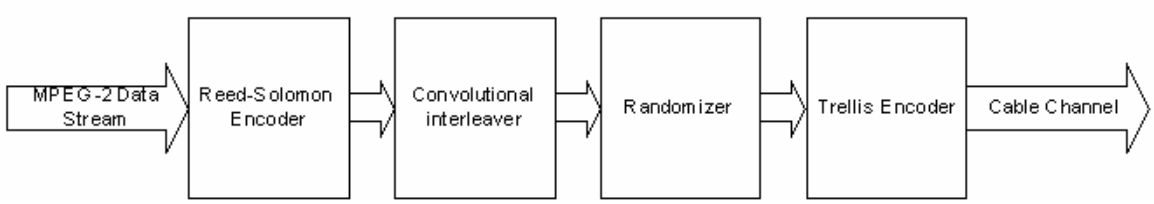


Fig. 2.3 Forward Error Correction block diagram

Two types of coding schemes and interleaving techniques are employed to ensure the reliability of the data transmission over the cable channel:

- Extended (128, 122) Reed Solomon (RS) coder – Provides block encoding and has the capability of correcting up to three symbols within an RS block.
- Convolutional interleaver – The input symbols delayed by different length result in spreading the errors over several code words. RS encoding together with interleaving provide for the correction of the burst errors that decoder can not correct [Sklar01]
- Randomizer – Randomizes the data over the channel adding the additional information to allow effective QAM demodulator synchronization [ITU J.83].

- Trellis Encoder – Provides convolutional encoding.

2.4.1 Reed-Solomon coding

Reed-Solomon codes are block-based error correcting codes and their error correcting ability is determined by the bit redundancy, (n, k). Reed-Solomon code can correct up to (n-k)/2 symbol errors. Reed-Solomon codes are particularly well suited to correct the burst errors and are widely applied in communication. A systematic RS coder (128, 122) over Galois Field (GF) (128) is employed to code the data of MPEG transport layer. This extended RS code can correct up to t=3 symbols errors contained in a codeword, where 2t = 128-122. Both 64-QAM and 256-QAM use this RS code.

The implementation of the extended RS encoder is described as follows. The extended RS code over GF(128) is constructed with the systematic encoder. The primitive polynomial used to generate all the elements over GF(128) is:

$$p(x) = x^7 + x^3 + 1 \tag{2.2}$$

The (128,122) RS codeword is generated using the generating polynomial:

$$\begin{aligned} g(x) &= (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4)(x+\alpha^5) \\ &= x^5 + \alpha^{52}x^4 + \alpha^{116}x^3 + \alpha^{119}x^2 + \alpha^{61}x + \alpha^{15} \end{aligned} \tag{2.3}$$

Where, the primitive element α is a root of a primitive polynomial of the field, i.e. $p(\alpha) = 0$ [ITU J.83].

The message block including 122, 7-bit symbols can be expanded in a polynomial format as follows

$$m(x) = m_{121}\, x^{121} + m_{120}\, x^{120} + \ldots + m_1\, x^1 + m_0 \tag{2.4}$$

First, five parity symbols can be obtained as

$$r(x) = r_4 x^4 + r_3 x^3 + r_2 x^2 + r_1 x + r_0 \tag{2.5}$$

Then, the first parity symbols in a systematic Reed-Solomon codeword are given by the remainder of $m(x) \cdot x^5 \bmod g(x)$.

The message block appended with the five parity symbols forms a new polynomial referred to as an even multiple of the generator polynomial, and can be represented as the following polynomial:

$$c(x) = m_{121}\, x^{126} + m_{120}\, x^{125} + \ldots + m_1\, x^6 + m_0 x^5 + r_4 x^4 + r_3 x^3 + r_2 x^2 + r_1 x + r_0 \tag{2.6}$$

The extended parity symbol c_{128} is generated by evaluating the code word at the sixth power of α [ITU J.83].

$$c_{128} = c(\alpha^6) \tag{2.7}$$

The code $c(x)$ appended with this extended parity symbol forms the final 128 symbol Reed-Solomon block:

$$\begin{aligned} code(x) &= xc(x) + c_{128} \\ &= m_{121}\, x^{127} + m_{120}\, x^{126} + \ldots + m_1\, x^7 + m_0 x^6 + r_4 x^5 + r_3 x^4 + r_2 x^3 + r_1 x^2 + r_0 x + c_{128} \end{aligned} \tag{2.8}$$

The codeword of this (128, 122) RS encoder is transmitted in the order of MSB first, $m_{121} m_{120} \ldots m_1 m_0 r_4 r_3 r_2 r_1 r_0 c_{128}$ with m_{121} transmitted as the first bit.

2.4.2 Interleaving

The Forney Convolutional interleaver, employed between the RS encoder and the randomizer, has the capability to correct the burst errors. The convolutional interleaver is used for both 64-QAM and 256-QAM.

Fig.2.4 shows the operation of the convolutional interleaver. The input symbols from RS encoder appear at the input commutator and the interleaved data is output from the output commutator. An (I, J) interleaver, has I branches. Each branch has a bank of registers (the width of the register is 7, the same as the RS symbol size) and each register has a delay of J symbol periods. Symbol period is given and equals to 7 bit clock periods. The register clock depends on its width. Clock for register width of 7 is 7 clock periods, while for a register width of 1 equals 1 clock period. There is no register at the first branch, labeled with 1. The k-th branch has (k-1) registers and the corresponding path delay equals (k-1)*J symbol periods delay. The input and output commutator can be referred to as the interleaving commutator. These two interleaving commutators are reset to the top-most branch and move to the next branch at the RS symbol frequency. After reaching the last branch, the interleaving commutators rotate back to the top-most branch. A series of bad symbols caused by the burst noise in the channel are spread over many RS blocks by the de-interleaver, such that the resultant symbol errors per block are within the range of the RS decoder correction capability [ITU J.83].

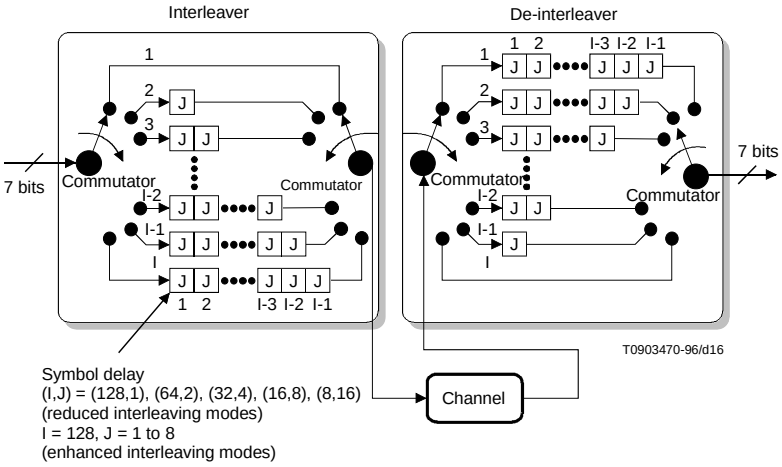


Fig. 2.4 Interleaving functional block diagram [ITU J.83]

Two distinct operating modes, level 1 and level 2, provide two distinct levels of interleaving capability. In level 1, a single interleaving depth (128, 1) is used in the 64-QAM modulation schemes. The level 2 shall encompass 64-QAM and 256-QAM transmission, and will for both modulation schemes be capable of supporting variable interleaving [ITU J.83]. The interleaving depths is enlarged and reduced by a certain factor. For instance, the reduced modes are (64,2), (32, 4), …, (8, 16). The synchronization bytes of the FEC frame contains the interleaving configuration parameters, which are available for the deinterleaver at the receiver side.

Table 2.1 describes the interleaver parameters for level 1 operation, with associated latency and burst protection [ITU J.83]. Table 2.2 describes the decoding of the 4-bit in-

band control word into the I and J interleaving parameters for level 2 operation, also with associated burst protection and latency [ITU J.83].

Table 2.1 Level 1 interleaving [ITU J.83]

Control word (4 bits)	**I (# of taps)**	**J (increment)**	**Burst protection**	**Latency**
Xxxx	128	1	95 μs	4.0 ms

Table 2.2 Level 2 interleaving [ITU J.83]

Control word (4 bits)	**I (# of taps)**	**J (increment)**	**Burst protection 64-QAM/256-QAM**	**Latency 64-QAM/256-QAM**
0001	128	1	95 μs /66 μs	4.0 ms/2.8 ms
0011	64	2	47 μs /33 μs	2.0 ms/1.4 ms
0101	32	4	24 μs /16 μs	0.98 ms/0.68 ms
0111	16	8	12 μs /8.2 μs	0.48 ms/0.33 ms
1001	8	16	5.9 μs /4.1 μs	0.22 ms/0.15 ms
1011	Reserved			
1101	Reserved			
1111	Reserved			
0000	128	1	95 μs /66 μs	4.0 ms/2.8 ms
0010	128	2	190 μs /132 μs	8.0 ms/5.6 ms
0100	128	3	285 μs /198 μs	12 ms/8.4 ms
0110	128	4	379 μs /264 μs	16 ms/11 ms
1000	128	5	474 μs /330 μs	20 ms/14 ms
1010	128	6	569 μs /396 μs	24 ms/17 ms
1100	128	7	664 μs /462 μs	28 ms/19 ms
1110	128	8	759 μs /528 μs	32 ms/22 ms

2.4.3 Randomization

As shown in Fig.2.5, the randomizer is the third function block in the FEC block diagram. The randomizer provides for even distribution of the symbols in the constellation, which enables the demodulator to maintain proper lock [ITU J.83]. 7-bit Pseudorandom Noise (PN) sequence is added to the data symbols to generate a random transmitted sequence.

For both 64- and 256-QAM, the randomizer is reset to all ones during the FEC frame trailer, so the randomization is not applied to the trailer. The randomizer can be represented as a polynomial over the GF(128). It can be realized as a feed back network with the finite field addition and delay terms. The randomizer is first initialized to be in "111_1111" state. All the operations are performed over the GF(128).

$$f(x) = x^3 + x + \alpha^3 \tag{2.9}$$

where α is an root of the primitive polynomial $p(x) = x^7 + x^3 + 1$ and $p(\alpha) = 0$.

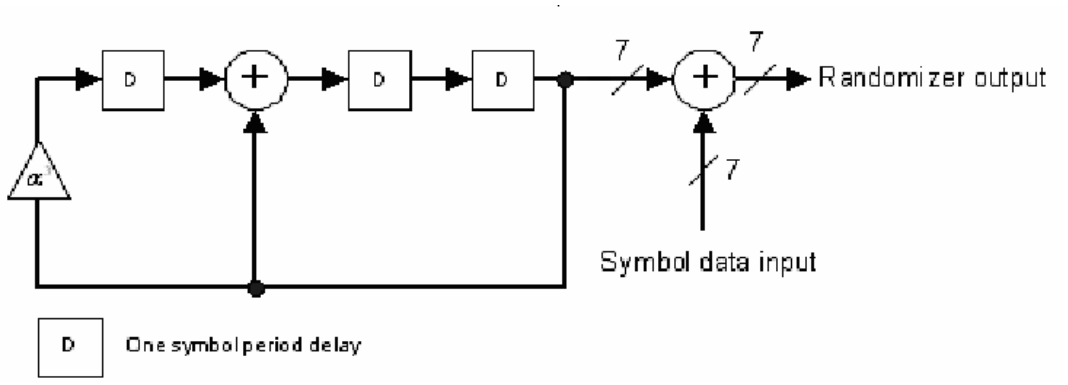


Fig. 2.5 Block diagram of 7-bit symbol randomizer

2.4.4 Trellis coded modulator

Trellis coded modulation (TCM) scheme has the high spectral efficiencies. This efficiency is achieved by expanding the size of the signal constellation, instead of expanding the bandwidth, which results in a high code performance.

2.4.4.1 64-QAM modulation mode

As shown in Fig.2.6, 64 QAM Trellis coded modulator is composed of four types of function blocks: Parser, differential pre-coder, binary puncture convolutional coder and QAM mapper. 28 bits data stream from the randomizer form an input group and the modulator generates 30 bits QAM symbols. So the overall code rate for this modulator is 14/15.

28 bits data groups consist of four 7-bit symbols referred to as ‘A’ and ‘B’ as shown in Fig. 2.7. The first two RS symbols are labeled as ‘A’ and the remaining two RS symbols are labeled as ‘B’. The LSB of ‘A’ and ‘B’ are first differentially pre-coded and then fed into the binary puncture convolutional coder. The convolutional coder generates two groups of five code bits, which are labeled as U5 U4 U3 U2 U1 and V5 V4 V3 V2 V1 from four input bits. These two five bits data streams are sent to the QAM mapper. The MSB of ‘A’ and ‘B’ are uncoded and sent to the QAM mapper directly. The whole bit stream is aligned with the 7-bit RS symbol and is serialized MSB first.

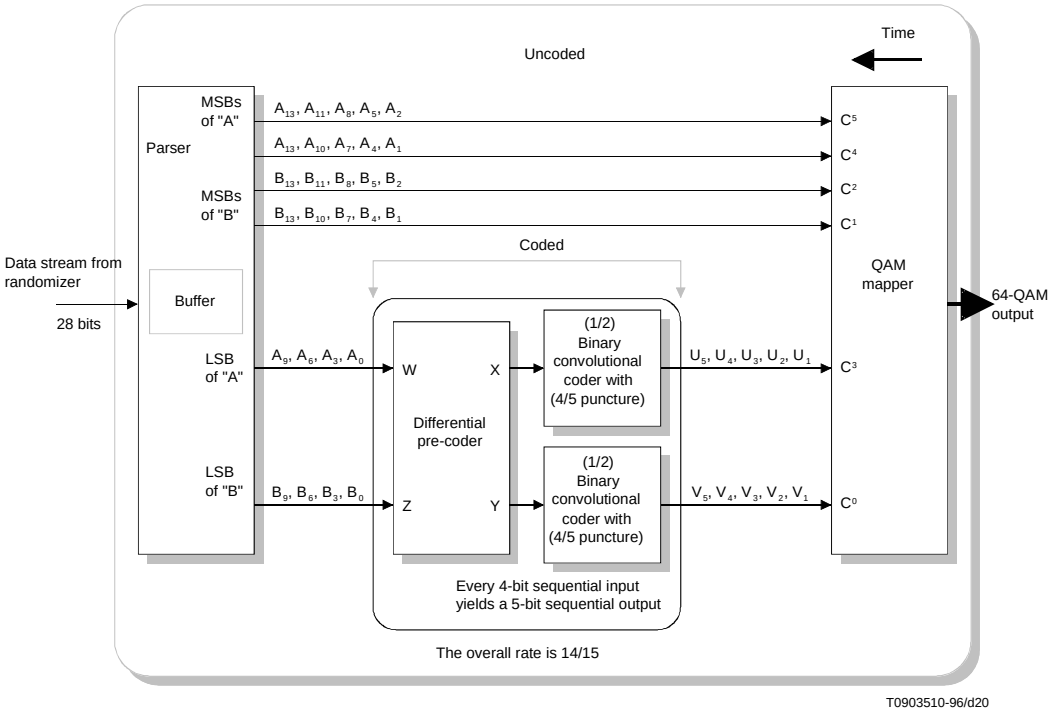


Fig. 2.6 64-QAM trellis coded modulator block diagram [ITU J.83]

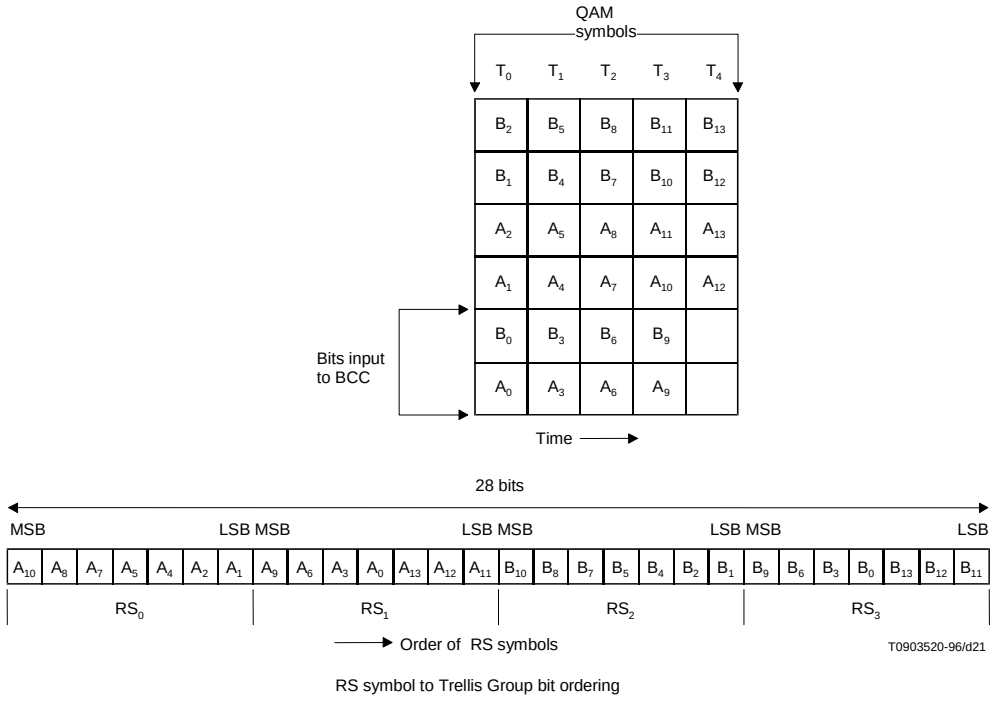


Fig. 2.7 64-QAM trellis group [ITU J.83]

2.4.4.2 256-QAM modulation mode

256-QAM trellis coded modulator is shown in Fig.2.8. As the 64-QAM modulator, 256-QAM modulator consists of four function blocks: data formatter, differential pre-coder, binary puncture convolutional coder and QAM mapper. There is slight difference between the 256-QAM and 64-QAM modulator. 38 bits data stream from the randomizer forms an input group. The modulator generates 40 bits 5-QAM symbols, so the overall code rate for this modulator is 19/20.

As shown in Fig. 2.9, the input 38 bits data have two different formats, which can be defined as non-sync group and sync group. In a non-sync group, only data bits form the input 38 bits data group. For a sync group, 30 bits data and 8 bits sync byte, which appear at the end of the FEC frame, contribute to the 38 bits data group. The MSB of the ‘A’ and ‘B’ are uncoded and fed into the QAM mapper directly. The so-called LSB of ‘A’ and ‘B’ are first differentially pre-coded and then sent to the binary puncture convolutional coder. The convolutional coder generates two groups of five code bits, which are labeled as U5 U4 U3 U2 U1 and V5 V4 V3 V2 V1 from four input bits. These two five bits data streams are sent to the QAM mapper. The difference between 64-QAM modulator and 256-QAM modulator lies in the first functional block. Data formatter, instead of data parser is used in 256-QAM modulator, to deal with non-sync group and sync group data formats. For the non-sync group the first bit of the RS symbol bit stream forms the LSB of ‘A’ and the second bit of the RS symbol bit stream is referred to as the LSB of ‘B’. For sync group, the 0^{th}, 2^{nd}, 4^{th} and 6^{th} bit of sync byte forms the ‘LSB’ of A and 1^{st}, 3^{rd}, 5^{th}, 7^{th} bits are referred as the ‘LSB’ of ‘B’. The bit stream is aligned with the 7-bit RS symbol and is serialized with the MSB first.

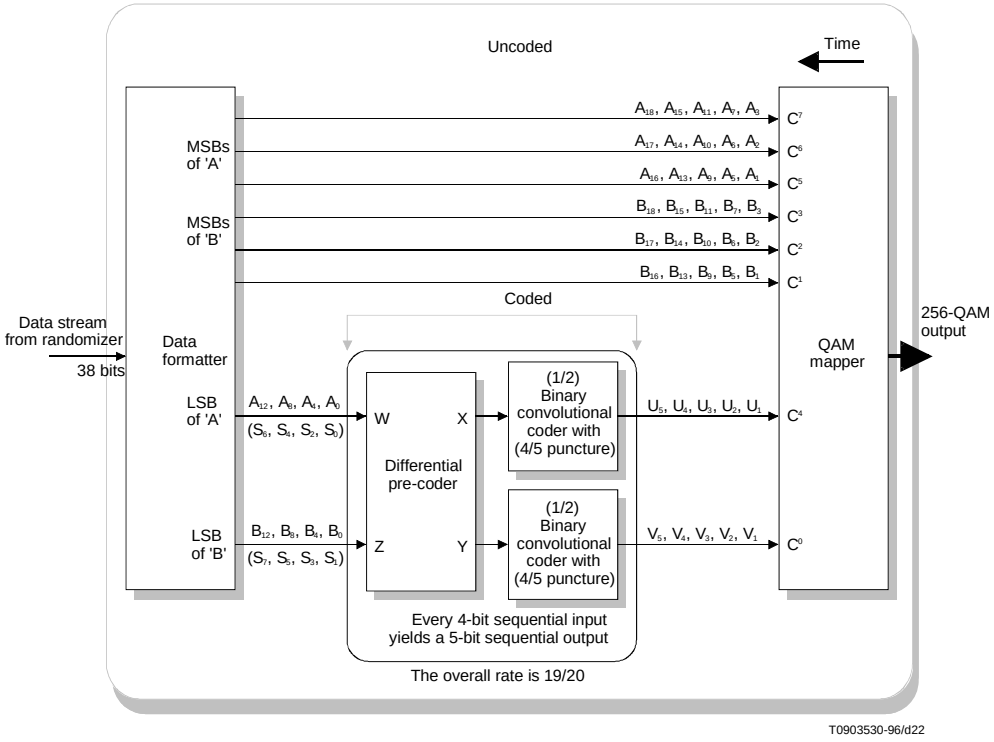


Fig. 2.8 256-QAM trellis coded modulator block diagram [ITU J.83]

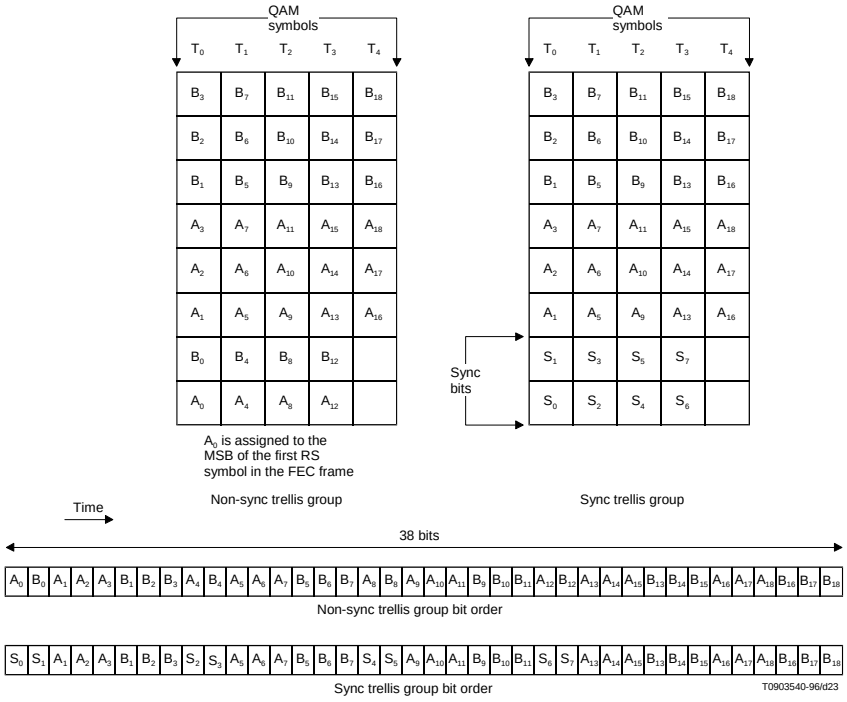


Fig. 2.9 256-QAM sync and non-sync trellis group [ITU J.83]

2.4.4.3 Rotationally invariant pre-coding

As shown in Fig.2.10, differential pre-encoder is employed to take advantage of the "rotationally invariant" property of the nonlinear, trellis encoder. For a code with 90° invariant property, the correct sequence can be recovered after decoding even if the decoder locks on the wrong phase (multiple of 90°). It is a desirable feature for a robust modem. Non-rotationally invariant coding requires resynchronization of the FEC when

the carrier phase tracking changes quadrant alignment, leading to a burst of errors at the FEC output [ITU J.83].

Both 64-QAM and 256-QAM modulation schemes use the rotationally invariant encoding. In Fig. 2.6, the inputs to the differential encoder are the 3rd and the 6th of 6-bit symbols, and C^3 and C^0 are the outputs of the differential encoder bits in 64-QAM; in Fig. 2.8 the inputs to the differential encoder are the 4th and the 8th of 6-bit symbols, and C^4 and C^0 is the output of the differential encoder bits in 256 QAM.

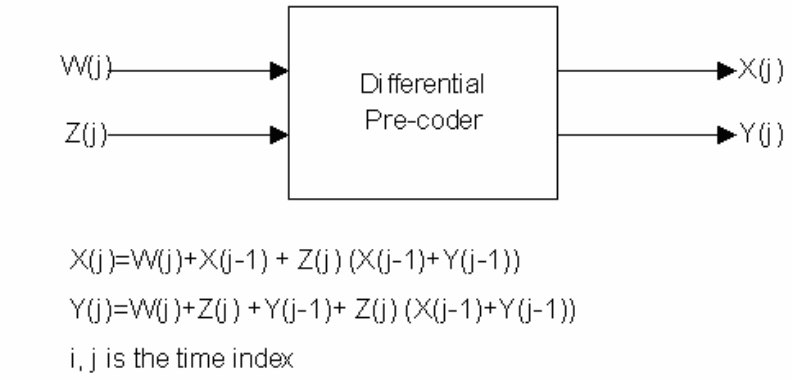
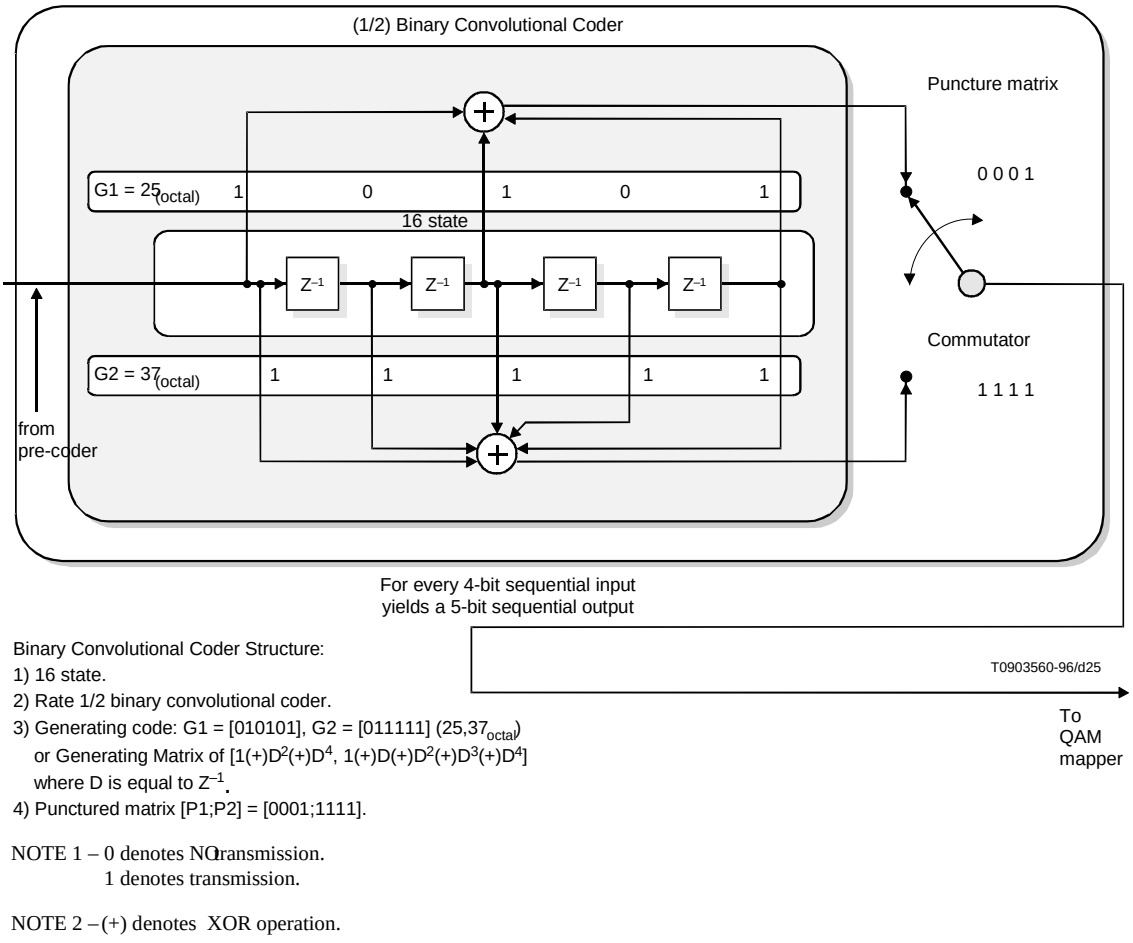


Fig. 2.10 Block diagram of the differential pre-encoder [ITU J.83]

2.4.4.4 Binary Convolutional Coder

Fig. 2.11 shows the binary non-systematic convolutional coder with constraint length equal to 4. There are two output branches, so the code rate is ½. The generators that characterize the encoding functions are G1 = 010 101 and G2 = 011 111 ($25,37_{octal}$). Four

data bits form a trellis group and are fed into this encoder at the bit rate. When the first bit is shifted into the encoder, encoded data bits are output from both upper and lower branches. Before the last data bit shifts out of the shift registers, the encoder generates eight bits data on both output branches. The output switch samples the data bits at the output branches according to the puncture matrix. The puncture matrix defines when the data sampled will be transmitted. The puncture matrix for this convolutional coder is $\begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ ("0" denotes NO transmission, "1" denotes transmission). It means the output switch is reset to the upper branch when the first bit is shifted into the encoder and move to the lower branch when all the four bits are shifted into the encoder. The puncture matrix essentially converts the rate 1/2 encoder to the rate 4/5, since only 5 of the 8 encoded bits are retained after puncturing [ITU J.83].



Binary Convolutional Coder Structure:
1) 16 state.
2) Rate 1/2 binary convolutional coder.
3) Generating code: G1 = [010101], G2 = [011111] ($25,37_{octal}$)
or Generating Matrix of [1(+)D^2(+)D^4, 1(+)D(+)D^2(+)D^3(+)D^4]
where D is equal to Z^{-1}.
4) Punctured matrix [P1;P2] = [0001;1111].

NOTE 1 – 0 denotes NO transmission.
1 denotes transmission.

NOTE 2 – (+) denotes XOR operation.

Fig. 2.11 Punctured Binary Convolutional Coder [ITU J.83]

2.4.4.5 QAM constellation mapping

For 64-QAM, 6-bit data from the MSB of ‘A’ and ‘B’ and outputs of convolutional encoder are converted to the 6-bit constellation symbol according to Fig. 2.12 in the QAM mapper.

For 256-QAM, 8-bit data from the MSB of ‘A’ and ‘B’ and outputs of convolutional encoder are converted to the 6-bit constellation symbol according to Fig. 2.13 in the QAM mapper.

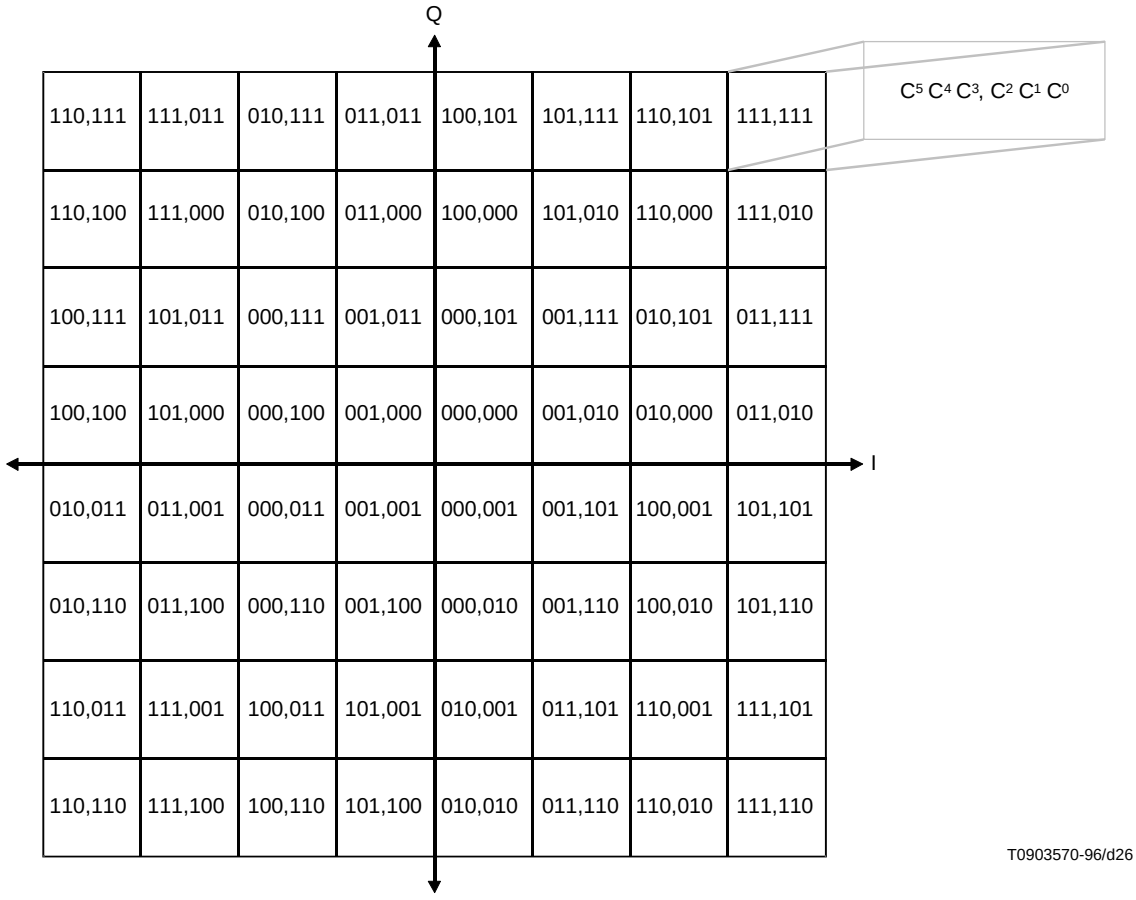
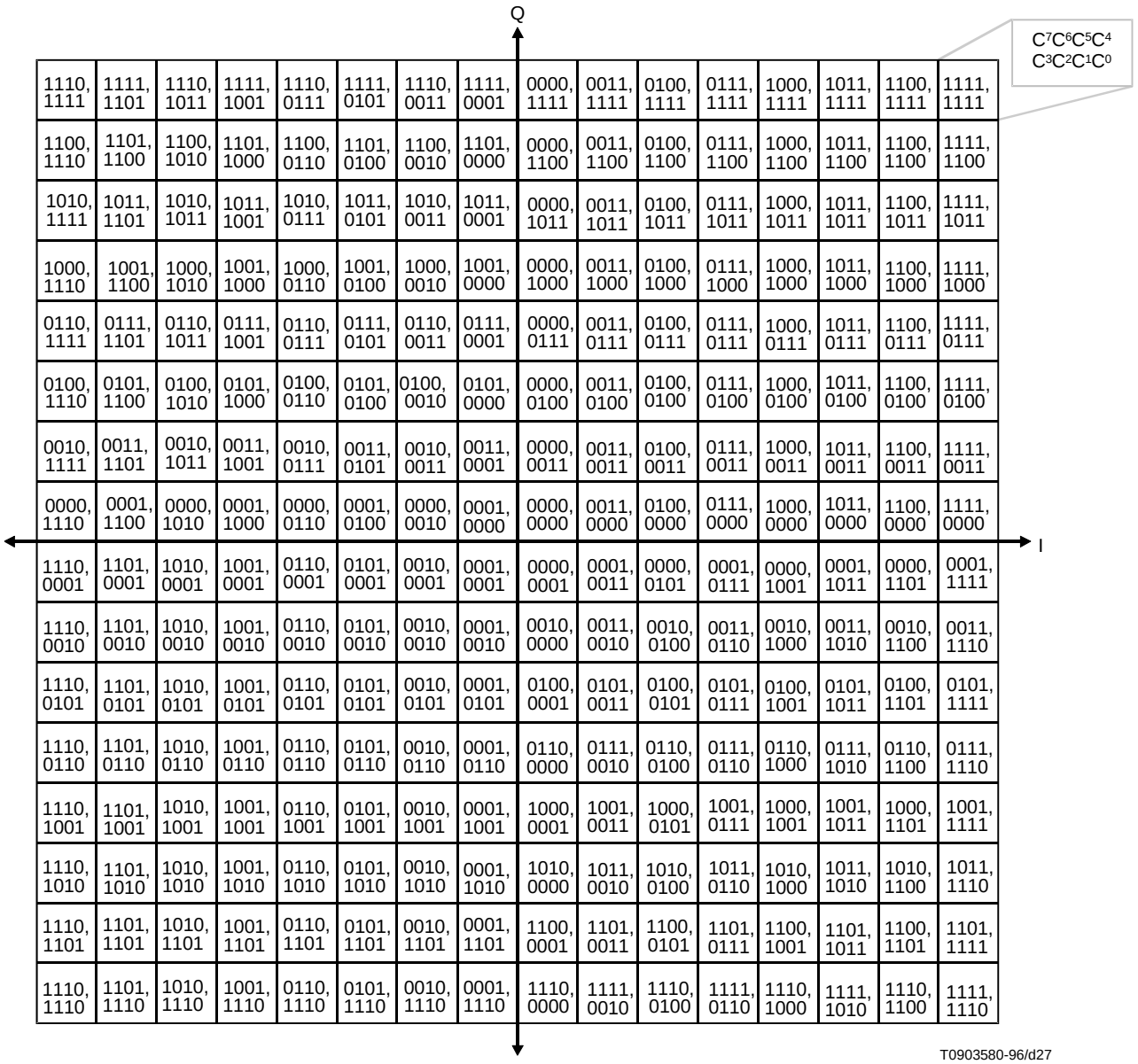


Fig. 2.12 64-QAM constellation [ITU J.83]



1110, 1111	1111, 1101	1110, 1011	1111, 1001	1110, 0111	1111, 0101	1110, 0011	1111, 0001	0000, 1111	0011, 1111	0100, 1111	0111, 1111	1000, 1111	1011, 1111	1100, 1111	1111, 1111
1100, 1110	1101, 1100	1100, 1010	1101, 1000	1100, 0110	1101, 0100	1100, 0010	1101, 0000	0000, 1100	0011, 1100	0100, 1100	0111, 1100	1000, 1100	1011, 1100	1100, 1100	1111, 1100
1010, 1111	1011, 1101	1010, 1011	1011, 1001	1010, 0111	1011, 0101	1010, 0011	1011, 0001	0000, 1011	0011, 1011	0100, 1011	0111, 1011	1000, 1011	1011, 1011	1100, 1011	1111, 1011
1000, 1110	1001, 1100	1000, 1010	1001, 1000	1000, 0110	1001, 0100	1000, 0010	1001, 0000	0000, 1000	0011, 1000	0100, 1000	0111, 1000	1000, 1000	1011, 1000	1100, 1000	1111, 1000
0110, 1111	0111, 1101	0110, 1011	0111, 1001	0110, 0111	0111, 0101	0110, 0011	0111, 0001	0000, 0111	0011, 0111	0100, 0111	0111, 0111	1000, 0111	1011, 0111	1100, 0111	1111, 0111
0100, 1110	0101, 1100	0100, 1010	0101, 1000	0100, 0110	0101, 0100	0100, 0010	0101, 0000	0000, 0100	0011, 0100	0100, 0100	0111, 0100	1000, 0100	1011, 0100	1100, 0100	1111, 0100
0010, 1111	0011, 1101	0010, 1011	0011, 1001	0010, 0111	0011, 0101	0010, 0011	0011, 0001	0000, 0011	0011, 0011	0100, 0011	0111, 0011	1000, 0011	1011, 0011	1100, 0011	1111, 0011
0000, 1110	0001, 1100	0000, 1010	0001, 1000	0000, 0110	0001, 0100	0000, 0010	0001, 0000	0000, 0000	0011, 0000	0100, 0000	0111, 0000	1000, 0000	1011, 0000	1100, 0000	1111, 0000
1110, 0001	1101, 0001	1010, 0001	1001, 0001	0110, 0001	0101, 0001	0010, 0001	0001, 0001	0000, 0001	0001, 0011	0000, 0101	0001, 0111	0000, 1001	0001, 1011	0000, 1101	0001, 1111
1110, 0010	1101, 0010	1010, 0010	1001, 0010	0110, 0010	0101, 0010	0010, 0010	0001, 0010	0010, 0000	0011, 0010	0010, 0100	0011, 0110	0010, 1000	0011, 1010	0010, 1100	0011, 1110
1110, 0101	1101, 0101	1010, 0101	1001, 0101	0110, 0101	0101, 0101	0010, 0101	0001, 0101	0100, 0001	0101, 0011	0100, 0101	0101, 0111	0100, 1001	0101, 1011	0100, 1101	0101, 1111
1110, 0110	1101, 0110	1010, 0110	1001, 0110	0110, 0110	0101, 0110	0010, 0110	0001, 0110	0110, 0000	0111, 0010	0110, 0100	0111, 0110	0110, 1000	0111, 1010	0110, 1100	0111, 1110
1110, 1001	1101, 1001	1010, 1001	1001, 1001	0110, 1001	0101, 1001	0010, 1001	0001, 1001	1000, 0001	1001, 0011	1000, 0101	1001, 0111	1000, 1001	1001, 1011	1000, 1101	1001, 1111
1110, 1010	1101, 1010	1010, 1010	1001, 1010	0110, 1010	0101, 1010	0010, 1010	0001, 1010	1010, 0000	1011, 0010	1010, 0100	1011, 0110	1010, 1000	1011, 1010	1010, 1100	1011, 1110
1110, 1101	1101, 1101	1010, 1101	1001, 1101	0110, 1101	0101, 1101	0010, 1101	0001, 1101	1100, 0001	1101, 0011	1100, 0101	1101, 0111	1100, 1001	1101, 1011	1100, 1101	1101, 1111
1110, 1110	1101, 1110	1010, 1110	1001, 1110	0110, 1110	0101, 1110	0010, 1110	0001, 1110	1110, 0000	1111, 0010	1110, 0100	1111, 0110	1110, 1000	1111, 1010	1110, 1100	1111, 1110

Fig. 2.13 256-QAM constellation [ITU J.83]

2.5 Modulation and demodulation

2.5.1 QAM characteristics

The cable transmission format is summarized in Table 2.3 for 64-QAM and 256-QAM and Table 2.4 contains a summary of the pertinent characteristics of the variable interleaving modes [ITU J.83].

Table 2.3 Cable transmission format [ITU J.83]

Parameter	**64-QAM format**	**256-QAM format**
Modulation	64-QAM, rotationally invariant coding	256-QAM, rotationally invariant coding
Symbol size	3 bits for "I" and 3 bits for "Q" dimensions	4 bits for "I" and 4 bits for "Q" dimensions
Transmission band	54 to 860 MHz	54 to 860 MHz
Channel spacing	6 MHz	6 MHz
Symbol rate	5.056941 Msps ± 5 ppm	5.360537 Msps ± 5 ppm
Information bit rate	26.97035 Mbps ± 5 ppm	38.81070 Mbps ± 5 ppm
Frequency response	Square root raised cosine filter (Roll-off ≈ 0.18)	Square root raised cosine filter (Roll-off ≈ 0.12)
FEC framing	42-bit sync trailer following 60 RS blocks	40-bit sync trailer following 88 RS blocks
QAM constellation mapping	6 bits per symbol	8 bits per symbol
NOTE – These values are specific to 6 MHz channel spacing. Additional sets of values for differing channel spacing are under study.		

Table 2.4 Variable interleaving modes [ITU J.83]

	Level 1	**Level 2**
QAM format	64-QAM (see Table 2.3)	64- or 256-QAM (see Table 2.3)
Interleaving	Fixed interleaving I = 128 J = 1	Variable interleaving I = 128,64,32,16,8 J = 1,2,3,4,5,6,7,8,16

Chapter 3 Low Power Design

3.1 Introduction

In this chapter, sources of the power consumption for FPGA are examined based on the existing research work [Shang02, Kusse98] and the existing low power design techniques [Chandrakasan95], and their applicability to the low power FPGA design is investigated. Finally, the low power design strategy for the digital modulator is derived.

3.2 Sources of power consumption

First, the power dissipation of the Xilinx Virtex II is discussed by reviewing Shang's recent work [Shang02]. Good understanding of the sources of the power consumption in FPGA is fundamental for the low power FPGA design and it will guide us in the right direction to develop the effective methods for low-power FPGA design.

Before presenting Shang's research results, we first briefly introduce the architecture of Xilinx Virtex chip. The detailed information can be found in Virtex datasheet [Xilinx00]. Virtex FPGA consists of two major types of resources: configurable logic blocks (CLB), and routing resources. CLBs are the computational elements or logic parts of the FPGA, and they are connected by the routing resources. The routing resources include three

types of wires (long lines, hex lines and double lines) and the switches. The long lines can distribute the signals across the device in two directions: vertical and horizontal; the hex lines route signals to every third or sixth CLB blocks in all four directions; the double lines route signals to every first or second CLB block in all four directions [Xilinx00]. The switches connect the CLB's inputs and outputs to these lines and are referred to as the input crossbar and the output crossbar [Shang02]. Also, there are some dedicated global routing resources for the global clock distribution, local routing resources for the carry propagation and the local clock, and direct connect lines.

In the digital CMOS circuits, two types of power consumption, static and dynamic power consumption, exist. Since the design of low power FPGA devices is beyond the research scope of this thesis, only the dynamic power consumption is considered. The dynamic power consumption in the digital CMOS circuits results from charging and discharging the capacitances and can be modeled as [Rabaey02]:

$$P_{dynamic} = \alpha_{0->1} C_L V_{dd} V_{swing} f_{clk} \qquad (3.1)$$

Where C_L is the capacitance, V_{swing} is the voltage swing, f_{clk} is the clock frequency and $\alpha_{0->1}$ is the switching factor. [Shang02] broke down the sources of the dynamic power dissipation and calculated the corresponding capacitance for each resource. Due to his privileged access to the FPGA schematic, he obtained the results on the transistor level by simulation and confirmed these results by measurements. His results of the effective capacitance of each resource are shown in the Table 3.1.

Table 3.1 Capacitance of the resources of Virtex II (2v1000FG256-5) [Shang02]

Type	Resource of Virtex II	Capacitance (PF)
Interconnect Per CLB	IXbar	9.44
	OXbar	5.12
	Double	13.20
	Hex	18.40
	Long(*)	26.10
Logic Per CLB	LUT inputs	26.40
	FF inputs	2.88
	Carray	2.68
Clocking	Global wiring(*)	300
	Local	0.72

* the value may change with the types of the device

Two conclusions can be drawn from the above table. First, the capacitance of the interconnection is comparable to that of the computational element in a CLB. Second, the capacitance of the interconnect line is not proportional to its length. The capacitance of a double line, which extends between two CLB blocks, is almost half of the capacitance of the long line, which goes across the whole chip, in device 2v1000FG256-5; and the capacitance of a double line is close to that of the hex line, which spans six CLB blocks. For the small devices, the long line capacitance becomes smaller, and then there is not much difference between the capacitance of the lines.

It is also reported in [Shang02] that 60% of power is dissipated in the interconnection by experiments with a set of typical designs. Clocking and logic resources consume 14% and 16% of the total power separately. Another important factor that impacts the dynamic

power consumption is the switch factor [Shang02]. Actually, the dynamic or switching power consumption of all these resources varies significantly with the switching activities. As in the VLSI, the clock distribution network is also a primary component of power dissipation.

3.3 Low power design techniques

Existing low power design techniques at two levels used in VLSI are explored and their effectiveness in low power FPGA design is examined in this section based on distribution of dynamic power dissipation in FPGA. The logic level design optimization of FPGA is performed during the synthesis, mapping and placing processes, which is done by Computer Aided Design (CAD) tools, and circuits level design involves the design of the FPGA device itself, so only the algorithmic and architectural levels optimization for low power are discussed in this section.

3.3.1 Algorithm level optimization

The most effective method for low power is performing optimization at the algorithm level. At the algorithm level, the optimization may involve performing algorithm modification, selecting new arithmetic operation or applying transformation described in [Chandrakasan95]. For instance, converting the complex arithmetic operations such as

multiplication and division to simple actions like addition and shifting can reduce the area many times as compared to typical implementation [Starzyk04].

This reduction in area results in the decrease of the effective capacitance, and thus the low power goal is achieved. Another way of reducing power through algorithmic modification is exploiting the advantages of constant values in arithmetic operations [Chandrakasan95]. This low power design technique is used in the design of Reed Solomon (RS) encoder. The finite field multiplication is the bottleneck of the high performance RS encoder. The fixed coefficient not only makes the pipelining of the design possible, but also minimizes the total number of the partial products in the new proposed structure.

Exploiting algorithm parallelism is an effective way in low power design for VLSI [Chandrakasan95]. Parallelizing the design to low power works in VLSI due to the power supply's quadratic impact on the power consumption. However, since the FPGA device is given before the design and the power supply is fixed, then the power supply reduction to low the power consumption is not applicable to FPGA design [Kusse98]. Yet, by properly exploring the concurrency in the data path on the algorithm level, the amount of the logic CLBs resources and interconnections may be reduced. This results from the logic combination by the LUT since each LUT can accommodate any four-input function. The optimum number of replications can be found by exploring the area power product in the design space. The example of this exploration is presented, in the design of message

checksum block of the digital modulator, where concurrency of the algorithm can be revealed by unfolding technique. Parallelizing the design speeds up the design, and the reduction of the area due to logic combination and less interconnection load may lower the power dissipation.

In addition, the algorithm can be reformulated to simplify the control structure and to reduce the amount of the interconnection, which is the dominating factor of the power dissipation in FPGA. This style is suitable for the low power FPGA design. The design example of such algorithm reformulation is presented in the interleaver design.

3.3.2 Architectural level optimization

Techniques of optimizing architecture for low power are examined in this section. In VLSI, parallel architectures can be employed to speed up the design. This creates an opportunity to slow down processing in each parallel block by reducing the supply voltage and thus saving the power. This effective approach results from the power's quadratic dependence on the supply voltage. As discussed in the algorithm level optimization section, the design space can be explored to obtain the optimum number of replications. Combining logic blocks by introducing parallelism on the architectural level reduces total effective capacitance improving the overall performance. At some optimum

number of replication, the total effective capacitance decreases when the logic reduction due to logic combination by LUT dominates.

Pipelining the architecture used successfully in low power VLSI is also suitable for the low power design in FPGA. The increase in effective capacitance introduced by pipeline registers is merely 11% percent. This is because each output of the CLB has an optional registered output and the effective capacitance of this flip-flop is approximately 11% percent of the total logic capacitance, according to the effective capacitance in Shang's table (Table 3.1). The data transfer frequency however can be reduced by a much higher factor and this results in a significant power savings. In addition, the pipelining has the lower area cost advantage over parallelizing. Therefore, the pipelined design is an effective low power design technique for FPGA

Another category for low power design in VLSI at the architectural level is minimizing the switching activities by choosing a proper binary number representation, for instance, switching 2's complement to sign-magnitude representation when the probability of data changing around zero is high [Chandrakasan95]. In addition, resynchronizations can be used to balance the signal paths to minimize the glitching activity [Chandrakasan95]. Due to the similar reasons, all these are suitable to low power FPGA design.

3.4 Low power design strategy for the digital modulator

Considering the application characteristics specifies the low power design strategy. Maintaining a given level of throughput is a common design concept in signal processing and other dedicated applications, in which there is no advantage in performing computing faster than some given rate, since the hardware will simply have to wait until further processing is required [Chandrakasan95]. However, scaling down the power supply voltage used in VLSI to slow down data transfer rate is not applicable to the low power FPGA design. Thus, the low power design strategy for the digital modulator targeted on FPGA is accomplished by slowing down the clock frequency, as well as minimizing the effective switching capacitances by design optimization at the algorithm and architectural levels.

Chapter 4 Low Power Design of Digital Modulator

4.1 Introduction

In this chapter, the low power design techniques are applied to the design of the several key modules including Reed Solomon Encoder, interleaver, TCM modulator and MPEG framer.

4.2 Reed Solomon encoder

The implementation of extended Reed Solomon (RS) encoder is discussed in this section. A systematic (128,122) encoder over GF(2^7) is used in the modulator. Its generator polynomial is g(x)=x^5+g4*x^4+g3*x^3+ g2*x^2+g1*x^1+g0, where g4=alpha^52, g3=alpha^116, g2=alpha^119, g1=alpha^61 and g0=alpha^15, where alpha is the root of the primitive polynomial p(x), i.e. p(alpha)=0. The primitive polynomial p(x)=x^7+x^3+1 generates all elements of GF(2^7).

The straightforward method to implement this Reed Solomon encoder uses a feed back network with finite field arithmetic operations. The block diagram of the encoder is shown in Fig. 4.1. Two types of operations are involved - one is addition, which is simply XOR operation using the standard basis to represent the finite field elements; another one

is multiplication, implemented by the finite field multiplier. The multiplier is the key element and has a great impact on the performance of the RS encoder. The first parity symbols in a systematic Reed-Solomon codeword are given by the remainder of $m(x) \cdot x^5 \bmod g(x)$ .The extended parity symbol c_{128} is generated by evaluating the code word at the sixth power of α [ITU J.83]. So when mux_sel1=1, the first five parity symbols are generated; when mux_sel1=0, they are shifted out and at the same time the extended parity symbols are calculated. Finally, the extended parity symbols are transmitted when mulx_sel2=0.

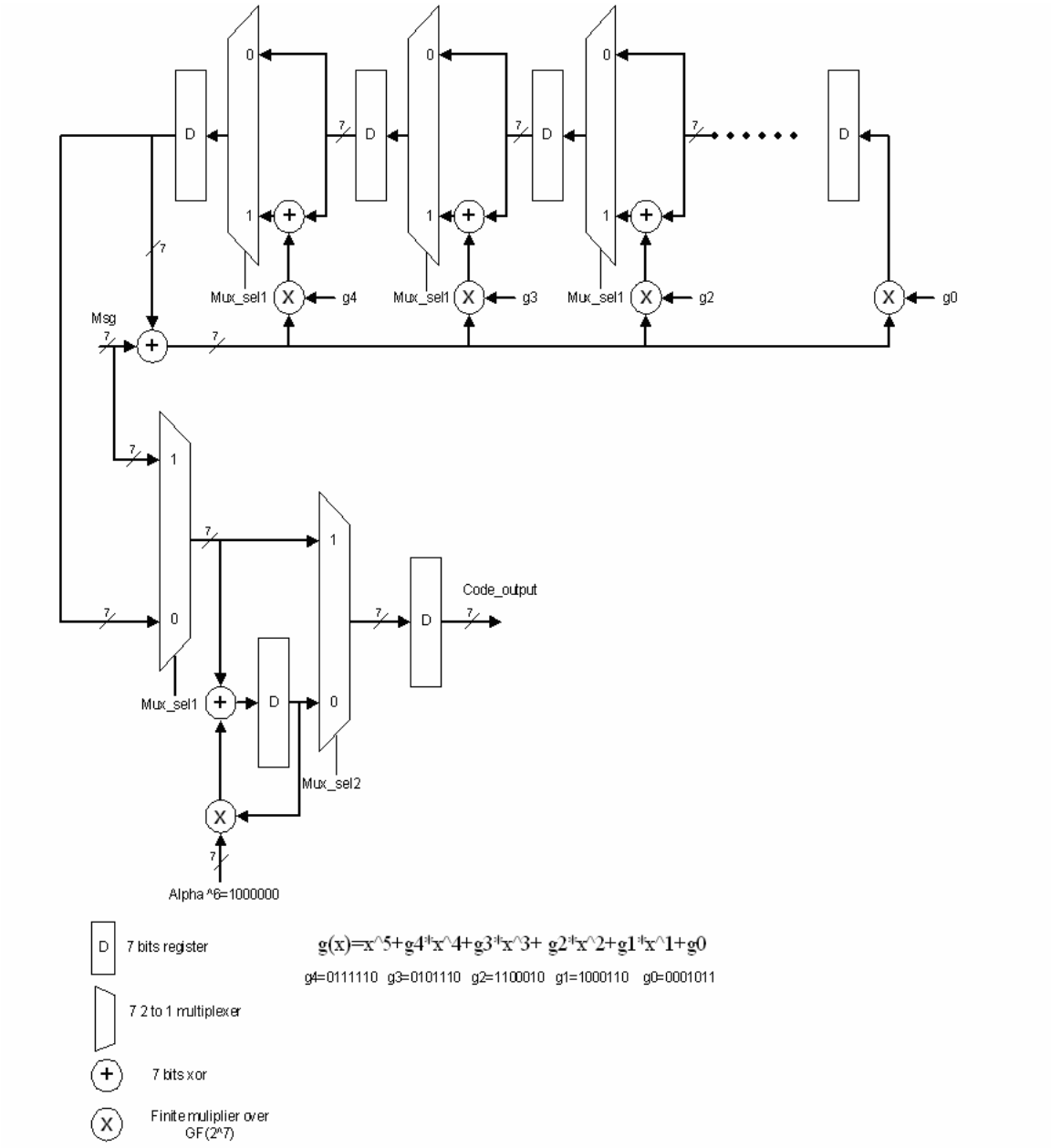


Fig. 4.1 Block diagram for the extended RS Solomon encoder

A number of architectures of this finite field multiplier were available [Berlekamp82] [Massey86, Jain98, Mastrovito91 and Sunar99]. Berlekamp and Massey-ommru bit serial multiplier, Mastrotio bit parallel and semi-systolic array multipliers are the four major

types of architectures. Bit serial structure is too slow for this design, though it has the area advantage over parallel structure. Bit parallel structure has higher performance, and thus lowers the speed requirement to lower the power. Two recently proposed parallel structures are investigated, and a new structure based on the fixed coefficient is proposed as follows.

First, Jain's parallel array-type multiplier is considered. Any nonzero elements in the finite field can be represented in two forms, the exponential form and the polynomial form. Using the polynomial representation based on the standard basis, the finite filed multiplication involves two types of operations, one is a polynomial multiplication and another is a modulo operation. In this structure, the multiplication and the modulo operation are executed alternatively. The algorithm and its implementation in the VLSI are discussed in Sunar's paper [Sunar99]. The notation in this part is adopted from Jain's paper [Jain98].

Let $C = AB \bmod p(x)$, where $A, B, C \in GF(2\wedge m)$ and $p(x)$ is a primitive polynomial over $GF(2\wedge m)$. The polynomial representation is used, so

$$A = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + ... + a_1\alpha^1 + a_0 \quad (4.1)$$

$$B = b_{m-1}\alpha^{m-1} + b_{m-2}\alpha^{m-2} + ... + b_1\alpha^1 + b_0 \quad (4.2)$$

$$C = c_{m-1}\alpha^{m-1} + c_{m-2}\alpha^{m-2} + ... + c_1\alpha^1 + c_0 \quad (4.3)$$

$$p(x) = \alpha^m + p_{m-1}\alpha^{m-1} + p_{m-2}\alpha^{m-2} + ... + p_1\alpha^1 + p_0 \quad (4.4)$$

Then, $C = b_{m-1}[A\alpha^{m-1} \bmod p(x)] + ... + b_2[A\alpha^2 \bmod p(x)] + b_1[A\alpha \bmod p(x)] + b_0 A$

We define $A^{(k)} = A\alpha^k \bmod p(x) = a_{m-1}^{(k)}\alpha^{m-1} + a_{m-2}^{(k)}\alpha^{m-2} + ... + a_1^{(k)}\alpha^1 + a_0^{(k)}$, then $A^{(k)} = A^{(k-1)}a \bmod p(x)$. Also define $C^{(k)} = A^{(k-1)}b_{k-1} + C^{(k-1)}$ with $C^{(0)} = 0$, then the final results of the multiplication is $C^{(m)}$. The algorithm developed by Jain can be described as the follows:

1) At the first m-1 steps (step 0 to step m-2), $A^{(k)}$ and $C^{(k)}$ are computed, where k changes from 1 to m-1

$$a_i^{(k)} = \begin{cases} a_{i-1}^{(k-1)} + a_{m-1}^{k-1}p_i, 1 \le i \le m-1 \\ a_{m-1}^{k-1}p_i, \qquad i = 0 \end{cases} \tag{4.5}$$

$$c_i^{(k)} = a_i^{(k-1)}b_{k-1} + c_i^{(k-1)} \tag{4.6}$$

2) At step m-1, the final result $C^{(m)}$ of the multiplication is obtained with

$$c_i^{(m)} = a_i^{(m-1)}b_{m-1} + c_i^{(m-1)} \tag{4.7}$$

As is shown in Fig 4.2, the block diagram of the multiplier over GF(2^7) is derived from the above algorithm for this RS encoder design. The square block implements equations (4.5) and (4.6), the inputs p6 to p0, b6 to b0, and a6 to a0 are the coefficients of p(x), B and A respectively. The final results c6_out to c0_out were obtained at the output of the final stage blocks (step6) as shown in Fig. 4.2.

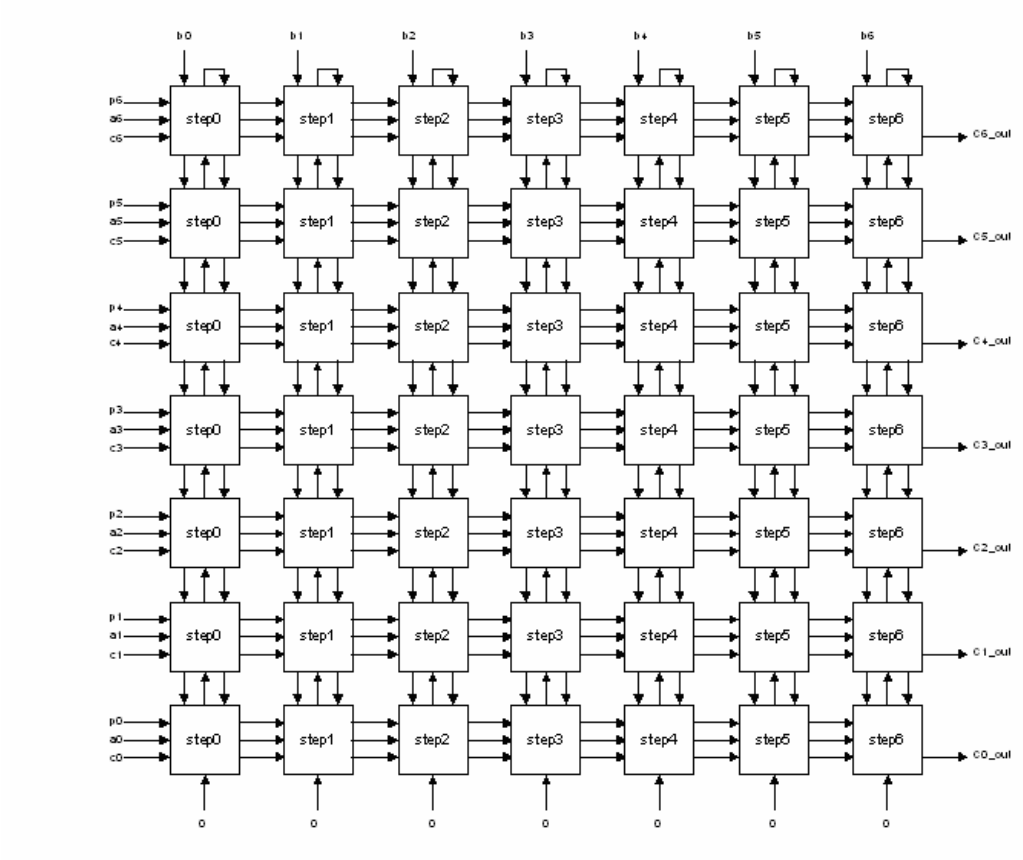


Fig. 4.2 Block diagram of the array multiplier over GF (2^7)

Although the pipelining of this array multiplier can improve the performance by a factor of m, for GF (2^m), the feed back network in RS encoder make pipelining very difficult. In addition, pipelining of this structure requires fixed multiplicand B. I implemented this architecture to compare with the approach developed in this thesis. As demonstrated in next section, this new proposed approach yields smaller area and lower power dissipation.

The proposed structure is very similar to the one described by Mastrovito [Mastrovito91]. However, the proposed multiplier has a significant advantage over Mastrovito's design,

namely it is not limited to trinomial. Both the Mastrovito's and the new proposed multiplier are fully parallel structures. In both of them, two fundamental operations, multiplication and modulo operation, are performed separately. The Mastrovito's multiplier can be used only in case when the primitive polynomial is x^m+x^1+1. Sunar extended the Mastrovito's multiplier to the all trinomials and we refer to it as the modified Mastrovito's multiplier. Fortunately, the modified Mastrovito's multiplier can be used for the primitive polynomial in our design, since p(x) = x^7+x^3+1 is a trinomial.

In what follows, I describe the theory and the basic operation of Sunar's multiplier, which is commonly used in error correction processing. For the multiplier over GF(2^m), this method constructs the multiplication matrix by pre-calculating higher order elements $x^m, x^{m+1}, ..., x^{2m-2}$ modulo p(x) using several equations. The notation in this part is adopted from [Sunar99].

Let $C = AB \bmod p(x)$, where $A, B, C \in GF(2\wedge m)$ and $p(x)$ is a primitive polynomial over $GF(2\wedge m)$. The polynomial representation is used, so

$$A = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + ... + a_1\alpha^1 + a_0 \quad (4.7)$$

$$B = b_{m-1}\alpha^{m-1} + b_{m-2}\alpha^{m-2} + ... + b_1\alpha^1 + b_0 \quad (4.8)$$

$$C = c_{m-1}\alpha^{m-1} + c_{m-2}\alpha^{m-2} + ... + c_1\alpha^1 + c_0 \quad (4.9)$$

$$p(x) = \alpha^m + p_{m-1}\alpha^{m-1} + p_{m-2}\alpha^{m-2} + ... + p_1\alpha^1 + p_0 \quad (4.10)$$

The multiplication matrix Z will be constructed in terms of the coefficients of A and the procedure for the construction is shown as follows.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{m-2} \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} z_0 & 0 & 0 & 0 & \cdots & 0 \\ z_1 & z_0 & 0 & 0 & \cdots & 0 \\ z_2 & z_1 & z_0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & z_0 & 0 & \vdots \\ z_{m-2} & z_{m-3} & z_{m-4} & \cdots & z_0 & 0 \\ z_{m-1} & z_{m-2} & z_{m-3} & z_{m-4} & \cdots & z_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{bmatrix} \tag{4.11}$$

The matrix Z equals to X+Y,

And $$X=\begin{bmatrix} a_0 & 0 & 0 & 0 & \cdots & 0 \\ a_1 & a_0 & 0 & 0 & \cdots & 0 \\ a_2 & a_1 & a_0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & a_0 & 0 & \vdots \\ a_{m-2} & a_{m-3} & a_{m-4} & \cdots & a_0 & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & a_{m-4} & \cdots & a_0 \end{bmatrix} \tag{4.12}$$

$Y=\sum_{i=0}^{k-1} T\left[\uparrow i(m-n)\right]+\sum_{i=0}^{k-1} U\left[\rightarrow i(m-n)\right]$, where $T\left[\uparrow k\right]$ represents the matrix shift up by k rows and $T\left[\rightarrow k\right]$ shift left by k columns, $k=\left\lfloor \frac{m-2}{m-n} \right\rfloor +1$, and the matrixes T and U are

$$T=\begin{bmatrix} 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 \\ 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ 0 & 0 & 0 & a_{m-1} & \ddots & \vdots \\ \vdots & \vdots & \vdots & 0 & a_{m-1} & \vdots \\ 0 & 0 & 0 & \cdots & 0 & a_{m-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{4.13}$$

$$U=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \cdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 \\ \vdots & \vdots & \vdots & \cdots & a_3 & a_2 \\ 0 & 0 & a_{m-4} & \cdots & \vdots & 0 \\ 0 & 0 & \cdots & a_{m-1} & \cdots & a_{m-n} \end{bmatrix} \tag{4.14}$$

For the derivation of the above results, please refer to the paper [Sunar99].

The multiplication matrix of the modified Mastrivito's multiplier over GF (2^7) for the primitive polynomial x^7+x^3+1 is derived as follows.

In this encoder, m= 7 and n=3. Then k=2 and

$$\begin{aligned} Y &= \sum_{i=0}^{k-1} T\left[\uparrow i(m-n)\right] + \sum_{i=0}^{k-1} U\left[\rightarrow i(m-n)\right] \\ &= \sum_{i=0}^{1} T\left[\uparrow i(7-3)\right] + \sum_{i=0}^{1} U\left[\rightarrow i(7-3)\right] \end{aligned} \tag{4.15}$$

where

$$T=\begin{bmatrix} 0 & a6 & a5 & a4 & a3 & a2 & a1 \\ 0 & 0 & a6 & a5 & a4 & a3 & a2 \\ 0 & 0 & 0 & a6 & a5 & a4 & a3 \\ 0 & 0 & 0 & 0 & a6 & a5 & a4 \\ 0 & 0 & 0 & 0 & 0 & a6 & a5 \\ 0 & 0 & 0 & 0 & 0 & 0 & a6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.16}$$

$$T\left[\uparrow 4\right]=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & a6 & a5 \\ 0 & 0 & 0 & 0 & 0 & 0 & a6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.17}$$

$$U=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a6 & a5 & a4 & a3 & a2 & a1 \\ 0 & 0 & a6 & a5 & a4 & a3 & a2 \\ 0 & 0 & 0 & a6 & a5 & a4 & a3 \\ 0 & 0 & 0 & 0 & a6 & a5 & a4 \end{bmatrix} \tag{4.18}$$

$$U\left[\rightarrow 4\right]=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a6 & a5 \\ 0 & 0 & 0 & 0 & 0 & 0 & a6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.19}$$

$$Y=T+T\left[\uparrow 4\right]+U+U\left[\rightarrow 4\right]=\begin{bmatrix} 0 & a6 & a5 & a4 & a3 & a2+a6 & a1+a5 \\ 0 & 0 & a6 & a5 & a4 & a3 & a2+a6 \\ 0 & 0 & 0 & a6 & a5 & a4 & a3 \\ 0 & a6 & a5 & a4 & a6+a3 & a5+a6+a2 & a4+a5+a1 \\ 0 & 0 & a6 & a5 & a4 & a6+a3 & a5+a6+a2 \\ 0 & 0 & 0 & a6 & a5 & a4 & a6+a3 \\ 0 & 0 & 0 & 0 & a6 & a5 & a4 \end{bmatrix}$$

Finally Z=X+Y

$$= \begin{bmatrix} a0 & a6 & a5 & a4 & a3 & a2+a6 & a1+a5 \\ a1 & a0 & a6 & a5 & a4 & a3 & a2+a6 \\ a2 & a1 & a0 & a6 & a5 & a4 & a3 \\ a3 & a2+a6 & a1+a5 & a0+a4 & a6+a3 & a5+a6+a2 & a4+a5+a1 \\ a4 & a3 & a2+a6 & a1+a5 & a0+a4 & a6+a3 & a5+a6+a2 \\ a5 & a4 & a3 & a2+a6 & a1+a5 & a0+a4 & a6+a3 \\ a6 & a5 & a4 & a3 & a2+a6 & a1+a5 & a0+a4 \end{bmatrix} \quad (4.20)$$

For the constant coefficient in the RS encoder, we pre-calculate these matrixes and the design example of g4=alpha^52 in the generator polynomial is $g4=\alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha^1$ .i.e a6=a0=0, all others are 1. The multiplication matrix

$$Z=\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Then, the diagram of the multiplier over GF(2^7) can be derived from the matrix shown in Fig. 4.3

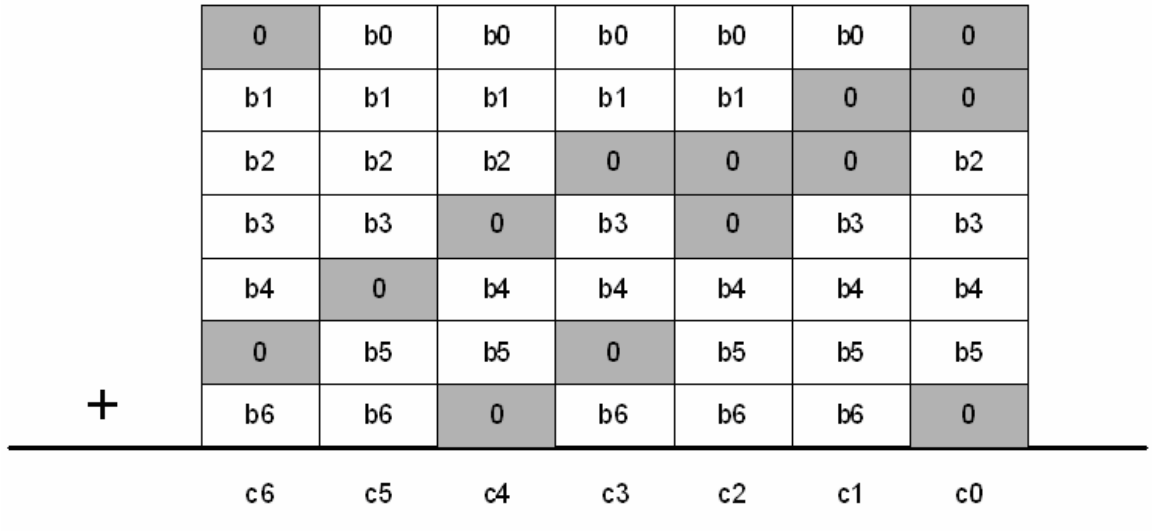


Fig. 4.3 Block diagram of the multiplier (C=A*alpha^52) over GF(2^7)

The bottleneck for speeding up the GF multiplier is always the modulo operation. Here, I propose a new version of this algorithm that reduces the number of the partial product and addition operations to speed up the design. In this proposed structure, we use both the polynomial and exponential representations instead of single polynomial or exponential terms. The key idea to simplify modulo operation is pre-computing, which takes advantage of the fixed coefficients. Let us consider $C = AB \bmod p(x)$, where $A, B, C \in GF(2^{\wedge} m)$ and $p(x)$ is primitive polynomial over $GF(2^{\wedge} m)$. The polynomial representation is used for A and C, so

$$A = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + ... + a_1\alpha^1 + a_0 \qquad (4.21)$$

$$C = c_{m-1}\alpha^{m-1} + c_{m-2}\alpha^{m-2} + ... + c_1\alpha^1 + c_0 \quad (4.22)$$

$$p(x) = \alpha^m + p_{m-1}\alpha^{m-1} + p_{m-2}\alpha^{m-2} + ... + p_1\alpha^1 + p_0 \quad (4.23)$$

Since every none zero element in the GF(2^m) can be represented as α^k, where $k = 0,1,2...,2^m - 1$, and assuming that the fixed coefficient B is α^k, then $AB = a_{m-1}\alpha^{m-1+k} + a_{m-2}\alpha^{m-2+k} + ... + a_1\alpha^{1+k} + a_0\alpha^k$.

We first calculate $\alpha^{k+j} \bmod p(x)$, *where* $j = 0,1,2...m-1$:

Assume $\left(\alpha^{k+j} \bmod p(x)\right) = r_{m-1}^j\alpha^{m-1} + r_{m-2}^j\alpha^{m-2} + ... + r_1^j\alpha^1 + r_0^j$,

so $C = AB \bmod p(x) = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i r_i^j \alpha^i$, and the hardware structure can be derived from this equation.

The design example of the multiplier with B equal to alpha^52 is shown as follows.

The coefficients of $\alpha^{52+j} \bmod p(x)$, where j=0,1,…6, is shown as the following matrix,

$$R = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

The hardware structure derived from the above result is similar to the modified Mastrovito multiplier shown in Fig. 4.3. The tree structure [Chandrakasan95] can be used to compute the individual column, if the number of elements is larger than four, which is

referred to as the granularity of the LUT for Xilinx Virtex FPGA. Since all the inputs from A arrive at the same time, the tree structure effectively reduces the glitching activities by balancing the signal paths and reducing the logic depth [Chandrakasan95].

Comparing to the Mastrovito's multiplier, the computing complexity is of the same order: m for GF(2^m). The proposed finite field multiplier takes advantage of the multiplication by a constant coefficient, while Mastrovito's multiplier is more general and both A and B are not limited to fixed values. On the other hand, the proposed structure is not limited to the trinomials, which is the limitation of the Mastrovito multiplier.

Table 4.1 shows the comparison between the proposed methods and Jain's array multiplier (From Fig. 4.2) in terms of power and area when they are employed in the RS encoder. From this table, we can see that the logic power and the signal power are reduced by about 243% and 350% respectively, which results from the great simplification of modulo operations in the proposed method. Also the total area cost is reduced by 233% by employing the new structure.

Table 4.1 Performance of RS encoders using different multipliers

Power in (MW)	**Clock power**	**Logic power**	**Signal power**	**Input power**	**Total Power**	**Area**	**Area*Power**
Proposed	0.462	0.331	0.375	0.199	1.367	62	84.754
Array	0.415	1.136	1.69	0.199	3.44	207	712.08

4.3 Trellis coded modulation

In this section, energy efficient design of trellis coded modulator is presented. The modulator consists of four functional blocks: data formatter, differential pre-coder, punctural binary convolutional encoder and QAM mapper. The signal graphs of the 64 QAM and 256 QAM have slight difference in the number of trellis groups and bit width, so 64 QAM design example is discussed.

The direct map method can be used in implementing both the differential pre-coder and QAM mapper. Although pipelining this feedback network pre-coder is possible as shown in [Chandrakasan95], it will not impact the performance of the whole modulator and the circuit complexity increases.

For the 4/5 puncture binary convolutional coder, the block diagram is shown in Fig. 4.4. The 4 bit data stream is an input at Din from the output of the differential pre-coder received at the bit rate, which is seven times larger than the symbol rate. 8-bit encoded data are generated at Dout. The multiplexer selects the data from upper or lower branch (g1 or g2) according to the puncture matrix. The first three bits are labeled as the invalid data.

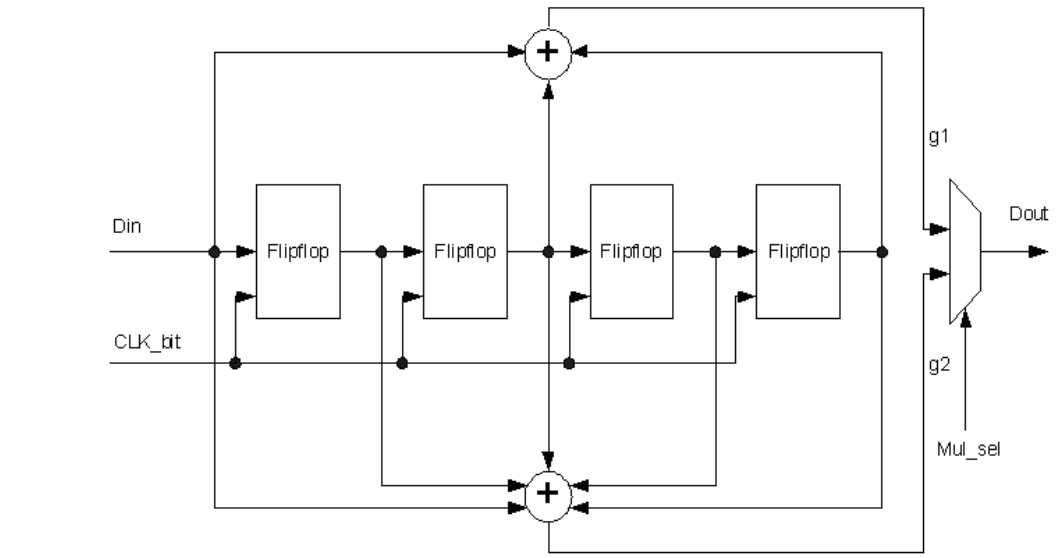


Fig. 4.4 Block diagram of the convolutional coder

The data formatter is the key module for this trellis coded modulator. The data formatter forms the trellis group from the RS symbols as shown in Fig. 4.5

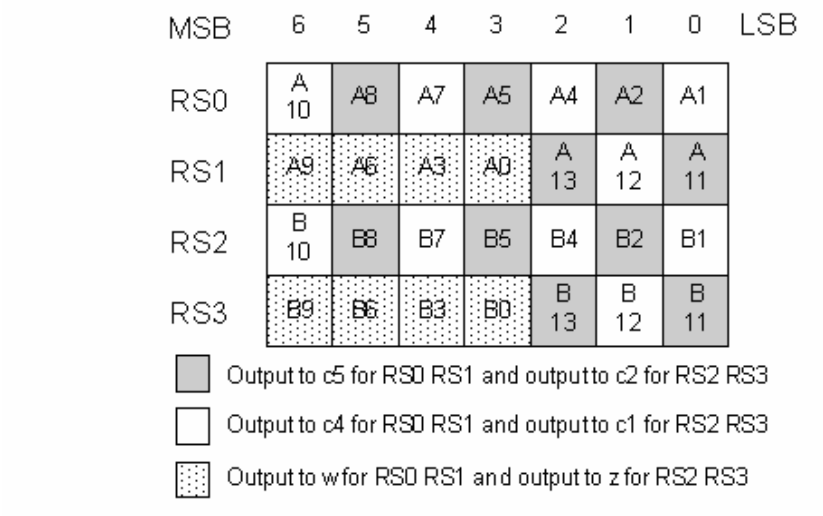


Fig. 4.5 RS symbols to Trellis Group bit ordering

We define four RS symbols as the single group for conversion to trellis group. Each of four symbols is referred to as a processing period named as RS0 symbol period, RS1, RS2 and RS3 symbol period. These four input RS symbols are reordered into six groups or six output bit streams: C5, C4, C2, C1, W and Z. The rules for this reordering are shown in Fig. 4.5. The rules for RS symbols to trellis group bit ordering:

- The odd bits of RS0 symbol and the even bits in RS1 symbol's LSB form the C5
- The even bits of RS0 symbol and the odd bits in RS1 symbol's LSB contribute to the C2
- The remaining four bits of the RS1 symbol, symbol's MSB, are fed into the W input of differential pre-coder.

For the C2, C1, and Z repeat the same reordering on the RS2 and RS3 symbols.

The hardware structure resulting from the direct map is show in Fig. 4.6.

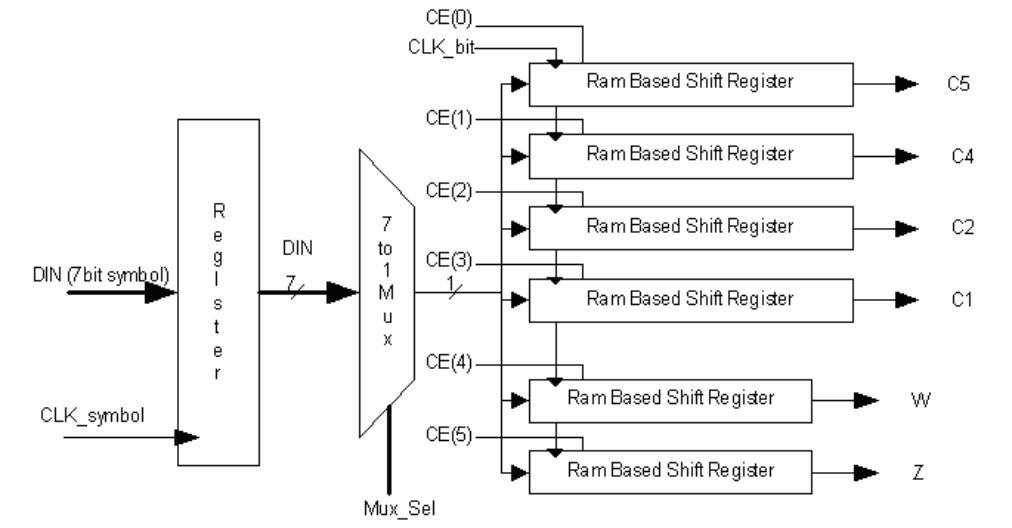


Fig. 4.6 Block diagram of the data formatter using direct map method

7-bit symbol data at DIN are fed into the register at the symbol frequency (CLK_symbol). During a loop period, data formatter accepts four symbols to form the trellis group. The 7 to 1 multiplexer functions as a parallel to serial converter. The selection lines of the multiplexer Mux_Sel change from 0 to 7 at the frequency of symbol rate/8 (clk_b). During the first seven clock cycles, the symbol data is distributed to the corresponding output shift register. The Random Access Memory (RAM) based shift register was employed. It is very efficient for this data in terms of both area and interconnects since only six LUTs and 12 interconnects are used. At the eighth clock period, new 7-bit RS symbol is loaded. The load enable signals of the RAM based shift register are enabled according to the rules of reordering as shown in Fig. 4.5. A lower 3 bits of 5-bit counter are used to count the bit sequence and the remaining 2 bits indicate the sequence of the RS symbols. Combining the counter and some additional combinational logic, the control logic is enhanced to deal with the resynchronization of the output data bits to the mapper, which is caused by the delay of further processing the w and z inputs through the differential pre-coder and puncture binary convolutional encoder. This prevents modulator from accepting the symbol data in a continuous way. In addition, the handshake logic has to be introduced.

One brutal way to remove this imbalance in the signal graph is replicating the same data path to speed up the pre-coding and convolutional encoding. Two identical data paths including the two RAM based registers for w and z, convolutional encoding and pre-coding are used, and each unit works at the symbol rate. The data path capacitance has

increased by a factor of 2, and the extra routing and overhead due to switching of the inputs and outputs make parallelizing of the design inefficient in terms of area and power consumption.

In this thesis a novel approach to design data formatter was developed. It is based on rescheduling the sequences of the reordering process. By introducing the buffer, which is implemented using bit serial RAM, we first deal with the RS1 and RS3 instead of RS0 and RS2. The bit stream is then fed into pre-encoder and convolutional encoder, and the RS0 and RS2 is reordered simultaneously. The sum of extra delay of the two encoders is one symbol period and then the synchronization problem automatically disappears. The block diagram is shown in Fig. 4.7.

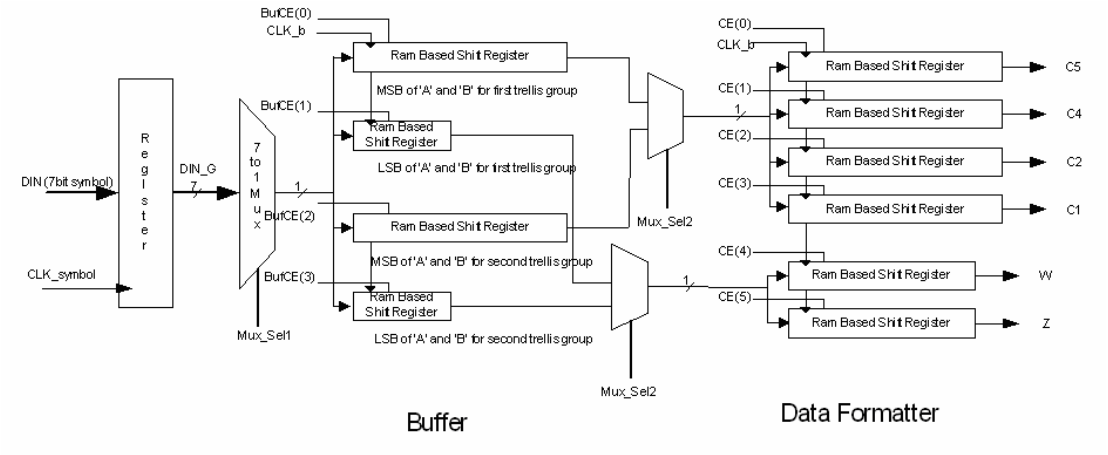


Fig. 4.7 Block diagram of the modified data formatter

There is also a slight increase in the total “effective” capacitances introduced by rescheduling, which costs the four RAM based shift registers as buffers, two multipliers

and extra control logic, while the previous methods increase the resources by doubling the logic and connections. From Table 4.2, the replication method consumes 74% more power and cost 72% more logic resources than the proposed rescheduling method. Both the interconnection and logic cost are reduced by reformulating the algorithm , which results in low power consumption. Power area product of the original method is higher by almost 200% than that in proposed approach.

Table 4.2 Comparison between two methods in area and power

	Power (mW)	**Area(Slice)**	**Area*Power**
Replicate data path	5.976	74	442.224
Rescheduling	3.43	43	147.49
Replicate / rescheduling	174.23%	172.09%	299.83%

4.4 Interleaver design

As shown in Fig. 4.8, the (I, J) convolutional interleaver consists of input and output commutators and I branches or delay lines. The interleaver accepts the Reed Solomon symbols at the input commutator, and the interleaved data streams are output at the output commutator. Reed Solomon symbols appear on the input commutator arm at the symbol rate, which is around 5.0Msymbol/sec for ITU J.83 standard [ITU J.83]. The topmost branch, labeled with 1, has zero delay. The second branch has J symbol periods delay, and the k-th branch is delayed by (k-1)*J symbol periods. The last branch has (I-1)*J symbol periods delay. Both input and output commutators are reset to the top most branch and move onto the next branch at the symbol frequency. After reaching the last

branch, they switch back to the first branch and repeat the above rotation. We define the commmutator switching from branch 1 to I as a loop.

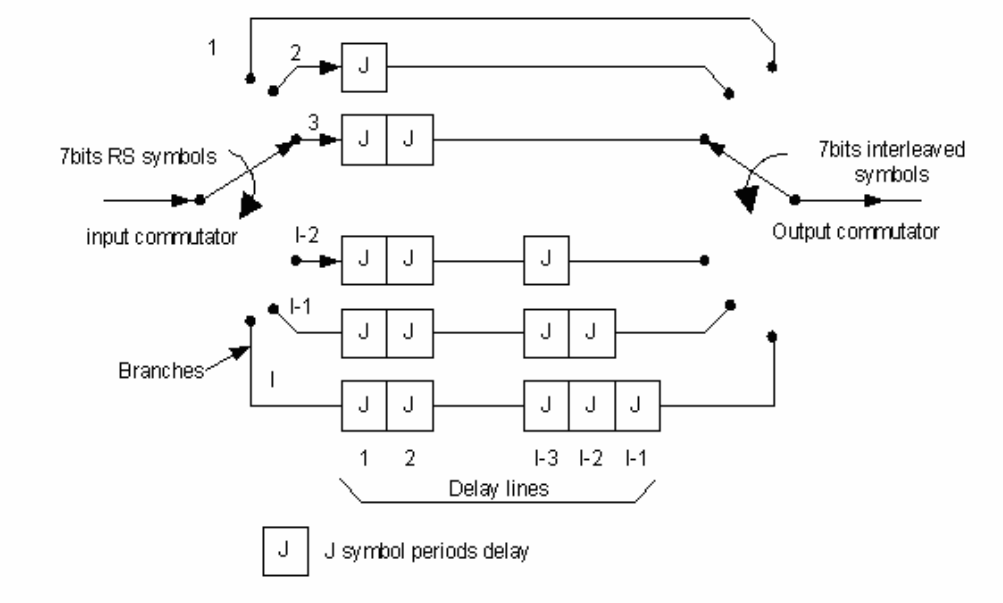


Fig. 4.8 ITU J.83 convolutional interleaver (I, J) functional block diagram

The direct map method is used to implement this convolutional interleaver. The key elements for the straightforward implementation are the delay lines. RAM based shift register is suitable to realize the delay elements. A single RAM based shift register can implement the delay lines with the value ranging form 0 to 15 clock periods. Then the delay line is constructed by cascading these RAM Based Shift Registers, named as RAM Based Shift Register Chain. This implementation is very efficient in terms of area and power consumption. The design example of (128,1) convolutional interleaver is shown in Fig. 4.9.

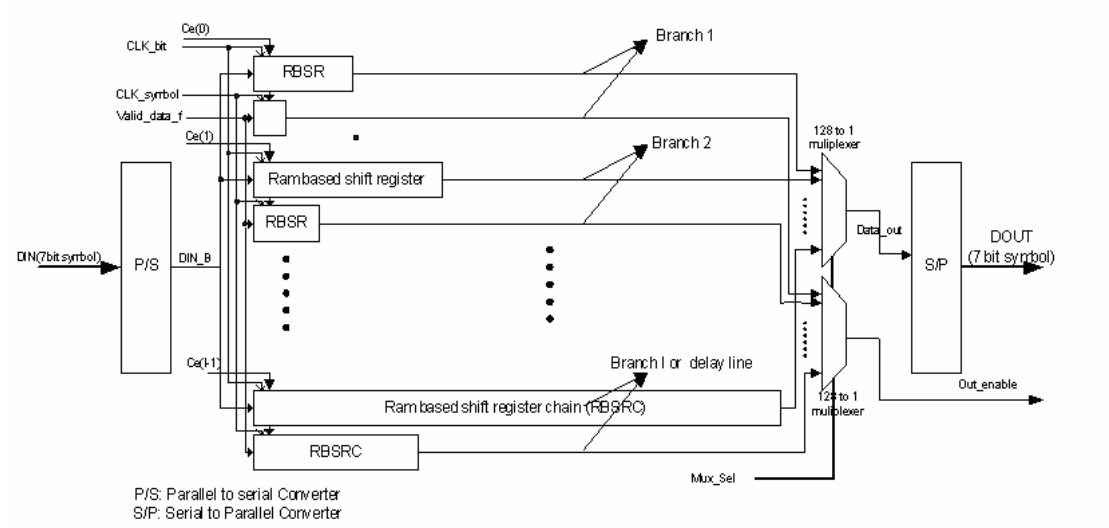


Fig. 4.9 Block diagram of the interleaver using the RAM based shift register

7-bit symbol data is fed into the interleaver in a bit stream format through the Serial to Parallel module. CLK_symbol is the symbol frequency and CLK_bit is the bit rate, which equals symbol rate * 7. Ce(i), i=0…I-1, is the shift enable signal. For a branch i, two delay lines were used. One is the delay line of data and its input is symbol data in a serial format. It has (i-1)*J clock (CLK_bit) periods delay. Another delay line is for data output enable and its input is valid data flag, valid_data_f, and it has (i-1) clock(CLK_symbol) periods delay. When valid data appears at the input of the delay line of data, the valid data flag is shifted into the data output enabling flag delay line simultaneously. Since this delay line shifts at the frequency same as the symbol rate, then the length of this delay line can be shrinked by factor of 7 bits per symbol. This reduces the area cost and the signal switching activities. These two delay lines share the same shift enable signal Ce(i), for branch i. The rotation of the commutator is implemented by enabling the delay line at one symbol period. The outputs of the delay lines connect to the

inputs of the 128 to 1 multiplexer. There are two 128 to 1 multiplexers, one is for the data output and another is the valid data flag or data output enable. The multiplexer selection signals of the multiplexer, Mux_Sel, are synchronized with the position of the commutators. Finally, valid data stream is converted to the 7 bit symbol data at the output and out_enable signal indicates whether it is valid or not. However, this design is not so efficient from the power consumption perspective. The interconnection consists of 128*2 shift enable signals, 128*1 data input signals, two types of clock signals, 128*2 output signals from the delay lines to the multiplexer outputs, and also the interconnecting signal between the shift register.

In what follows, the interleaver algorithm is reformulated for low power purpose. It is based on a new concept of using dual port memory to realize the interleaving. The interleaving can be considered as a sequences reordering process for the input symbols. The design example of (I, J) convolutional interleaver is presented as the follows. The following lemma helps to identify output groups and time instances at which valid data was obtained. Let R(0), R(1), R(2),…,R(m) be the input Reed-Solomon symbols and 0, 1, 2, …,m represent the time indices. Divide the output into groups of symbols R(i), where symbols form a group when they are output at the same loop (i.e. when the commutator rotates from the first branch to the last branch.)

Lemma: The output sequence after the interleaving is a sequence of $(\lfloor m/I \rfloor + 1)$ groups: R(I*K), R(I*K-(I-1)), R(I*K-(I-1)*2),…, R(I*K-(I-1)*(I-1)), where $K = 0,...,\lfloor m/I \rfloor$.

When I*K-(I-1)*j >=0, and I*K-(I-1)*j <= m, the output is a valid output; otherwise, the output is not a valid output.

Proof:

Define a single loop as the commutator rotates from the first branch, branch 0, to the last branch, branch I-1. At each loop, index of the RS symbols increases by one at the symbol frequency. For the first loop, the R(0+j) RS symbol appears on the j-th branch. At the ith loop, R(i*I+j) RS symbol presents at the j-th branch. During the first loop, only R(0+0) is output since the first branch has no delay. Since all the other indexes in this output group, I*0-(I-1)*p (p=1 to I-1), are negative, the lemma is proven for n=0. During the second loop, the first branch outputs the data R(I*1), which appears on this branch at this loop; the second branch output is R(I*1-(I-1))=R(0*I+1), which is fed into the second branch at the first loop and it is shifted out after one symbol delay, and for all the other indexes in this output group ,I*1-(I-1)*p (p=2 to I-1), are negative. The lemma is then proven for n=1. Assume that the lemma is valid for n=K. At the (K+1)th loop, the output of the first branch is the data just fed into this branch and it is equal to R(I*(K+1)). For the second branch, it has one symbol period delay, so its output has I symbols delay of data output at the Kth loop, which is equal to R((K-1)*I+2+I), and for i-th branch, the data that appears on the output also have I symbols delay of data output at the Kth loop, which is equal to R(I*K-(I-1)*i+I)= R(I*(K+1)-(I-1)*i). So the lemma is proven for n=k+1 and as a results of induction for all n>=0.

The proposed dual port memory design uses the above rules to interleave the input data. Assume that we have an interleaver with I branches. We map the RAM with column address and row address. Let I=2^(m+1). The column address can be represented as $a^c_{m-1}, a^c_{m-2}, ..., a^c_1, a^c_0$, and the row address lines are $a^r_{m-1}, a^r_{m-2}, ..., a^r_1, a^r_0$, so the RS symbol is written into the RAM with the following address pattern. The address is initialized to all zeros at the beginning of the interleaving, and increases by $\underbrace{00...1}_{m}\underbrace{00...1}_{m}$ instead of 1 at the symbol frequency; at the same time, data is read at the other port with the address increased by 1 at the symbol frequency and is reset to all ones. We can prove that data output have the same sequence as described in above rules. As shown in Fig. 4.10, we map the dual port RAM to a matrix with the column address equal to the column number and row address equal to the row number.

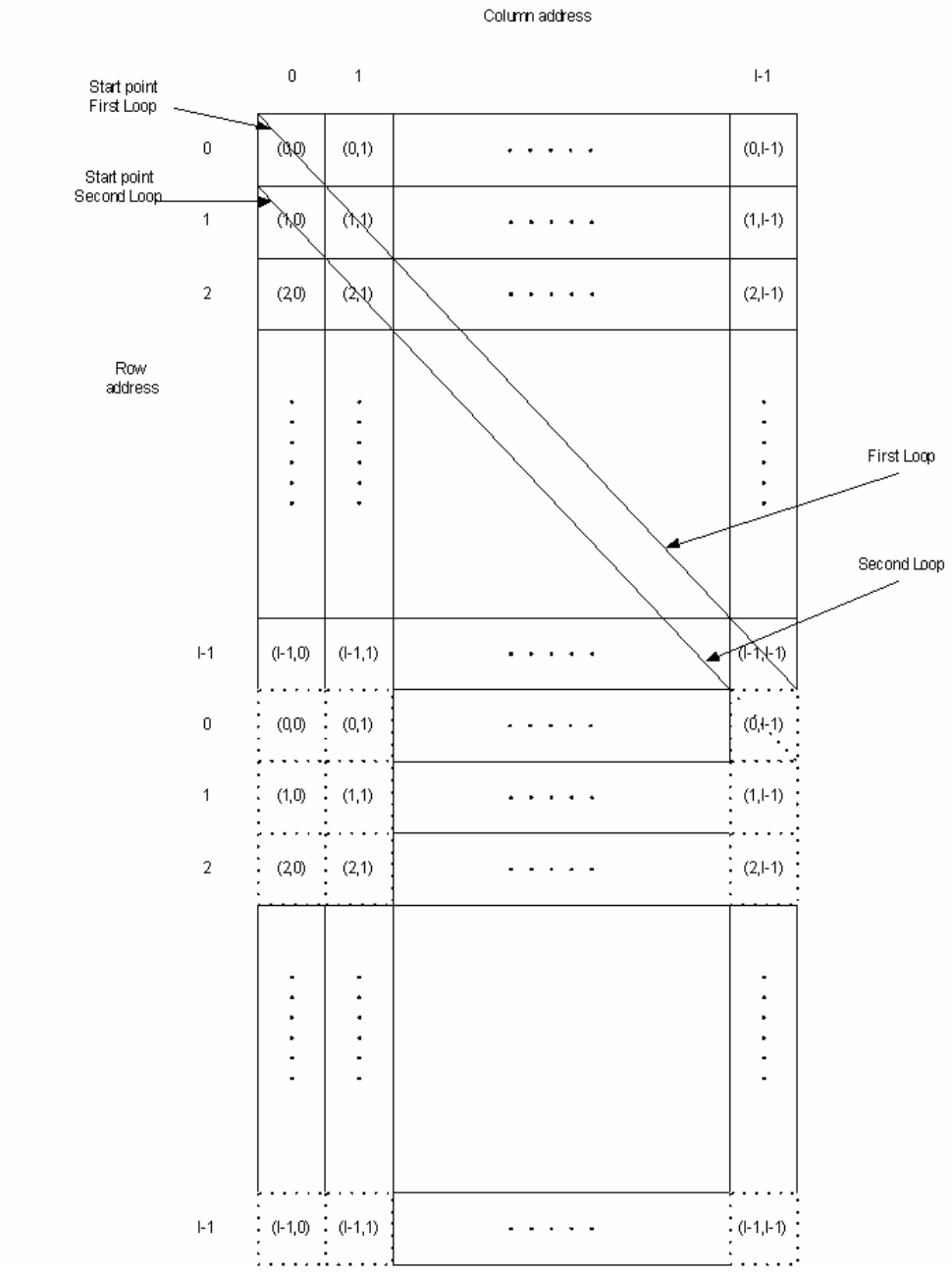


Fig. 4.10 Memory map of the Interleaver using matrix format

For each loop, the RS input symbol data is written to the diagonal of the above matrix (RAM). During the first loop period, the data is written to the diagonal line from (0,0) to (I-1,I-1). At the next loop, the data is written to the successive diagonal line, which starts at (1,0) and ends at (I mod I, I-1)=(0,I-1), since the address is increased by $\underbrace{00...1}_{m}\underbrace{00...1}_{m}$, i.e. $\underbrace{I-1}_{m}\underbrace{I-1}_{m}+\underbrace{00...1}_{m}\underbrace{00...1}_{m}=\underbrace{00...1}_{m}\underbrace{00...0}_{m}$. The dashed line matrix is an extension from the original matrix (RAM), so the data written to the matrix (RAM) in a single loop is simply a diagonal line. At the i-th loop the data is inputted to the diagonal line starting from ((i-1) mod I, 0). The interleaved data is obtained by reading the data row by row at the other port of the RAM. At the same column of the neighboring line, the time difference of these two data equals to (I-1) symbol periods. This is because the time difference at the start point of the successive lines is equal to I symbol periods, which is time period of the loop. The time index differences between these two symbols of the same column and their start points are m and n, and the absolute value of m-n is 1 symbol delay time. So, the total time difference between these two symbols equals (I-1). The symbol index of the kth column always starts with I*K, where $K = 0,...,\lfloor m/I \rfloor$.

To eliminate the invalid data at the beginning of the interleaving, the RAM is initialized to be zero and the data bus width equals 8 instead of 7 bit symbol data. One MSB bit is used as the valid data flag. Also each time the data is read from the RAM, zero is written back to the same position to invalidate the data at that position. To avoid the conflict that occurs when data is written at the same address on the two ports, the write operations at

two ports are performed at different clock cycles. Also the counter is employed to make the read operation occurring only when data is available and to provide the synchronization mechanism for the data input and output.

The reformulation of the algorithm by using time sequence reordering significantly simplifies the control structure and reduces the switching activities, since the clock frequency is reduced by factor of seven (from the bit rate to the symbol rate), and the interconnection lines,(which are the major power consumption sources in FPGA), are reduced by a factor of around twenty. Unfortunately, there is a slight increase in the area by using the block RAM. Unlike the outside RAM, the power consumption of block RAM is comparable to the logic resources. However, the logic resource only consumes 16% power according to [Shang02]'s result. Therefore, the low switching activities and smaller interconnection compensate cost due to logic increase in the RAM. This new method has the advantages both in area, delay and power consumption. Unfortunately, the XPower tools from Xilinx does not provide the power estimation of the Block RAM, thus no quantitative results can be obtained. Based on the reduction of the clock frequency and the number of interconnection lines, the power consumption can be reduced by a factor of 15 without considering the power dissipated by the block RAM. Therefore, this reformulation is well suitable for the interleaving with fewer branches since it requires less memory space.

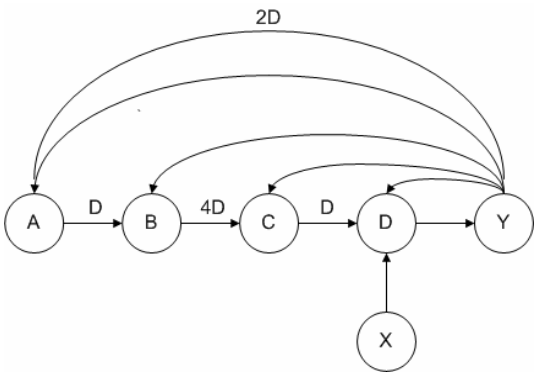
4.5 MPEG framer

In this section, the design of MPEG checksum processing block is discussed. Linear Feed Back Shift Register $f(x)$ is used to generate the MPEG checksum.

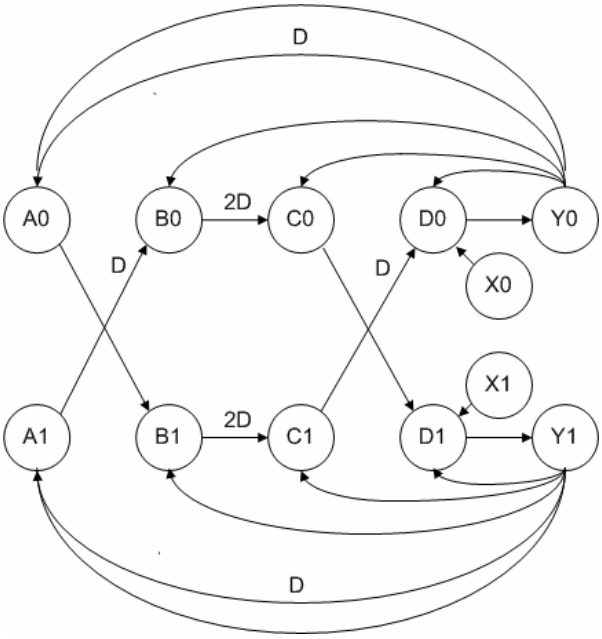
$f(x) = [1 + x^{1496} b(x)] / g(x)$ [ITU J.83]

Where $g(x) = 1 + x + x^5 + x^6 + x^8$ and $b(x) = 1 + x + x^3 + x^7$.

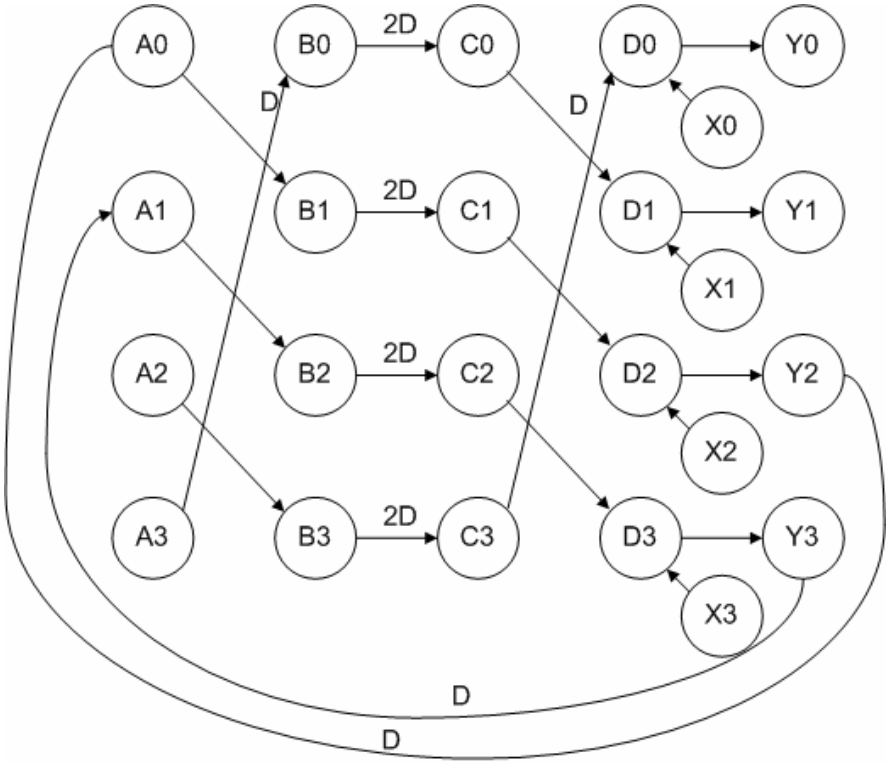
The structure of this checksum generation function is shown in Fig. 2.2. All the addition operations are modulo 2 and the LFSR is reset to all zeros before the calculation of a new frame. We present this design with the design space exploration example for parallelizing the design to lower the power consumption. Data flow graph (DFG) is used to represent the feedback network g(x) as shown in Fig. 4.11. The nodes represent modulo 2 addition operation and the edges represent the data path between the nodes. The number associated with the edge represents the delay. For instance, D represents one bit period delay, 2D represents two-bit period delay and so on. In Fig. 4.11(a), A, B, C, and D is modulo 2 addition and X, Y are input and output nodes.



(a)



(b)

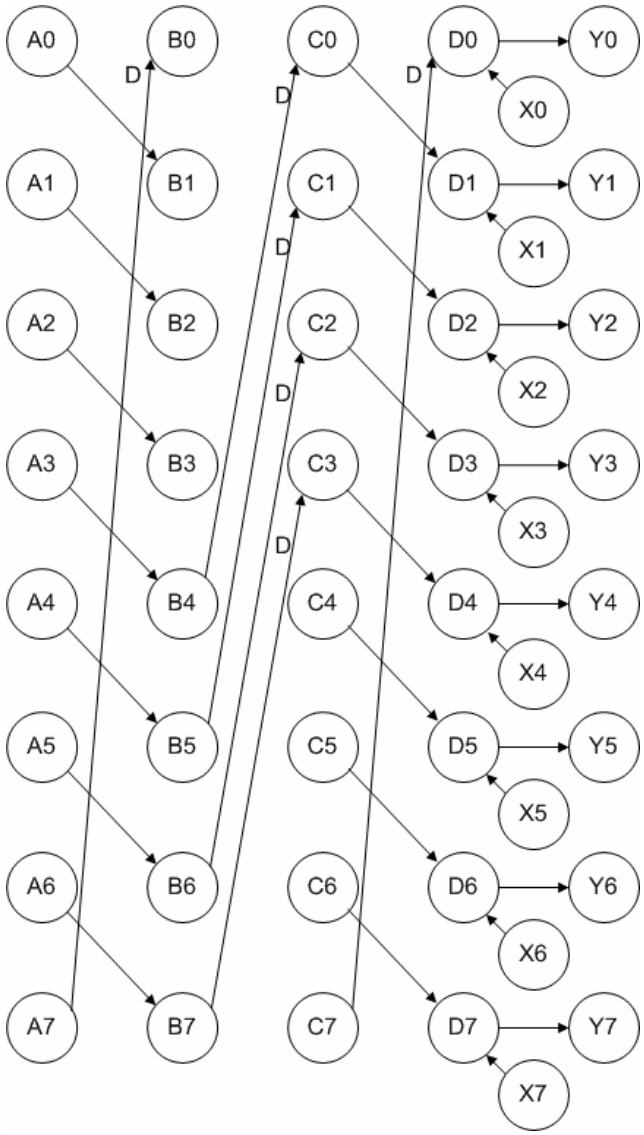


The following connection without any delay are not drawn in above diagram:

Yi->Ai, Yi->Bi, Yi->Ci and Yi->Di, where i=0, 1, 2, 3

Y0->A2 and Y1->A3

(c)



The following connection without any delay are not drawn in above diagram:

Yi->Ai, Yi->Bi, Yi->Ci and Yi->Di, where i=0, 1, 2, …,7

Bi->Ci+4, where i=0, 1, 2,3

Yi->Ai+2, where i=0,1, 2,3,4,5

The following connection with one delay are not drawn in above diagram: Y6->A0 and Y7->A1

(d)

Fig. 4.11 Replicating the data path in (a) by factor 2 in (b), 4 in (c) and 8 in (d)

Applying the unfolding to the DFG in (a) with the unfolding factor equal to 2, 4 and 8, the unfolded DFGs are obtained as shown in Fig. 4.11(b), (c), and (d). These DFGs were implemented on Xilinx Virtex and their performance in terms of power and area are shown in Fig. 4.12. From this plot, it is clear that the unfolding factor 2 is the optimum number to replicate the data path, while bit serial structure is not necessarily the best choice for low power design though it has the smallest area. The increase of the unfolding factor to 4 and 8 degrades the performance in power and power area product. Thus, the optimum number of the design parallelization is obtained by the design space exploration.

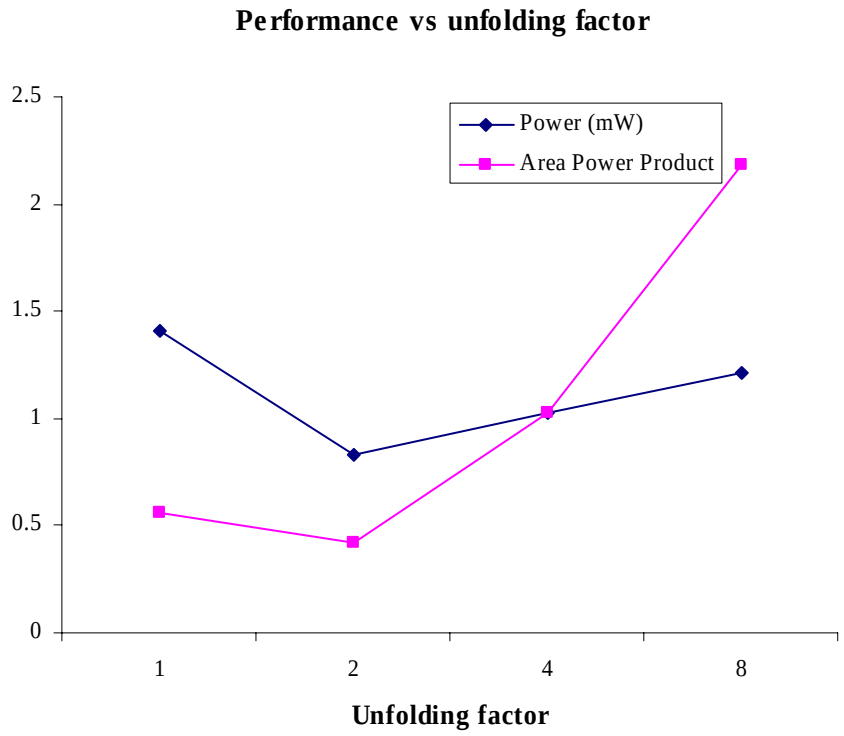


Fig. 4.12 Performance of the parallelizing the design by a factor of 1, 2, 4 and 8

A detailed analysis of various sources of power dissipation in these structures is illustrated by Fig. 4.13. Fig.4.13 shows that though the clock power is reduced by a factor proportional to the unfolding factor, the logic and signal power increase by a factor from 3 to 5 when unfolding factor equals to 4 or 8. When this increase of logic and interconnection dominates, the power reduction due to clock slowing down can not compensate the extra cost due to parallelism. Then, further increase in the parallelism only increases the power consumption. Thus design space has to be explored to optimize the design using the parallelism.

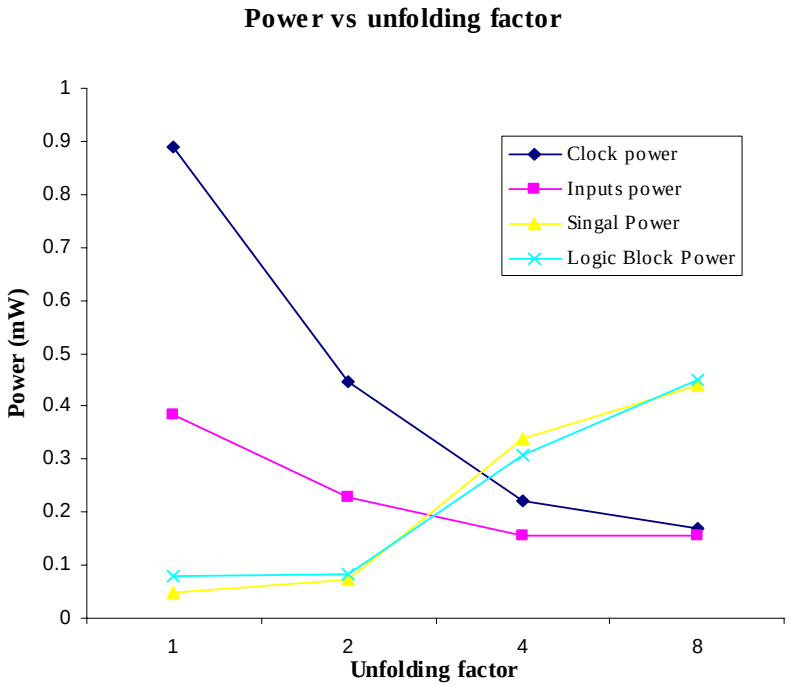


Fig. 4.13 Power of the parallelizing the design by a factor of 1, 2, 4 and 8

In this chapter, the low power design techniques were applied to the design of several key modules including Reed Solomon Encoder, interleaver, TCM modulator and MPEG framer.

Chapter 5 Simulation and results

5.1 Introduction

The implemented designs were tested using the simulation tools from the Active-VHDL. The functional simulation environment has been built in Matlab. It generates the test vectors and expected results for the VHDL implementation. In this section, the simulation results are presented by individual module of the design.

5.2 RS encoder

122 symbol data were generated using the Matlab to be applied as the test vectors to the testbench of the RS encoder in VHDL. The six parity symbols were computed in Matlab :30,113,116 83, 95 and 18. The simulation waveforms of RS encoder in Active VHDL are shown in Fig. 5.1. The final six-byte symbols on data_output bus, (where the data valid signal indicates that the output data is valid), are the same as the Matlab results.

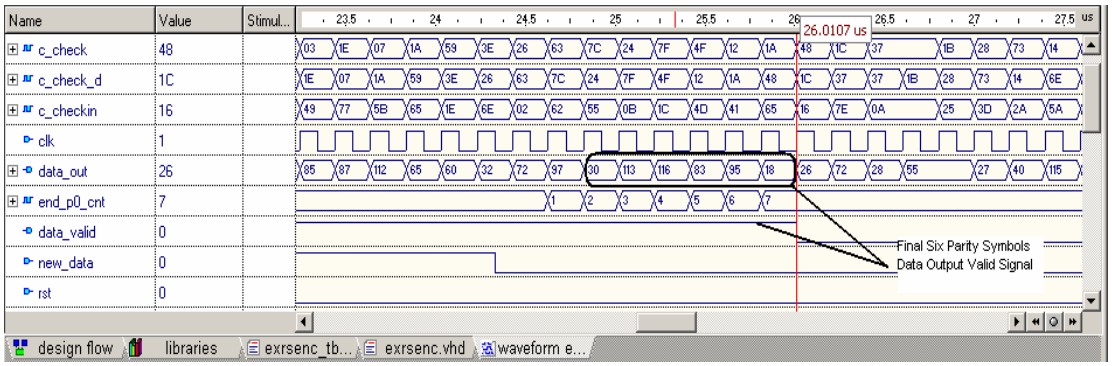


Fig. 5.1 Simulation waveforms of RS encoder

5.3 Trellis coded modulation

The Trellis coded modulation has four sub blocks: data parser, differential pre-coder, binary convolutional coder and QAM mapper. The functionality of these sub blocks are verified by Matlab and VHDL simulations as follows.

5.3.1 Data parser

First, data parser forms the trellis group as shown in Fig. 2.7. Four symbols data are fed into this block when the block is ready to accept new symbol data, that is r_rdy is high, at the rising edge of the symbol clock and the falling edge of the bit clock (symbol *8) . MSBs and LSBs of ‘A’ and ‘B’ shown in Fig. 2.6 are saved to the LFSR as shown in the following simulation wave forms. The ce_line(0 to 5) shown in Fig. 5.2 are the shift enable signals of the LFSR for trellis group of C5, C4, C2, C1, W and Z. The signal

vector sel_line selects the bit data in the symbol in LSB first and when sel_line=”7”, new symbol data is loaded.

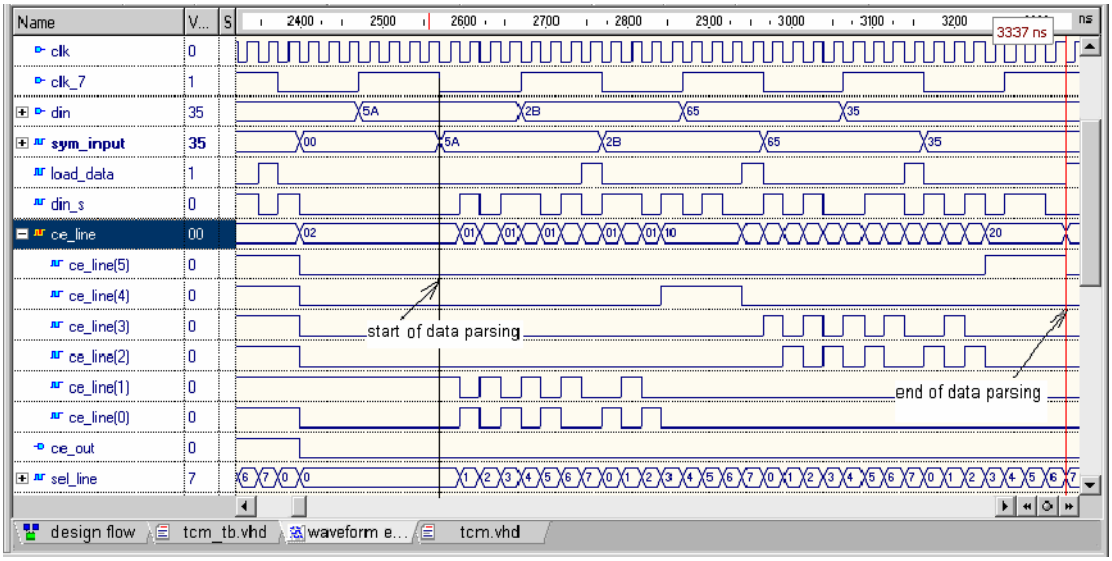


Fig. 5.2 Simulation waveforms of data parser

5.3.2 Differential pre-coder

As shown in Fig 5.3, the signals dformat_out(4), dformat_out(5) and df_x and df_y are connected to inputs w and z and outputs x and y in Fig 2.6. The signal ce_in_df initializes the differential pre-coder to zeros before the data is shifted out from the data formatter LFSR. The high value of ce_line(5) for four bit clock period before the cursor that indicates the last four bits of the fourth RS symbol is saved to the LFSR, which means the end of the data formatting. Although the pre-coder output df_x and df_y are

still active after four clocks, these bits will be masked by the input_mask signal of the convolutional coder.

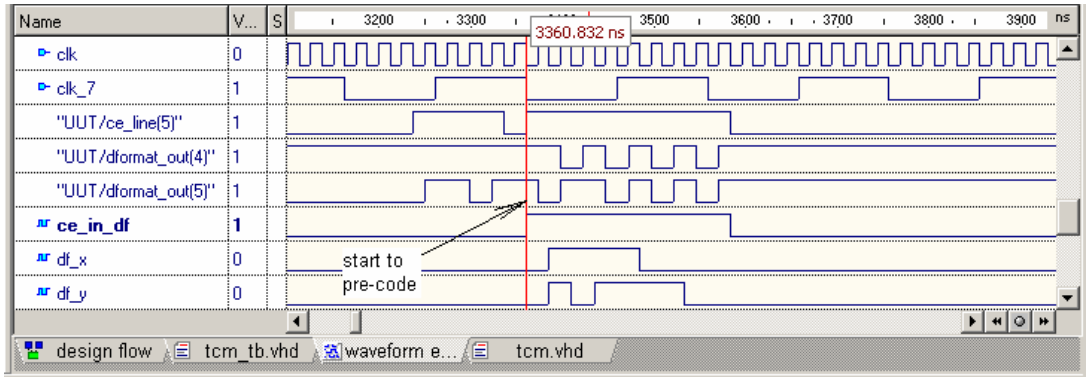


Fig. 5.3 Simulation waveforms of differential pre-coder

5.3.3 Binary convolutional coder

As shown in Fig 5.4, df_x and df_y signals are connected to outputs of the differential pre-coder and u and v is the output of the binary convolutional encoder in Fig. 2.6. The pulse of start_conv signal resets the registers of the convolutional encoder. The input_mask signal lasts four bit clock cycles to ensure data inputs is valid. The input_mask signal is also the select signal for the multiplexer of the output and switch output from the upper to the lower branch.

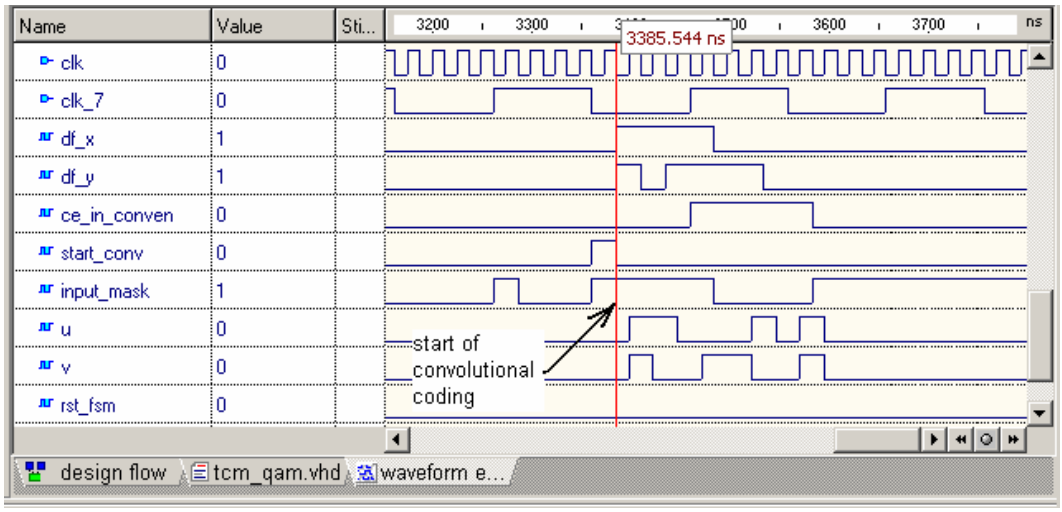


Fig. 5.4 Simulation waveforms of differential pre-coder

Finally, when the convolutional encoding is done, the 5QAM symbols are output from the dout signal and the ce_out indicates its validity as shown in Fig. 5.5. These outputs consists of two parts: first are the MSBs of ‘A’ and ‘B’, which are saved in LFSR during data formatting stage; second are the outputs of the convolutional encoder, u and v signals. All these six outputs will be fed into the QAM mapper to obtain the corresponding 6-bit constellation symbols.

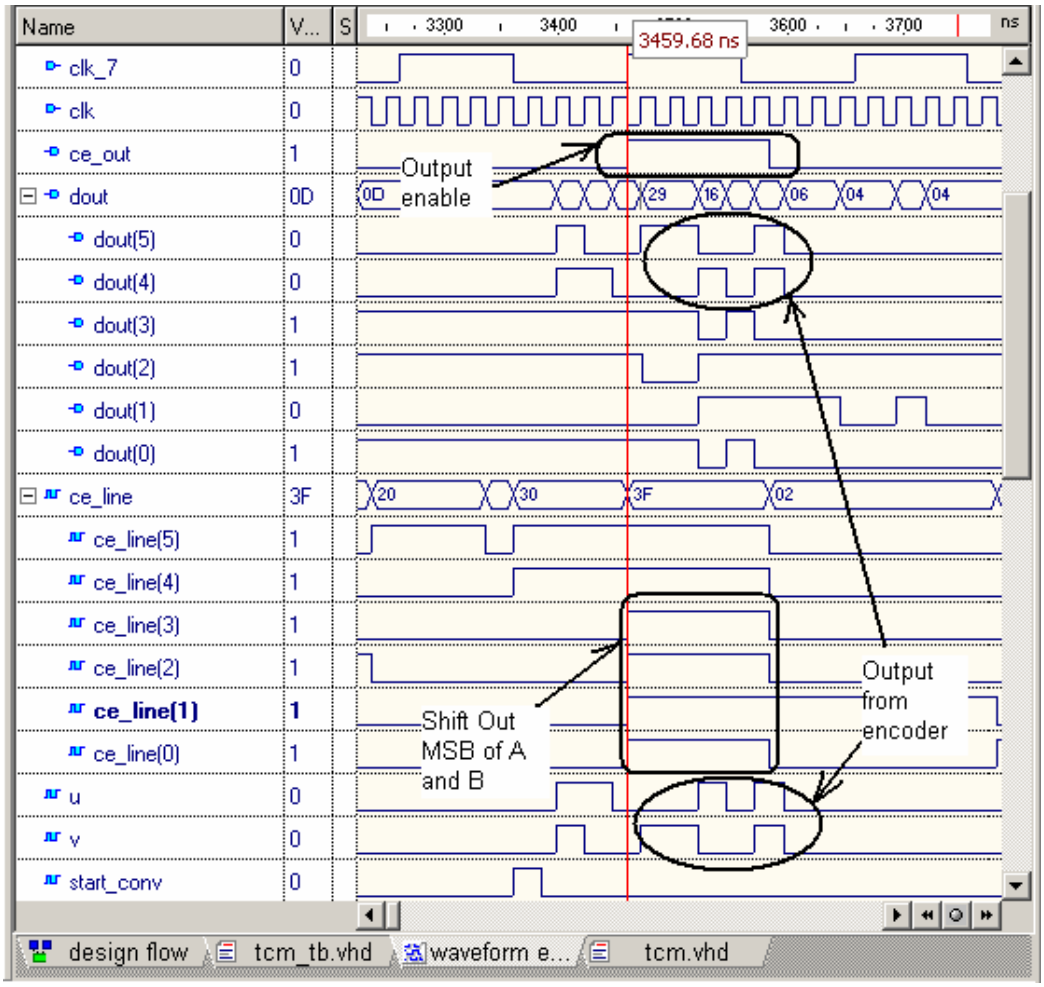


Fig. 5.5 Simulation waveforms of formation of the 5QAM symbols

5.3.4 QAM mapper

QAM constellation mapping is implemented using the look up table. The six 5-QAM symbols c5 to c0 are the inputs to the QAM mapper as shown in Fig. 2.6. The simulation waveform shown in Fig 5.6 is a 64-QAM mapper. The I and Q output are valid when QAM_ce is high.

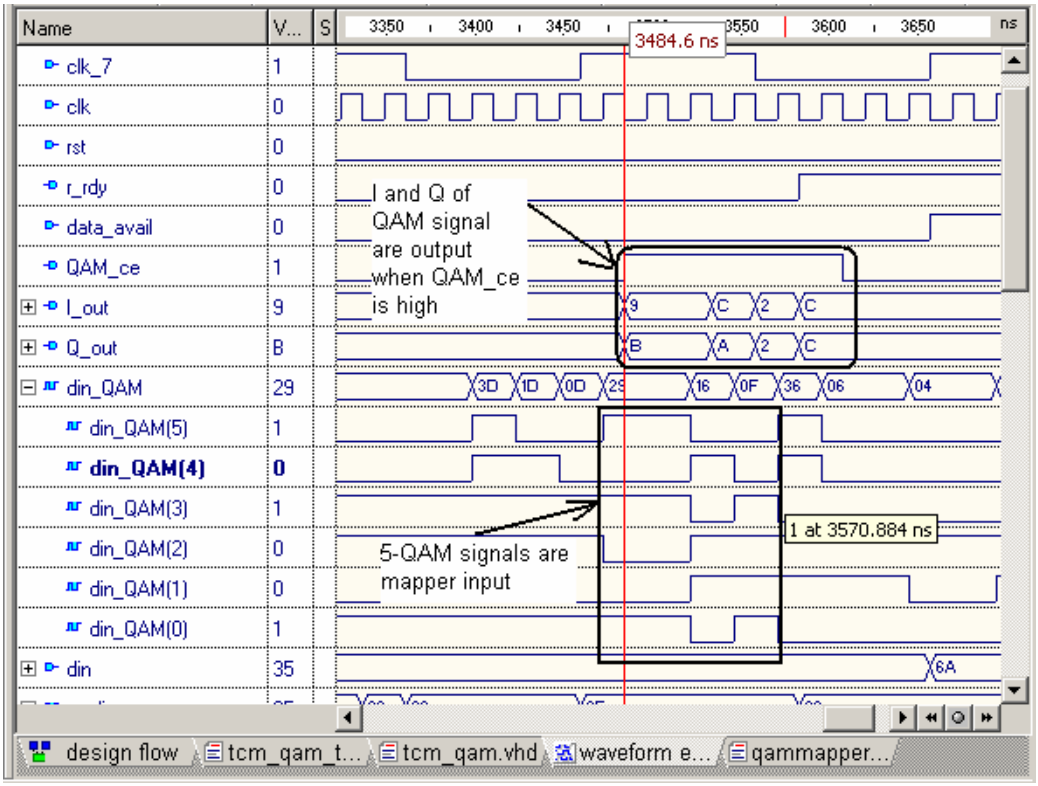


Fig. 5.6 Simulation waveforms of formation of the 5QAM symbols

5.4 Interleaver design

The interleaver is tested using two methods, first Matlab and testbench simulation on large amount of data; another is simulation wave based on a small scale of data. In the first method, test vectors were generated by Matlab and fed into the test bench of the interleaver. The interleaver data symbols generated from this test bench are compared with the Matlab results. The second method using simulation in a small scale is presented as follows. The simulation waveforms shown in Fig.5.7 and Fig. 5.8 show the response of

the interleaver (4,1). The input data is valid when data_avail sigal is high, and the valid output of the interleaved data is obtained when ce_out is high. The test vector contains integer data from 3c to 4c. In Chapter 3, two special cases, starting interleaving and ending interleaving were discussed. In this simulation, these two special cases are tested. Fig. 5.7 shows how the interleaving starts. Data input, din, have the integer data from 3c to 49 and the output sequence is 3c, 40, 44,41,3e, which is correct. Fig 5.8 shows how the interleaving ends. At the final stage, the output sequence correctly shows 4C 49 46 43, 4A 47 and 4B.

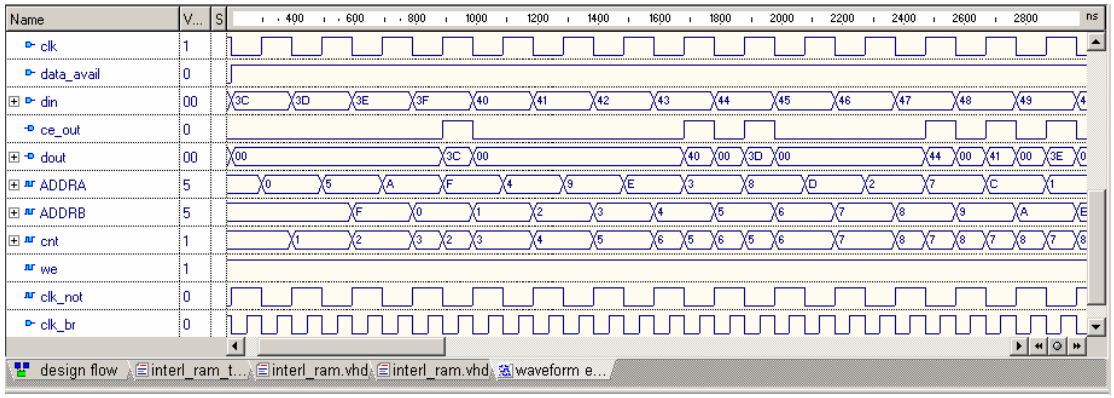


Fig. 5.7 Signal waveforms at the interleaving starts

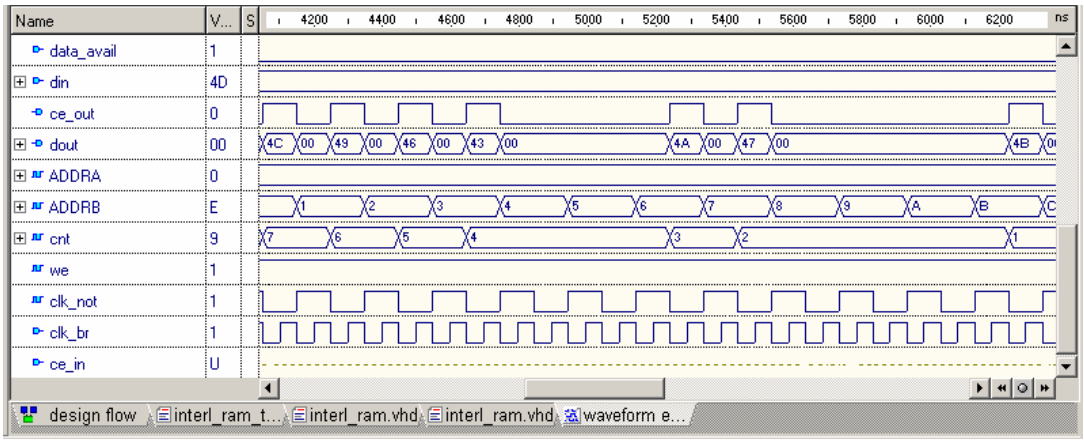


Fig. 5.8 Signal waveforms at the interleaving ends

5.5 Randomizer

The simulation waveforms are shown in Fig 5.9. As shown in Fig 2.5, the data_in signal is the “Data in” and the data_out is the “Data out”, and the fb_x_as_q, fb_x_as_q1, and fb_x_as_q2 are the output of the register from left to right. The test data fed into the data_in are 00 to 0A. The output of the randomizer is compared with that of the Matlab code. Then functionality of the randomizer is verified in VHDL.

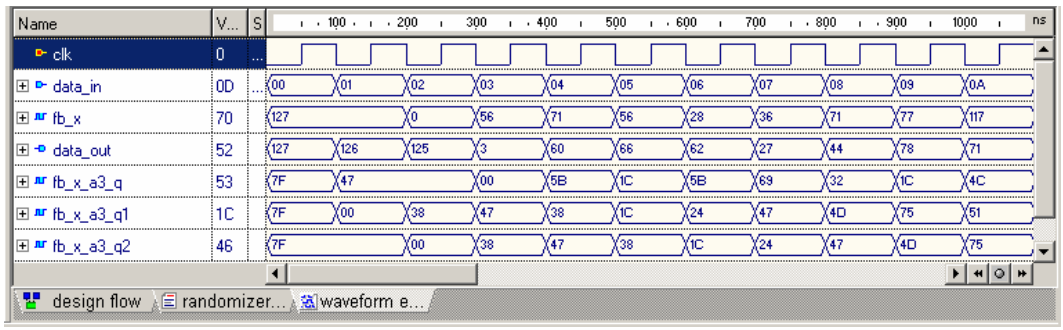


Fig. 5.9 Simulation waveform of randomizer

5.6 MPEG framer

Matlab generates the test vectors and computes the output signature for the VHDL implementation. The functionality of the module is verified by comparing this signature with the one generated by Matlab.

Chapter 6 Conclusions and Future Work

6.1 Conclusions

This thesis research is focused on the design of the digital modulator for a cable terminal system targeted on FPGA with emphasis on low power dissipation. The specification of the digital modulator is presented, which is then virtually implemented in the structural VHDL on Xilinx FPGA Virtex II. Based on the power distribution in FPGA, we found that the existing VLSI low power design techniques are effective for energy efficient design in FPGA. By applying these techniques to the digital modulator design and exploring the design space of the area and power product, an acceptable low-power design is obtained.

Existing research results on power distribution in FPGA by Shang shows that the dominant components of the dynamic power consumption in FPGA are interconnections, while clock and logic modules consume around 14-16% of total power each. The power dissipation in FPGA changes in proportion to the switching activities. By limiting these switching activities in the implementation of algorithms, a significant part of the power can be saved.

The effectiveness of the low power VLSI design techniques at algorithm and architecture levels in FPGA design are investigated based on the characteristics of the power distribution in FPGA. First, performing the optimization at the algorithm level is well suitable to the low power FPGA design. Performing the algorithm modification, to speed up the design, can lower the clock frequency requirements of the design reducing its switching activities. Also, the algorithm can be reformulated to simplify the control structure and thus to reduce the amount of interconnections, saving hardware resources and energy dissipation. Then, converting the complex arithmetic operations to simple operations and taking advantage of the constant values in these operations can reduce the effective switching capacitances and improve the speed by reducing the design area. The optimization performed on the architectural level is also effective for the low power design in FPGA. Pipelining the design, improves the performance with little overhead. Also the switching and glitching activities can be minimized by choosing proper binary number representation and balancing the signal paths. Thus, for the digital modulator design discussed in this thesis, the strategy employed is improving the performance of design and slowing down the clock frequency, as well as minimizing the effective switching capacitances by optimizing the design at algorithm and architecture level.

By applying these techniques to the design, all the design options are examined towards the low power design goal. In the design of the RS encoder, the bottleneck of the performance improvement is removed by employing a new structure for the finite field multiplier. For the interleaver design, dual port memory scheme is used to reduce the

switching activities and the amount of the interconnection. Then rescheduling is employed to solve the real time requirement for the TCM modulator. This reformulation reduces the area cost and increases the performance. Parallelizing the design is examined in the design of the MPEG checksum generator. And the results show that it is not as effective as in VLSI since the voltage is fixed in the FPGA design. Therefore, we conclude that most of the VLSI low power design techniques, except parallelizing, are very effective for energy efficient design in FPGA.

6.2 Future work

Perhaps the most effective way for the low power design in FPGA involves the design of the devices themselves, since the effective capacitance is very large comparing to the equivalent capacitance in the VLSI technology. In addition, power estimation tools providing more accurate and detailed picture of the power dissipation of the resources is highly desirable for the low power design and will facilitate the exploration of the design space. Existing tools lack of such support.

References

[Anderson02] J. H. Anderson and F. N. Najm, "Power-Aware Technology Mapping for LUT-Based FPGAs," *IEEE Int. Conf. on Field-Programmable Technology (FPT)*, Hong Kong, pp. 211-218, 2002.

[Berlekamp82] E. R. Berlekamp. "Bit-Serial Reed-Solomon Encoders," *IEEE Trans. on Information Theory*, Vol.28, No.6, pp. 869–874, Nov. 1982.

[Chandrakasan95] A.P. Chandrakasan and R.W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. of the IEEE*, Vol.83, No.4, pp.498-523, April 1995.

[Doyle02] B. Doyle et al., "Transistor Elements for 30nm Physical Gate Lengths and Beyond", *Intel Tech. Journal*, Vol. 6, No. 2, pp. 5-13. May 2002.

[George99] V. George, H. Zhang, J. Rabaey, "The design of a low energy FPGA", *ACM Int. Symp. on Low Power Design*, pp. 188-193, 1999.

[ITUH.222] ITU-T Recommendation H.222.0 (1995) | ISO/IEC 13818-1:1996, *Information technology – Generic coding of moving pictures and associated audio information: Systems*.

[ITU J.83] ITU-T Recommendation J.83, Annex-B, Apr. 1997, *Digital Multi-Programme Systems For Television, Sound And Data Services For Cable Distribution*.

[Jain98] S. Jain, L. Song and K.K. Parhi, "Efficient Semi-Systolic VLSI Architectures for Finite Field Arithmetic", *IEEE Trans. on VLSI Systems*, Vol. 6, No. 1, pp. 101-113, March 1998.

[Kusse98] E. Kusse and J. Rabaey, Low-Energy Embedded FPGA Structures in *1998 Int. Symp. on Low Power Electronics and Design*, pp. 155- 160, Aug. 1998.

[Massey86] J. L. Massey and J. K. Omura. “Computational Method and Apparatus for Finite Field Arithmetic”. *US Patent No.4,587,627*, 1986.

[Mastrovito91] E. D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Linkoping University, Department of Electrical Engineering, Linkoping, Sweden, 1991.

[Rabaey02] J. Rabaey and A. Chandrakasan and B. Nikolic *Digital Integrated Circuits: A Design Perspective 2nd Edition*, Prentice Hall, 2002.

[Shang02] L. Shang, A. S. Kaviani, and K. Bathala “Dynamic power consumption in VirtexTM-II FPGA,” *10th ACM Int. Symp. on Field-Programmable Gate Arrays(FPGA)*, pp.157-164, Feb. 2002.

[Sklar01] B. Sklar, *Digital Communications Fundamentals and Applications*, Prentice-Hall PTR, New Jersey, 2001.

[Starzyk04] J. A. Starzyk and F. Wang, “Dynamic Probability Estimator for Machine Learning”, *IEEE Trans. on Neural Networks*, Vol. 15, No. 2, pp 298- 308, March 2004.

[Sunar99] B. Sunar and Ç.K. Koç, “Mastrovito Multiplier for All Trinomials,” *IEEE Trans. on Computers*, Vol. 48, No.5, pp. 522-527, May 1999.

[Wang85] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, I. S. Reed, “VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$,” *IEEE Trans. Computers*, Vol. 34, No.8, pp. 709-717, Aug. 1985.

[Xilinx00] Xilinx Inc., Virtex-II Platform FPGA Handbook, 2000.