Nat.Lab. Technical Note PR-TN-2004/00933

Date of issue: 2004/11

High Throughput Reed Solomon Decoder for Ultra Wide Band

Akash Kumar

Industrial Mentor: Sergei Sawitzki

Author's address data: A. Kumar; akakumar@natlab.research.philips.com

Technical Note: PR-TN-2004/00933

Title: High Throughput Reed Solomon Decoder for Ultra Wide Band

Author(s): Akash Kumar

Part of project: Ultra Wide Band

Customer: Company Research, PS, POS

Keywords: Reed-Solomon Decoder, syndrome computation, dual line, error correction, Berlekamp Massey

Abstract: Reed Solomon (RS) codes have been widely used in a variety of communication systems such as space communication link, digital subscriber loops, and wireless systems as well as in networking communications and magnetic and data storage systems. Continual demand for ever higher data rates makes it necessary to devise very high-speed implementations of RS decoders. This report summarizes the most recent algorithms and architectures used for implementing high-speed RS decoders. The architecture which promised to be the best in terms of area, latency, power consumption and speed was then chosen for VHDL implementation. The implementation was tested using Cadence SimVision and optimised for high speed and low area.

Conclusions: A uniform comparison was drawn for various algorithms proposed in the literature. This helped in selecting the appropriate architecture for the intended application. Modified Berlekamp Massey algorithm was chosen for the VHDL implementation. Further, dual line architecture was used which is as fast as serial and has low latency as that of a parallel approach. The decoder implemented is capable of running at 200 MHz in ASIC implementation, which translates to 1.6Gbps and requires only about 12K design cells and an area of $0.22mm^2$ with CMOS12 technology. The system has a latency of only 284 cycles for RS(255,239) code. The power dissipated in the worst case is 14mW including the memory block when operating at 1Gbps data rate.

Contents

List of Figures

List of Tables

List of Abbreviations

DVD	Digital Versatile Disc
FF	Flip-Flop
FFM	Finite Field Multiplier
FIFO	First In First Out
FPGA	Field Programmable Gate Array
Gbps	Giga bits per second
IC	Integrated Circuit
Mbps	Mega bits per second
MUX	Multiplexer
RAM	Random Access Memory
RT	Register Transfer
RTL	Register Transfer Level
SNR	Signal-to-Noise Ratio
UWB	Ultra Wide Band
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1 Motivation

Reed Solomon (RS) codes have been widely used in a variety of communication systems such as space communication links, digital subscriber loops and wireless systems, as well as in networking communications and magnetic and data storage systems. Continual demand for ever higher data rates and storage capacity makes it necessary to devise very high-speed implementations of RS decoders. Newer and faster implementations of the decoder are being developed and implemented. A number of algorithms for decoding are available and this often makes it difficult to determine the best choice, due to the number of variables and trade-offs available. Before making a good choice for the application, therefore, thorough research is needed into the decoders available.

For the IEEE 802.15-03 standard proposal (commonly known as UWB) in particular, very high data rates for transmission are needed. According to the current standard, the data rate for UWB will be as high as 480 Mbps. Since the standard is also meant for portable devices, power consumption is of prime concern, and at the same time the silicon area should be kept as low as possible. As such, a low power and high throughput codec is needed for the UWB standard. Reed Solomon is seen as a promising codec for such a standard.

2 Introduction to Reed Solomon

Reed Solomon codes are perhaps the most commonly used in all forms of transmission and data storage for forward error correction (*FEC*). The basic idea of FEC is to systematically add redundancy at the end of the messages so as to enable the correct retrieval of messages despite errors in the received sequences. This eliminates the need for retransmission of messages over a noisy channel. RS codes are a subset of Bose-Chaudhuri-Hocquenghem (*BCH*) codes and are linear block codes. Figure 1 shows a general system employing RS codes for error correction.

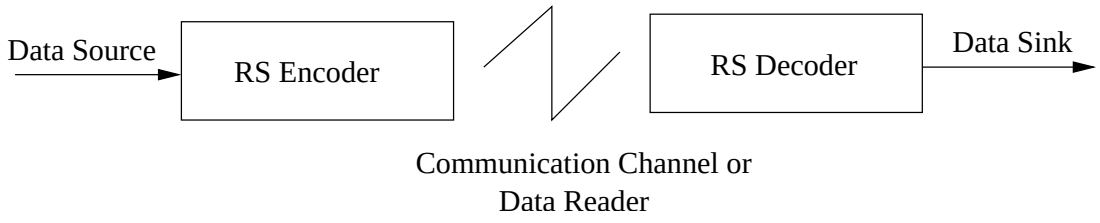

Figure 1: *A typical system employing RS codes*

2.1 Properties of Reed Solomon Codes

A $RS(n, k)$ code implies that the encoder takes in k symbols and adds $n-k$ parity symbols to make it an n-symbol code word. Each symbol is at least of m bits, where $2^m > n$.

Conversely, the longest length of code word for a given bit-size m, is $2^m - 1$. For example, $RS(255, 239)$ code takes in 239 symbols and adds 16 parity symbols to make 255 symbols overall of 8 bits each. Figure 2 shows an example of a systematic RS code word. It is called systematic code word as the input symbols are left unchanged and only the parity symbols are appended to it.

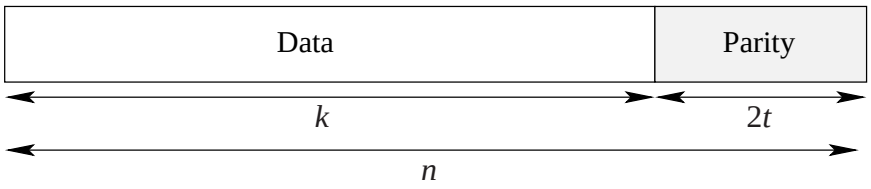

Figure 2: *A typical RS code word*

Reed Solomon codes are best for burst errors. If the code is not meant for erasures, the code can correct errors in up to t symbols where $2t = n - k$. A symbol has an error if at least one bit is wrong. Thus, $RS(255, 239)$ can correct errors in up to 8 symbols or 50 continuous bit errors. It is also interesting to see that the hardware required is proportional to the error correction capability of the system and not the actual code word length as such.

2.2 Description of the algorithm

This section gives a short description of the algorithm. It is assumed that the reader is familiar with the Galois Field arithmetic. Further details on Galois arithmetic can be found in [1]. For readers interested in detailed explanation of Reed Solomon decoders, they may refer to [1] and [2]. An important property of Galois field arithmetic, however, is that the result of arithmetic operations ($+$, $-$, $/$, $\times$, etc) is always in the same field.

2.2.1 Systematic Form Encoding

Consider RS codes with symbols from $GF(2^m)$, and let α be a primitive element in $GF(2^m)$. The generator polynomial of a primitive t-error correcting RS code of length $2^m - 1$ is:

$$g(x) = (x + \alpha)(x + \alpha^2) \ldots (x + \alpha^{2t}) \tag{1}$$

Let

$$a(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} \tag{2}$$

be the message to be encoded, $k = n - 2t$. The remaining $2t$ parity check symbols are the co-efficients of the remainder:

$$b(x) = b_0 + b_1 x + \ldots + b_{2t-1} x^{2t-1} \tag{3}$$

resulting from dividing the message polynomial $x^{2t}a(x)$ by the generator polynomial $g(x)$. Thus, we get the following code word overall:

$$c(x) = b_0 + b_1x + \ldots + b_{2t-1}x^{2t-1} + a_0x^{2t} + a_1x^{2t+1} \ldots + a_{k-1}x^{2t+k-1} \quad (4)$$

and the entire code word satisfies the property $c(x) \bmod g(x) \equiv 0$.

2.2.2 Decoding

When a code word is received at the receiver, it is often not the same as the one transmitted, since noise in the channel introduces errors in the system. Let us say if $r(x)$ is the received code word, we have

$$r(x) = c(x) + e(x) \quad (5)$$

where $c(x)$ is the original codeword and $e(x)$ is the error introduced in the system. The aim of the decoder is to find the vector $e(x)$ and then subtract it from $r(x)$ to recover the original code word transmitted. It should be added that there are two aspects of decoding - error detection and error correction. As mentioned before, the error can only be corrected if there are a maximum of t errors. However, the Reed Solomon algorithm still allows one to detect if there are more than t errors. In such cases, the code word is declared *uncorrectable.*

Syndrome Computation
One of the first steps to decoding a code word is the computation of syndrome. $2t$ syndrome coefficients are computed as defined in the following equation.

$$s_i = \sum_{j=o}^{N-1} r_j(\alpha^{i+1})^j, i = 0, 1, \ldots 2t-1 \quad (6)$$

If there is no error in the code word, all the syndromes computed are zero. Non-zero syndromes imply an error in the code word and these are then passed to subsequent blocks for computing the error value and error location.

Key Equation Solver
The syndromes are used to compute the error locator and error evaluator polynomial. Since we have m-bit symbol, knowing there is an error in a symbol is not enough. We also need to determine the value of the error occurred in order to determine the transmitted symbol. If there are e errors in the received code word, we can define the *error locator* polynomial $\Lambda(x)$ of degree e and the *error evaluator* polynomial $\Omega(x)$ of degree at most $e-1$ to be

$$\Lambda(x) = 1 + \lambda_1x + \lambda_2x^2 + \ldots + \lambda_ex^e \quad (7)$$

$$\Omega(x) = 1 + \omega_1x + \omega_2x^2 + \ldots + \omega_{e-1}x^{e-1} \quad (8)$$

which are related to the syndrome polynomial $S(x)$ through the *key equation* [3]

$$\Lambda(x)S(x) \equiv \Omega(x) \mod x^{2t} \quad (9)$$

where

$$S(x) = s_0 + s_1 x + \ldots + s_{2t-1} x^{2t-1} \tag{10}$$

Solving the above key equation is perhaps the hardest part of the algorithm. Once solved for $\Omega(x)$ and $\Lambda(x)$, we can determine the locations where the error occurred and the value of the error. The actual code word transmitted can then be easily determined. It is to be noted however, that it is only possible in the event that $e \leq t$. There are many algorithms to solve the key equation and they shall be covered in the section 4.2.

Chien/Forney Algorithm

Chien search involved checking whether $\Lambda(\alpha^{-j}) = 0$ for each $j, 0 \leq j \leq n-1$. If it is, then an error has occurred at j^{th} location in the received code word. The next step is to compute the value of error, Y_i that has occurred. This is computed by Forney's error value formula [1]

$$Y_i = -\frac{\Omega(x)|_{x=\alpha^{-j}}}{x\Lambda'(x)|_{x=\alpha^{-j}}} \tag{11}$$

where $\Lambda'(x)$ denotes the formal derivative of $\Lambda(x)$, which is simply (for Galois arithmetic)

$$\Lambda'(x) = \lambda_1 + \lambda_3 x^2 + \ldots. \tag{12}$$

Thus, we get

$$x\Lambda'(x) = \lambda_1 x + \lambda_3 x^3 + \ldots \tag{13}$$

which is the summation of the terms of odd degree in the computation of $\Lambda(x)$. Thus, it can be computed during Chien search itself.

Error detection in the case of more than t errors can be done during Chien search. If the number of roots computed is equal to the degree of $\Lambda(x)$, the number of errors, e is less than or equal to t; otherwise we know that $e > t$.

3 Channel Model

Before we proceed to the actual decoder implementation, it is important to look at the channel model itself. Since UWB (Ultra Wide Band) is not very well explored yet, it is important to analyse how the channel would behave at the frequency and the data rate under consideration. Most of the error-correcting codes are often concerned with situations where the channel is assumed to be memory-less, as it allows for easy theoretical analysis. When the model becomes too complicated, it is often possible to retain only the essential properties of the channel and use a less complex model. One of the most common models used for modelling transmission over land mobile channels is the Gilbert-Elliott model. In this model a channel can be either in a good state or a bad state depending on the signal-to-noise ratio *(SNR)* at the receiver. For different state, the probability of error is different. As expected, in a good state, the probability of error is lower than that of the channel in the bad state. The dynamics of the channel are modeled as a first order Markov chain, a model which Wang and Moayeri [19] and Wang and Chang [20] , in spite of

its simplicity, showed to be very accurate for a Rayleigh fading channel. In [21] , Ahlin presented a way to match the parameters of the GE model to the land mobile channel, an approach that was generalized in [19] to a Markov model with more than two states. In [22], Wilhelmsson and Laurens evaluated the performance of block error-correcting codes over the GE Channel. They also provided a good and easy to understand analysis of obtaining the parameters for a land mobile channel.

3.1 Gilbert Elliott Channel Model

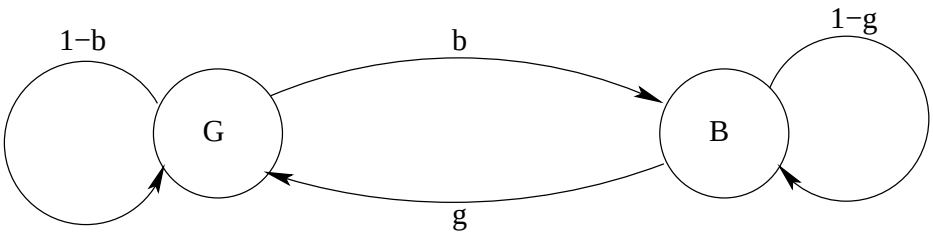

Figure 3: *The Gilbert-Elliott Channel Model.*

Figure 3 shows the GE Channel Model. Two states are shown represented by G and B representing the good and the bad state respectively. Further, the transition probability from the good state to the bad state is shown as b and from the bad to the good state as g. The probability for error in state G and B is denoted by $P(G)$ and $P(B)$ respectively. What follows is a concise explanation of the model. A more detailed analysis can be found in [22] and [23].

To obtain the relation between the physical quantities and the parameters of the model, Rayleigh fading was considered. The amplitude α of the received signal is therefore

$$f(\alpha) = \frac{2\alpha}{\overline{\gamma}} e^{-\alpha^2/\overline{\gamma}}, \quad \alpha \geq 0 \tag{14}$$

and the SNR is exponentially distributed, given by

$$f(\gamma) = \frac{1}{\overline{\gamma}} e^{-\gamma/\overline{\gamma}}, \quad \gamma \geq 0 \tag{15}$$

where $\overline{\gamma}$ is the average SNR of the received signal. Since, we have two states in the GE Channel, let γ_t be the threshold for the SNR, where the channel changes the state. The stationary probabilities for the two states are given by

$$P^{stat}(B) = 1 - e^{-\rho^2} \tag{16}$$

$$P^{stat}(G) = e^{-\rho^2} \tag{17}$$

where $\rho^2 = -\gamma_t/\overline{\gamma}$. From these we arrive at the channel transition probabilities given by the following equations,

$$g = \frac{\rho f_D T_s \sqrt{2\pi}}{e^{\rho^2} - 1} \tag{18}$$

$$b = \rho f_D T_s \sqrt{2\pi} \tag{19}$$

where $f_D = \frac{\nu f_c}{c}$. Here ν is the relative speed of the objects communicating, f_c is the frequency of the carrier and c is the velocity of light. T_s is the symbol duration. f_D indicates the doppler frequency while $f_D T_s$ signifies the normalized doppler frequency. The error probabilities in different states can be computed as follows:

$$P_e(B) = \frac{1}{P^{stat}(B)} \int_0^{\gamma_t} f(\gamma) P_e(\gamma) d\gamma. \tag{20}$$

and

$$P_e(G) = \frac{1}{P^{stat}(G)} \int_{\gamma_t}^{\infty} f(\gamma) P_e(\gamma) d\gamma. \tag{21}$$

where $P_e(\gamma)$ is the symbol error probability given the value of γ and $f(\gamma)$ is as defined above. $P_e(\gamma)$ depends on the type of modulation used, but for BPSK (Binary Phase Shift Keying) - one of the common modulation schemes, we have $P_e(\gamma) = Q(\sqrt{r2\gamma})$, where [24]

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-t^2/2} dt \tag{22}$$

3.2 Simulation

Following were the parameters set for the simulation of the Ultra Wide Band channel:

- carrier frequency = 4.0 GHz
- information rate = 480 Mbps

Two sets of simulation were run for different threshold reading, and each for different velocity. The threshold here signifies the SNR level at which the channel changes states, and the velocity the relative velocity of the communicating agents. The first set was with the threshold set to 5dB lower than the average SNR and the other with 10dB less than the average. As the choice of threshold can affect the accuracy of the model significantly at times, different values were taken. Two different values of velocity were also considered - 1 m/s for slow movement and 8 m/s for fast movement. However, due to the very high data bit rate involved the transition probability is very small. Therefore, these channel transitions become very rare events and simulations determined the error probabilities for codewords beginning in a certain state. These were then weighted by the steady state probability of the corresponding state and added together to obtain the overall probability rate. Two measures, the bit error rate and the symbol error rate are computed and plotted. The simulation was run for 10,000 codewords to get a good estimate for each state.

	5 dB		10 dB	
	vel = 1 m/s	vel = 8 m/s	vel = 1 m/s	vel = 8 m/s
$f_D T_s$	2.78E-08	2.22E-07	2.78E-08	2.22E-07
ρ	0.562341325		0.316227766	
g	1.05E-07	8.42E-07	2.09E-07	1.67E-06
b	3.92E-08	3.13E-07	2.20E-08	1.76E-07
$P^{stat}(G)$	0.728893414		0.904837418	
$P^{stat}(B)$	0.271106586		0.095162582	

Table 1: *The common parameters obtained from Mathematica*

	5 dB			10 dB		
$\overline{\gamma}$	γ_t	$P_e(B)$	$P_e(G)$	γ_t	$P_e(B)$	$P_e(G)$
13	8	0.047379902	1.39392E-05	3	0.123893547	0.001177155
14	9	0.037957829	2.21E-06	4	0.10298993	0.000543123
15	10	0.030346588	2.34E-07	5	0.084368039	0.000219542
16	11	0.024227994	1.50E-08	6	0.068306711	7.53028E-05
17	12	0.019323014	5.08E-10	7	0.054848762	2.10503E-05
18	13	0.015398433	7.81E-12	8	0.043824921	4.56E-06
19	14	0.012262918	4.42E-14	9	0.034928742	7.17E-07
20	15	0.009760765	9.75E-17	10	0.027806509	7.55E-08
21	16	0.007765911	0	11	0.022124086	4.81E-09
22	17	0.006176696	0	12	0.017596651	1.63E-10
23	18	0.004911393	0	13	0.01399196	2.50E-12
24	19	0.003904464	0	14	0.01112334	1.41E-14
25	20	0.003103451	0	15	0.008841354	1.99E-16
26	21	0.002466439	0	16	0.007026583	0

Table 2: *The probability parameters obtained from Mathematica*

Mathematica software was used to solve the complex mathematical equations and obtain the channel model parameters for the physical quantities under consideration. As can be seen from the afore-mentioned equations, ρ depends only on the difference in the average and threshold SNR. Therefore the steady state probability for the two states remain the same regardless of the velocity, and so does the probability of error. Table 1 shows some of the common parameters obtained from Mathematica, while Table 2 shows the probability data obtained for both 5 dB and 10 dB threshold.

3.2.1 Simulation Results

As can be seen from the Figures 4-7, the error probabilities decrease with increase in SNR as expected. All the figures show the symbol and the bit error probability observed.

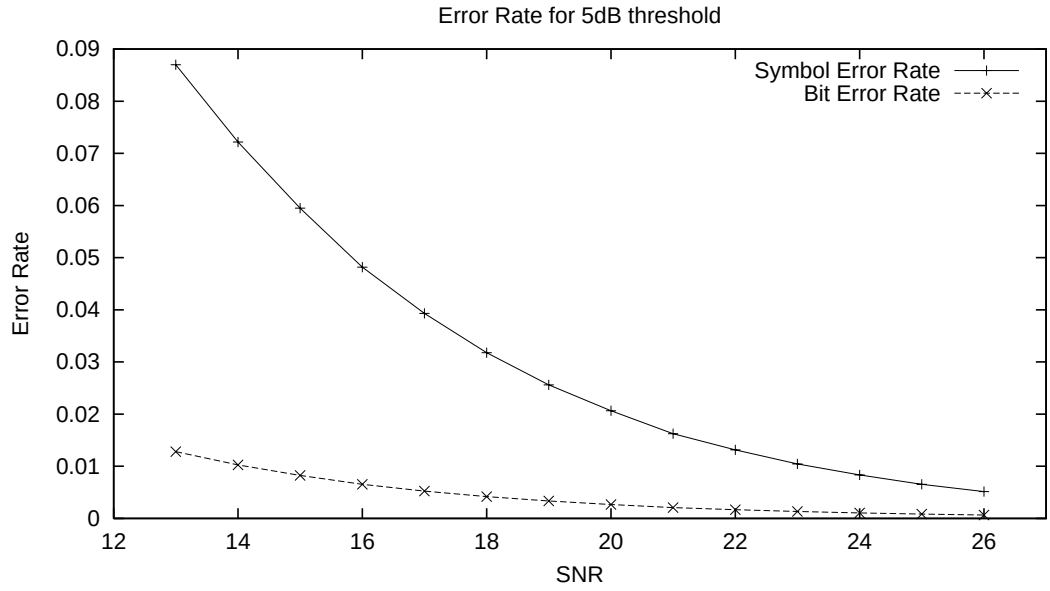

Figure 4: *The Error probability for symbol and bits when the bad channel threshold is kept at 5 dB below the average SNR level. This graph uses normal scale for representation along Y-axis.*

Figure 4 shows the error rate when the threshold for the bad state is set at 5dB lower than the average SNR, while Figure 5 shows the same graph but using a logarithmic scale for the Y-axis. Figures 6 and 7 show the corresponding graphs when the threshold is set at 10dB below the average SNR. As expected the error rates follow a linear relationship with the increasing SNR on the logarithmic scale. It can also be noticed that the symbol error rate is almost 8 times that of the bit error rates, which is expected as each symbol has 8 bits. We see that the bit error rates for the two cases (5dB and 10dB) are almost exactly same while the symbol error rates becomes equal around 20dB average SNR level. Also, we notice that around 20dB average SNR, the symbol error rate is about 0.02, which corresponds to an average of 5 symbol errors in a code word of 255 symbols.

4 Decoder Structure

This section explains the architecture of various blocks in more detail. The conventional architecture is presented first and later the modifications and improvements suggested are summarized.

4.1 Syndrome Computation

Figure 8 shows how a typical syndrome computation cell looks like. $2t$ syndrome cells are connected either in parallel or in series depending on how the output is desired. This in

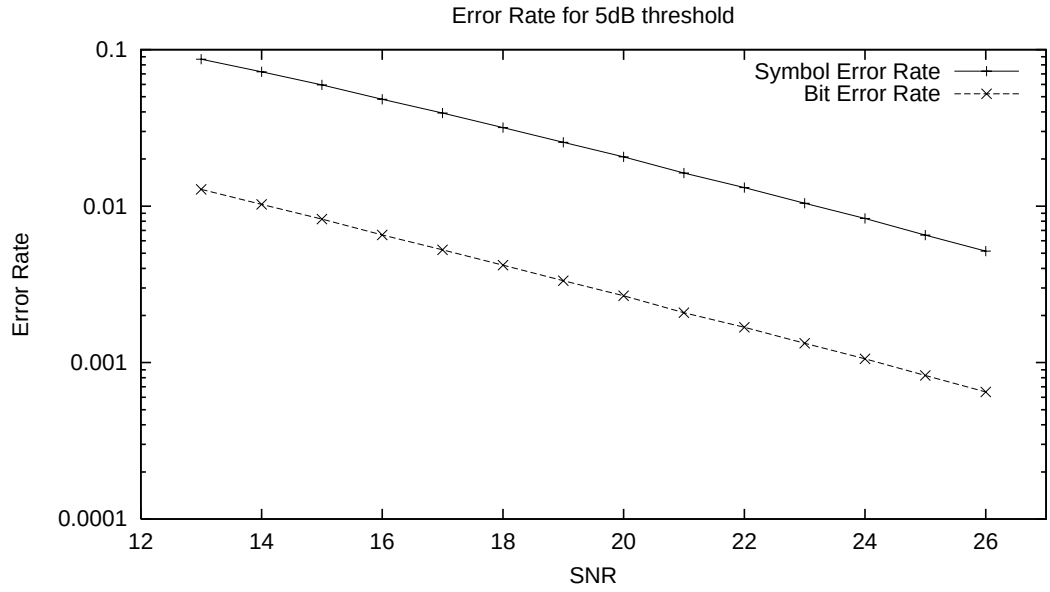

Figure 5: *The Error probability for symbol and bits when the bad channel threshold is kept at 5 dB below the average SNR level. This graph uses logarithmic scale for representation along Y-axis.*

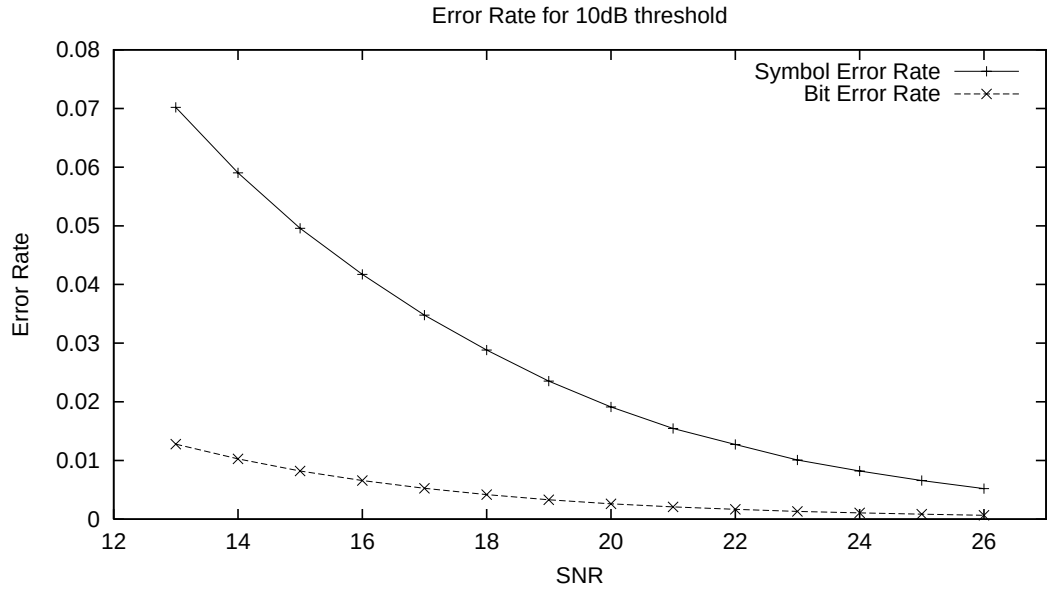

Figure 6: *The Error probability for symbol and bits when the bad channel threshold is kept at 10 dB below the average SNR level. This graph uses normal scale for representation along Y-axis.*

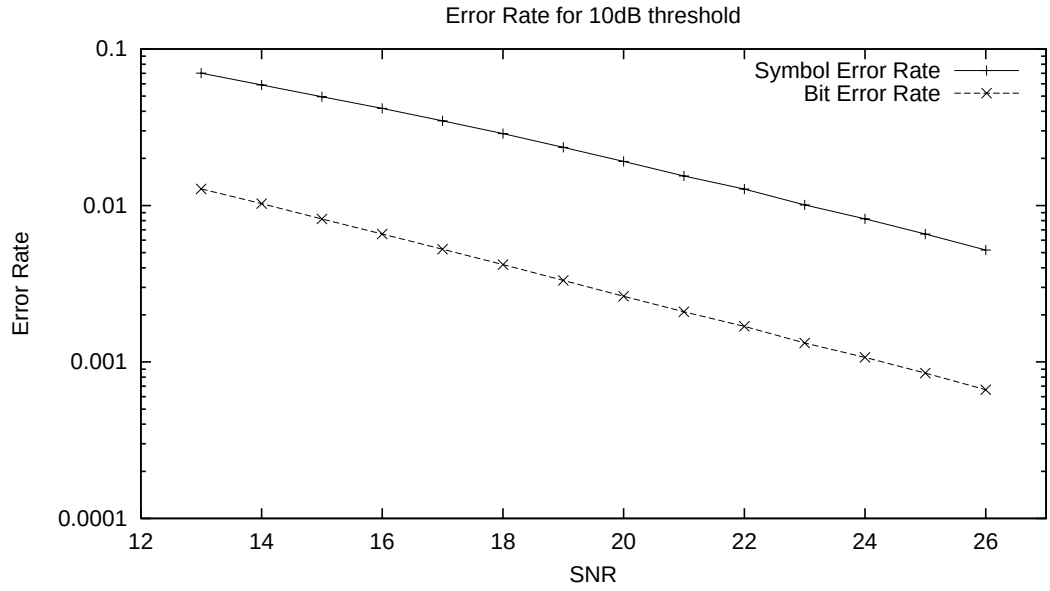

Figure 7: *The Error probability for symbol and bits when the bad channel threshold is kept at 10 dB below the average SNR level. This graph uses logarithmic scale for representation along Y-axis.*

turn depends on the algorithm used for solving the key equation, e.g. Euclidean algorithm takes the input serially [4], while Berlekamp Massey requires all the syndromes in parallel [3].

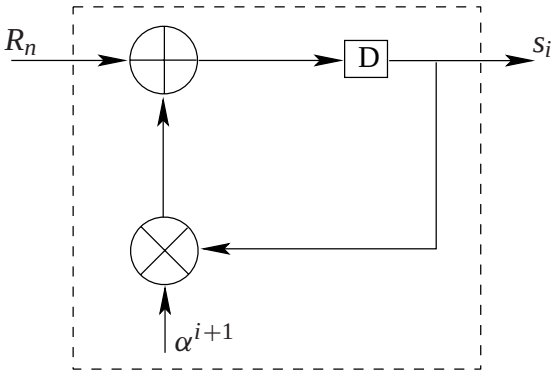

Figure 8: *A typical syndrome computation cell.*

As shown in figure 8, each cell requires the following resources:

- 1 delay FF
- 1 FFM (constant-variable multiplier)
- 1 adder

It is to be noted that each of these resources are meant for m-bit symbol and that $2t$ of such cells are required for the entire structure. An extra MUX and FF is needed for serial output of syndromes. The symbols received are input in the order R_n, $R_{n-1}, \ldots R_1$. Thus, it takes n cycles to compute the syndromes in the serial implementation.

4.2 Key Equation Solver

We now arrive at the most difficult part of the entire flow, the *Equation Solver.* A number of algorithms are available for this particular section. Trade-offs occur between the latency of the algorithm and the silicon area needed for the implementation. Critical time delay is also an important consideration as it determines the maximum frequency of operation.

4.2.1 Euclidean Algorithm

This is one of the most commonly employed algorithm. The original Euclidean algorithm was accepted for ITU G.975 recommendation. More details on it can be found in [6].

Original Euclidean
The original Euclidean consists of $2t$ divider and t multiply blocks. In this architecture ROM is used for FFI*(Finite Field Inversion).* The critical path delay as mentioned in [6] is (ROM + AND + 2 $\times$ MULT + ADD + 2$\times$MUX). The overall latency for this block is $2t$ cycles.

A slight variation to the original Euclidean algorithm has been presented in [7]. It is called *Configurable Multi-mode Design.* The design is very regular and can be adapted for different RS Codes. Extra hardware is needed in this design, but promises lower critical delay.

Modified Euclidean
This is a division free algorithm and hence no ROM is needed for this block. $2t$ blocks are connected like a systolic array [6]. The critical path delay for the algorithm is (MULT + ADD + MUX). The overall latency is $3t + 37$, but it can be operated at a frequency that is 1.8 times faster than the original algorithm. Variants of this algorithm with a fully pipelined FFM have achieved even higher frequencies with extra hardware [4].

Decomposed inversion-less Euclidean algorithm
This was proposed in [8]. Hardware is reused to reduce the hardware needed. The main motivation is that the key equation solver has a much lower latency as compared to the syndrome computation block and as such, most of the time the hardware for key-equation solver is not used. In this architecture only 3 FFM's are used, but the latency is very high. Therefore, a larger FIFO buffer is needed.

4.2.2 Berlekamp Massey

This algorithm is believed to have the least hardware complexity. The reason is that the hardware can be reused to compute the error evaluator polynomial after the error locator polynomial has been computed [5]. [3] provides a very good description of BM algorithm.

Modified BM
The error evaluator polynomial is computed after the error locator polynomial. It leads to fewer multiplications and additions i.e. for decoding one code word, fewer multiplications and additions are needed in the algorithm in modified BM as compared to BM or Euclidean algorithm [5]. A separated approach is used which results in power and hardware savings.

Decomposed inversion less modified BM
This algorithm does not require the use of inverters and is explained in [9]. Some parallelism is introduced in solving the key equation and cleverly schedules only 3 FFM's. Latency is higher for this implementation.

Parallel Approach for BM
An example of the parallel approach for BM can be seen in [10]. In this approach, extra hardware is needed and also the critical time delay is higher due to the presence of two multipliers and adders in the critical path. The latency however, is very low and hence, smaller FIFO buffer is needed.

Serial approach for BM
This approach has been demonstrated in [11]. The hardware requirement is low and as always, the latency is higher for this architecture. Larger FIFO buffer is therefore, needed for the architecture. The critical path delay is lower, and can therefore support high frequency rates.

Dual Line approach
The dual line approach was proposed in [10]. The suggested approach has a low latency like a parallel structure and has a very low critical path delay. Besides, the structure is very regular and easy to implement. However, it requires more computational elements. An example of dual-line architecture is shown in Figure 9. As can be seen in the figure, there are two series of registers, namely C and D. More details on this particular algorithm are explained in the section on Implementation Details.

Reformulated Inversion-less BM
This algorithm was discussed in [3]. Though, it requires slightly more hardware, there are tremendous gains in terms of critical time delay. No implementation has however been proposed as yet.

4.2.3 Peterson Gorenstein Zierler Algorithm

This algorithm is only mentioned for the sake of completeness. It works rather well for $t < 4$, but doesn't scale well [12].

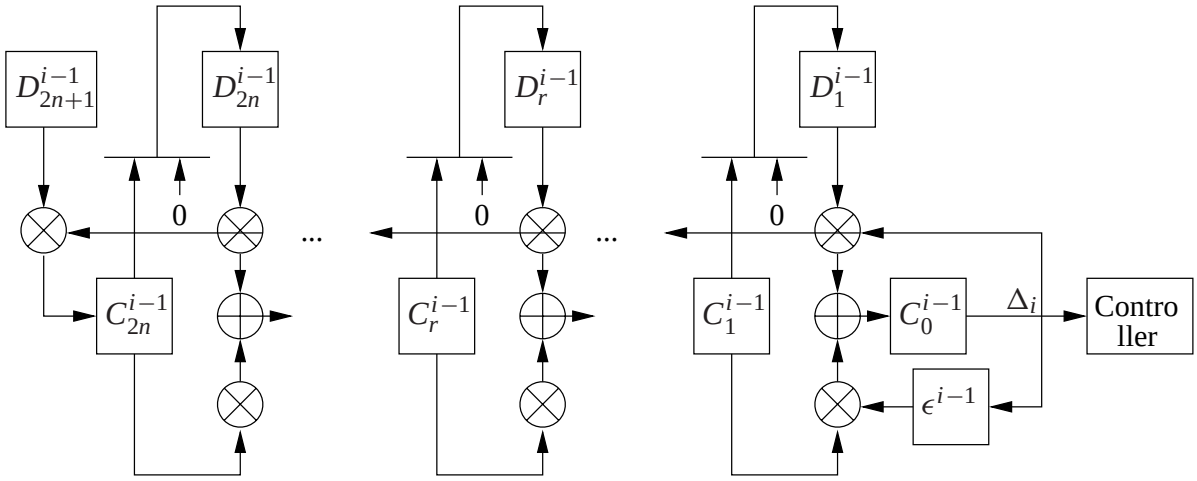

Figure 9: *Dual-line architecture for modified Berlekamp Massey.*

4.3 Chien/Forney Algorithm

Figure 10 shows a typical cell used in both Chien search and Forney evaluator [4]. Not many variations for this block are suggested. It is a common practice, however, to increase the throughput rate by multiplying hardware. However, syndrome computation block also needs to be duplicated in that case.

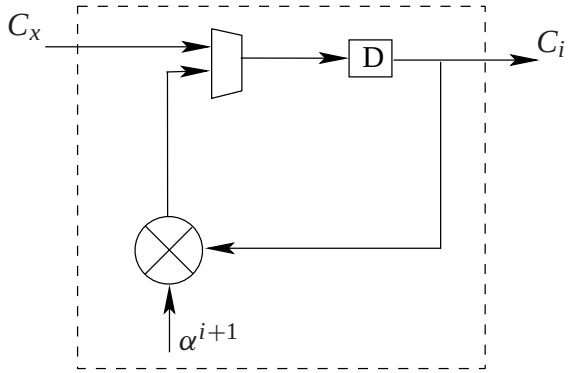

Figure 10: *A typical computation cell used in Chien/Forney.*

4.4 Finite Field Multiplier

Finite Field Multiplier (*FFM*) is the most resource intensive computation element in terms of gates needed. Therefore, various designs have been proposed in the literature for the same. There are two kinds of multipliers, namely constant-variable and variable-variable. Constant-variable multiplies are used in the syndrome computation and Chien search block, while variable-variable multipliers are used in key-equation solver. Constant-

variable multiplier can be implemented with much fewer gates as it can be optimised accordingly. It normally requires around 3 to 24 XOR gates depending on the constant [14] while a variable-variable multiplier requires around 77 XOR and 64 AND gates. These figures are for a 2-input gate.

4.4.1 Fully Parallel Multiplier

A fully parallel multiplier was proposed in [4]. As with most ideas, there is a trade-off involved between hardware and speed. This architecture is capable of being operated at a very high speed as it can be fully pipelined and thus provide a lower critical path delay. However, it requires more hardware than normal - about 52 XOR and 80 AND gates.

4.4.2 Composite Field Multiplier

Many papers have also suggested use of a Galois multiplier on composite field, e.g. in [15]. This multiplier often requires about 25% less hardware as compared to a conventional multiplier.

Various other ideas for optimising hardware requirement for a multiplier have been discussed in [16], [17] and [18].

5 Taxonomy in Design Space

Figure 11 shows the various architectures available. Table 3 shows the hardware requirements of computational elements used in various architectures. Estimates have been made from the figures drawn in the papers when actual counts could not be obtained for a particular architecture. It should be noted that this is only the estimate of computational elements and, therefore, more gates will be needed for control overhead. Total latency of the various blocks will determine the size of FIFO.

5.1 Design Decisions

In order to choose a good architecture for the application, various things have to be taken into account.

- Gate count: Determines the silicon area to be used for development. A one time production cost but can be critical if it is too high.
- Latency: Latency is defined as the delay between the received code word and the decoded code word. The lower the latency, the smaller is the FIFO buffer size required and therefore, it also determines the silicon area to a large extent.
- Critical path delay: It determines the minimum clock period, i.e. maximum frequency that the system can be operated at.

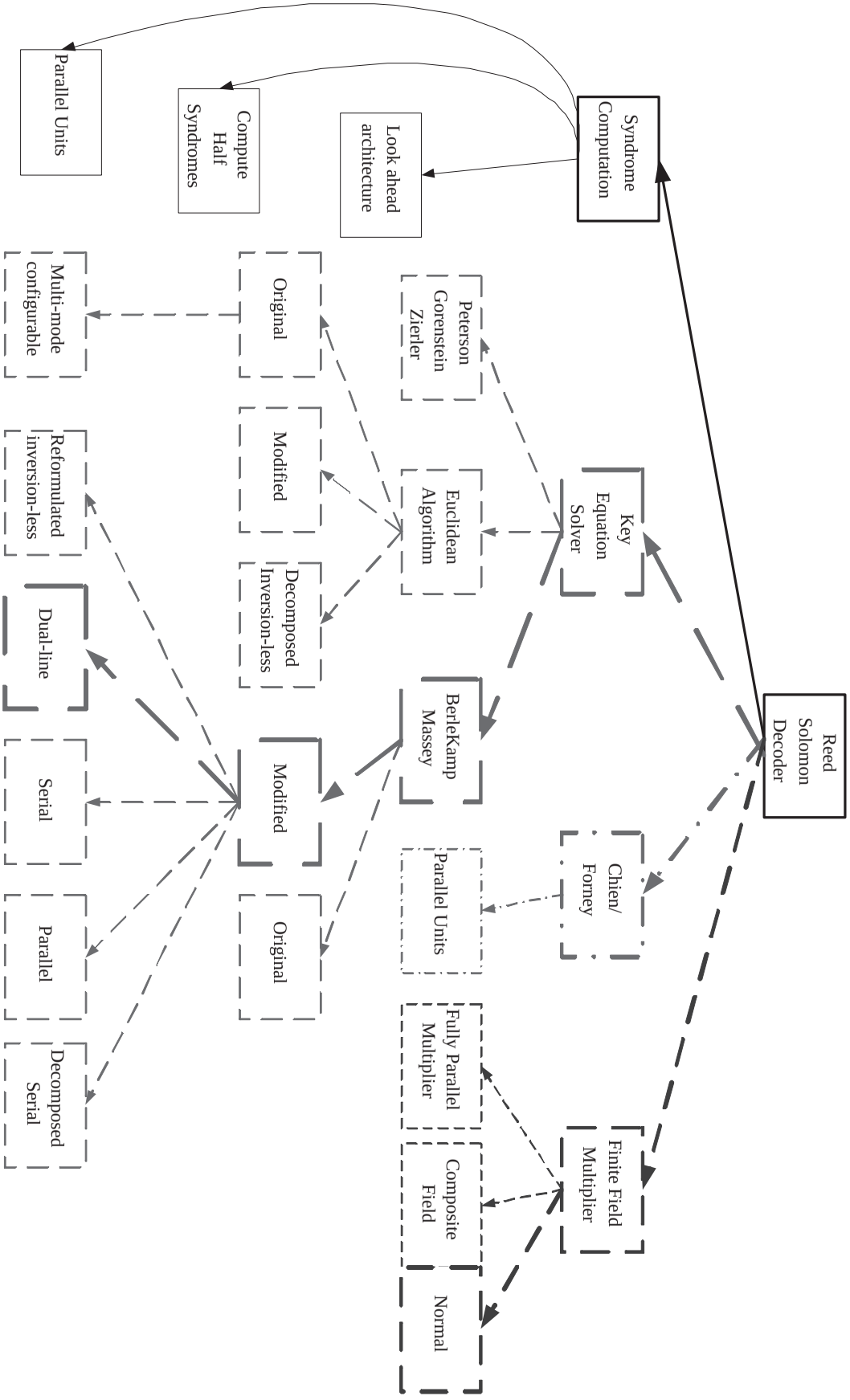

Figure 11: *Design Space Exploration*

Architecture	Blocks	Adders	Multipliers	Muxes	Latches	Latency	Critical Path Delay
Syndrome Computation [4]	2t	1	1	1	2		
Total		2t	2t	2t	4t	n	Mul + Add + Mux
Look ahead architecture (x units)	2t	x	x	1	2		
Total		2xt	2xt	2t	4t	n/x	Mul + Add + Mux
Original Euclidean [6]							
Divider Block	2t	1	1	3	2		
Multiply Block	t	2	1	3	3		
Total (Estimates)		4t	3t	9t	7t		
Actual [13]		4t + 1	3t + 1	11t + 4	14t + 6	4t - 3	ROM + 2×Mul + Add + 2×Mux
Modified Euclidean [6]							
Degree Computation Block	2t	2	0	7	7		
Polynomial Arithmetic Block	2t	2	4	8	19		
Total (Estimates)		8t	8t	30t	52t		
Actual [13]		8t	8t	40t + 2	78t + 4	10t + 8	Mul + Add + Mux
Decomposed inversion-less [8]		1	3	1	3t + 1	2t×(t+1)	Mul + Add + Mux
Modified BerleKamp Massey							
Serial		1	3	4	3t + 2	2t×(2t+2)	Mul + Add + Mux
Decomposed inversion-less [9]		2	3	2	5	2t×(t+1)	Mul + Add + Mux
Parallel		t	3t + 2	t	3t + 1	2t	2×Mul + 2×Add + Mux
Dual-line [10]		2t	4t + 1	2t	4t + 1	3t + 1	Mul + Add
Reformulated inversion-less [3]		3t + 1	6t + 2	3t + 1	6t + 2	2t	Mul + Add
Chien/Forney		2t	2t + 2	2t + 2	2t + 10	4	max(Mul + Add, ROM)

Table 3: *Summary of hardware utilization of various architectures*

Table 3 shows a summary of all the above mentioned parameters. For our intended UWB application, speed is of prime concern, as it has to be able to support data rates as high as 480 Mbps, and perhaps even 1 Gbps in the near future. At the same time, power has to be kept low, as it is to be used in portable devices as well. This implies that the active hardware at any time should be minimised. Also, the overall latency and gate count of computational elements should be low, since that would determine the total silicon area of the design.

5.1.1 Key Equation Solver

Reformulated inversion-less and dual line implementation of the modified Berlekamp Massey have the smallest critical path delay among all the alternatives of the Key Equation Solver. Astute reader would have noticed that the critical path delay of syndrome computation block seems to be higher than that of Key Equation Solver. However, the multiplier used in syndrome computation and Chien blocks is a constant-variable multiplier, which has lower critical path delay (and also less hardware) than that of Key Equation solver, which uses a variable-variable multiplier. When comparing inversion-less and dual-line implementation, dual line is a good compromise in latency and computational elements needed. The latency is one of the lowest and it has the least critical path delay of all the architectures summarized above. Thus, dual-line implementation of the BM algorithm was chosen for the key-equation solver. Another benefit of this architecture is that the design is very regular and hence easy to implement.

5.1.2 Syndrome and Chien/Forney

These sections are not as critical as the KE solver as mentioned earlier. Hardware could be duplicated if even higher data rates are desired. Power saving measures can be applied in addition, regardless of what architecture is chosen for KE solver.

5.1.3 RS Code

As we can see from Table 3, the hardware requirement for the entire block is a function of t, the error correction capability, and the latency is a function of both n and t. Thus, while we want to have a code with high error correction capability, we can not have a very high value of t as the hardware needed is proportional to it. The value of n determines the bit-width of the symbol and therefore the hardware needed, but only logarithmically. However, one would want to have a value of $n = 2^m - 1$, to derive maximum benefit out of the hardware. $RS(255, 239)$ is a very common code used, since it works on 8-bit symbol, and has an error correction capability of 8.

5.2 Highlights

Table 4 shows the various parameters for choosing dual line architecture with $n = 255$, $k = 239$ and $t = 8$. The overall critical path delay is hence Mul + Add. However, it should be noted that in Table 4 different kind of multipliers (constant-variable and variable-variable) are grouped together for a rough estimate.

Architecture	Adders	Multipliers	Muxes	Latches	Latency
Syndrome Computation	2t	2t	2t	4t	n
Dual-line	2t	4t + 1	2t	4t + 1	3t + 1
Chien/Forney	2t	2t + 2	2t + 2	2t + 10	4
Total	6t	8t + 3	6t + 2	10t + 11	3t + n + 5
For Parameters above	48	67	50	91	284

Table 4: *Summary of hardware utilization for Dual-line architecture*

6 Implementation Details

6.1 Design Flow

Figure 12 shows the design flow for development of the decoder. As shown in the figure, the first step was to develop a C-model for the decoder. 'Gcc' compiler was used to compile the code and to check if the code worked correctly. Output of each intermediate stage was compared with the expected output according to the algorithm with the aid of an example. Details of C-code development will be explained in a later section.

Once the algorithm was fully developed and tested in C, VHDL-code development started. One of the options was to use an automated tool like ART-builder for generating the VHDL-code from C. However, it was decided to hand write the VHDL, since it gives more flexibility and it can be often coded more efficiently. The VHDL code was structured such so it could be completely synthesized with ease. A wrapper class was written around it, in order to test it. This VHDL code was compiled and tested using Cadence tools. 'Ncsim' was used to simulate the system and generate the output stream for the same input tests as were used for testing C code. The output stream from VHDL and C were then compared.

When this output was found to be matched for various input test cases, synthesis experiments were started. Precision RTL by Mentor Graphics was first used to see if the code was synthesible, and later to optimize the design. Quartus II tool from Altera was also used to see the usage and frequency of operation for Altera chips. The *edif* netlist generated from Precision RTL was also synthesized on the Quartus tool to see the timing characteristics of the chip. The results from both the flows are discussed in the *Results* section. Ambit from Cadence was later used to analyse the hardware usage and frequency of operation after various optimisation settings. All the synthesis tools, namely Precision

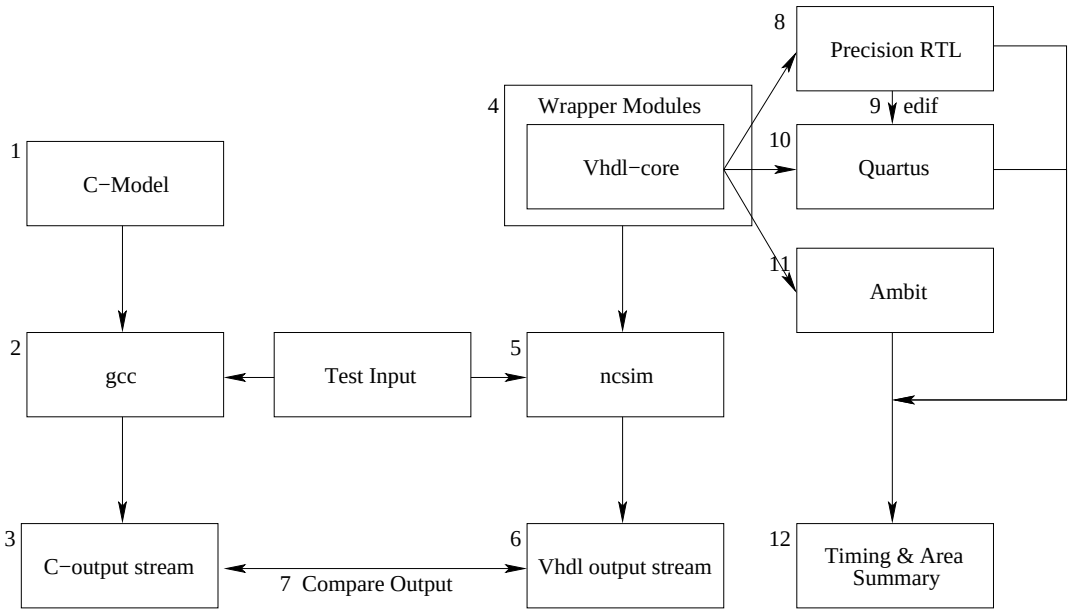

Figure 12: *Design flow of development process*

RTL, Quartus and Ambit were used to obtain an estimate of timing and area.

6.1.1 Design Flow for Power Estimation

The design flow needed for power estimation has been explained in Figure 13. As shown in the figure, the core VHDL modules are optimised and synthesized using *ambit*. The synthesized model is written out into a verilog netlist using *ambit* itself. Once the netlist is obtained, this is then compiled using *ncvlog* into the work library together with the technology library. The library used is for the same technology as the one used for synthesis. As can be seen, the wrapper modules are actually written in VHDL, while the compiled core was from the verilog. Thus, to allow interaction between the two, the top interface of the work library, is extracted into a VHDL file and then compiled into the work library. This is done using *ncshell* and *ncvhdl* respectively. This being done, the wrapper modules can now be compiled into the work library and the design is now ready for elaboration and simulation.

From this point onwards, two approaches can be used. Either *ncelab* and *ncsim* can be used purely for simulating the synthesized design, or *dncelab* and *dncsim* can be invoked which are essentially the same tools, but also includes the *DIESEL* routines for estimating the power dissipated in the design. *Diesel* is an acronym for DIssipation Estimation Software Extension for Logic simulation. As the name says, it provides existing logic simulators with additional functionality. Diesel basically keeps track of the instantaneous signal transitions that occur during a simulation. By combining this transition information with a one-time library characterization, it determines the instantaneous supply current, and derivatives thereof. *Diesel* is an internal tool developed within Philips and estimates

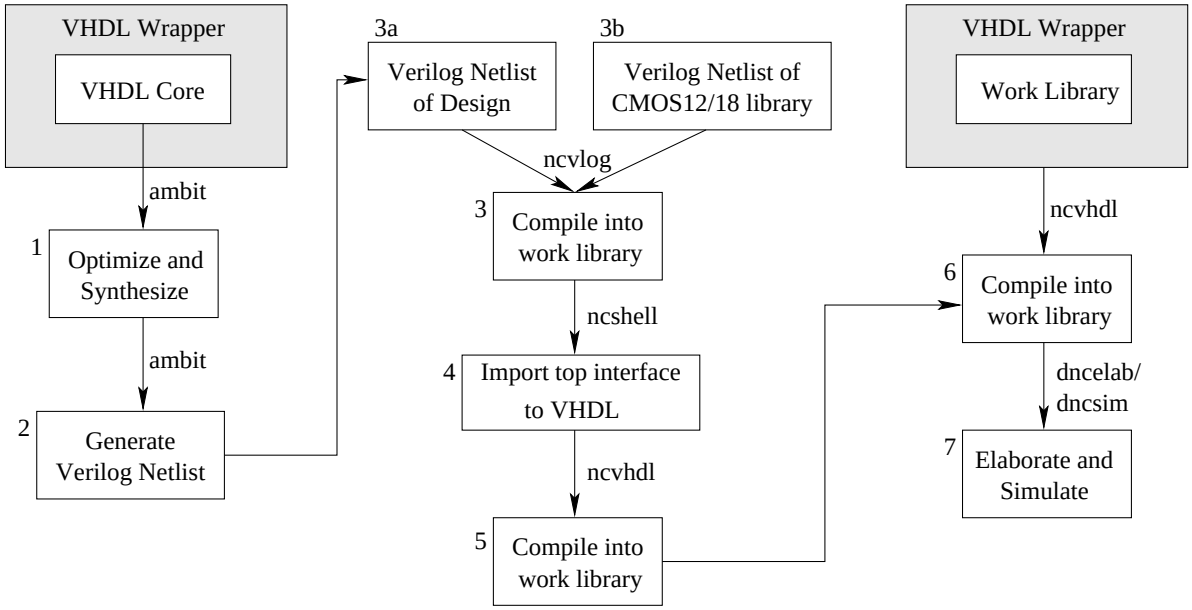

Figure 13: *Design flow for the estimation of power*

the power for the simulated design, and hence the accuracy of the results depend on the input taken.

6.2 C-Code Development

As mentioned earlier, the syndrome computation and Chien/Forney is very standard. The key-equation solver, however, can vary to a great extent. Therefore, a very brief description of that block is presented here. As stated earlier the dual-line algorithm presented in [10] was used for this. However, not all the details are presented in it.

6.2.1 Algorithm for Key Equation Solver

There are essentially two series of registers C and D which are continually updated. As mentioned earlier, key equation solver needs to compute $\Omega(x)$ and $\Lambda(x)$. They are initialised as

$$C_k^{(0)} = \begin{cases} S_{k+1} & ,\text{for } 0 \leq k \leq 2t-1 \\ 1 & ,\text{for } k = 2t \end{cases} \tag{23}$$

$$D_k^{(0)} = \begin{cases} S_k & ,\text{for } 1 \leq k \leq 2t-1 \\ 0 & ,\text{for } k = 2t \\ 1 & ,\text{for } k = 2t+1 \end{cases} \tag{24}$$

These are then updated as follows:

$$C_k^{(i)} = \begin{cases} \epsilon^{(i-1)} \times C_{(k+1)}^{(i-1)} & ,\text{when } \Delta_i = 0 \\ \epsilon^{(i-1)} \times C_{(k+1)}^{(i-1)} + \Delta_i \times D_{(k+1)}^{(i-1)} & ,\text{when } \Delta_i \neq 0 \end{cases} \tag{25}$$

$$D_k^{(i)} = \begin{cases} 0 & ,\text{when } k = 2t - i \\ C_{(k)}^{(i-1)} & ,\text{when } k \neq 2t - i \wedge (\Delta_i \neq 0 \wedge 2L_{i-1} \leq i - 1) \\ D_{(k)}^{(i-1)} & ,\text{otherwise} \end{cases} \tag{26}$$

When i becomes $2t$, i.e. after $2t$ iterations, the first $t + 1$ registers of C, contain $\Lambda(x)$. (Please refer to [10] for details on how other variables are updated). After that the registers are re-initialised as follows.

$$C_k^{(2t)} = \begin{cases} \text{unchanged} & ,\text{for } 0 \leq k \leq t \\ 0 & ,\text{for } t + 1 \leq k \leq 2t \end{cases} \tag{27}$$

$$D_k^{(2t)} = \begin{cases} 0 & ,\text{for } 1 \leq k \leq t \\ S_k & ,\text{for } t + 1 < k \leq 2t + 1 \end{cases} \tag{28}$$

The same formula is applied for update of C registers, except that ϵ remains unchanged now and D registers are not updated at all. After $t + 1$ iterations, the error evaluator polynomial $\Lambda(x)$ is contained in C registers.

6.2.2 Structure of the Code

The basic structure of the code mimics the decoder structure as well. The code has been made highly modular for easy debugging and understanding of the code. Further, the code has been commented using JavaDoc format to follow the commenting convention such that it makes easier to understand. A quick overview is presented in the Algorithm 1. The algorithm was progressively tested for various symbol sizes from 3-bit onwards all the way to 8-bit symbol. This was to ensure that the code was fully customisable for any number of bits and to also allow for easy testing. It was easy to debug the code for a lower bit RS Code.

6.3 VHDL Development

After an intensive test of the code developed in C, VHDL code for the same was developed. Figure 14 shows the block diagram for the VHDL code developed. As can be seen in the figure, there are five main blocks in the core VHDL module, one for each basic function in the algorithm. The 'memory block' is a passive element, providing only a FIFO buffer. In the actual model some more inputs have been defined in order to account for global resets and valid signals.

In addition to the five modules, an RS package was defined which contained the lookup table for Galois Field. The lookup table was needed for Forney evaluator. A C-routine was written to automatically generate the package body to suit the RS Code specification.

Generate $GF(2^m)$ and $G(x)$
Read the input $a(x)$
Compute the remainder $b(x)$ when $a(x) \times x^{2t}$ is divided by $G(x)$
Generate transmit buffer
Introduce errors {Simulate noise}
Compute Syndromes
if $S(x) = 0$ **then**
    Declare no error
else {Error in code word}
    Compute $\Omega(x)$ and $\Lambda(x)$ {Key Equation Solver}
    for all j such that $0 \leq j \leq n-1$ **do**
        if $\Lambda(\alpha^{-j}) = 0$ **then** {Chien Search}
            Compute Y_i {Forney Evaluator}
            Add Y_i to the received symbol.
        end if
    end for
end if
Compare the output with the original code word

Algorithm 1: Pseudo Code for RS Decoder

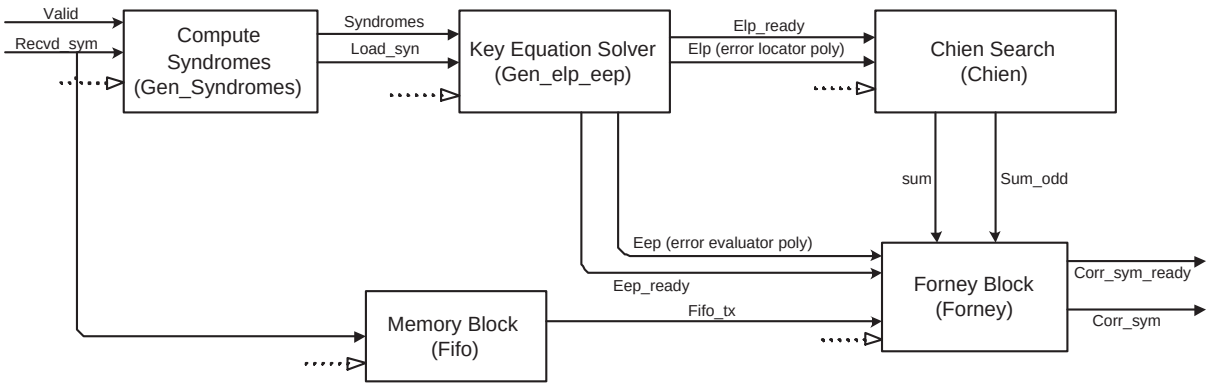

Figure 14: *Block diagram of the decoder developed in VHDL*

The modules and the RS package itself are completely customisable to suit any RS code. The Galois field multiplier was also defined in RS package body. This is to allow for modifications in the Galois field multiplier and test the results for different implementations of the multiplier. When coding the VHDL modules, Philips CoReUSE guidelines were followed.

A wrapper class was written around this core module for testing. The test bench used for the testing was the same as the one used in C. The VHDL code listing is also provided for in the Appendix. Figure 15 shows the classes built around the core module to enable proper testing of the module.

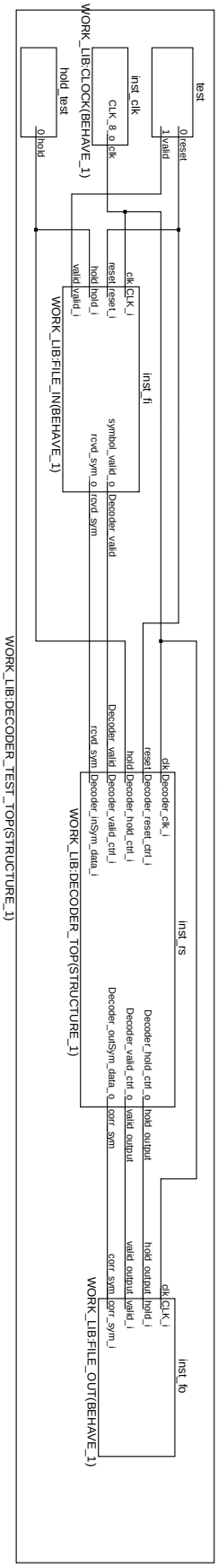

Figure 15: *Wrapper Modules around the core module*

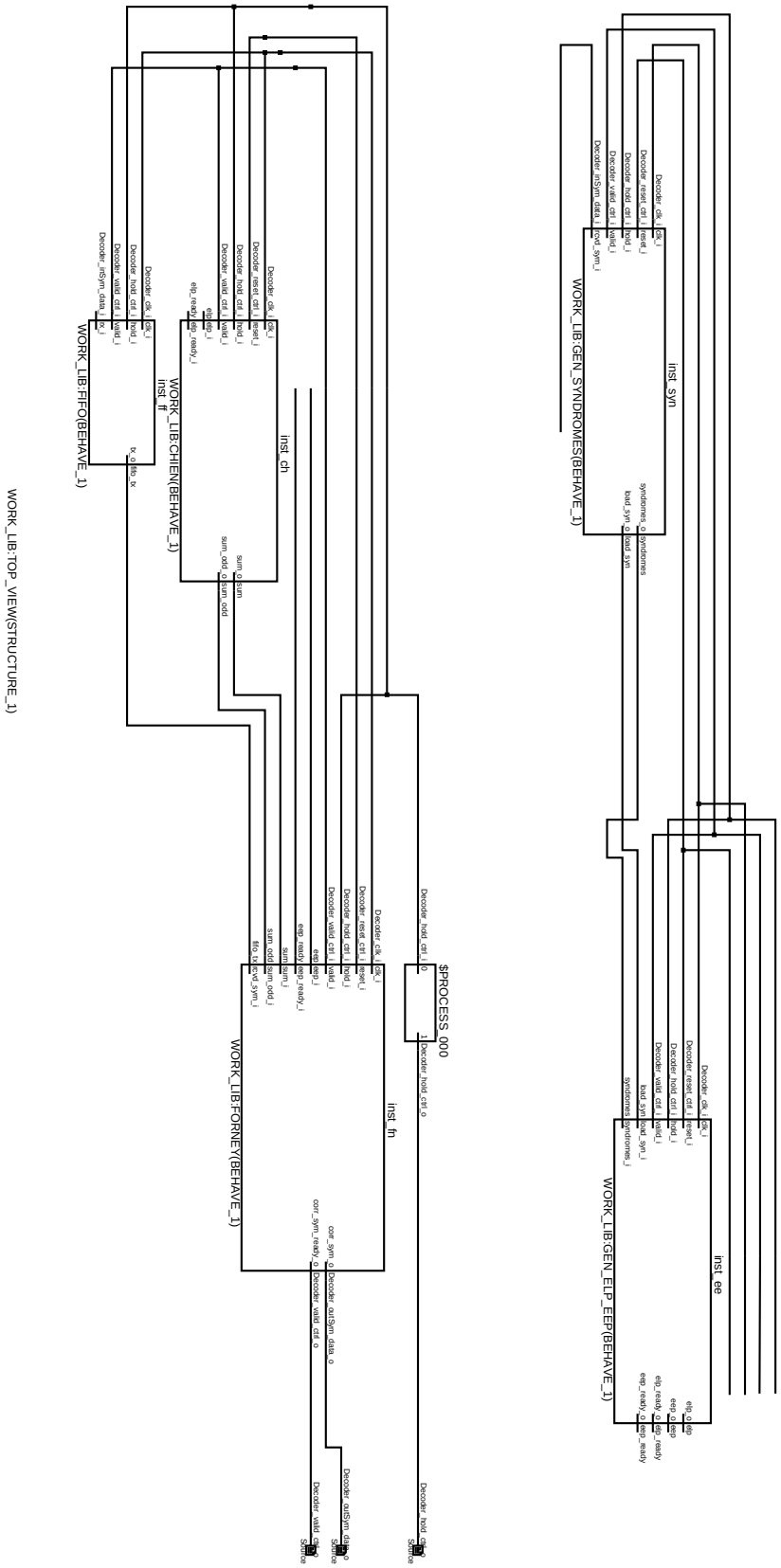

Figure 16: *Full Schematic of the top view*

6.4 Simulation

Simulation was carried using Cadence SimVision. After compilation and elaboration, the model was continually tested in SimVision to ensure that each module worked correctly. Besides testing the functionality, it was also important to test for latency in the model.

As shown in Figure 15, the test bench for simulation consisted of the file input, file output, clock generator and two test modules. The test module was written to test the functionality of reset and valid signals. Various patterns of these were supplied to test the functionality and the output in the simulator observed. A *hold_test* block was also written to test the functionality of holding the system in the event a hold signal was received. The file input and output blocks were used to allow easy input from a file and to dump the output in a file respectively. The output file allowed easy comparison with the output generated from the C-program.

6.5 Synthesis

Arriving at the synthesizable model from the original VHDL model required some minor changes. The enclosing 'if' structure was slightly modified to not test for any event change on 'reset'. Also, only 'reset' and 'clock' was tested in it. Figures 17 - 21 show the respective blocks developed in VHDL. The input ports of the block are on the left side, while the output ports are on the right side.

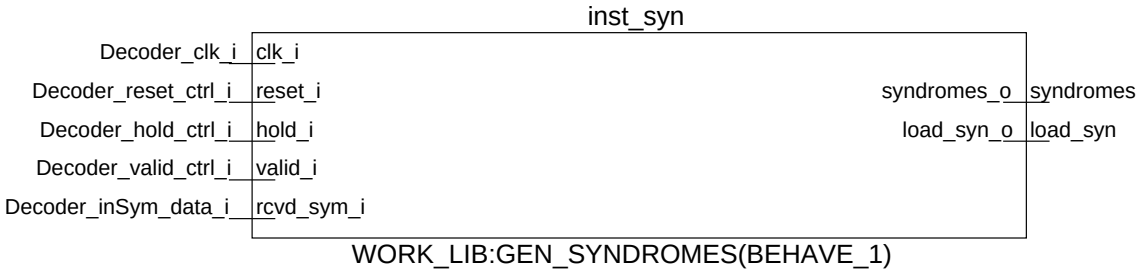

Figure 17: *Block diagram of the syndrome computation block*

The first synthesis experiments were carried out with Precision RTL by Mentor Graphics. Minor corrections as mentioned above were made to make the code synthesizable. The experiments provided an initial idea of the resource consumption of the decoder on Altera FPGA. Later, Ambit was used to get an idea of the area required for ASIC development. Various optimisation options were played around with to see the limitations of the algorithms. Quartus II from Altera was also used in the end to check for the resource utilization and the timing analysis report on Altera FPGA.

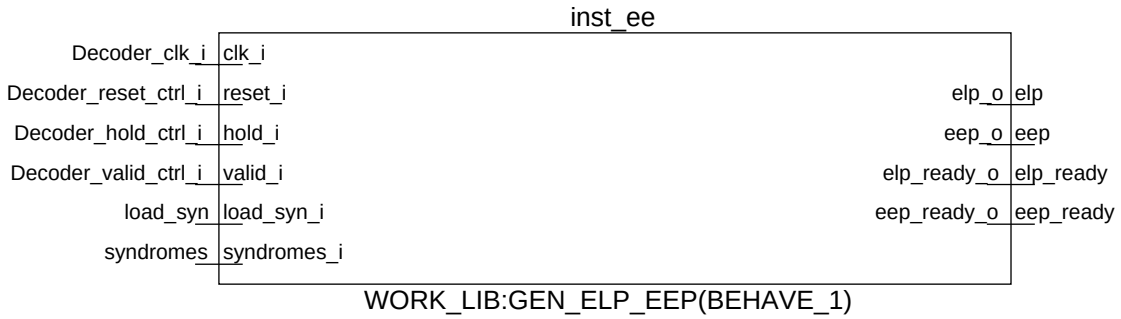

Figure 18: *Block diagram of the ELP and EEP computation block*

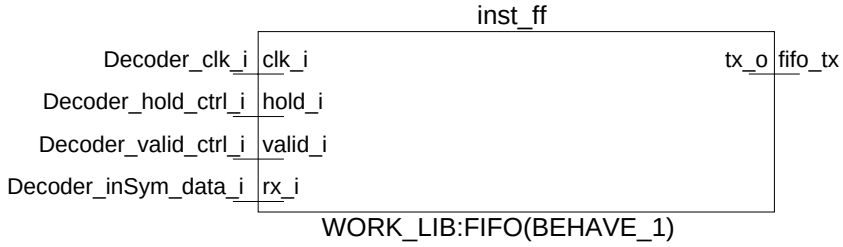

Figure 19: *Block diagram of the FIFO buffer*

6.6 Power Estimation

Co-simulating the synthesized and the VHDL wrapper as explained earlier was carried out using *ncsim.* This however, presented a problem. The synthesized design had a number of constraints on it. The one which caused the main problem was the *setup-hold* constraint. The *setup* time is defined as the time before the clock edge in during which the data should be stable and *hold* is the duration during which the data should be stable after the clock edge. In case of the co-simulation a hold-violation was encountered which was caused due to immediate availability of the data from the wrapper modules at the clock edge. This was actually a false alarm as in a real-circuit this would never happen due to wire delays and slight delay in availability of the data. The problem was solved by adding a small delay in the availability of the data at the input of the core design.

7 Results

This section covers the results of various synthesis experiments conducted. Resource utilization, timing analysis and the power consumption were used as benchmarking parameters for various tools used. The detailed report for the following can be found in the Appendix. This section only highlights the basic results.

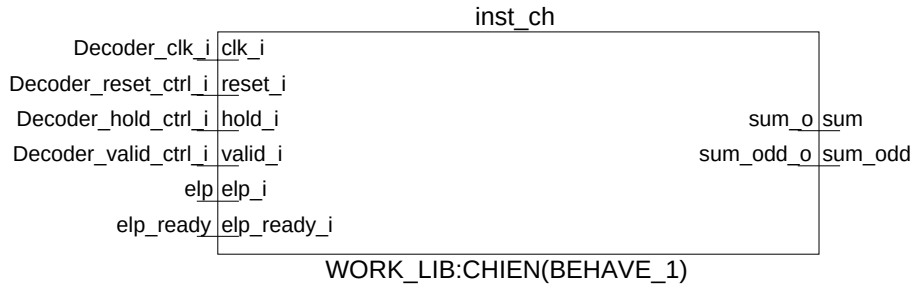

Figure 20: *Block diagram of the Chien search block*

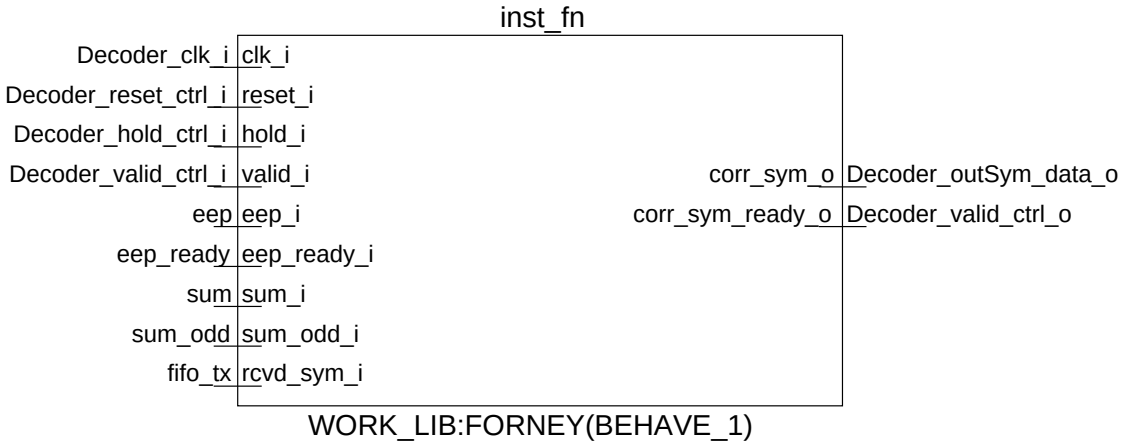

Figure 21: *Block diagram of the Forney evaluator*

7.1 Decoder

Since the decoder consisted of more than one module, a detailed account for each module wherever appropriate has been provided.

7.1.1 Precision RTL

In Precision RTL, chip EP2S15F484C from Altera in Stratix II series was selected for benchmarking. About 3.55% of LUT's (Look-up Tables) were utilised and about 7% of the input-output ports were utilized. No DSP elements, however, could be assigned by this tool. The timing report shows the minimum period as 12.414 ns i.e. a maximum frequency of 80.554 MHz. In other words it can support the data rate of $80.554 \times 8 = 644.43$ Mbps when no constraints are applied onto it. However, when the frequency constraint of 100Mhz is applied, the minimum period was found to have increased marginally to 12.886 ns. However, the values obtained by Precision RTL are only an estimate of the actual timing. The tool does a high level optimisation and generates a netlist. The actually value of timing can only be obtained once *Place and Route* is done. This is done using Quartus II.

7.1.2 Quartus II

The package was allowed to choose an appropriate chip and incidentally the one chosen was EP2S15F484C indeed, the same as the one selected for Precision RTL. Quartus II could also identify blocks in the code that were assigned to the DSP block elements. Two 9-bit DSP blocks out of the available 96 were assigned by the tool.

Various optimisation settings were tried out for time and area trade-off. When optimised for time, the critical path delay of the FPGA was found to be 10.89 ns. Thus, the FPGA could be run at a frequency of 91.82 MHz. This translates to the total bit rate of $91.82 \times 8 = 734.56$ Mbps, since we have 8-bit symbols. When it was optimised for area, the critical path delay was found to be 10.92 ns, i.e. an operating frequency of 91.52 MHz. Thus, we don't lose that much in terms of the frequency. However, we do gain a lot in terms of area. The number of ALUT's used is 19% for the case in which the design is optimised for area in contrast to 23% used in the design optimised for time. The number of FIFO memory elements in both cases is the same which is $< 1\%$. Interestingly, the balanced option produces the lowest timing delay of 10.272 ns. This would allow the FPGA to run at 97.35M Hz.

Experiments were also carried out using the *edif* netlist from Precision RTL. As expected, the timing obtained after doing optimisation on Precision was better than directly compiling and optimising in Quartus. The clock period is found to be 9.153 ns which is 11% lower than the timing provided in Quartus II directly. Thus, this can be run at a higher frequency of 109.25 MHz.

7.1.3 Ambit

Ambit was run with the library *PcCMOS18corelib*. The silicon area required was analysed for various timing constraints. A rough comparison for area of the decoder is shown in Table 5. This table shows the area requirement when the constraint was set to 5 ns. The tool succeeded in producing the design with the time delay of only 5.554 ns, which translates to a frequency of 180 MHz or 1.44 Gbps. Interestingly, not much gain was made in terms of area when the timing constraint was relaxed, even though, the critical path delay increased to 12 ns. The total number of design cells used, including the memory, for the library *PcCMOS18corelib* was 12,768.

Module	Module Area(μm^2)
Chien	15675.392
FIFO	148684.807
Forney	52936.705
Key Equation	186404.866
Syndromes	34754.560
Top View	438472.713

Table 5: *Resource utilization for the decoder in CMOS18*

Experiments were also carried out with the library *PcCMOS12corelib.* Table 6 shows the area requirement for the same. The timing constraint was set to be the same as for CMOS18 library and a critical time path of 5.092 ns was achieved, which can support close to 200 MHz frequency, i.e. 1.6 Gbps. Also, when comparing Tables 5 and 6, we can see that the area required for the same chip is almost exactly half of the one needed for CMOS18. The total number of design cells used for the library *PcCMOS12corelib* was 12,613, which is almost the same as that for *PcCMOS18corelib.*

Module	Module Area(μm^2)
Chien	7663.343
FIFO	83183.278
Forney	21608.247
Key Equation	89602.009
Syndromes	17828.014
Top View	219913.131

Table 6: *Resource utilization for the decoder in CMOS12*

7.1.4 Diesel

For power estimation, *cap_wire_exclusive* option of Diesel was enabled, and a capacitance of 2.1fF was used per load for the CMOS12 library and 2.2fF for the CMOS18 library. Rest of the options were left as default. The analysis was carried out for different time intervals and then summarized to obtain a precise dissipation of each module. It is to be noted, however, that the values provided in this section do not include the input drive power. The input drive power refers to the power needed to drive the input signals like clock, reset signals and some other control signals transferred between the testbench and the design. Needless to say, the clock signal dominates all of them. The input drive power is mentioned separately wherever necessary. Also, the input drive power estimate is only obtained for the overall block, not individual modules.

Variation With Frequency

As mentioned above the design was optimized for running at 200 MHz. However, the power estimates were obtained for three different frequencies of operation - 2.5 MHz, 25 MHz and 125 MHz. Table 7 shows the variation of power consumed for different frequencies of operation for both *PcCMOS12corelib* and *PcCMOS18corelib.* The power figures presented here are for the entire block and for an arbitrary number of errors. The input data used for these simulations is identical. Another thing to be noted is that the input drive power is a function of the frequency only and not the input data, whereas the dissipated power depends on both of them. As can be seen, both the power dissipated and the input drive power is almost exactly linear with the frequency. The reason for this lies in the number of transitions that occur in a given time, which is exactly dependent on the frequency of operation of the design. For a different input data at a given frequency, only the dissipated power varies, while the input drive power remains same.

	Power(μw)			
	CMOS12		CMOS18	
Frequency (MHz)	Diss.	Drive	Diss.	Drive
2.5	117	190	340	611
25	1, 170	1, 900	3, 390	6, 110
125	5, 720	9, 500	16, 650	30, 570

Table 7: *Power dissipation for the entire decoder for different frequencies.*

Variation With Number of Errors

Figure 22 shows the variation of power with the number of errors found in the codeword. The graph is shown for *PcCMOS12corelib* with the design operating at 125 MHz. For a different frequency of operation and library exactly the same trend is observed, and hence the results are not included here. As can be seen from the graph obtained, the power dissipated for the FIFO block is independent of the number of errors found, which is expected and self-explanatory. The power dissipation in the syndrome computation block is also independent of the number of errors. This can be again explained by the number of transitions. In syndrome computation, regardless of the number of errors, all the $2 \times t$ syndrome computation units are always active. The only difference is that if indeed the number of errors is zero, in the final iteration the syndromes become zero; this does not cause any significant reduction in terms of power.

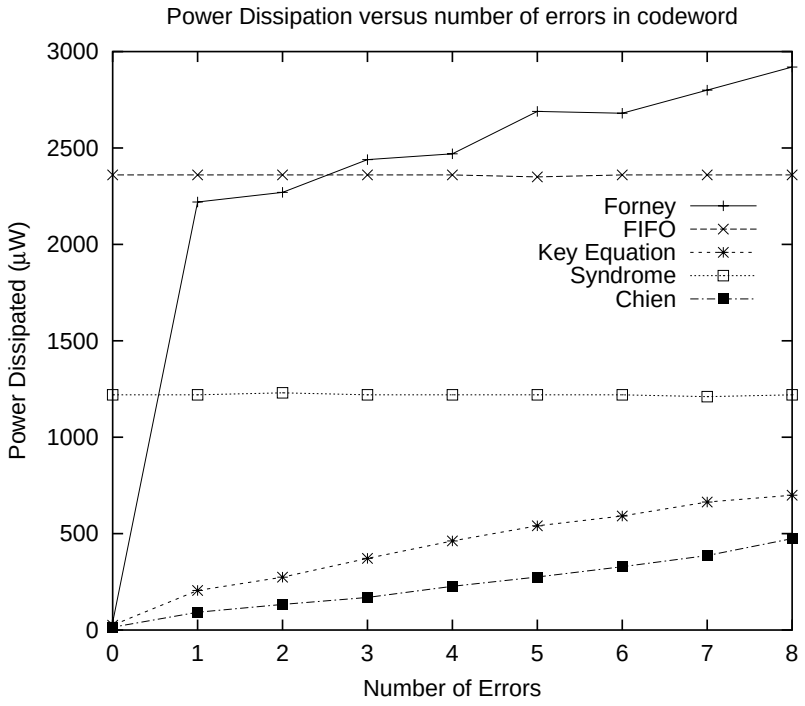

Figure 22: Variation of power dissipated with number of errors for different modules.

For the Key Equation Solver, it is clearly seen that the power dissipated increases linearly with the number of errors. The degree of the ELP is equal to the number of errors present in the codeword. Therefore, the number of transitions needed to compute the ELP also depends on the number of errors present. The degree of EEP is not the same as the number of errors, but still depends on it. Hence, the linear graph is obtained. The Chien search block also shows a linear increase in the power dissipated due to the similar reason. This block as mentioned earlier, checks if a root exists at a particular location, found by evaluating the ELP at a given location. The power consumed in evaluating the ELP is directly proportional to its degree, as it determines the number of multiplications needed to be done.

The behaviour of Forney evaluator is a bit different from the other modules. We see that the power dissipated for the codeword with an even number of errors is not significantly larger to the one with the previous number of errors. The reason lies in the fact that the degree of EEP for codeword with one error is often the same as the one with two errors, and so on and so forth. However, as a general rule, there is still an increase in the power dissipation, because of some computation that is done for each error found.

Distribution of Power in Different Modules

Figure 23 shows a distribution of power when the maximum number of errors correctable in the code word are found, while Figure 24 shows the distribution when the code word is received intact. As can be seen, in the case of no errors, bulk of the power is consumed in computing syndromes, apart from the memory. In the event of maximum errors detected, the Forney block consumes the maximum power. As mentioned earlier, this does not include the power that is dissipated in clocking of the circuit. Another thing to note is that the Key Equation Solver is not always active. Since these polynomials are only computed once for the entire codeword, it is active for only 25 cycles, which is only a tenth of the codeword size. The power figure mentioned here is the averages power estimate. This is to say that the actual power dissipated by this module when it is active is ten times that of the average estimate provided here.

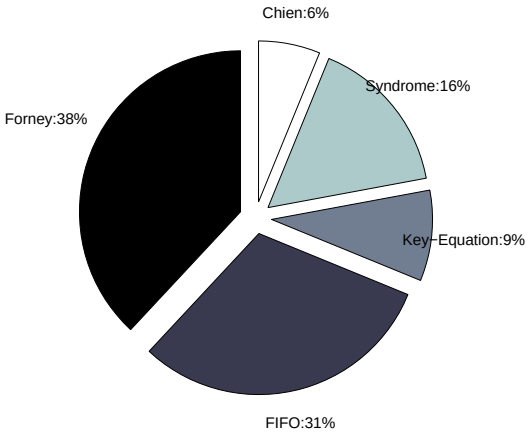

Figure 23: Power consumed by various blocks when 8 errors are found

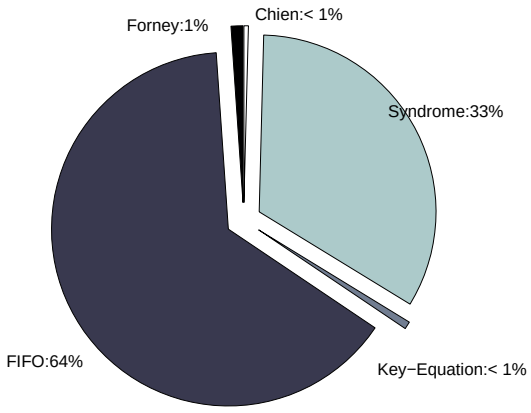

Figure 24: Power consumed by various blocks when no errors are found

7.2 Encoder

Encoder consisted of only one single module. Also, this module is functionally very similar to the syndrome computation block of the decoder. Therefore, the results obtained for this block are very close to it.

7.2.1 Precision RTL

The same chip for the encoder was selected as the one in the decoder i.e. EP2S15F484C from Altera in Stratix II series. As expected, the same number of Input/Output ports were utilized as that of the decoder. This is due to the fact that the interface for both the encoder and decoder is exactly same. The percentage of LUT's (Look-up Tables) utilized is however, 5.5% which is about 2% more than the decoder. The timing report obtained is almost similar to the one obtained for the decoder, and it is expected to run at 78MHz. However, as mentioned earlier, this is merely an estimate of the actual timing which can only be obtained after the actual *Place and Route* is done.

7.2.2 Quartus II

With Quartus II, there is only marginal difference from the data obtained from Precision RTL where the resource utilization is concerned. The timing, however, is vastly different. Quartus II provides a maximum speed of operation as 207 MHz when the actual VHDL module was compiled and 219 MHz when the *edif* netlist from Precision RTL was used.

7.2.3 Ambit

As with the decoder, experiments were carried out with both *PcCMOS12corelib* and *PcCMOS18corelib* libraries. Since, there was only one module in it and the critical path for this module was not so long, it was possible to optimize the design for higher frequencies. With *PcCMOS12corelib* it was even possible to optimize the encoder for a frequency as high as 400 MHz. With *PcCMOS18corelib*, however, the design could be only optimised to 330MHz. Table 8 shows the comparison of area required for the encoder using different libraries and optimising for different clock periods. For the period of 5ns, the total number of design cells was 936 for *PcCMOS18corelib* and 1126 for *PcCMOS12corelib.*

	Area(μm^2)	
Period (ns)	CMOS12	CMOS18
3.3	17906.685	34504.705
5.0	16746.795	31567.873

Table 8: *Resource utilization for the encoder for different libraries.*

7.2.4 Diesel

Encoder has only one core module and is almost always active, unlike decoder which has some blocks active only for a short while. Table 9 shows the power consumption for the encoder when it is run at different clock frequencies. The table also shows the input drive power, which formed about 15% of the total power consumption for *PcCMOS12corelib* and about 21% for *PcCMOS18corelib.* As can be seen from the table, the drive power for this block is much lesser as compared to the decoder. This is because of the low activity modules in decoder like the Key Equation Solver, which are only active for a short interval of time. Besides, some modules are only active when the error is indeed found in the codeword. In contrast to this, the encoder is always active, and hence the data activity is much higher than the decoder.

	Power(μw)			
	CMOS12		CMOS18	
Frequency (MHz)	Diss.	Drive	Diss.	Drive
2.5	51	9	110	29
25	510	90	1, 103	287
125	2, 560	450	5, 490	1, 430

Table 9: *Power dissipation for encoder for different frequencies.*

8 Optimisations to Design

From the results, it was observed that the FIFO and the Forney block consumed most of the power. These blocks were investigated further and redesigned to improve the performance. The original design of FIFO involved a serial arrangement of shift-registers. This design was the most compact in terms of area but consumed more power since at every cycle all the elements were shifted by one. The design was hence, modified to have only one read and write every clock cycle. This increased the design area, but significantly reduced the power. Area of the new design of FIFO is now 109,000 μm^2 (with *PcCMOS12corelib*), while the power consumed is only 970 μW, 60% lower than the earlier design.

For the Forney block, design was optimised by combining two table lookups into one for computing the inverse of elements. This resulted in a better circuit in terms of area and also decreased the power significantly. The optimised design for Forney now occupies an area of 13,000 μm^2, about 38% lower than original design. The power consumption is lower by atleast 1.5 mW for all cases. Table 10 shows the new area distribution of the decoder.

Power analysis was repeated for the optimised design. Figure 25 shows the power distribution in various modules when there are 8 errors in the received codeword, while Figure 26 shows the distribution when the codeword is received intact. As we can see, the FIFO

Module	Module Area(μm^2)
Chien	7655.274
FIFO	108906.613
Forney	13408.329
Key Equation	89587.888
Syndromes	17719.085
Top View	237414.359

Table 10: *Resource utilization for the decoder in CMOS12 in optimised design*

now takes less than half the power in no-error case, as compared to two-thirds in the original design. In the case of 8-errors, the power consumption of Forney has now reduced to about a quarter as compared to one-third in the original design.

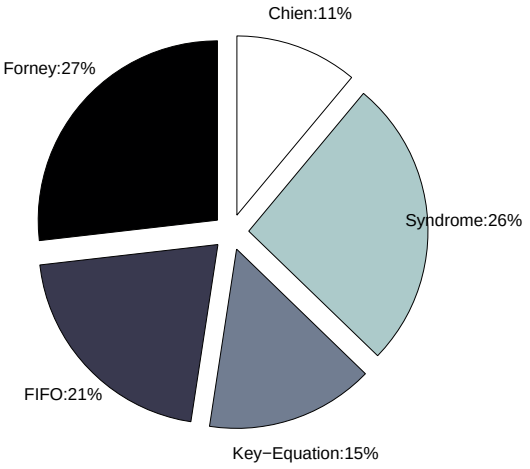

Figure 25: Power consumed when 8 errors are found in optimised design

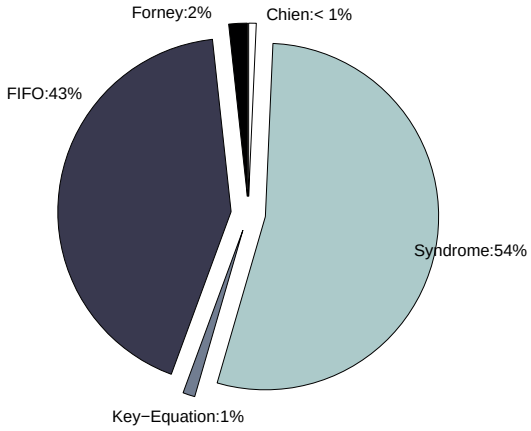

Figure 26: Power consumed when no errors are found in optimised design

Figure 27 shows the variation of power with the number of errors. The trend in the power consumption of Forney is the same as before the optimisation. The total power consumption of the design now lies between 12mW to 14mW depending upon the no-error case to when maximum errors are found. It should be noted that 9.5mW of power is consumed in driving the input. Thus, only about 2.5mW to 4.5mW is actually consumed in the transitions in the design.

8.1 Embedded Memory for FIFO

Using embedded memory for FIFO was also analysed, since in the final design embedded memory shall be used. Table 11 shows an estimate of the area and power consumption for various libraries. In the table, x refers to the number of rows, y to the number of rows per block and z to the number of blocks. The total number of words in memory is $x \times y \times z$.

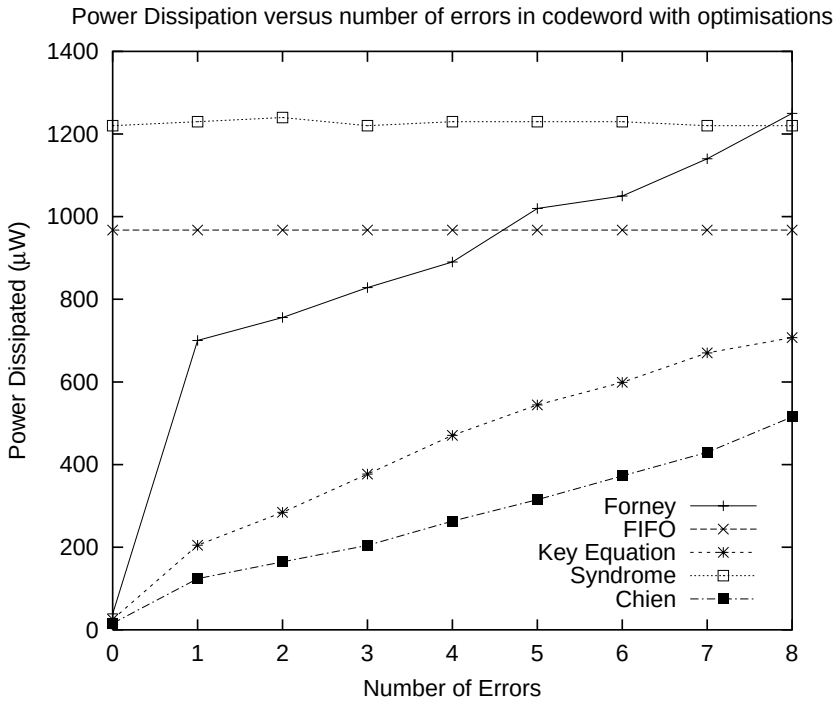

Figure 27: Variation of power dissipated with number of errors for different modules with modifications in the design.

As can be seen *AMDC C12ESRAM* is the only one that provides a lower power than our simulated design. However, all of them seem to have a better area than the synthesized design. When *AMDC C12ESRAM* is used for the final design, the area would be reduced by $0.08mm^2$, that is 80% lower than the synthesized design. The power consumption, however, remains the same.

	Size	Words			Area	Power	
Library	(bits)	x	y	z	(mm2)	(uW/MHz)	at 125 MHz
AMDC C12XSRAM	2.2K	36	8		0.02	13.03	1628.75
LTG C12FSRAM	2.2K	72	4		0.03	11.32	1415
LTG C12FSRAM	2.2K	36	8		0.02	11.57	1446.25
AMDC C12ESRAM	2.2K	72	2	2	0.02	7.2	900
LTG C12FDSRAM	2.2K	72	4		0.06	11.86	1482.5
LTG C12FDSRAM	2.2K	36	8		0.05	12.88	1610
AMDC C12EDSRAM	2.2K	72	4		0.04	10.7	1337.5
LTG C12FTSRAM	2.2K	72	4		0.03	13.9	1737.5

Table 11: *Memory Estimates for various libraries and designs*

9 Conclusions

A uniform comparison was drawn for various algorithms that have been proposed in literature. This helped in selecting the appropriate architecture for the intended application. Besides, some modifications were also suggested. Modified Berlekamp Massey algorithm was chosen for the VHDL implementation. Further, a dual line architecture was used which is as fast as serial and has low latency as that of a parallel approach.

The decoder developed was well tested and simulated in various tools available. It was tested for both FPGA and ASIC implementation. The throughput requirement has been met and the area and power estimates for the chip are also provided.

The decoder implemented is capable of running at 200 MHz in ASIC implementation, which translates to 1.6Gbps and requires only $0.44mm^2$ with CMOS18 technology and only $0.22mm^2$ with CMOS12 technology. The system has a latency of only 284 cycles for RS(255,239) code. The power dissipated in the worst case is 14mW for the decoder including the memory block when operating at 1Gbps data rate.

References

[1] R. E. Blahut; *Theory and Practice of Error Control Codes*, Addison-Wesley, 1983.

[2] S. B. Wicker and V. K. Bhargava; *Reed Solomon Codes and Their Applications*, Piscataway, NJ: IEEE Press, 1994.

[3] D. V. Sarwate and N. R. Shanbhag; *High-speed architectures for Reed-Solomon decoders*, IEEE transactions on VLSI Systems 2001.

[4] H. Lee; *High-speed VLSI architecture for parallel Reed-Solomon decoder* IEEE transactions on VLSI Systems 2003.

[5] H. C. Chang, C. C. Lin and C. Y. Lee; *A low-power Reed-Solomon decoder for STM-16 optical communications* IEEE Asia-Pacific Conference on ASIC 2002

[6] H. Lee, M. L. Yu, and L. Song; *VLSI design of Reed-Solomon decoder architectures* IEEE International Symposium on Circuits and Systems 2000.

[7] H. Y. Hsu and A. Y. Wu; *VLSI design of a reconfigurable multi-mode Reed-Solomon codec for high-speed communication systems* IEEE Asia-Pacific Conference on ASIC 2002.

[8] H. C. Chang and C. Y. Lee; *An area-efficient architecture for Reed-Solomon decoder using the inversion less decomposed Euclidean algorithm* IEEE International Symposium on Circuits and Systems 2001.

[9] H. C. Chang and C. B. Shung; *A (208,192;8) Reed-Solomon decoder for DVD application* IEEE International Conference on Communications, 1998.

[10] H. J. Kang and I. C. Park; *A high-speed and low-latency Reed-Solomon decoder based on a dual-line structure* IEEE International Conference on Acoustics, Speech, and Signal Processing, 2002

[11] H. C. Chang and C. B. Shung; *New serial architecture for the Berlekamp-Massey algorithm* IEEE Transactions on Communications, 1999

[12] S. F. Wang, H. Y. Hsu and A. Y. Wu; *A very low-cost multi-mode Reed-Solomon decoder based on Peterson-Gorenstein-Zierler algorithm* IEEE Workshop on Signal Processing Systems, 2001.

[13] H. Lee; *An area-efficient Euclidean algorithm block for Reed-Solomon decoder* IEEE Computer Society Annual Symposium on VLSI, 2003

[14] H. C. Chang, C. B. Shung and C. Y. Lee; *A Reed-Solomon product-code (RS-PC) decoder chip for DVD applications* IEEE Journal of Solid-State Circuits, 2001

[15] Y. You, J. Wang, F. Lai and Y. Ye; *VLSI design and implementation of high-speed RS(204,188) decoder* IEEE International Conference on Communications, Circuits and Systems and West Sino Expositions, 2002

[16] C. Paar and M. Rosner; *Comparison of arithmetic architectures for Reed-Solomon decoders in reconfigurable hardware* IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[17] G. C. Ahlquist, M. Rice and B. Nelson; *Optimal Finite Field Multipliers for FPGAs* FPL 1999

[18] L. Song, and K. K. Parhi; *Efficient finite field serial/parallel multiplication* International Conference on ASAP, 1996.

[19] H. S. Wang and N. Moayeri; *Finite-state Markov channel - A useful model for radio communication channels* IEEE Transaction on Vehicular Technology, 1995.

[20] H. S. Wang and P. C. Chang; *On verifying the first-order Markovian assumption for a Rayleigh fading channel model* IEEE Transaction on Vehicular Technology, 1996.

[21] L. Ahlin; *Coding methods for the mobile radio channel* Nordic Seminar on Digital Land Mobile Communications, 1985.

[22] L. Wilhemsson and L. B. Milstein; *On the effect of imperfect interleaving for the Gilbert-Elliott channel* IEEE Transactions on Communications, 1999.

[23] G. Sharma, A. Dholakia, and A. Hassan; *Simulation of error trapping decoders on a fading channel* IEEE Transaction on Vehicular Technology, 1996.

[24] J. G. Proakis; *Digital Communications* New York: Mc Graw Hill, 2001

A Ambit

A.1 Area Report for CMOS18

```
+-----------------------------------------------------+
| Report  | report_area                               |
|---------+-------------------------------------------|
| Options | -hierarchical -cells > ./top_view.area.log |
+---------+-------------------------------------------+
| Date    | 20040729.122531                           |
| Tool    | bg_shell                                  |
| Release | v5.10-s081                                |
| Version | Jul 31 2003 16:13:20                      |
+---------+-------------------------------------------+
| Module  | top_view                                  |
+-----------------------------------------------------+

+-------------------------------------------------------------+
|  Block report for module 'top_view'  | Current | Cumulative |
|                                      |  Module |            |
|--------------------------------------+---------+------------|
| Number of combinational instances    |       1 |      10428 |
| Number of noncombinational instances |       0 |       3143 |
| Number of hierarchical instances     |       5 |          9 |
| Number of blackbox instances         |       0 |          0 |
| Total number of instances            |       6 |      13580 |
| Area of combinational cells          |  16.384 | 226525.188 |
| Area of non-combinational cells      |   0.000 | 211947.525 |
| Total cell area                      |  16.384 | 438472.713 |
| Number of nets                       |     313 |      13586 |
| Area of nets                         |   0.000 |      0.000 |
| Total area                           |  16.384 | 438472.713 |
+-------------------------------------------------------------+

+-------------------------------------------------------------------------------+
|                              Cell Usage Table                                 |
|-------------------------------------------------------------------------------|
|       Cellref |    Library |  Number of |  Cell Type |  Cell Area | Total Area |
|               |            |  Instances |            |            |            |
|---------------+------------+------------+------------+------------+------------|
|        bf1tx2 |    corelib |          1 |       comb |     16.384 |     16.384 |
|         Chien |    netlist |          1 |       hier |  15675.392 |  15675.392 |
|          FIFO |    netlist |          1 |       hier | 148684.807 | 148684.807 |
|        Forney |    netlist |          1 |       hier |  52936.705 |  52936.705 |
|   gen_elp_eep |    netlist |          1 |       hier | 186404.866 | 186404.866 |
| gen_syndromes |    netlist |          1 |       hier |  34754.560 |  34754.560 |
+-------------------------------------------------------------------------------+
```

A.2 Area Report for CMOS12

```
+-----------------------------------------------------+
| Report  | report_area                               |
|---------+-------------------------------------------|
| Options | -hierarchical -cells > ./top_view.area.log |
+---------+-------------------------------------------+
| Date    | 20040823.162211                           |
| Tool    | bg_shell                                  |
| Release | v5.10-s081                                |
| Version | Jul 31 2003 16:13:20                      |
+---------+-------------------------------------------+
| Module  | top_view                                  |
+-----------------------------------------------------+

+-------------------------------------------------------------+
|  Block report for module 'top_view'  | Current | Cumulative |
```

	Module	
Number of combinational instances	2	9714
Number of noncombinational instances	0	3143
Number of hierarchical instances	5	9
Number of blackbox instances	0	0
Total number of instances	7	12866
Area of combinational cells	26.224	102280.110
Area of non-combinational cells	0.000	117094.429
Total cell area	26.224	219374.539
Number of nets	314	12872
Area of nets	0.000	0.000
Total area	26.224	219374.539

Cell Usage Table

Cellref	Library	Number of Instances	Cell Type	Cell Area	Total Area
bfx1	corelib	1	comb	8.069	8.069
bfx4	corelib	1	comb	18.155	18.155
Chien	netlist	1	hier	7631.068	7631.068
FIFO	netlist	1	hier	82850.440	82850.440
Forney	netlist	1	hier	21630.436	21630.436
gen_elp_eep	netlist	1	hier	89472.908	89472.908
gen_syndromes	netlist	1	hier	17763.464	17763.464

A.3 Timing Report for CMOS18

Report	report_timing
Options	> ./top_view.timing.log
Date	20040729.122529
Tool	bg_shell
Release	v5.10-s081
Version	Jul 31 2003 16:13:20
Module	top_view
Timing	LATE
Slew Propagation	WORST
Operating Condition	CTLF_OP_COND
PVT Mode	max
Tree Type	worst_case
Process	1.500
Voltage	1.650
Temperature	125.000
time unit	1.000 ns
capacitance unit	1.000 pF
resistance unit	1.000 kOhm

```
Path 1: VIOLATED Setup Check with Pin inst_fn/corr_sym_o_reg_5/CP
Endpoint:   inst_fn/corr_sym_o_reg_5/D (v) checked with  leading edge of 'IDEAL_
CLOCK'
Beginpoint: inst_ch/sum_odd_o_reg_0/Q  (v) triggered by  leading edge of 'IDEAL_
CLOCK'
Other End Arrival Time     0.000
- Setup                    0.210
+ Phase Shift              5.000
= Required Time            4.790
- Arrival Time             5.344
= Slack Time              -0.554
Clock Rise Edge                 0.000
```



```
= Beginpoint Arrival Time       0.000
+-------------------------------------------------------------------------------+
|         Instance          |       Arc       |  Cell   | Delay | Arrival | Required |
|                           |                 |         |       |  Time   |   Time   |
|---------------------------+-----------------+---------+-------+---------+----------|
|                           | Decoder_clk_i ^ |         |       |   0.000 |   -0.554 |
| inst_ch                   | clk_i ^         | Chien   |       |   0.000 |   -0.554 |
| inst_ch/sum_odd_o_reg_0   | CP ^ -> Q v     | fd4sqx2 | 0.453 |   0.453 |   -0.100 |
| inst_ch                   | sum_odd_o[0] v  | Chien   |       |   0.453 |   -0.100 |
| inst_fn                   | sum_odd_i[0] v  | Forney  |       |   0.453 |   -0.100 |
| inst_fn/i_574             | A v -> Z ^      | ivx05   | 0.105 |   0.558 |    0.004 |
| inst_fn/i_1270            | B ^ -> Z ^      | an2x2   | 0.236 |   0.794 |    0.240 |
| inst_fn/i_1254            | A ^ -> Z v      | nd2x2   | 0.105 |   0.899 |    0.345 |
| inst_fn/i_940             | A v -> Z v      | bf2tx7  | 0.198 |   1.097 |    0.544 |
| inst_fn/i_2313            | A v -> Z ^      | nd2     | 0.082 |   1.179 |    0.626 |
| inst_fn/i_1428            | A ^ -> Z v      | ao7c    | 0.103 |   1.282 |    0.729 |
| inst_fn/i_1742            | B v -> Z ^      | nd2     | 0.106 |   1.388 |    0.835 |
| inst_fn/i_747             | B ^ -> Z v      | nr2     | 0.081 |   1.469 |    0.916 |
| inst_fn/i_1043            | B v -> Z ^      | nd2     | 0.118 |   1.587 |    1.033 |
| inst_fn/i_262             | B ^ -> Z v      | nr3     | 0.124 |   1.711 |    1.157 |
| inst_fn/i_1907            | A v -> Z ^      | ivx05   | 0.126 |   1.837 |    1.283 |
| inst_fn/i_2273            | B ^ -> Z v      | nd2x05  | 0.119 |   1.955 |    1.402 |
| inst_fn/i_2247            | A v -> Z ^      | iv      | 0.172 |   2.128 |    1.574 |
| inst_fn/i_1984            | A ^ -> Z ^      | bf1tx3  | 0.187 |   2.314 |    1.761 |
| inst_fn/i_2248            | A ^ -> Z ^      | bf2tx6  | 0.155 |   2.469 |    1.915 |
| inst_fn/i_1173            | A ^ -> Z v      | nd2     | 0.072 |   2.541 |    1.987 |
| inst_fn/i_1211            | A v -> Z ^      | nd2     | 0.074 |   2.615 |    2.062 |
| inst_fn/i_036928          | A ^ -> Z v      | nd2     | 0.068 |   2.683 |    2.129 |
| inst_fn/i_663             | A v -> Z ^      | nd2     | 0.088 |   2.771 |    2.217 |
| inst_fn/i_1924            | B ^ -> Z v      | nr2     | 0.091 |   2.862 |    2.308 |
| inst_fn/i_645             | A v -> Z ^      | nd4     | 0.119 |   2.980 |    2.427 |
| inst_fn/i_547             | A ^ -> Z v      | nr2     | 0.089 |   3.070 |    2.516 |
| inst_fn/i_913             | B v -> Z ^      | nd3     | 0.141 |   3.210 |    2.657 |
| inst_fn/i_5               | A ^ -> Z v      | nr2     | 0.103 |   3.314 |    2.760 |
| inst_fn/i_1004            | A v -> Z ^      | nr2     | 0.120 |   3.434 |    2.880 |
| inst_fn/i_618             | A ^ -> Z v      | ivx05   | 0.119 |   3.553 |    3.000 |
| inst_fn/i_880             | A v -> Z ^      | nd2     | 0.100 |   3.653 |    3.099 |
| inst_fn/i_794             | B ^ -> Z v      | nd2     | 0.064 |   3.717 |    3.164 |
| inst_fn/i_760             | A v -> Z ^      | nr2     | 0.113 |   3.830 |    3.276 |
| inst_fn/i_621             | A ^ -> Z v      | iv      | 0.083 |   3.913 |    3.359 |
| inst_fn/i_858             | A v -> Z ^      | nd2     | 0.075 |   3.987 |    3.434 |
| inst_fn/i_822             | B ^ -> Z v      | nd2     | 0.075 |   4.062 |    3.508 |
| inst_fn/i_1245            | B v -> Z ^      | nd2     | 0.146 |   4.208 |    3.654 |
| inst_fn/i_559             | A ^ -> Z v      | en2x05  | 0.290 |   4.498 |    3.944 |
| inst_fn/i_34              | A v -> Z ^      | nr2     | 0.159 |   4.657 |    4.103 |
| inst_fn/i_2524735         | C ^ -> Z v      | ao6     | 0.181 |   4.837 |    4.284 |
| inst_fn/i_1723            | B v -> Z ^      | nd2     | 0.159 |   4.996 |    4.442 |
| inst_fn/i_1726            | C ^ -> Z v      | ao7     | 0.108 |   5.104 |    4.550 |
| inst_fn/i_1734            | A v -> Z ^      | nd2     | 0.127 |   5.231 |    4.677 |
| inst_fn/i_1740            | A ^ -> Z v      | ao4     | 0.113 |   5.344 |    4.790 |
| inst_fn/corr_sym_o_reg_5  | D v             | fd2sqx2 | 0.000 |   5.344 |    4.790 |
+-------------------------------------------------------------------------------+
```

A.4 Timing Report for CMOS12

```
+-----------------------------------------+
| Report           | report_timing        |
|------------------+----------------------|
| Options          | > ./top_view.timing.log |
+------------------+----------------------+
| Date             | 20040823.155715      |
| Tool             | bg_shell             |
| Release          | v5.10-s081           |
| Version          | Jul 31 2003 16:13:20 |
+------------------+----------------------+
| Module           | top_view             |
| Timing           | LATE                 |
```



```
| Slew Propagation | WORST                 |
| PVT Mode         | max                   |
| Tree Type        | worst_case            |
| Process          | 1.500                 |
| Voltage          | 1.650                 |
| Temperature      | 125.000               |
| time unit        | 1.000 ns              |
| capacitance unit | 1.000 pF              |
| resistance unit  | 1.000 kOhm            |
+------------------------------------------+
Path 1: VIOLATED Setup Check with Pin inst_ee/C_r_reg_11_4/CP
Endpoint:   inst_ee/C_r_reg_11_4/D (v) checked with  leading edge of 'IDEAL_
CLOCK'
Beginpoint: inst_ee/D_r_reg_12_7/Q (v) triggered by  leading edge of 'IDEAL_
CLOCK'
Other End Arrival Time     0.000
- Setup                    0.128
+ Phase Shift              5.000
= Required Time            4.872
- Arrival Time             5.092
= Slack Time              -0.220
Clock Rise Edge                0.000
= Beginpoint Arrival Time      0.000
+-------------------------------------------------------------------------------+
|       Instance       |       Arc       |    Cell     | Delay | Arrival | Required |
|                      |                 |             |       |  Time   |   Time   |
|----------------------+-----------------+-------------+-------+---------+----------|
|                      | Decoder_clk_i ^ |             |       |   0.000 |   -0.220 |
| inst_ee              | clk_i ^         | gen_elp_eep |       |   0.000 |   -0.220 |
| inst_ee/D_r_reg_12_7 | CP ^ -> Q v     | df2sqx05    | 0.301 |   0.301 |    0.080 |
| inst_ee/i_35529      | A v -> Z v      | bfx1        | 0.282 |   0.583 |    0.362 |
| inst_ee/i_17998      | B v -> Z ^      | nr2ax05     | 0.442 |   1.025 |    0.804 |
| inst_ee/i_613        | D ^ -> Z v      | ao31x05     | 0.358 |   1.382 |    1.162 |
| inst_ee/i_115        | C v -> Z ^      | ao7abx05    | 0.245 |   1.627 |    1.407 |
| inst_ee/i_626        | D ^ -> Z v      | ao32abx05   | 0.169 |   1.796 |    1.576 |
| inst_ee/i_350        | D v -> Z ^      | ao32x1      | 0.205 |   2.002 |    1.781 |
| inst_ee/i_639        | D ^ -> Z v      | ao37x05     | 0.164 |   2.165 |    1.945 |
| inst_ee/i_4683       | D v -> Z ^      | ao37abx05   | 0.353 |   2.518 |    2.298 |
| inst_ee/i_4660       | B ^ -> Z v      | xns2x1      | 0.357 |   2.876 |    2.655 |
| inst_ee/i_1186       | B v -> Z v      | xns2x1      | 0.240 |   3.116 |    2.896 |
| inst_ee/i_1649       | C v -> Z ^      | ao7x05      | 0.198 |   3.314 |    3.093 |
| inst_ee/i_192        | D ^ -> Z v      | ao34x05     | 0.220 |   3.533 |    3.313 |
| inst_ee/i_1011       | B v -> Z v      | xns2x05     | 0.319 |   3.852 |    3.632 |
| inst_ee/i_427        | A v -> Z v      | xns2x05     | 0.338 |   4.190 |    3.969 |
| inst_ee/i_1665       | D v -> Z ^      | ao34x05     | 0.244 |   4.434 |    4.214 |
| inst_ee/i_4651       | D ^ -> Z v      | ao37x05     | 0.280 |   4.714 |    4.494 |
| inst_ee/i_1695       | B v -> Z ^      | nd2x05      | 0.180 |   4.894 |    4.674 |
| inst_ee/i_55217      | C ^ -> Z v      | ao44x1      | 0.198 |   5.092 |    4.872 |
| inst_ee/C_r_reg_11_4 | D v             | df2sqx05    | 0.000 |   5.092 |    4.872 |
+-------------------------------------------------------------------------------+
```

B Quartus

B.1 Direct Compilation

B.1.1 Fit Summary

```
Flow Status : Successful - Tue Aug 17 16:44:24 2004
Quartus II Version : 4.1 Build 181 06/29/2004 SJ Web Edition
Revision Name : decoder
Top-level Entity Name : decoder_top
Family : Stratix II
Total ALUTs : 2,749 / 12,480 ( 22 % )
Total registers : 888
```



```
Total pins : 22 / 343 ( 6 % )
Total memory bits : 2,248 / 419,328 ( < 1 % )
DSP block 9-bit elements : 2 / 96 ( 2 % )
Total PLLs : 0 / 6 ( 0 % )
Total DLLs : 0 / 2 ( 0 % )
Device : EP2S15F484C3
Timing Models : Preliminary
```

B.1.2 Map Summary

```
Flow Status : Successful - Tue Aug 17 16:42:33 2004
Quartus II Version : 4.1 Build 181 06/29/2004 SJ Web Edition
Revision Name : decoder
Top-level Entity Name : decoder_top
Family : Stratix II
Total combinational functions : 2396
Total registers : 888
Total pins : 22
Total memory bits : 2,248
DSP block 9-bit elements : 2
Total PLLs : 0
Total DLLs : 0
```

B.1.3 Timing Analyzer Summary

```
------------------------------------------------------------------------------------
Timing Analyzer Summary
------------------------------------------------------------------------------------

Type           : Worst-case tsu
Slack          : N/A
Required Time  : None
Actual Time    : 6.785 ns
From           : Decoder_hold_ctrl_i
To             : gen_syndromes:inst_syn|syndromes_o[6][7]
From Clock     :
To Clock       : Decoder_clk_i
Failed Paths   : 0

Type           : Worst-case tco
Slack          : N/A
Required Time  : None
Actual Time    : 5.335 ns
From           : forney:inst_fn|corr_sym_o[1]
To             : Decoder_outSym_data_o[1]
From Clock     : Decoder_clk_i
To Clock       :
Failed Paths   : 0

Type           : Worst-case tpd
Slack          : N/A
Required Time  : None
Actual Time    : 6.507 ns
From           : Decoder_hold_ctrl_i
To             : Decoder_hold_ctrl_o
From Clock     :
To Clock       :
Failed Paths   : 0

Type           : Worst-case th
Slack          : N/A
Required Time  : None
Actual Time    : 0.741 ns
From           : Decoder_inSym_data_i[2]
To             : gen_syndromes:inst_syn|syn_v[0][2]
From Clock     :
```



```
To Clock       : Decoder_clk_i
Failed Paths   : 0

Type           : Worst-case Minimum tco
Slack          : N/A
Required Time  : None
Actual Time    : 5.034 ns
From           : forney:inst_fn|corr_sym_ready_o
To             : Decoder_valid_ctrl_o
From Clock     : Decoder_clk_i
To Clock       :
Failed Paths   : 0

Type           : Worst-case Minimum tpd
Slack          : N/A
Required Time  : None
Actual Time    : 6.507 ns
From           : Decoder_hold_ctrl_i
To             : Decoder_hold_ctrl_o
From Clock     :
To Clock       :
Failed Paths   : 0

Type           : Clock Setup: 'Decoder_clk_i'
Slack          : -2.272 ns
Required Time  : 125.00 MHz ( period = 8.000 ns )
Actual Time    : 97.35 MHz ( period = 10.272 ns )
From           : gen_elp_eep:inst_ee|lpm_counter:i_v_rtl_0|cntr_rs8:auto_generated|safe_q[1]
To             : gen_elp_eep:inst_ee|l_v[5]
From Clock     : Decoder_clk_i
To Clock       : Decoder_clk_i
Failed Paths   : 294

Type           : Total number of failed paths
Slack          :
Required Time  :
Actual Time    :
From           :
To             :
From Clock     :
To Clock       :
Failed Paths   : 0

---------------------------------------------------------------------------------------
```

B.2 Compilation from EDIF netlist

B.2.1 Fit Summary

```
Flow Status : Successful - Wed Aug 25 18:10:54 2004
Quartus II Version : 4.1 Build 181 06/29/2004 SJ Web Edition
Revision Name : EdifCheck
Top-level Entity Name : decoder_top
Family : Stratix II
Total ALUTs : 3,120 / 12,480 ( 25 % )
Total registers : 888
Total pins : 22 / 343 ( 6 % )
Total memory bits : 2,248 / 419,328 ( < 1 % )
DSP block 9-bit elements : 0 / 96 ( 0 % )
Total PLLs : 0 / 6 ( 0 % )
Total DLLs : 0 / 2 ( 0 % )
Device : EP2S15F484C3
Timing Models : Preliminary
```


B.2.2 Map Summary

```
Flow Status : Successful - Wed Aug 25 18:09:21 2004
Quartus II Version : 4.1 Build 181 06/29/2004 SJ Web Edition
Revision Name : EdifCheck
Top-level Entity Name : decoder_top
Family : Stratix II
Total combinational functions : 2720
Total registers : 888
Total pins : 22
Total memory bits : 2,248
DSP block 9-bit elements : 0
Total PLLs : 0
Total DLLs : 0
```

B.2.3 Timing Analyzer Summary

```
--------------------------------------------------------------------------------
Timing Analyzer Summary
--------------------------------------------------------------------------------

Type           : Worst-case tsu
Slack          : N/A
Required Time  : None
Actual Time    : 8.259 ns
From           : Decoder_hold_ctrl_i
To             : gen_elp_eep:inst_ee|D_r_14_0
From Clock     :
To Clock       : Decoder_clk_i
Failed Paths   : 0

Type           : Worst-case tco
Slack          : N/A
Required Time  : None
Actual Time    : 5.615 ns
From           : forney:inst_fn|corr_sym_o_7
To             : Decoder_outSym_data_o[7]
From Clock     : Decoder_clk_i
To Clock       :
Failed Paths   : 0

Type           : Worst-case tpd
Slack          : N/A
Required Time  : None
Actual Time    : 6.069 ns
From           : Decoder_hold_ctrl_i
To             : Decoder_hold_ctrl_o
From Clock     :
To Clock       :
Failed Paths   : 0

Type           : Worst-case th
Slack          : N/A
Required Time  : None
Actual Time    : 0.399 ns
From           : Decoder_inSym_data_i[2]
To             : gen_syndromes:inst_syn|syn_v_5_2
From Clock     :
To Clock       : Decoder_clk_i
Failed Paths   : 0

Type           : Worst-case Minimum tco
Slack          : N/A
Required Time  : None
Actual Time    : 5.088 ns
From           : forney:inst_fn|corr_sym_ready_o
To             : Decoder_valid_ctrl_o
```



```
From Clock     : Decoder_clk_i
To Clock       :
Failed Paths   : 0

Type           : Worst-case Minimum tpd
Slack          : N/A
Required Time  : None
Actual Time    : 6.069 ns
From           : Decoder_hold_ctrl_i
To             : Decoder_hold_ctrl_o
From Clock     :
To Clock       :
Failed Paths   : 0

Type           : Clock Setup: 'Decoder_clk_i'
Slack          : N/A
Required Time  : None
Actual Time    : 109.25 MHz ( period = 9.153 ns )
From           : chien:inst_ch|NOT_sum_odd_o_4
To             : forney:inst_fn|corr_sym_o_5
From Clock     : Decoder_clk_i
To Clock       : Decoder_clk_i
Failed Paths   : 0

Type           : Total number of failed paths
Slack          :
Required Time  :
Actual Time    :
From           :
To             :
From Clock     :
To Clock       :
Failed Paths   : 0
```

C Precision RTL

C.1 Area Report

```
************************************************
Device Utilization for EP2S15F484C
************************************************
Resource                Used    Avail   Utilization
------------------------------------------------
IOs                     22      310       7.10%
LUTs                    2673    7800     34.27%
Registers               879     419328    0.21%
Memory Bits             2264    96     2358.33%
DSP block 9-bit elems   2       0          inf%

------------------------------------------------
WARNING: This design does not fit in the device specified!
This design does not fit into any device in this technology!

*****************************************************

Library: work    Cell: decoder_top    View: structure_1

*****************************************************

  Cell                        Library  References     Total Area

 INBUF                        stratixii    12 x
```



```
 OUTBUF                        stratixii    10 x
 altshift_taps_8_283_1_3_0     OPERATORS     1 x   2264   2264 Memory Bits
 Chien                         work          1 x     96     96 Registers
                                                      0      0 LCs
                                                    167    167 LUTs
 Forney                        work          1 x     81     81 Registers
                                                      0      0 LCs
                                                    525    525 LUTs
 gen_elp_eep                   work          1 x    437    437 Registers
                                                      0      0 LCs
                                                      2      2 DSP block 9-bit elems
                                                   1834   1834 LUTs
 gen_syndromes                 work          1 x    146    146 LUTs
                                                    265    265 Registers
 stratixii_lcell_comb          stratixii     1 x      1      1 LUTs

 Number of ports :                      22
 Number of nets :                      343
 Number of instances :                  28
 Number of references to this view :     0

Total accumulated area :
 Number of DSP block 9-bit elems :       2
 Number of LCs :                         0
 Number of LUTs :                     2673
 Number of Memory Bits :              2264
 Number of Registers :                 879
 Number of accumulated instances :    3628
```

C.2 Timing Report

```
-- CTE report summary..
                 CTE Report Summary

                  Clock Frequency Report

Domain                  Clock Name                          Min Period (Freq)
------                  ----------                          -----------------
ClockDomain1            Decoder_clk_i                       12.886 (77.604 MHz)
End CTE Report Summary ..... CPU Time Used: 4 sec.
-- CTE report summary..
                 CTE Report Summary

Analyzing setup constraint violations 10

Setup Slack Path Summary

               Data
       Setup   Path
Index  Slack   Delay   Source Clock    Dest. Clock        Data Start Pin              Data End Pin
-----  ------  ------  -------------  -------------  ------------------------  ---------
----------------  ----
  1    -4.886  12.726  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(0)/clk     inst_ee/reg_l_v(7)/datain
  2    -4.428  12.268  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(1)/clk     inst_ee/reg_l_v(7)/datain
  3    -4.392  12.232  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(2)/clk     inst_ee/reg_l_v(7)/datain
  4    -4.356  12.196  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(3)/clk     inst_ee/reg_l_v(7)/datain
  5    -4.320  12.160  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(4)/clk     inst_ee/reg_l_v(7)/datain
  6    -4.284  12.124  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(5)/clk     inst_ee/reg_l_v(7)/datain
  7    -4.230  12.070  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(6)/clk     inst_ee/reg_l_v(7)/datain
  8    -4.176  12.016  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_i_v(7)/clk     inst_ee/reg_l_v(7)/datain
  9    -2.841  10.681  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_C_r(0)(4)/clk  inst_ee/reg_l_v(7)/datain
 10    -2.817  10.657  Decoder_clk_i  Decoder_clk_i  inst_ee/reg_C_r(0)(3)/clk  inst_ee/reg_l_v(7)/datain

End CTE Report Summary ..... CPU Time Used: 0 sec.
-- CTE report timing..
                 CTE Critical Path Report
-- CTE get true worst setup path..
```



```
Critical path #1, (path slack = -4.89):

SOURCE CLOCK: name: Decoder_clk_i period: 8.000000
     Times are relative to the 1st rising edge
  DEST CLOCK: name: Decoder_clk_i period: 8.000000
     Times are relative to the 2nd rising edge

NAME                                  GATE                  DELAY    ARRIVAL DIR  FANOUT
inst_ee/reg_i_v(0)/clk              stratixii_lcell_ff                    0.00   up
inst_ee/reg_i_v(0)/regout           stratixii_lcell_ff        0.61        0.61   up
inst_ee/i_v(0)                      (net)                     0.00                    2
inst_ee/i_v_inc_201/ix23/datac      stratixii_lcell_comb                  0.61   up
inst_ee/i_v_inc_201/ix23/sumout     stratixii_lcell_comb      1.68        2.29   up
inst_ee/i_v_inc_201/nx28            (net)                     0.00                   20
inst_ee/modgen_dec_203/ix23/datac   stratixii_lcell_comb                  2.29   up
inst_ee/modgen_dec_203/ix23/cout    stratixii_lcell_comb      0.44        2.72   up
inst_ee/modgen_dec_203/nx29         (net)                     0.00                    1
inst_ee/modgen_dec_203/ix34/cin     stratixii_lcell_comb                  2.72   up
inst_ee/modgen_dec_203/ix34/cout    stratixii_lcell_comb      0.04        2.76   up
inst_ee/modgen_dec_203/nx40         (net)                     0.00                    1
inst_ee/modgen_dec_203/ix45/cin     stratixii_lcell_comb                  2.76   up
inst_ee/modgen_dec_203/ix45/cout    stratixii_lcell_comb      0.04        2.80   up
inst_ee/modgen_dec_203/nx51         (net)                     0.00                    1
inst_ee/modgen_dec_203/ix56/cin     stratixii_lcell_comb                  2.80   up
inst_ee/modgen_dec_203/ix56/cout    stratixii_lcell_comb      0.04        2.83   up
inst_ee/modgen_dec_203/nx62         (net)                     0.00                    1
inst_ee/modgen_dec_203/ix67/cin     stratixii_lcell_comb                  2.83   up
inst_ee/modgen_dec_203/ix67/cout    stratixii_lcell_comb      0.04        2.87   up
inst_ee/modgen_dec_203/nx73         (net)                     0.00                    1
inst_ee/modgen_dec_203/ix78/cin     stratixii_lcell_comb                  2.87   up
inst_ee/modgen_dec_203/ix78/cout    stratixii_lcell_comb      0.04        2.90   up
inst_ee/modgen_dec_203/nx84         (net)                     0.00                    1
inst_ee/modgen_dec_203/ix89/cin     stratixii_lcell_comb                  2.90   up
inst_ee/modgen_dec_203/ix89/cout    stratixii_lcell_comb      0.04        2.94   up
inst_ee/modgen_dec_203/nx95         (net)                     0.00                    1
inst_ee/modgen_dec_203/ix100/cin    stratixii_lcell_comb                  2.94   up
inst_ee/modgen_dec_203/ix100/sumout stratixii_lcell_comb      0.73        3.67   up
inst_ee/modgen_dec_203/nx105        (net)                     0.00                    2
inst_ee/modgen_gt_204/ix101/datad   stratixii_lcell_comb                  3.67   up
inst_ee/modgen_gt_204/ix101/cout    stratixii_lcell_comb      0.35        4.02   up
inst_ee/modgen_gt_204/nx107         (net)                     0.00                    1
inst_ee/modgen_gt_204/ix112/cin     stratixii_lcell_comb                  4.02   up
inst_ee/modgen_gt_204/ix112/cout    stratixii_lcell_comb      0.04        4.06   up
inst_ee/modgen_gt_204/nx118         (net)                     0.00                    1
inst_ee/i123bx2/datad               stratixii_lcell_comb                  4.06   up
inst_ee/i123bx2/combout             stratixii_lcell_comb      1.39        5.45   up
inst_ee/n123bx2                     (net)                     0.00                   18
inst_ee/l_v_mult_206/ix34/datab(0)  lpm_mult_1                            5.45   up
inst_ee/l_v_mult_206/ix34/result(0) lpm_mult_1                4.97       10.42   up
inst_ee/l_v_mult_206/nx33           (net)                     0.00                    1
inst_ee/l_v_add_209/ix31/datac      stratixii_lcell_comb                 10.42   up
inst_ee/l_v_add_209/ix31/cout       stratixii_lcell_comb      0.44       10.85   up
inst_ee/l_v_add_209/nx37            (net)                     0.00                    1
inst_ee/l_v_add_209/ix42/cin        stratixii_lcell_comb                 10.85   up
inst_ee/l_v_add_209/ix42/cout       stratixii_lcell_comb      0.04       10.89   up
inst_ee/l_v_add_209/nx48            (net)                     0.00                    1
inst_ee/l_v_add_209/ix53/cin        stratixii_lcell_comb                 10.89   up
inst_ee/l_v_add_209/ix53/cout       stratixii_lcell_comb      0.04       10.92   up
inst_ee/l_v_add_209/nx59            (net)                     0.00                    1
inst_ee/l_v_add_209/ix64/cin        stratixii_lcell_comb                 10.92   up
inst_ee/l_v_add_209/ix64/cout       stratixii_lcell_comb      0.04       10.96   up
inst_ee/l_v_add_209/nx70            (net)                     0.00                    1
inst_ee/l_v_add_209/ix75/cin        stratixii_lcell_comb                 10.96   up
inst_ee/l_v_add_209/ix75/cout       stratixii_lcell_comb      0.04       11.00   up
inst_ee/l_v_add_209/nx81            (net)                     0.00                    1
inst_ee/l_v_add_209/ix86/cin        stratixii_lcell_comb                 11.00   up
```

```
inst_ee/l_v_add_209/ix86/cout      stratixii_lcell_comb       0.04     11.03   up
inst_ee/l_v_add_209/nx92           (net)                      0.00                 1
inst_ee/l_v_add_209/ix97/cin       stratixii_lcell_comb                11.03   up
inst_ee/l_v_add_209/ix97/cout      stratixii_lcell_comb       0.04     11.07   up
inst_ee/l_v_add_209/nx103          (net)                      0.00                 1
inst_ee/l_v_add_209/ix108/cin      stratixii_lcell_comb                11.07   up
inst_ee/l_v_add_209/ix108/sumout   stratixii_lcell_comb       0.70     11.77   up
inst_ee/l_v_add_209/nx113          (net)                      0.00                 1
inst_ee/i9f82x1/dataa              stratixii_lcell_comb                11.77   up
inst_ee/i9f82x1/combout            stratixii_lcell_comb       0.96     12.73   up
inst_ee/n9f82x1                    (net)                      0.00                 1
inst_ee/reg_l_v(7)/datain          stratixii_lcell_ff                  12.73   up

Initial edge separation:       8.00
Source clock delay:      -     1.79
Dest clock delay:        +     1.79
                         -----------
Edge separation:               8.00
Setup constraint:        -     0.16
                         -----------
Data required time:            7.84
Data arrival time:       -    12.73
                         -----------
Slack (VIOLATED):             -4.89

-- CPU Time Used: 0 sec.
End CTE Analysis ..... CPU Time Used: 0 sec.
```


Author(s) Akash Kumar

Title High Throughput Reed Solomon Decoder for Ultra Wide Band

Distribution

Nat.Lab./PI	WB-5	
PRL	Redhill, UK	
PRB	Briarcliff Manor, USA	
LEP	Limeil–Brévannes, France	
PFL	Aachen, BRD	
CIP	WAH	
Director:	G. Beenker	WAY-5-057
Department Head:	A. van der Werf	WDC 3-034

Abstract

Members group van der Werf		
J.P.J. Heemskerk	IP&S	SFF-8
J.L. Bakx	POS	SFJ-1

Full report

G.F.M. Beenker	Nat.Lab.	WAY 5-055
A.J.W.M. ten Berg	Nat.Lab.	WAY 4-069
C.H. van Berkel	Nat.Lab.	WDC 3-020
N.C. Bird	Nat.Lab.	WDC 5-101
E.J. Dijkstra	Nat.Lab.	WDC p-033
J.T.J. van Eindhoven	Nat.Lab.	WDC p-045
M.J.M. Heijligers	Nat.Lab.	WAY 4-065
D.J. de Jong	Nat.Lab.	WAH 1-002
J.-P. Linnartz	Nat.Lab.	WY 6-023
L. Nederlof	Nat.Lab.	WAY 5-059
A. van der Werf	Nat.Lab.	WDC 3-034
P. van der Wolf	Nat.Lab.	WDC 3-028
D. Birru	Research Briarcliff, USA	
N. Tan	Research Briarcliff, USA	
T. Helbig	Research Red Hill, UK	
M. Weiss	PS BL Connectivity Dresden, Germany	
W. Drescher	PS BL Connectivity Dresden, Germany	
T.Y. Pu	PS BL Connectivity APIC, Singapore	
A. Deoliveira	PS Eindhoven, The Netherlands	

M. van Lier PS Eindhoven, The Netherlands
P. van Otterloo PS Eindhoven, The Netherlands
J.-B. Theeten PS Eindhoven, The Netherlands
R. von Vignau PS Eindhoven, The Netherlands

