

# High Throughput and Low Power Reed Solomon Decoder for Ultra Wide Band

A. Kumar; S. Sawitzki  
akakumar@natlab.research.philips.com

## Abstract

Reed Solomon (RS) codes have been widely used in a variety of communication systems. Continual demand for ever higher data rates makes it necessary to devise very high-speed implementations of RS decoders. In this paper, a uniform comparison was drawn for various algorithms and architectures proposed in the literature, which helped in selecting the appropriate architecture for the intended application. Dual-line architecture of modified Berlekamp Massey algorithm was chosen for the final design. Using *PcCMOS12corelib* the area of the design is  $0.22mm^2$  and a throughput of 1.6 Gbps. The design dissipates only 17mW of power in the worst case, including memory, when operating at 1.0 Gbps data rate.

## 1 Motivation

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 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 are available and this often makes it difficult to determine the best choice due to the number of variables and trade-offs available. Therefore, before making a good choice for the application a thorough research is needed into the decoders available.

For 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 the 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 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 add redundancy at the end of the messages systematically so as to enable the retrieval of messages correctly despite errors in the received sequences. This eliminates the need of retransmission of messages over a noisy channel. RS codes are a subset of Bose-Chaudhuri-Hocquenghem (*BCH*) codes and are linear block codes. [1] is one of the best references for RS Codes.

An  $RS(n, k)$  code implies that the encoder takes in  $k$  symbols and adds  $n - k$  parity symbols to make it a  $n$  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 1 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.

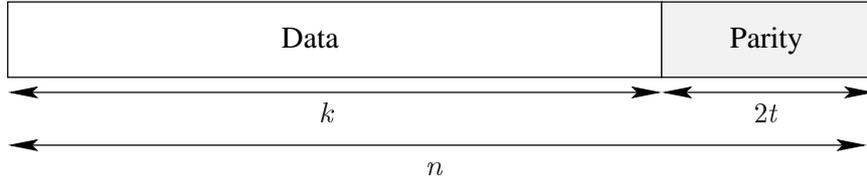


Figure 1: A typical RS code word

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.

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

$$r(x) = c(x) + e(x) \quad (1)$$

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 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 fewer than or equal to  $t$  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 as *uncorrectable*.

The basic decoder structure is shown in Figure 2. A detailed explanation on Reed Solomon decoders can be found in [1] and [2]. Decoder essentially consists of four modules. The first module computes the syndrome polynomial from the received sequence. This is used to solve a key equation in the second block, which generates two polynomials for determining the location and value of these errors in the received code word. The next block of Chien search uses the Error Locator Polynomial obtained from the second block to compute the error location, while the fourth block employs Forney algorithm to determine the value of error occurred. The correction block merely adds the values obtained from the output of the Forney block and the FIFO block. Please note that in Galois arithmetic, addition and subtraction are equivalent.

### 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.

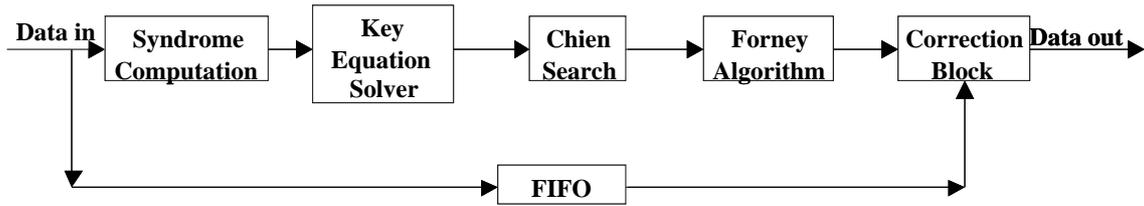


Figure 2: Decoder flow

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 states, the probability of error is different. In [3], Ahlin presented a way to match the parameters of the GE model to the land mobile channel, an approach that was generalized in [4] to a Markov model with more than two states.

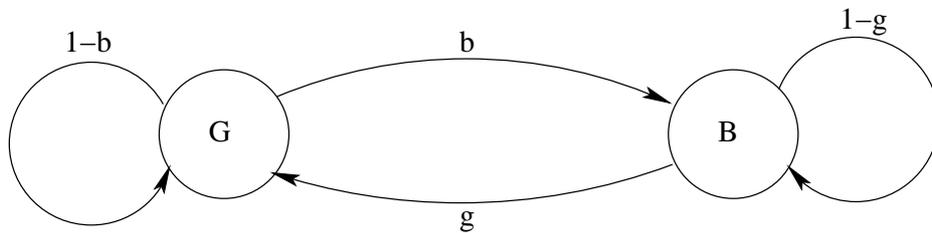


Figure 3: The Gilbert-Elliott Channel Model.

Figure 3 shows the GE Channel Model. Two states are shown represented by  $G$  and  $B$  indicating 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. A detailed analysis can be found in [5] and [6].

### 3.1 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. The threshold here signifies the SNR level at which the channel changes states. The first set was with the threshold set to 5dB lower than the average SNR and the other with 10dB less than the average. Due to the very high data bit rate involved the transition probability is very small. Therefore, channel transition 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

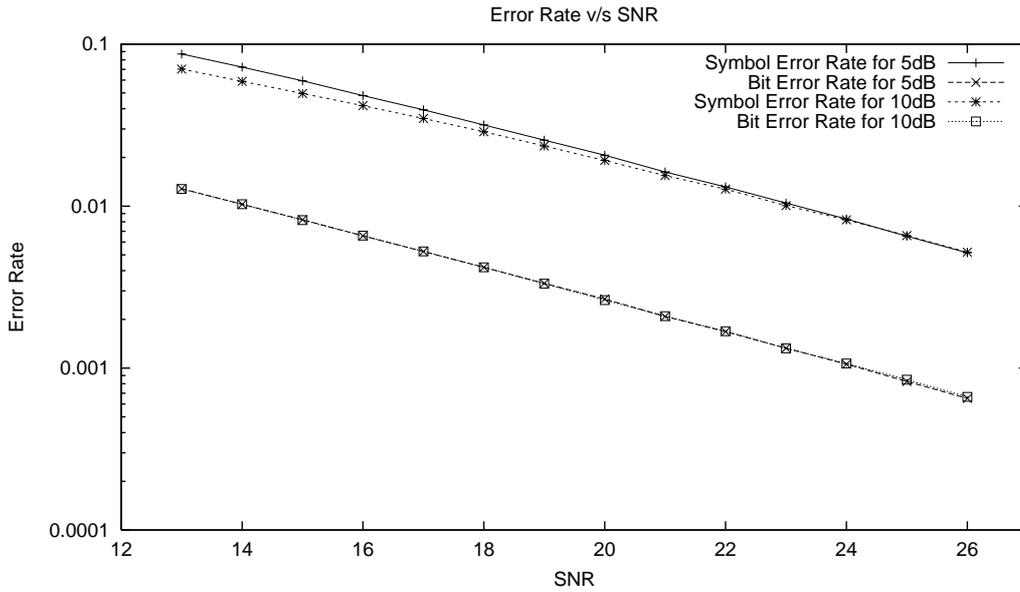


Figure 4: The symbol error rate and bit error rates for different thresholds.

symbol error rate are computed and plotted. The simulation was run for 10,000 codewords to get a good estimate for each state. Mathematica software was used to solve the complex mathematical equations and obtain the channel model parameters for the physical quantities under consideration.

### 3.2 Simulation Results

As can be seen from the Figure 4, the error probabilities decrease with increase in SNR as expected. The figure shows the symbol and the bit error probabilities observed. As expected the error rates follow a linear relationship with the increasing SNR on the logarithmic scale. We notice that around 20dB average SNR for both the thresholds, the symbol error rate is about 0.02, which corresponds to an average of 5 symbol errors in a code word of 255 symbols. From the results, an error correction capability of 8 is seen as a good choice, as when the SNR is above 20dB, the likelihood of more than 8 errors in a codeword of 255 is very low.

## 4 Architecture Design Options

Having decided on the codeword, investigation was carried out to determine appropriate algorithm and architecture. Figure 5 shows the various architectures available. Table 1 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 any architecture. It should be noted that this is only the estimate of computational elements and, therefore, additional hardware will be needed for control overhead. Total latency of the various blocks will determine the size of FIFO.

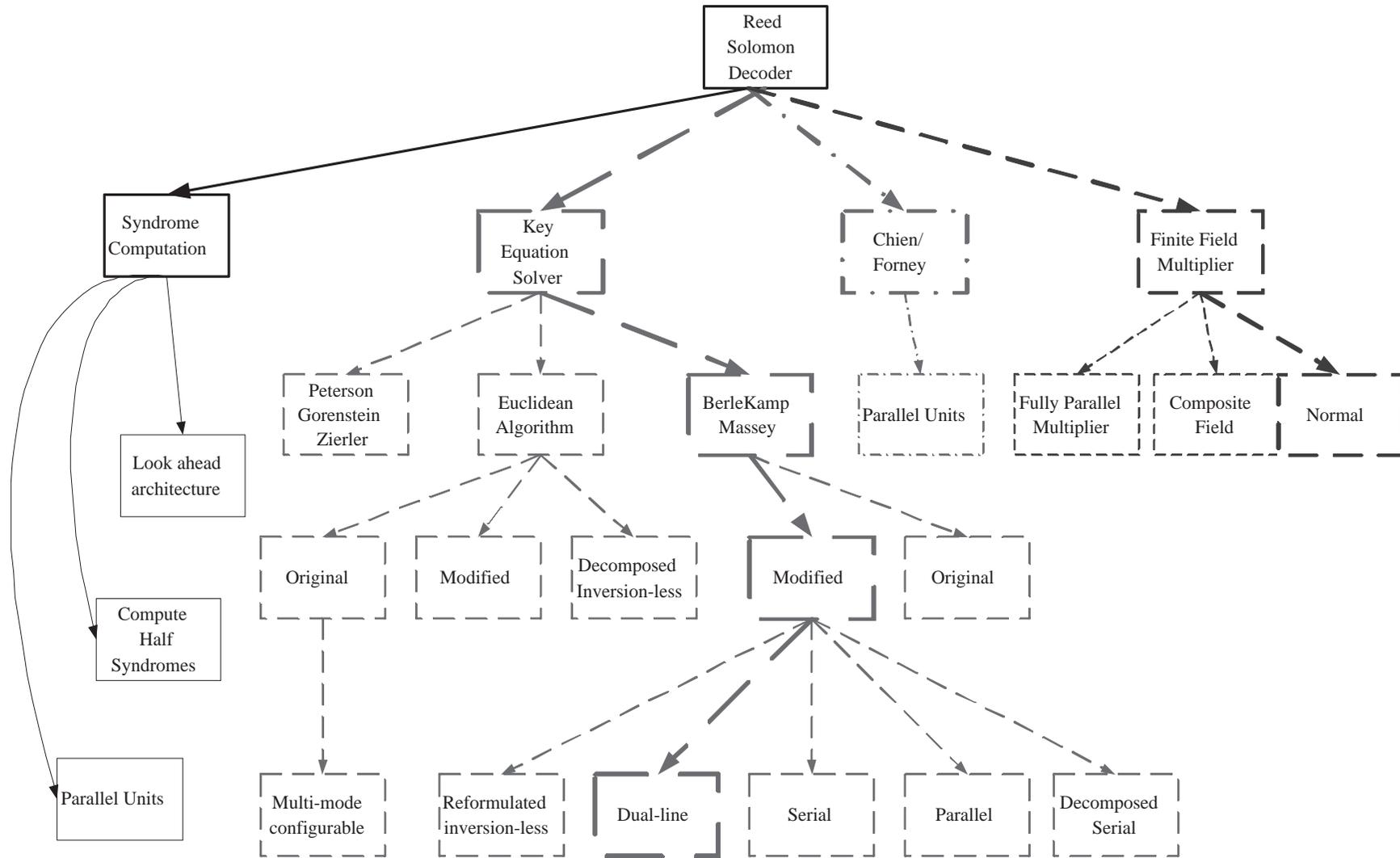


Figure 5: Design Space Exploration

Architecture	Blocks	Adders	Multipliers	Muxes	Latches	Latency	Critical Path Delay
Syndrome Computation [7]	2t	1	1	1	2		
Total		2t	2t	2t	4t	n	Mul + Add
Look ahead architecture (x units)	2t	x	x	1	2		
Total		2xt	2xt	2t	4t	n/x	Mul + Add
<b>Original Euclidean [8]</b>							
Divider Block	2t	1	1	3	2		
Multiply Block	t	2	1	3	3		
Total (Estimates)		4t	3t	9t	7t		
Actual [10]		4t + 1	3t + 1	11t + 4	14t + 6	4t - 3	ROM + 2×Mul + Add + 2×Mux
<b>Modified Euclidean [8]</b>							
Degree Computation Block	2t	2	0	7	7		
Polynomial Arithmetic Block	2t	2	4	8	19		
Total (Estimates)		8t	8t	30t	52t		
Actual [10]		8t	8t	40t + 2	78t + 4	10t + 8	Mul + Add + Mux
Decomposed inversion-less [11]		1	3	1	3t + 1	2t×(t+1)	Mul + Add + Mux
<b>Modified BerleKamp Massey</b>							
Serial		1	3	4	3t + 2	2t×(2t+2)	Mul + Add + Mux
Decomposed inversion-less [12]		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 [13]		2t	4t + 1	2t	4t + 1	3t + 1	Mul + Add
Reformulated inversion-less [14]		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 1: Summary of hardware utilization of various architectures

## 4.1 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 corresponding 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.

Table 1 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 kept low. Also, the overall latency and gate count of computational elements should be low since that would determine the total silicon area of the design.

### 4.1.1 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. 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. 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.

### 4.1.2 RS Code

As we can see from Table 1, 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. The value of  $t$  is often chosen to be a power of 2 in order to maximise the hardware utilised in design. Taking into account the results of Channel Model Simulation  $RS(255, 239)$  is chosen, since it has an error correction capability of 8.

## 4.2 Highlights

Table 2 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.

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 2: Summary of hardware utilization for Dual-line architecture

## 5 Design Flow

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.

Once the algorithm was fully developed and tested in C, VHDL-code was developed. The VHDL code was structured such so it could be easily synthesized. 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. Ambit from Cadence was used to analyse the hardware usage and frequency of operation after various optimisation settings.

The design flow needed for verification of synthesized design and power estimation has been explained in Figure 6. As shown in the figure the core VHDL modules were optimised and synthesized using *ambit*. The synthesized model was written out into a verilog netlist using *ambit* itself. Once the netlist was obtained, it was compiled using *ncvlog* into the work library together with the technology library. The library used was for the same technology as the one used for synthesis. As can be seen, the wrapper modules were 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, was extracted into a VHDL file and then compiled into the work library. This was done using *ncshell* and *ncvhdl* respectively. This being done, the wrapper modules were compiled into the work library.

From this point onwards, two approaches were used. *Ncelab* and *ncsim* were used purely for simulating the synthesized design, and *dncelab* and *dnctsim* were used to obtain power estimate, which were essentially the same tools, but included the *DIESEL* routines for estimating the power dissipated in the design. *Diesel* is an internal tool developed within Philips which estimates the power for the simulated design, and hence the accuracy of the results depends on the input provided.

## 6 Results

This section covers the results of various synthesis experiments conducted. Resource utilization, timing analysis and the power consumption were used as benchmarking parameters.

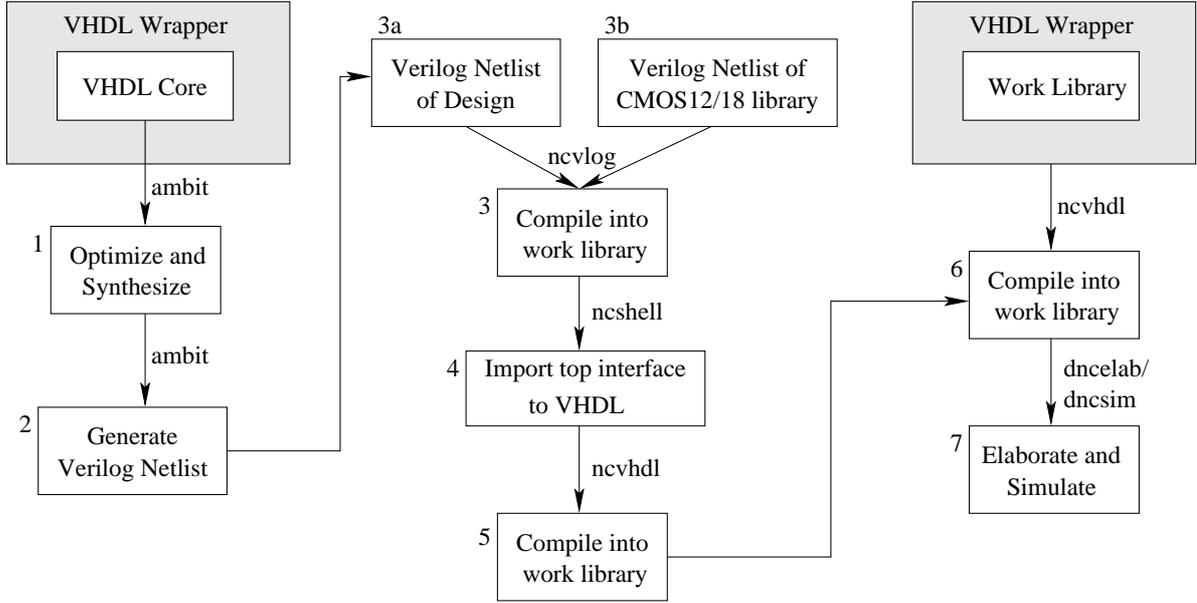


Figure 6: Design flow for design verification and estimation of power

## 6.1 Area Analysis

Ambit was run with the libraries *PcCMOS12corelib* and *PcCMOS18corelib*. The silicon area required was analysed for various timing constraints. A comparison for area of the decoder is shown in Table 3. This table shows the area requirement when the constraint was set to 5 ns, which can support 200 MHz frequency, i.e. 1.6 Gbps. The total number of design cells used, including the memory, were 12,768 and 12,613 for *PcCMOS18corelib* and *PcCMOS12corelib* respectively.

Module	Module Area( $\mu\text{m}^2$ )	
	CMOS12	CMOS18
Chien	7663.343	15675.392
FIFO	83183.278	148684.807
Forney	21608.247	52936.705
Gen_elp_eep	89602.009	186404.866
Gen_syndromes	17828.014	34754.560
top_view	219913.131	438472.713

Table 3: Resource utilization for the decoder in CMOS12

## 6.2 Power Analysis

The power estimates provided in this section are for design operation at 125 MHz, which translates to data rate of 1Gbps.

### 6.2.1 Variation With Number of Errors

Figure 7 shows the variation of power with the number of errors found in the codeword for *PcC-MOS12corelib*. As can be seen from the graph obtained, the power dissipated for the FIFO and syndrome computation block is independent of the number of errors as expected. For the block that computes the ELP (Error Locator Polynomial) and EEP (Error Evaluator Polynomial), it is clearly seen that the power dissipated increases linearly with the number of errors. The Chien search block also shows a linear increase in the power dissipated.

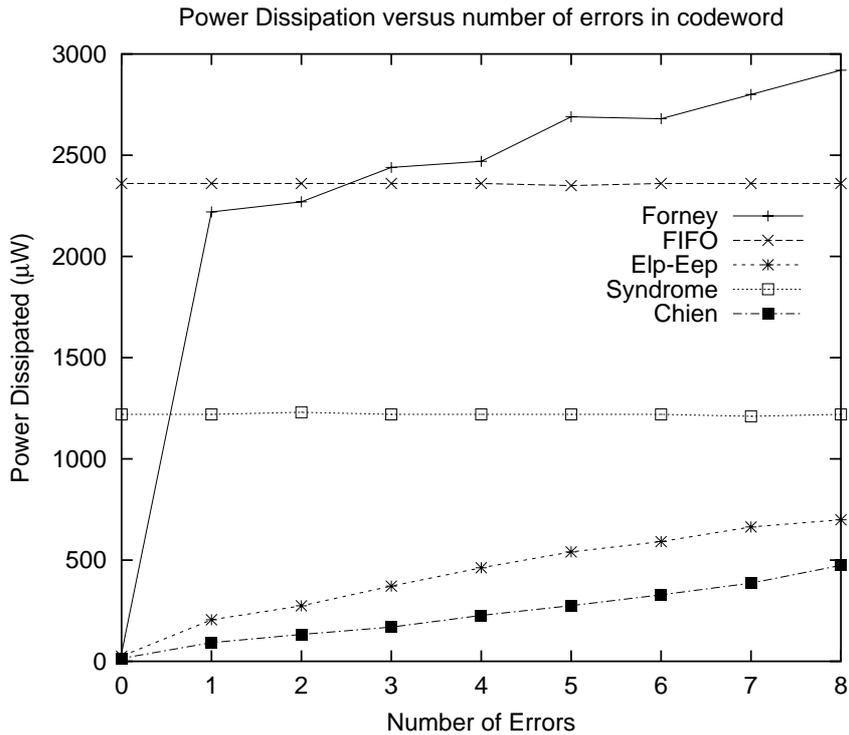


Figure 7: Graph showing variation of power dissipated with number of errors for different modules.

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.

### 6.2.2 Distribution of Power in Different Modules

Figure 8 shows a distribution of power when there are maximum number of errors correctable in the received code word, while Figure 9 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.

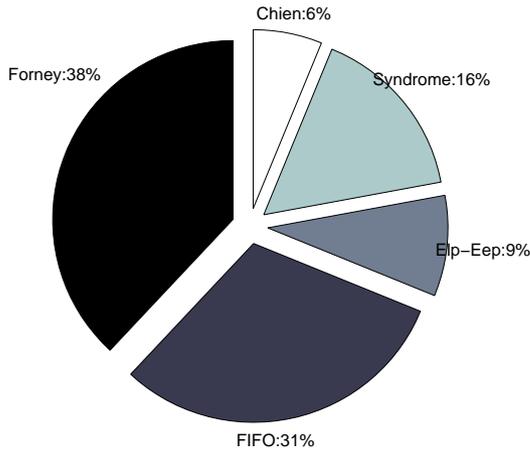


Figure 8: Power consumed by various blocks when 8 errors are found

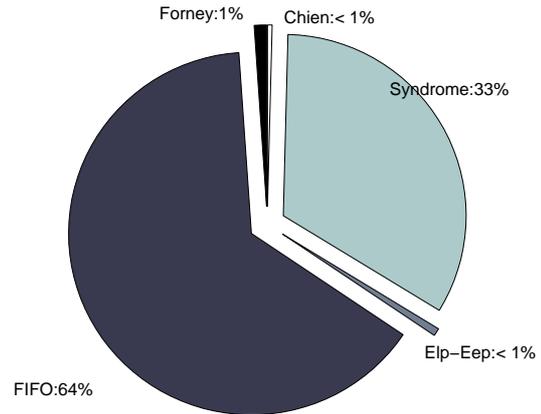


Figure 9: Power consumed by various blocks when no errors are found

## 7 Benchmarking

Please note that for all the designs  $RS(255, 239)$  code has been used for benchmarking. The design using modified Euclidean Algorithm is very hardware intensive. The design proposed in [7] uses roughly 115K gates for  $0.13\mu m$  CMOS technology operating at 6 Gbps excluding memory. The proposed design only uses 12K cells including memory in both  $0.12\mu m$  and  $0.18\mu m$  technology. The results are better even when they are normalised for throughput and technology. The latency of the design is only 284 cycles when compared to 355 cycles in [7].

In terms of power, a design was proposed by Chang in [9] for low power. In that design, 62mW of power is used in the best case, including memory, using  $0.25\mu m$  CMOS technology, and 100mW are consumed in the worst case. In our design, only 17mW of power is used in the worst case using  $0.12\mu m$  technology. The area of the chip proposed in [9] using  $0.25\mu m$  CMOS technology is  $5mm^2$ , while the area of the proposed design is  $0.22mm^2$  with  $0.12\mu m$  technology.

## 8 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. Modified Berlekamp Massey algorithm was chosen for the VHDL implementation. 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 17mW including the memory block when operating at 1Gbps data rate.

## References

- [1] S. B. Wicker and V. K. Bhargava; *Reed Solomon Codes and Their Applications*, Piscataway, NJ: IEEE Press, 1994.
- [2] Richard E. Blahut; *Theory and Practice of Error Control Codes*, Addison-Wesley, 1983.
- [3] L. Ahlin; *Coding methods for the mobile radio channel* Nordic Seminar on Digital Land Mobile Communications, 1985.
- [4] H. S. Wang and N. Moayeri; *Finite-state Markov channel - A useful model for radio communication channels* IEEE Transaction on Vehicular Technology, 1995.
- [5] L. Wilhemsson and L. B. Milstein; *On the effect of imperfect interleaving for the Gilbert-Elliott channel* IEEE Transactions on Communications, 1999.
- [6] G. Sharma, A. Dholakia, and A. Hassan; *Simulation of error trapping decoders on a fading channel* IEEE Transaction on Vehicular Technology, 1996.
- [7] Hanho Lee; *High-speed VLSI architecture for parallel Reed-Solomon decoder* IEEE transactions on VLSI Systems 2003.
- [8] H. Lee, M. L. Yu, and L. Song; *VLSI design of Reed-Solomon decoder architectures* IEEE International Symposium on Circuits and Systems 2000.
- [9] 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
- [10] H. Lee; *An area-efficient Euclidean algorithm block for Reed-Solomon decoder* IEEE Computer Society Annual Symposium on VLSI, 2003
- [11] 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.
- [12] H. C. Chang and C. B. Shung; *A (208,192;8) Reed-Solomon decoder for DVD application* IEEE International Conference on Communications, 1998.
- [13] 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
- [14] D. V. Sarwate and N. R. Shanbhag; *High-speed architectures for Reed-Solomon decoders*, IEEE transactions on VLSI Systems 2001.