Low Power/High Speed design of a Reed Solomon Decoder

Arun Raghupathy and K. J Ray Liu
Electrical Engineering Department and Institute for Systems Research
University of Maryland, College Park

ABSTRACT OF THE PAPER

With the spread of Reed Solomon codes to portable applications, low power RS decoder design has become important. This paper discusses how the Berlekamp Massey Decoding algorithm can be modified in order to obtain a low power architecture. In addition, modifications that speed-up the syndrome and error computations are suggested. Then the VLSI design of a low power/high speed decoder is described. The power reduction when compared to the normal design is estimated.

I. INTRODUCTION

Reed Solomon (RS) codes, in recent years, have found widespread use in applications range from consumer products (such as audio CD players) to space communications. RS codes are good candidates for use in portable wireless receivers as a part of concatenated coding systems with convolutional codes. In portable systems low power operation is extremely desirable because of the need to extend battery lifetimes. In this paper, we propose a modified Berlekamp Massey (BM) algorithm that leads immediately to an efficient high speed/low power VLSI architecture for RS decoding.

Various approaches have been used for decoding RS codes ([2],[3],Chap.5 in [4]). The approach we have used first involves the computation of the syndrome. This is followed by the application of the BM algorithm to compute the error locator and error evaluator polynomials. Finally, a Chien search is performed in conjunction with Forney's method to find the error locations and error values.

Algorithm/architecture modifications can be used to favorably impact the power consumption of a VLSI design. Using parallelism enables the circuit to operate at lower speeds while maintaining the overall throughput constant [5],[6]. Basically, area is tradedoff for low power operation. We apply an interesting transformation to create additional parallelism in the BM algorithm while reducing the number of iterations from $2t$ to t, where t is the error correcting capability of the RS code. In addition, parallelism is exploited in the syndrome and error computations.

In section II, we propose modifications to the BM decoding algorithm for an (n,k) RS code that can correct t errors, where $n-k=2t$. In the section III, we discuss the VLSI architecture of a low power decoder and estimate the power reduction when compared to a normal design.

*This work is supported in part by the NSF NYI Award MIP9457397 and the ONR grant N00014-93-10566.

II. REED SOLOMON DECODING

Let the syndromes of an $(n, n-2t)$ RS code,be denoted by $v_j, j=0,1,\cdots,2t-1$. We will try to follow as closely as possible the notation used in [1]. The syndrome computation is just an FFT computed on a Galois field.

$$S_j = \sum_{i=0}^{n-1} \alpha^{i(b+j)} v_i \quad j=0,1,\cdots,(2t-1) \tag{1}$$

Each syndrome can be computed by performing a series of multiply-adds. The computation can be reorganized as

$$S_j = \sum_{l=0}^{\lfloor (n+1)/2 \rfloor} \alpha^{2l(b+j)} v_{2l} + \alpha^{b+j} \sum_{l=0}^{\lceil (n-1)/2 \rceil} \alpha^{2l(b+j)} v_{2l-1} \tag{2}$$

so that the number of iterations required to compute the syndromes gets halved. Therefore, the input data rate can be doubled.

The error computation can be written as

$$e_i = \begin{cases} 0 & \text{if } \Lambda(\alpha^{-i}) \neq 0 \\ -\frac{\Gamma(\alpha^{-i})}{\alpha^{(b-1)i}\Lambda'(\alpha^{-i})} & \text{if } \Lambda(\alpha^{-i}) = 0 \end{cases} \tag{3}$$

where $\Lambda(x)$ is the error locator and $\Gamma(x)$ is the error evaluator polynomial [1]. In this case, two e_i's can be computed in parallel. Again this leads to a 2 times speed up in computation.

The modifications required in the BM Algorithm are more involved. The BM Algorithm can be written as follows:

Berlekamp Massey Algorithm

1. **for** $r = 1$ **to** $2t$
2. $\quad \Delta_r = \sum_{j=0}^{L_{r-1}} \Lambda_j^{(r-1)} v_{r-1-j}$
3. $\quad$ **if** $\Delta_r \neq 0$ **then** $b1 = 0$ **else** $b1 = 1$
4. $\quad$ **if** $2L_{r-1} <= (r-1)$ **then** $b2 = 0$ **else** $b2 = 1$
5. $\quad \begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} = M_r(x) \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix}$ (refer Table 1)
6. $\quad L_r = f_r(L_{r-1})$ (refer Table 1)
7. **end for**

Condition	$M_r(x)$	$f_r(L_{r-1})$
$\overline{b1}\ \overline{b2}$	$\begin{bmatrix} 1 & \Delta_r x \\ \Delta_r^{-1} & 0 \end{bmatrix}$	$r - L_{r-1}$
$b1 + b2$	$\begin{bmatrix} 1 & \Delta_r x \\ 0 & x \end{bmatrix}$	L_{r-1}

Table 1: Updating $\Lambda(x)$,$B(x)$ and L for the BM algorithm

A number of observations can be made. The degree of $\Lambda(x)$ increases in iteration r only when $\Delta_r \neq 0$ and $2L \leq r-1$. This indirectly implies that while $\Lambda(x)$ can

0-7803-3583-X/97 $10.00 ©1997 IEEE

change every iteration, its degree can increase only once every two iterations.

In addition, we note that $B(x)$ and L need to be updated only when the degree of $\Lambda(x)$ increases. Otherwise $B(x)$ is just shifted and L remains the same. This seems to suggest that if we looked at updating the pair of polynomials $(\Lambda^{(2k-2)}(x), B^{(2k-2)}(x))$ to $(\Lambda^{(2k)}(x), B^{(2k)}(x))$ without going through $(\Lambda^{(2k-1)}(x), B^{(2k-1)}(x))$ we can halve the number of iterations. Ofcourse, this modified iteration will be more complicated than a single iteration of the original algorithm. In general, the modified iteration takes time $T^{'} > T$. In order to get an improved algorithm in terms of speed/power, it is enough if $T^{'} < 2T$. In particular, the modified algorithm must expose additional parallelism so that the above holds. Ideally, we would like to have $T^{'}$ as close to T as possible to get a doubling in speed.

Let Δ_{2k-1} be the discrepancy in the k th iteration of the modified algorithm. We want to design the new algorithm, so that this matches the discrepancy Δ of the original algorithm at the odd iteration. Let δ_{2k} be the predicted discrepancy at the even step of the original algorithm.

Let us look at 2 consecutive iterations (one odd and one even) in the original algorithm. There are 5 possibilities for updating $\Lambda(x)$. In the first case , $\Lambda(x)$ is unchanged. The next 2 cases correspond to $\Lambda(x)$ getting updated in an odd iteration with or without an increase in L, respectively. Cases 4 and 5 correspond to $\Lambda(x)$ getting updated in an even iteration with or without an increase in L, respectively. As far as $B(x)$ is concerned there are 4 possibilities. In the first case, $B(x)$ is just shifted in both iterations. The next two cases, correspond to an update in one of the iterations and a shift in the other. The final case in which updates occur in both iterations is impossible.

We note that the 2 tests on L_{2k-2} and L_{2k-1} (i.e $2L_{2k-2} \leq 2k-2$ and $2L_{2k-1} \leq 2k-1$) are equivalent for our purposes. To show why, we consider two cases. If the degree of $\Lambda(x)$ does not change in the odd iteration, then $L_{2k-2} = L_{2k-1}$. In this case, both the tests are equivalent. This is because $2L_{2k-2} \leq 2k-2 \Rightarrow 2L_{2k-1} = 2L_{2k-2} \leq 2k-1$ and $2L_{2k-1} = 2L_{2k-2} \leq 2k-1 \Rightarrow 2L_{2k-2} \leq 2k-2$ (since $2L_{2k-2}$ cannot equal $2k-1$). The other case occurs when the degree of $\Lambda(x)$ increases in the odd iteration then we are uninterested in testing whether $2L_{2k-1} \leq 2k-1$ since the degree of $\Lambda(x)$ cannot increase in 2 consecutive iterations.

In the case when $\Delta_{2k-1} = 0$, δ_{2k} equals Δ_{2k} because $\Lambda^{(2k-1)}(x) = \Lambda^{(2k-2)}(x)$. We also have $L_{2k-1} = L_{2k-2}$ and $B^{(2k-1)}(x) = B^{(2k-2)}(x)$. If now, $\delta_{2k} = 0$ then $\Lambda^{(2k)}(x) = \Lambda^{2k-2}(x)$, $L_{2k} = L_{2k-2}$ and $B^{(2k)}(x)$ is just a twice shifted version of $B^{(2k-2)}(x)$ (See 1st row of Table 2). On the other hand, if $\delta_{2k} \neq 0$ there are 2 subcases corresponding to $2L_{2k-1} \leq 2k-1$ and $2L_{2k-1} > 2k-1$ respectively. When $2L_{2k-1} \leq 2k-1$, the degree of $\Lambda(x)$ can increase at the even iteration and $B(x)$ needs to hold the the old contents of $\Lambda(x)$ as $B^{(2k)}(x) = \delta_{2k}^{-1}\Lambda^{(2k-2)}(x)$ (See 2nd row of Table 2). When $2L_{2k-1} > 2k-1$, $\Lambda(x)$ is updated in the even iteration without degree changing and $B^{(2k)}(x)$ is updated by just shifting it as $B^{(2k)}(x) = x^2\ B^{(2k-2)}(x)$ (See 3rd row of Table 2).

The case when $\Delta_{2k-1} \neq 0$ is more difficult to handle. The first subcase occurs when $2L_{2k-2} \leq 2k-2$, so that $\Lambda(x)$ can increase in degree in the odd iteration and $B(x)$ is updated to hold the old contents of $\Lambda(x)$ as $B^{(2k-1)} = \Delta_{2k-1}^{-1}\Lambda^{(2k-2)}(x)$ (See 4th row of Table 2). In the even iteration, as mentioned before, $\Lambda(x)$ cannot increase in degree, therefore $B^{(2k)} = x\ B^{(2k-1)}$. The update for $\Lambda(x)$ can be derived as below. The derivation proceeds along the lines used in [1] to prove the BM Theorem. The odd iteration update can be written as

$$\Lambda^{(2k-1)}(x) = \Lambda^{(2k-2)}(x) + A_1 x^l\ \Lambda^{(m-1)}(x)$$

The even iteration can be written as

$$\Lambda^{(2k)}(x) = \Lambda^{(2k-2)}(x) + A_1 x^l\ \Lambda^{(m-1)}(x) + A_2 x\ \Lambda^{(2k-2)}(x)$$

Let us choose m as the last iteration at which an increase of degree in $\Lambda(x)$ occurred. If we now choose l such that $2k-1-l = m$ then we get

$$\Delta_{2k-1}^{'} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-1)} v_{2k-2-j} = \Delta_{2k-1} + A_1\Delta_m$$

$$\delta_{2k}^{'} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k)} v_{2k-1-j} = \delta_{2k} + A_2\Delta_{2k-1} + A_1\gamma_m$$

where $\gamma_m = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(m-1)} v_{2k-1-j-l}$. Using the conditions, $\Delta_{2k-1}^{'} = 0$ and $\delta_{2k}^{'} = 0$, we get $A_1 = -\Delta_m^{-1}\Delta_{2k-1}$ and $A_2 = -\Delta_{2k-1}(\delta_{2k} - \Delta_m^{-1}\gamma_m\Delta_{2k-1})$.

Note in table 2 that we update $\Delta_m^{-1}\gamma_m$ instead of both Δ_m and γ_m.This variable changes whenever L changes. If L changes at an odd iteration, $\Delta_m = \Delta_{2k-1}$ and $\gamma_m = \delta_{2k}$ so that $\Delta_m^{-1}\gamma_m$ must be updated to hold $\Delta_{2k-1}\delta_{2k}$. When we look at the case when L changes at an even iteration, we see that $\Delta_m = \delta_{2k}$. In order to find γ_m for this case another variable η_{2k} has to be introduced. Then we have $\gamma_m = \eta_{2k}$.

The second subcase occurs when $\Delta_{2k-1} \neq 0$ and $2L_{2k-2} > 2k-2$. In this case $\Lambda(x)$ cannot increase in degree during either of the 2 iterations. This means that $B^{(2k)} = x^2B^{(2k-2)}$ (see 5th row of Table 2).

The odd iteration update can be written as before. The even iteration can be written as

$$\Lambda^{(2k)}(x) = \Lambda^{(2k-2)}(x) + A_1 x^l\ \Lambda^{(m-1)}(x) + A_2 x^{l+1}\ \Lambda^{(m-1)}(x)$$

where l and m are chosen as before.

$$\begin{aligned} \Delta_{2k-1}^{'} &= \Delta_{2k-1} + A_1\Delta_m \\ \delta_{2k}^{'} &= \delta_{2k} + A_2\Delta_m + A_1\gamma_m \end{aligned}$$

Again choosing A_1 and A_2 such that $\Delta_{2k-1}^{'} = 0$ and $\delta_{2k}^{'} = 0$, we get $A_1 = -\Delta_m^{-1}\Delta_{2k-1}$ and $A_2 = -\Delta_m^{-1}(\delta_{2k} - \Delta_m^{-1}\gamma_m\Delta_{2k-1})$.

The complete algorithm is shown in Table 2. Upto this point, we have discussed the computation of $\Lambda(x)$ without referring to $\Gamma(x)$. We can compute $\Gamma(x)$ by a set of parallel iterations on $(\Gamma(x), A(x))$ with the same update matrix $M_k(x)$ (See Chap. 8 in [1]). The only difference is that the initialization is $\Gamma(x) = 0$ and $A(x) = -x^{-1}$.

Modified Berlekamp Massey Algorithm

0. Initialize $\Lambda(x) = 1$, $B(x) = 1$

Condition	$M_k(x)$	$f_k(L_{2k-2})$	$g_k(\alpha_{2k-2})$
$b1\ b0$	$\begin{bmatrix} 1 & 0 \\ 0 & x^2 \end{bmatrix}$	L_{2k-2}	α_{2k-2}
$b1\ \overline{b0}\ \overline{b2}$	$\begin{bmatrix} 1 & \delta_{2k}x^2 \\ \delta_{2k}^{-1} & 0 \end{bmatrix}$	$2k - L_{2k-2}$	$\delta_{2k}^{-1}\eta_{2k}$
$b1\ \overline{b0}\ b2$	$\begin{bmatrix} 1 & \delta_{2k}x^2 \\ 0 & x^2 \end{bmatrix}$	L_{2k-2}	α_{2k-2}
$\overline{b1}\ \overline{b2}$	$\begin{bmatrix} G(x) & \Delta_{2k-1}x \\ x\Delta_{2k-1}^{-1} & 0 \end{bmatrix}$	$2k - 1 - L_{2k-2}$	$\Delta_{2k-1}^{-1}\delta_{2k}$
$\overline{b1}\ b2$	$\begin{bmatrix} 1 & G1(x) \\ 0 & x^2 \end{bmatrix}$	L_{2k-2}	α_{2k-2}

Table 2: Updating $\Lambda(x)$,$B(x)$ and L for the BM algorithm where $G(x) = 1 + \Delta_{2k-1}^{-1}\beta_{2k-1}x$ and $G1(x) = \beta_{2k-1}x^2 + \Delta_{2k-1}x$

1. **for** $k = 1$ **to** t
2. $\quad \Delta_{2k-1} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-2)} v_{2k-2-j}$
3. $\quad \delta_{2k} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-2)} v_{2k-1-j}$
4. $\quad \eta_{2k} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-2)} v_{2k-j}$
5. $\quad \beta_{2k-1} = \delta_{2k} - \Delta_{2k-1}\alpha_{2k-2}$
6. $\quad$ **if** $\delta_{2k} \neq 0$ **then** $b0 = 0$ **else** $b0 = 1$ **fi**
7. $\quad$ **if** $\Delta_{2k-1} \neq 0$ **then** $b1 = 0$ **else** $b1 = 1$ **fi**
8. $\quad$ **if** $2L_{2k-2} <= (2k-2)$ **then** $b2 = 0$ **else** $b2 = 1$ **fi**
9. $\quad \begin{bmatrix} \Lambda^{(2k)}(x) \\ B^{(2k)}(x) \end{bmatrix} = M_k(x) \begin{bmatrix} \Lambda^{(2k-2)}(x) \\ B^{(2k-2)}(x) \end{bmatrix}$
10. $\quad L_{2k} = f_k(L_{2k-2})$
10. $\quad \alpha_{2k} = g_k(\alpha_{2k-2})$
11. **end for**

III. VLSI ARCHITECTURE AND DESIGN

In this section, we describe the VLSI architecture used for 2 decoders for the (63,57) RS code over GF(64) that can correct upto 3 errors. The first one is a design based on the normal BM Algorithm and the second is a design that incorporates the algorithm modifications that we suggested in earlier sections. Both designs were implemented as a 3 stage pipeline comprised of a syndrome stage, the BM stage and the error stage. Each stage was implemented as a state machine.

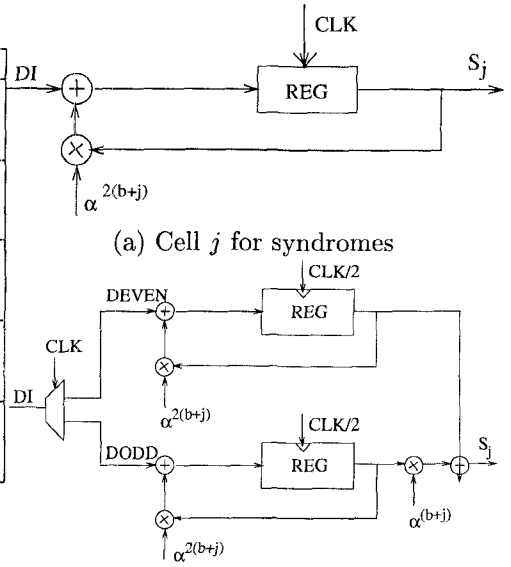
The Galois field multipliers and inverters were implemented using the "recursive" representation of the Galois field in terms of its subfield (the Galois field is the quadratic extension of the subfield)(Chap. 10 in [4]). This allows for an efficient implementation. In addition, the binary representation was chosen to minimize area while not sacrificing speed.

A. Syndrome Computation Stage

The summation in the normal case can be computed in a straightforward manner using Horner's rule as

$$S_j = ((\cdots(v_{n-1}\alpha^{b+j} + v_{n-2})\alpha^{b+j} + \cdots)\alpha^{b+j} + v_0 \qquad (4)$$

In the modified case, again Horner's rule can be applied to compute the 2 sub-summations. The 2 sub-summations can be computed in parallel. Figure 1 shows



(a) Cell j for syndromes

(b) Modified Cell j for syndromes

Figure 1: Syndrome Computation

the computation for the 2 cases.

B. BM Algorithm Stage

In the normal case, $2t$ iterations are required whereas just t iterations are needed in the modified case. This means the modified BM stage can accept syndromes at twice the rate of the normal BM stage as long as the critical path remains the same. In both cases, the inner products required to evaluate Δ, δ and η are computed serially in $t+1$ iterations (since the degree of $\Lambda(x)$ is atmost t. Each inner product is computed using just 1 multiplier and 1 adder. In addition, the polynomial updates required in the algorithm are computed serially in $t+1$ iterations (since the degrees of $\Lambda(x)$ and $B(x)$ are atmost t).

C. Error Computation

Note that we need to compute $\Lambda(\alpha^{-i})$, $\Lambda'(\alpha^{-i})$ and $\Gamma(\alpha^{-i})$ in order to compute e_i. In any field of characteristic 2,

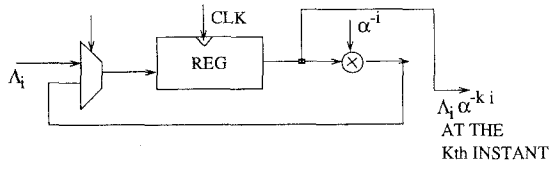
$$\Lambda'(x) = \sum_{k=0}^{t} k\Lambda_k x^{k-1} = \sum_{k:odd} \Lambda_k x^{k-1} \qquad (5)$$

therefore, $(\alpha^{-i})\Lambda'(\alpha^{-i})$ can be obtained as part of the computation for $\Lambda(\alpha^{-i})$. In addition, the circuit computes $\Gamma 1(x) = x\Gamma(x)$. We can write the error computation in these terms as

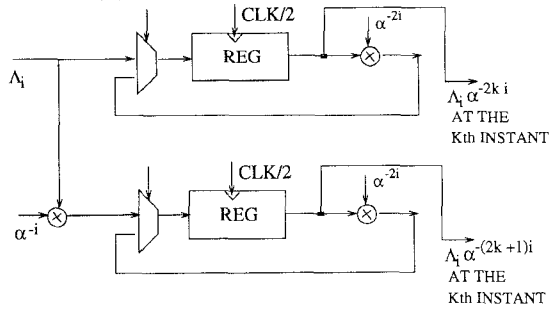
$$e_i = \begin{cases} 0 & \text{if } \Lambda(\alpha^{-i}) \neq 0 \\ -\frac{\Gamma 1(\alpha^{-i})}{\alpha^{(b-1)i}\alpha^{-i}\Lambda'(\alpha^{-i})} & \text{if } \Lambda(\alpha^{-i}) = 0 \end{cases} \qquad (6)$$

This implies that choosing $b = 1$ avoids the scaling by $\alpha^{(b-1)i}$.

The Figure 2 shows one of the modules in the normal and modified cases. In the normal case, adding the outputs of the odd cells at time instant k gives $(\alpha^{-k})\Lambda_{odd}(\alpha^{-2k})$ while the sum of the even cells gives $\Lambda_{even}(\alpha^{-2k})$. Summing the 2 results gives $\Lambda(\alpha^{-k})$. In the modified case, adding the upper outputs of all the odd cells gives $(\alpha^{-2k})\Lambda_{odd}(\alpha^{-4k})$ while the sum of the



(a) Cell i for error computation



(b) Modified Cell i for error computation

Figure 2: Error Computation

upper outputs of all even cells gives $(\Lambda_{even}(\alpha^{-4k})$. On the other hand, adding the lower outputs of all odd numbered cells gives $(\alpha^{-(2k+1)})\Lambda_{odd}(\alpha^{-(4k+2)})$ while the sum of the lower outputs of all even cells gives $(\Lambda_{even}(\alpha^{-(4k+2)})$. Since the critical path remains the same, this means that the error computation cells can produce output at twice the rate of the normal cells because of the 2 fold parallelism.

D. Power Estimate

Let us assume that the normal circuit can work at a maximum clock speed of f and the modified one at a maximum clock speed of $f^{'}$. We can operate the modified design in 2 modes - high speed and low power. In the high speed mode the modified design is operated at the same voltage as the normal design. The throughput will be $2 * f^{'}/f$ times the throughput of the normal design. So a speedup can be expected as long as $f^{'} > f/2$. In the low power mode, the throughput is maintained a constant and the voltage of the modified design is scaled down. This means that the clock rate of the low power design can be slowed down to $f/2$. If the critical paths of the normal and modified designs at the maximum voltage V are $T = 1/f$ and $T^{'} = 1/f^{'}$ respectively, then the voltage can be reduced to $V^{'}$ (at this voltage the critical path in the modified design becomes $T'' = 2T$). The ratio of capacitances can be estimated as the ratio of active areas (A) of the designs. So the ratio of the power consumption in the modified design to the normal design can be estimated as

$$\frac{A_M}{A_N}(\frac{V^{'}}{V})^2\frac{f/2}{f} \tag{7}$$

These kind of simple first order estimates can be fairly accurate based on our prior experience [6]. The behavioral decription was specified in the hardware description language(sfl) for PARTHENON [7] and then synthesized into a layout using a 2μ double metal SCMOS library.

Table 3 shows the transistor counts and the active area of the layout. We note that the transistor count is more or less doubled while the ratio A_M/A_N is 2.37. Also, note that $T^{'} = 35ns$ and $T = 30ns$. This gives us a bit rate speedup of $2T/T^{'} = 1.71$. Based on the first order model for the delay in terms of the power supply [5], we can scale the voltage for the modified design from $V = 5V$ to $V^{'} = 3.6V$ (the delay increases from $T^{'} = 35ns$ to $T'' = 60ns$). Using the above equation, this leads to a power saving of about 35%. All our speed estimates are based on IRSIM, in which each node has a capacitance and each transistor is modeled as a resistor in series with a voltage-controlled switch. At present, we are in final testing stages before we can send our chips for fabrication.

Parameter	Normal	Design
No. of Gates	7171	13945
No. of Transistors	24226	48570
Active Area	$5415\Lambda \times 5161\Lambda$	$8080\Lambda \times 8035\Lambda$
Clk. Speed(IRSIM)	f =33MHz	$f^{'}$=28.6MHz
Bit Rate(IRSIM)	198Mb/s	343Mb/s

Table 3: Comparison of Normal and Modified Designs

IV. CONCLUSION

In this paper, we first proposed an algorithm that leads to a high speed /low power RS decoder. Then we showed how this leads to an efficient VLSI architecture. Our present IRSIM estimates indicate that a speed-up of 1.71 can be obtained with the modified algorithm. Alternatively, we can scale down the voltage to reduce the power consumption by 35% when compared with the normal design.

REFERENCES

[1] Blahut, *Algebraic methods for Signal processing and Error control coding*, Springer-Verlag, 1992.

[2] S. Whitaker,J. Canaris and K. Cameron, "Reed Solomon VLSI Codec for Advanced Television," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, no. 2,pp. 230-236, June 1991.

[3] H. M. Shao, T. K. Truong, L. J. Deutsch, J. H. Yuen and I. S. Reed, "A VLSI Design of a Pipeline Reed-Solomon Decoder," *IEEE Transactions on Computers*,vol. C-34, no. 5,pp.393-402 May 1985.

[4] Wicker and Bhargava,*Reed-Solomon codes and applications*, IEE Press, 1994.

[5] A. P Chandrakasan and R. W Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," *Proceedings of the IEEE*, Vol. 83, no. 4, pp.498-523, April 1995.

[6] A.-Y Wu, K. J. R Liu, Z. Zhang, K. Nakajima, A. Raghupathy, "Low-Power Design Methodology for DSP Systems using Multirate Approach," *Proceedings IEEE Symposium on Circuits and Systems*, May 1996.

[7] Y. Nakamura, K. Oguri and A. Nagoya, "Synthesis from pure behavioral descriptions," *in High Level VLSI Synthesis* (R. Camposano and W. Wolf,eds.), Kluwer Academic Publishers, 1991,pp.205-229.