

A Low Complexity, High Speed, Regular and Flexible Reed Solomon Decoder for Wireless Communication

Abid Rashid, Frank H.P. Fitzek, Ole Olsen, Yannick Le Moullec
 Aalborg University, CTIF
 Denmark
 Email: {aras03|ff|oo|ylm}@kom.aau.dk

Morten Gade
 Freescale
 Denmark
 Email: mgade1@freescale.com

Abstract—This paper proposes a low complexity, high speed, regular and flexible architecture for VLSI implementation of Reed Solomon decoder. With this architecture the error locator and error evaluator polynomials of the decoder can be computed in parallel and the same datapath can be reused to realize a partially pipelined parallel RS decoder. By architecture reuse the number of required resources for the RS decoder can be adjusted to a specific application, while maintaining internal parallel computation of each RS procedure. The resulting architecture contains one type of program element, which simplifies test and fabrication.

I. INTRODUCTION

Reed Solomon (RS) error codes are used in many communication links to reduce the effect of burst errors. RS codes are also used in the DVB-H standard providing TV and multimedia services on handheld devices. To accomplish power savings and to improve the quality of service, the concept of Time Slicing and Multiprotocol Encapsulation with Forward Error Correction (MPE-FEC) using Reed Solomon decoding is introduced in the DVB-H standard [8].

An RS(n,k) encoder can encode k length information into n length code word. The RS decoder can correct maximum of $t = (n - k)/2$ errors (ρ) and $d = n - k$ erasures (ϵ) or $2 \times \rho + \epsilon \leq d$ of errors and erasures (an erasure occurs when the location of error in the code word is known). Figure 1 shows a block diagram of a typical RS decoder [5] containing;

- **Syndrome Calculator:** syndrome computation is performed to detect if the code word contains errors or not. If there are errors, decoding is performed using the syndrome $S(x)$ at later stages.
- **Error & Erasure Locator:** this block computes the error locator polynomial $\lambda(x)$ from the available erasure information and syndrome. This polynomial is used to find locations and magnitude of errors in the code word.
- **Root Finder:** this block evaluates the $\lambda(x)$ polynomial to find locations of errors using the Chien Search.
- **Error Evaluator:** the error evaluator block is used to compute the error evaluator polynomial $\omega(x)$ which is used in finding magnitude of errors at the inverse of root locations.

- **Magnitude Finder:** this block computes the magnitude of error ($E(\alpha^{-1})$) at inverse of root locations by evaluating the derivative $\lambda'(x)$ polynomial and the $\omega(x)$ polynomial using the Forney Algorithm.
- **Error Correction:** the final step of the decoder is to correct the code word using $E(\alpha^{-1})$ information and the incoming erroneous code word (Recd).

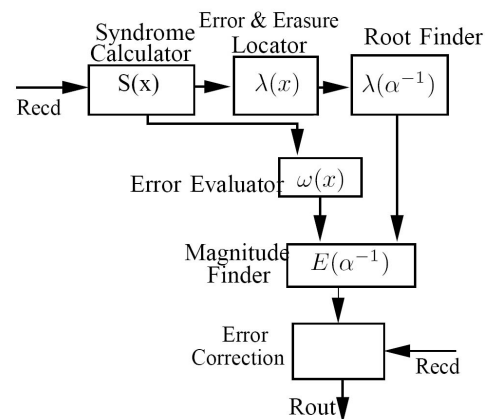


Fig. 1. Block diagram of a typical RS decoder

Sequential execution of the RS decoder for MPE-FEC is very slow. As estimated in [7] the time to decode one frame in the MPE-FEC arriving at a rate of 2Mbps [8], is about 22 minutes on the ARM9E-S RISC architecture. Therefore, high speed implementation of the RS decoder is required to incorporate the forward error correction.

High speed implementation of the RS decoder is a well studied topic as given in [10]-[12]. In the RS decoders, the bottleneck lies in solving the key equations [2] which is done using the Extended Euclidean (eE) algorithm or the Berlekamp Massey (BM) algorithm. Recently a novel algorithm transformation is proposed in [2] based upon the BM algorithm in which the critical path length of the error locator and the evaluator polynomials of the RS decoder is reduced to

T_{add} and T_{mul} , where T_{add} and T_{mul} are the delays of Galois Field (GF) addition and multiplication. This amount

of critical path is much smaller than that of the previous RS decoder implementations using the BM algorithm and comparable to the best known result of implementations based on the eE algorithm [2].

A high speed systolic architecture based on this methodology is expensive as it requires many Processing Elements (PE) for its implementation, e.g., to compute the error evaluator polynomial of data word of length 255 having error ρ and erasure ϵ correction capability $2 \times \rho + \epsilon \leq 64$ [5] as used in MPE-FEC, 65 PEs are required and each PE contains two multipliers and one adder using binary eight bit GF arithmetic. When the error locator polynomial is computed in parallel with the error evaluator, this will require 65 more PEs. If this systolic architecture is realized with Celoxica DK-Suite environment it consumes 36k flip flops with GF(2^8) multiplication carried as explained in IV-A.

Sequential execution of any of the RS procedures can significantly increase the execution time [7], therefore all the individual tasks should be computed in a parallel fashion. If syndrome computation is performed in parallel, it will require 64 more GF adders and GF multipliers. Similarly, Chien Search, Forney algorithm and error correction blocks will consume the resources.

The number of PEs for a fully parallel pipelined implementation of a Reed Solomon Error Decoder RS(n,k) based upon this methodology are given in table I (Refer to section III for equations of each RS procedures).

RS Decoder Block	Multipliers	Adders	PEs	Iterations
Syndrome	d	d	d	n
$\lambda(x)$ & $\omega(x)$	$2 \times (2 \times d)$	$2 \times d$	$2 \times d$	d
Root Finder	d	d	d	$(4 \times d)$
$\omega(\alpha^{-i})$	d	d	d	d
$\lambda'(\alpha^{-i})$	d	d	d	d
Error Correction	d	d	d	1
Total	$9 \times d$	$7 \times d$	$7 \times d$	$7 \times d + n + 1$

TABLE I

TIME-AREA ESTIMATES FOR FULLY PARALLEL RS PROCEDURES

Complexity defined as number of PEs times number of iterations of the RS decoder increases quadratically with increase in error correction capabilities d of the decoder. Different classes of PEs have to be used to implement the RS decoder resulting in a complicated architecture. Therefore any architecture and algorithm transformation that could facilitate the acceleration of the decoding process with reduced complexity and one type of simple PEs of minimal latency is highly desired.

In [1], algorithm transformations for computation of various RS procedures are proposed using one type of PEs but computation of the $\omega(x)$ polynomial is dependent on the $\lambda(x)$ polynomial. The erasure polynomial requires one separate block of PEs as well. Thus the execution time of the decoder is increased as compared to the methodology proposed in [2].

Secondly PEs in [1] use multiplexers to switch between the RS procedures which results in a complex architecture.

To achieve a fast decoding with reduced complexity, this paper presents an architecture containing a simple and regular

type of PEs. With this architecture not only the error locator and the evaluator polynomial can be computed concurrently but syndrome computation, Chien Search, polynomial evaluation and error correction can be performed on the same parallel architecture. Due to parallel computation of each RS procedure it is fast and by the architecture reuse the complexity is reduced.

The main idea behind the methodology is to exploit the data dependency in general and the fact that all the blocks of the RS decoder can be implemented with the equations of the type $\kappa = (\alpha \otimes \chi) \oplus (\beta \otimes \eta)$, where \otimes and \oplus are operations in GF and $\kappa, \alpha, \chi, \beta$ and η are the elements of GF as shown in figure 2. The architecture is devised by an extension of the concepts described in references [1] -[4] to achieve the features of low complexity, high speed, regularity and flexibility.

The remainder of the paper is organized as follows: Section II presents basic PE and a building unit of PEs. With this building unit main datapath computations of all the RS procedures can be performed. Section II also presents a restructured [2] algorithm. To maintain the accelerated computation of the error locator and the evaluator polynomials it is proposed to use two building units as given in [2]. Section III describes the transformations for other RS procedures to achieve architecture reusability. In section IV a scheduling scheme is proposed to realize a partially pipelined parallel RS decoder with the same regular architecture and resource estimation and clock cycles for a DVB-H code word example required for the architecture are estimated with the Celoxica DK-Suite. Finally in section V the various RS decoder solutions are presented to meet certain time and area constraints.

II. ARCHITECTURE FOR THE ERROR EVALUATOR AND THE ERROR LOCATOR POLYNOMIAL

A. Basic Program Element

Figure 2 shows the basic program element used to implement the architecture of the RS decoder. Each of the PE contains two GF multipliers and one adder. The critical path length is only one adder (GF) and one multiplier (GF) long.

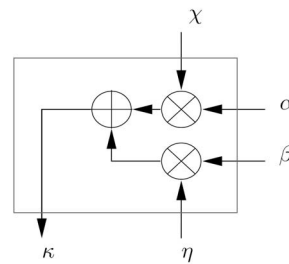


Fig. 2. Basic PE of the architecture

B. Basic Building Block of RS Decoder

Figure 3 shows the basic building block used to implement all the main blocks of the RS decoder. The building block contains one block of PEs having $d + 1$ elements.

```

1 Initialize;
2  $k = 1$  ,  $l = 0$  and  $decode\_flag = 0$ ;  $\omega^A(x) = S(x)$  ,
    $\lambda^A(x) = 1$  ,  $\omega^B(x) = S(x)$  ,  $\lambda^B(x) = 1$  ,  $\omega^C(d) = 1$ ;
3 if  $k < s$  then
4   |  $decode\_flag = 1$ ;
5 end
6  $\omega^B(x) = x\omega^B(x)$  ,  $\lambda^B(x) = x\lambda^B(x)$  ;
7 if  $decode\_flag == 0$  then
8   |  $\delta = 1$  ,  $\gamma = Z_k$  ,  $\omega^C(x) = 0$ 
9 else
10  |  $\delta = \omega^B(d)$  ,  $\gamma = \omega^A(d)$ ;
11 end
12  $\omega^C(x) = (\delta \otimes \omega^A(x)) \oplus (\gamma \otimes \omega^B(x))$ ;
13  $\lambda^C(x) = (\delta \otimes \lambda^A(x)) \oplus (\gamma \otimes \lambda^B(x))$ ;
14 if  $decode\_flag == 0$  then
15  |  $\omega^A(x) = \omega^C(x)$  ,  $\lambda^A(x) = \lambda^C(x)$ ;
16  | if  $\delta \neq 0$  &  $2l \leq k-1$  then
17    |  $l = l - k$ ;
18  | end
19 else
20  | if  $\delta \neq 0$  &  $2l \leq k-1$  then
21    |  $l = l - k$ ;  $\omega^A(x) = \omega^C(x)$  ,  $\lambda^A(x) = \lambda^C(x)$ ;
22  | else
23    |  $\omega^A(x) = \omega^B(x)$  ,  $\lambda^A(x) = \lambda^B(x)$ ;
24  | end
25 end
26  $\omega^B(x) = \omega^C(x)$  ,  $\lambda^B(x) = \lambda^C(x)$ ;
27  $k = k + 1$ ;
28 if  $k \leq d - 1$  then
29   | then go to line 4
30 end

```

Algorithm 1: Re-structured [2] algorithm for the error locator and the evaluator polynomials

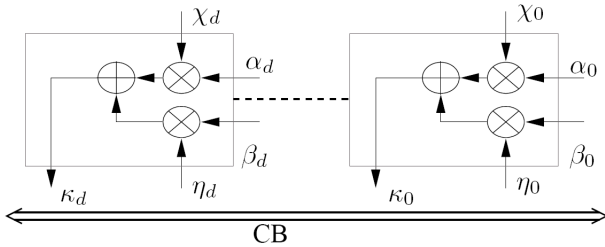


Fig. 3. Basic building unit of the architecture

C. Computation of Error Locator and Evaluator Polynomial with Modified Architecture

According to the algorithm [2], error locator and evaluator polynomials can be computed concurrently. For complete derivation we refer the reader to [2] and present only reformulated equations here. Note the change of notations $\hat{\delta}_i$, $\hat{\delta}_{(i+1)}$, $\hat{\theta}_{(r-1)}$, $\hat{b}_{(i-1)}$ and $\hat{\lambda}^{(r-1)}$ is replaced by ω^C , ω^B , ω^A , λ^A , λ^B and index variables are removed.

Note that a similar set of equation for error evaluator and locator is proposed in [4]. But with this structure and

initialization of the variables of the algorithm intermediate polynomials as required by [4] are avoided and the same architecture can be used to realize a parallel implementation of [4] to solve the key equations. The main datapath equations are 1 and 2 that can be performed with two BBUs surrounded by a control bus (CB).

$$\omega^C(x) = (\delta \otimes \omega^A(x)) \oplus (\gamma \otimes \omega^B(x)) \quad (1)$$

$$\lambda^C(x) = (\delta \otimes \lambda^A(x)) \oplus (\gamma \otimes \lambda^B(x)) \quad (2)$$

Each polynomial $\omega^C(x)$ and $\lambda^C(x)$ contains $d + 1$ elements which can be obtained by carrying out addition and multiplication operations between polynomials $\omega^A(x)$, $\omega^B(x)$ and $\lambda^A(x)$, $\lambda^B(x)$ with δ and γ (elements in GF) respectively. Each operation on the polynomial elements can be computed in one PE. At least d iterations of the PEs are required to compute the error locator and evaluator polynomials concurrently. Other operations of [2] are register to register movements under given control conditions which should be parallelized to accomplish fast processing. Note that the main difference between this architecture and the architecture proposed in [2] is that we have omitted the polynomial shift operation out of each PE which results in polynomial shift operation to be performed after computation of $\lambda^C(x)$ and $\omega^C(x)$. It simplifies architecture of each PE and the architecture reusability is made possible.

III. TRANSFORMATIONS TO COMPUTE RS PROCEDURES WITH THE MODIFIED ARCHITECTURE

Other RS procedures can be computed with the same regular and fast architecture. We need certain transformations to be applied. Such transformations are proposed in [1].

A. Computation of Syndrome Calculator with the Parallel Architecture

For the RS(n,k) decoder, there are d syndrome elements to be computed. The computation of all the elements is independent from each other which can be exploited for accelerated implementation of syndrome calculator. d PEs of the block can be used to compute the syndrome. The basic equation to compute syndrome are given in 3. α 's are the elements from GF. Initially each syndrome element is initialized with zero and each element requires n iterations of d program elements to compute full syndrome.

$$S_j = (S_{i-1,j} \otimes \alpha_{j-1}) \oplus (1 \otimes Recd[n - i]) \quad i \leq n \quad (3)$$

Mapping of PE to j 'th syndrome element is given as $\alpha = S_{i-1,j}$, $\chi = \alpha_{j-1}$, $\beta = 1$, $\eta = Recd[n - i]$, $\kappa = S_j$. $Recd$ is array of received code word.

B. Computation of Root Finder on the Parallel architecture

Once the error locator and the evaluator polynomial are computed, the next step of the decoder is to find the roots of the polynomial which can be scheduled on one of the building

blocks as shown in the figure 3. The roots j elements from 0 to $n - 1$ can be evaluated by 4.

$$\lambda(\alpha_i) = (\alpha_i \otimes \lambda(\alpha_i)) \oplus (1 \otimes \lambda(x)) \quad 0 \leq i \leq n \quad (4)$$

There are n elements that can be the possible roots of the $\lambda(x)$ polynomial. By using block of parallel PEs, this computation can be accelerated. First $d + 1$ locations can be checked on the architecture simultaneously. They will require iterations equal to degree of the $\lambda(x)$ polynomial. Once the first $d + 1$ elements are computed, the next $d + 1$ elements can be computed on the same block. Thus maximum of $(d + 1) \times 4$ iterations of PE are required to compute the roots of the polynomial.

C. Polynomial Evaluation with the Parallel architecture

Similarly $\omega(\alpha_j^{-1})$ and $\lambda'(\alpha_j^{-1})$ can be computed with an equation similar as 4. The number of iterations required is equal to the degree of $\lambda(x)$ polynomial.

D. Error Correction with the Parallel architecture

The Forney algorithm is used to compute the magnitude of error at the inverse locations of the roots of the polynomial $\lambda(x)$. If we take the inverse of the polynomial elements $\lambda'(\alpha_j^{-1})$ the error correction at j 'th location of the roots can be carried out with the equation 5

$$\tilde{Y}_j = (\omega(\alpha_j^{-1}) \otimes \lambda'(\alpha_j^{-1})) \oplus (1 \otimes Recd[j]) \quad (5)$$

From the above discussion, we can see that except for error locator and evaluator polynomials, all the other blocks require one GF multiplier and adder. Therefore, their computation can be performed with less expensive PE blocks, but regularity in terms of structure of PEs will not be retained in the architecture.

IV. IMPLEMENTATION RESULTS

To implement the RS decoder with the new architecture we have used Celoxica DK environment with C-style, Handel-C language for software compiled hardware. Due to its higher level of abstraction for implementation, Handel-C is a better candidate to estimate resources and speed of an architecture in a short span of time. In Handel-C every assignment executes in one clock cycle. We have used DK design suite to simulate the code and to estimate the resources and clock cycles required. Lower level optimizations for place and route tool are not considered. We followed DVB-H standard as a case study which uses GF(28) RS(255,191) decoder at the data link layer with irreducible polynomial of $x^8 + x^4 + x^3 + x^2 + 1$ [9].

A. Resource and Speed Estimation of the RS decoder with Handel-C

The basic components for RS decoder are GF multiplier, adder and inverse operations. Under GF mathematics addition is carried out with a simple XOR operation and inversion can be carried with simple table look up methodology in which against each element of GF its inverse is stored. Multiplication

is complex and effects the performance of the decoder. For VLSI implementation of RS decoder, we choose bitwise GF(2⁸) multiplication methodology as proposed by [6].

Such a GF(2⁸) multiplication can be carried out in one clock cycle. But an eight bit multiplication requires 64 AND gates and 55 Xor gates. If bitwise multiplication is carried out in one clock cycle, then the maximum combinatorial path contains 24 AND gates and 21 XOR gates. If this multiplier is realized in one clock cycle then 69 flipflops and 1654 Nand gates are required. Such path delay limits the driving clock frequency which limits the performance of the decoder. To reduce combinatorial delay we splitted each multiplier into two stages and maximum delay contains 8 AND gates and 7 XOR gates.

Following code shows how Gf(2⁸) multiplication is carried out in Handel-C

```
gfMul(unsigned int 8 x, unsigned int 8 y){
unsigned int 1 z0,z1,z2,,,,,,,,z14;
unsigned int 1 o0,o1,,,,,,,,o7;
par{z0 = x[0]&y[0];
z1 = (x[1]&y[0]) (x[0]&y[1]);
.....
z7 = (x[7]&y[0]) (x[6]&y[1]) ..... (x[0]&y[7]);
.....
z14 = x[0] & y[7];}
par{o0 = z11z12z13z7;
o1 = z14z10z11z12z6;
.....
o7 = z12z13z14z8z0;}
return o0o1.....o7;}
```

The price paid is in a form of higher number of resources and two clock cycles instead of one. With this approach each multiplication is carried out in two clock cycles with 114 flipflops and 1501 Nand gates. Note that the multiplication operation is optimized in terms of resources with nine bit irreducible polynomial $x^8 + x^4 + x^3 + x^2 + 1$. With keyword par parallel hardware is generated for each instruction within braces with Handel-C.

B. RS Decoder Architecture with Two Building Units

From the discussion in section II-C we can find that two blocks can be used to compute error locator and evaluator polynomial concurrently.

We used two GF multipliers running in parallel and one adder operation to concatenate their operation for each PE to achieve accelerated implementation reducing combinatorial path at the same time. Therefore one iteration of PE requires 3 clock cycles in Handel-C.

To estimate the resources, RS(255,191) code word $C = 1 + x + x^2 \dots x^{254}$ is used and added with 31 errors. The error magnitude is increased from 0 to 31 in first 32 code elements of the code word. Therefore the decoder has following input as a code word $x + 2x^2 + 3x^3 \dots 31x^{31} + x^{32} \dots x^{254}$.

1) Syndrome Computation: The following code shows computation of syndrome in Handel-C.

```
while(i < 255){
c = Recd[i];
par{PE1(tmp1,alpha_1,1,c);
.....
PE64(tmp64,alpha_64,1,c);
i++;}
}
```

Each PE function in this example generates a hardware for PE which contains two parallel GF multipliers followed by a GF addition. Each function takes four arguments. Following code shows inner structure of each PE.

```
PE(unsigned int 8 x,unsigned int 8 y,
unsigned int 8 z, unsigned int 8 k){
par{
first = gfmull1(x,y);
second = gfmull2(z,k);
tmp = first ^ second;
}
}
```

tmp1, tmp2,.....tmp64 are the syndrome elements. α 's are the elements in the GF. After 255 iterations syndrome polynomial $S(x) = 0 + 46x + 181x^2 + \dots + 184x^{64}$ is computed.

2) *Computation of Error Locator and Evaluator Polynomial:* Two blocks, each of 65 PEs, are used to compute error locator and evaluator polynomial. One block is the same as used to compute the syndrome on which error evaluator is performed and another is added to compute error locator polynomial.

```
while (k < d){
par{delta = omega_A65 ;gamma = omega_B65 ;}
par{
par{PE0(omega_B1,delta,omega_a0,gamma);
.....
PE64(omega_B64,delta,omega_a64,gamma);}
par{xPE0(lambda_B1,delta,lambda_a0,gamma);
.....
xPE64(lambda_B64,delta,lambda_a64,gamma);}
}
```

In this code xPEs represent second block of PEs. Variables delta, gamma, omega as, omega bs, lambda as and lambda bs represent elements of $\delta, \gamma, \omega^a(x), \omega^b(x), \lambda^a(x)$ and $\lambda^b(x)$ in GF Initialization and other operations for the computation are assignment operations which are performed under conditions given in [2] with a par statement. 184, 133, 19,....., 0 and 1, 184, 184,, 255 are the values of δ and γ for 1, 2, 3,, 64 iterations respectively. After d iterations $\omega^C(x) = 42x^{34} + 77x^{35} + 182x^{36} + \dots + 56x^{63}$ and $\lambda^C(x) = 205x^{33} + 109x^{34} + 247x^{35} + \dots + 45x^{63} + 173x^{64}$ are obtained.

3) *Computation of the roots of Error Locator:* The elements of $\lambda(x)$ can be saved in an array and following code can be used to evaluate the roots.

```
while(i >= Degree){
c = Lambda[i];
par{PE0(tmp1,alpha_1,c,1);
PE1(tmp2,alpha_2,c,1);
.....
PE64(tmp64,alpha_64,c,1);
i--;}
}
if(tmp1==0){par{root[count]=tmp1;count++;}}
if(tmp2==0){par{root[count]=tmp2;count++;}}
```

After iterations equal to "Degree" of the polynomial $\lambda(x)$, each tmp variable should be checked to see either it is zero or not. The roots are saved in an array named root. Such 4 iterations of the code are required to evaluate $\lambda(x)$ at locations. In our case we find the roots at 0,2,3,4,5,6,.....,31 positions.

4) *Polynomial Evaluation:* After finding the roots, the next step is to find magnitude of errors at inverse of root locations and apply error correction. The $\lambda'(\alpha_j^{-1}), \omega(\alpha_j^{-1})$ and error correction is implemented in the similar way as

given in root finder with one block of PEs. Error polynomial $1 + 3x^2 + \dots + 32x^{31}$ is obtained which is used to recover original data after implementing the equation 5 with the same parallel architecture reflected by the following code.

```
par{if(root1) PE1(omegaA1,lambdaA1,c,1);
if(root2) PE2(omegaA2,lambdaA2,c,1);
.....
if(root64) PE64(omegaA64,lambdaA64,c,1);}
```

In this code omegaAs represent the values of $\omega(x)$ and lambdaAs represent inverse of the values of $\lambda'(x)$ computed at the inverse of the root locations. We have presented here the main datapath calculations by reusing the architecture. Other assignment statements are not coded here. With par statement all the assignment operations are performed in parallel. We found that this approach requires 3049 clock cycles to correct the data in the code word with 37k flip flops and 513K Nand Gates estimated with the DK-Suite simulator. Note that all the buffers are declared without ram keyword of Handel-C, which enables buffer data to be shared among different parallel tasks. All Optimizations features by Celoxica DK-Suite were enabled.

C. Partially Pipelined Parallel RS Systolic Architecture

From the above discussion we can see that one block of the PEs remains idle after computation of the error locator and the evaluator polynomial that can be reused to compute the syndrome of next code word in parallel with root finder and error correction procedures. Second block was used to compute the syndrome of the next code word and figure 4 shows clock cycles required and the various RS decoding tasks with two blocks of PEs, PA1 and PB1.

Figure 4 shows clock cycles required and various RS decoding tasks with two blocks of PEs, PA1 and PB1.

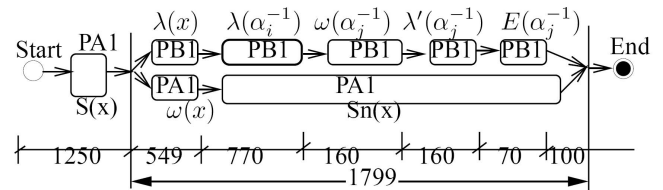


Fig. 4. Time-activity diagram for RS decoder with two PE building blocks

Note that computation of $\lambda(x)$ and $\omega(x)$ for next code word is synchronized with next code word syndrome $S_n(x)$ computation. Therefore, a partially pipelined parallel RS decoder is possible with two blocks of PEs. It was estimated that first code word requires 3049 clock cycles and next code word will be decoded in 1799 clock cycles with 37K flip flops and 546K Nand gates.

D. Estimation of Time to Decode DVB-H Frame

We used place and route tool of Xilinx-ISE for Spartan II XC2S200-5-FG456 FPGA to estimate maximum clock frequency with which the RS decoder based upon this methodology can be operated. Maximum combinatorial path delay was 104nsec. With this information we estimate that a RS(255,191)

decoder can achieve a bandwidth of almost 10Mbps. A complete 2Mbps data (Maximum burst size with MPE-FEC) can be decoded in almost 200msec.

V. DESIGN SPACE EXPLORATION FOR THE PARALLEL RS DECODER

The architecture is flexible as if higher decoding speed is required then more PEs can be added in the pipeline to increase the throughput. If lower speed is allowed then the architecture containing a subset of PEs can be time sliced to achieve an adequate performance at lower cost in area. Such a flexibility leads various solutions to the RS decoder to meet certain time and area constraints and provide certain degree of freedom of the choice of the RS decoder solutions. Different combinations of BBUs for the RS decoder maintaining the internal parallel computation of each RS procedure can be used as shown in Figure 5 and Table II.

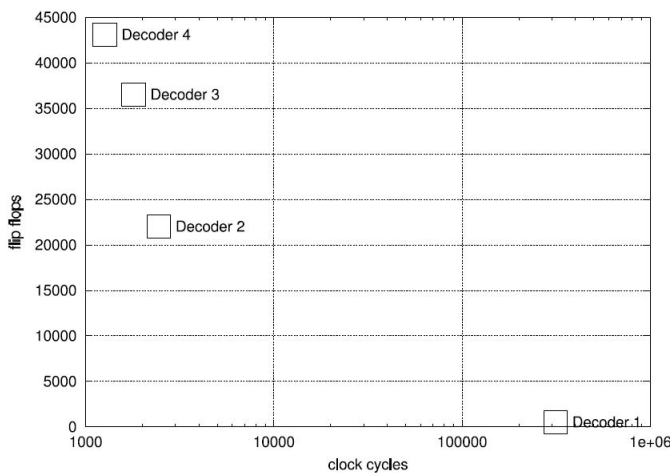


Fig. 5. Resource and time comparison for the various RS decoder forms

We are presenting the estimate of four of such solutions.

- Complete RS decoder can be implemented with a cell of one multiplier and adder. But the solution will be not be able to exploit the VLSI features. It will require 313207 clock cycles with 518 flip flops and 45612 Nand gates. All the assignments are performed in a sequential fashion.
- Complete RS decoder can be implemented with one BBU. It can compute all RS procedures in a parallel way internally providing significant computational speed. Solution with one BBU require 22k flip flops and 301K Nand gates and decode the code word with 2448 clock cycles. All the assignments are made with parallel statement.
- The solution with two BBUs is discussed and explained above. The clock cycles required are 1799 with 36523 flip flops and 573K Nand gates.
- To achieve higher throughput complete RS decoder can be implemented with three blocks two of which will be BBUs used to compute the error locator and the evaluator polynomial concurrently, root finder, polynomial evaluation, error correction and one can be used to compute the

syndrome in a parallel. Thus complete code word can be decoded in 1263 clock cycles with 43096 flip flops and 713K Nand gates.

RS Version	Nand Gates	Flipflops	Cycles	Time Area 106
1	45612	519	313207	15098
2	301123	22694	2448	1006
3	573212	36523	1799	1228
4	713213	43096	1263	1172

TABLE II

TIME-AREA PRODUCT ESTIMATES FOR VARIOUS RS SOLUTIONS

In the estimate the execution time is estimated with number of clock cycles being a technology independent factor. The area should be estimated in terms of Control Logic Blocks (CLBs) of the FPGA but Handel-C does not provide number of CLBs but number of flipflops and Nand gates can be estimated. Therefore for area estimate, we have used an approximation given by the number of flipflops*5+Nand gates.

VI. CONCLUSIONS

We have presented a high speed partially pipelined parallel architecture for Reed Solomon decoder and its performance is estimated with the Celoxica DK-Suite. Measurements show that a parallel RS(255,191) decoder can be implemented with only 130 PEs and can achieve a bandwidth of 10Mbps. The architecture is regular as the same type of PEs are used and is flexible to meet higher decoding rates.

REFERENCES

- [1] Keiichi Iwamura, Yasunori Dohi, Hideki Imai: *A Design of Reed-Solomon Decoder with Systolic Array Structure*. IEEE Transactions on Computers Vol 44 No.1 1995.
- [2] Tong Zhang and Keshab K. Parhi: *On the High-Speed VLSI Implementation of Error-and Erasres Correcting Reed-Solomon Decoders*. IEEE Transactions, 2003.
- [3] Dilip V. Sarwate and Naresh R. Shanbagh: *High-Speed Architectures for Reed-Solomon Decoders*. IEEE Transactions on VLSI Systems, 9 (5):641-655, Oct. 2001.
- [4] J.H.Jeng , T.K.Troung, T.C.Cheng: *New Decoding Algorithm for Correcting Both Erasures and Errors of Reed-Solomon Codes*. 12 ACM Great Lakes symposium on VLSI New York USA pp 89-93, 2002.
- [5] Nevio Benvenuto : *Algorithms for Communications Systems and their Applications*, Wiley Sons 2002..
- [6] Nick Iliev, James Stine, Nathan Jachimiec :*Digital Finite-Field Multiplier for Reed-Solomon Channel code in GF(2⁸) with programmable basis Polynomial*. IIT VLSI Labs, 2003.
- [7] Abid Rashid, AAU Denmark *Technical Report, Design Space Exploration for RS Decoder*, June 2005.
- [8] Final Draft ETSI EN 301-192, *Digital Video Broadcasting(DVB); DVB specification for data broadcasting*. 2001.
- [9] Shu Lin and Daniel J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*, Prentice Hall 1983.
- [10] Maurizio Martina, Guido Masera, Gianluca Piccinin, Fabrizio Vacca and Maurizio, *VLSI Reed Solomon Decoder Architecture for Networked Multimedia Applications*. IEEE 2001.
- [11] Dong Hoon Lee, Seung Wook Lee, Jong Tae Kim, *A Reed Solomon Decoder with the Efficient Recursive Cell Architecture for DVD Application*. IEEE 2001.
- [12] Arun Raghupathy and K.J.R. Lie *Algorithms-Based Low-Power/High-Speed Reed-Solomon Decoder Design* IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Vol 47, No. 11 November 2000.