A Pipeline Frequency-Domain Reed-Solomon Decoder for Application in ATM Networks*

Antonio Gabriel Lomeña†, Juan Carlos López† and Ander Royo‡

† Universidad Politécnica de Madrid
E.T.S.I. Telecomunicación
Ciudad Universitaria s/n
28040 Madrid (Spain)
e-mail: {lomena, lopez}@die.upm.es

‡ Lucent Technologies
Microelectrónica España
Tres Cantos s/n Zona Oeste
28760 Tres Cantos - Madrid (Spain)
e-mail: aro@tapies.micro.lucent.com

Abstract.

Reed-Solomon (RS) codes are considered one of the most powerful algebraic codes and have found many applications in telecommunications during the last years. In the present paper, we develop the hardware implementation of a frequency-domain RS decoder to work in an ATM network.

1 Introduction.

Asynchronous transfer mode (ATM) is a high-speed communication technology that has reached an almost universal acceptance in the last years. Since the involved transmission rates are quite high, the achievement of a good error control in the transmission of the information has become one of the most important problems. Therefore, hardware implementations appear as the most appropriate solution for encoding and decoding the information, moreover as future specifications tend to offer higher speed services.

In the following sections a circuit for an ATM embedded RS decoder will be described. The goal will be the design of a small area circuit compliant with the ATM speed requirements. The paper is organized as follows: section 2 presents a brief summary of the state-of-the-art involving RS decoder implementations. In section 3 the problem to solve is formulated and important concepts are described. In section 4 different Galois Field arithmetic operations are analyzed in order to select the better implementations. Section 5 presents some general considerations, which have been followed throughout the design. The implementation of the different stages are described in section 6 and section 7. In section 8 the global pipelining of the circuit is explained and some synthesis results are presented. Finally, the conclusions are outlined in section 9.

2 Related work.

There are several decoder implementations in the bibliography. In [1] and [3] a time-domain decoder is developed. In spite of its versatility, its high complexity does not make it suitable for the ATM requirements. In [2] a systolic array RS decoder is implemented. Again, the complexity of this architecture is somewhat high and it involves too much area. In reference [5] a pipeline RS decoder is devised. Although this circuit present features which are suitable for an ATM embedded design, the decoder which will be implemented in the present paper has better performance in stages such as the error correction modules. A new pipeline RS decoder is presented in [4]. Though this design is thoroughly scalable and shows a high throughput it also presents a somewhat complex control due to its recursive cells that does not make it the better choice for the ATM specifications.

3 Problem statement.

3.1 Important concepts and notation.

- *Galois Field, GF(p)*. It is a finite set composed of p elements which will allow an algebraic, methodical treatment of error correcting codes. Each GF has a *primitive element*, α, meaning that any other element in that field can be expressed as a power of α. Further details can be found in [10].
- *Error:* it happens when one symbol of the transmitted word is swapped to another valid symbol due to the channel noise. The receiver will not detect anything wrong and the decoder will have to find out the location of the error and its value.
- *Erasure:* it happens when the noise changes one symbol in such a way that the receiver detects it ambiguously. The receiver will mark the symbol as faulty, and the decoder will only have to correct it (since the location in the word is already known).
- An error correcting code is determined by two parameters expressed in the form (n,k), where k is the number of information symbols and n is the total number of symbols in the encoded word. For RS codes it is verified that $2t = n-k$, where t is the number of errors that can be corrected. The polynomials involved in the construction of a RS code over $GF(p^m)$ where p is a prime and m is a positive integer would be the following:

* This work has been funded by CICYT under project HADA (TIC97-0928)

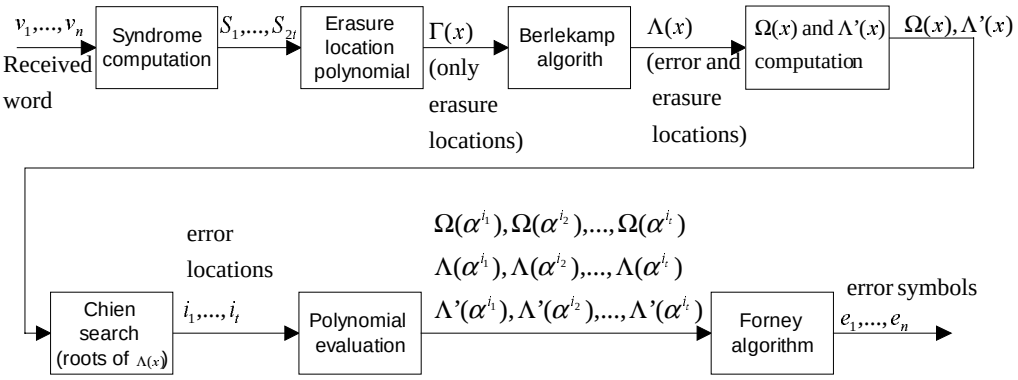

Figure 1— Decoding stages.

Information word:

$$i(x) = i_{n-1}x^{n-1} + i_{n-2}x^{n-2} + \cdots + i_1 x + i_0$$

Generator polinomial:

$$g(x) = (x - \alpha^{b_0})(x - \alpha^{b_0+1})(x - \alpha^{b_0+2}) \cdots (x - \alpha^{b_0+2t-1})$$

Transmitted word:

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1 x + c_0$$

Error word:

$$e(x) = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + \cdots + e_1 x + e_0$$

Received word:

$$v(x) = v_{n-1}x^{n-1} + v_{n-2}x^{n-2} + \cdots + v_1 x + v_0$$

where $i_i, c_i, v_i, e_i \in GF(p^m)$ $i = 0,\ldots,n-1$, b_0 is any arbitrary integer that we choose and α is a primitive element of $GF(p^m)$

So it is verified that:

$$v(x) = c(x) + e(x) = i(x)g(x) + e(x) \qquad (1)$$

3.2 The decoding problem in ATM networks.

The ATM specifications recommend a FEC (*Forward Error Correcting*) approach implemented with a $(n,k) = (128,124)$ RS code over $GF(2^8)$. Therefore each code word is composed of 128 eight-bit symbols; 124 will be information symbols and 4 will be redundant symbols. The number of errors and erasures that can be corrected is given by:

$$n - k \geq 2t + \rho \qquad (2)$$

where t is the number of errors that happened and ρ is the number of erasures. Consequently, for the case of a (128,124) RS code, 2 errors, 4 erasures or a mix of both, verifying the equation (**2**), can be corrected:

It is useful to remark that the RS code implemented in ATM networks is a shortened RS. The true code that is obtained from the algebraic theory would be a (255,251) code. The communication features of ATM networks only require 124 information symbols instead of 251. Thus a shortened code in which 127 symbols out of 251 are padded to zero is employed.

Several considerations lead to the conclusion that a frequency-domain Reed-Solomon decoder which implements Berlekamp-Massey plus Forney algorithm is the most appropriate option to reach the requirements of the ATM specifications [11]. The main stages that will compose this decoder are shown in Figure 1. Deeper details about the involved algorithms can be found in [10]. Briefly, the steps to be performed are the following:

1. To compute the syndromes, which is equivalent to a Galois Field Fourier Transform that converts the time-domain data into the frequency domain.
2. To find out the erasure and error locations.
   a) To construct the erasure-location polynomial , $\Gamma(\ )$,whose roots will indicate the location of the erasures in the received word.
   b) To compute the error-location polynomial $\Lambda(x)$, whose roots will indicate the location of the errors and erasures in the received frame. It is achieved by the Berlekamp-Massey algorithm. The initial value of $\Lambda(x)$ will be the erasure location polynomial.
3. To find the value of each error.
   a) To construct the error evaluator polynomial, $\Omega(x)$, where $\Omega(x) = S(x)\Lambda(x) (\mathrm{mod}\, x^{2t})$ and the Galois field derivative of $\Lambda(x)$, $\Lambda'(x)$.
   b) To perform a Chien search in which every element of the field $(0,1,\alpha,\alpha^2,\alpha^3,\ldots)$ is tested to see if it is a root of $\Lambda(x)$. The exponent of the roots will indicate the error locations.
   c) To apply the Forney algorithm, as is shown in the following equation:

$$e_{i_l} = \frac{-\Omega(\alpha^{-i_l})\alpha^{-(b_0-1)i_l}}{\Lambda'(\alpha^{-i_l})} \qquad l = 1,\ldots,t \qquad (3)$$

where e_{i_l} is the error symbol in the location i_l and α^{-i_l} is a root of $\Lambda(x)$.

4 Arithmetic operations in Galois Fields.

The computations which appear in the decoding algorithms are basically the addition, the multiplication, the inversion and the exponentiation of finite field elements. There are two main ways to implement these operations depending on the selected representation for the elements of the Galois field.

a) If they are represented as powers of the primitive element of the field, α, in the form

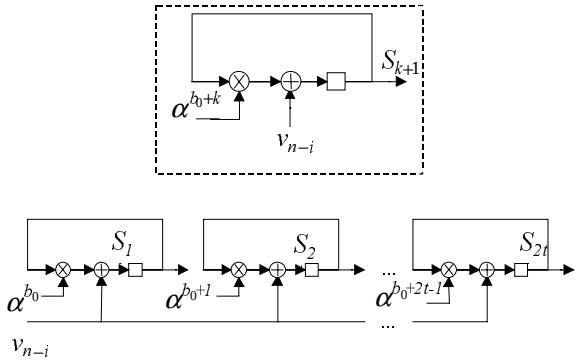

Figure 2— Syndrome generation.

$GF(q) = GF(p^m) = \{0,1,\alpha,\alpha^2,\alpha^3,\ldots,\alpha^{q-2}\}$ then it is really easy to perform multiplication, inversion and exponentiation since they are obtained adding or subtracting the exponents modulo $p^m - 1$. However the addition will be difficult to achieve and either complicated algorithms or a look-up table will be needed.

b) If they are represented as polynomials in the form $GF(q) = GF(p^m) = GF(p)[x]/f(x)$, where $f(x) \in GF(p)[x]$ is a primitive polynomial of degree *m*, then the addition will be easy since it is obtained adding the polynomial coefficients modulo *p*. In the case of $p = 2$ this is achieved with XOR gates. But now the other operations will be complicated and will require elaborated algorithms or look-up tables.

As it will be shown, the number of multiplications and additions in the algorithms is quite high being the amount of multiplications slightly superior to the number of additions. Besides there are two inversions and one exponentiation in the global decoding process. Nevertheless, the polynomial representation seems to be more appropriate for our system. It is so because it does not need conversions from the raw input data. Moreover, there has been extensive work on Galois field multiplication algorithms and feasible VLSI multipliers are attainable [7],[8].

Therefore the polynomial representation will be adopted. Look-up tables will be used to implement inversion and exponentiation. Hence there will be 3 look-up tables of 256 bytes each one in the circuit. Concerning the multiplication various alternatives can be implemented:

a) Polynomial base[7], dual base [6] or normal base [8] multipliers. In order to avoid the conversion between the multiplication basis and the representation basis, a polynomial basis will be chosen for the multiplier.

b) Classic or systolic multiplication. In spite of the speed advantages of systolic circuits a non-systolic multiplier will be implemented. It is due to a speed-area trade-off since multipliers must be massively used in the circuit and the increase in area would be too high.

Details about the implemented multiplier can be found in [7].

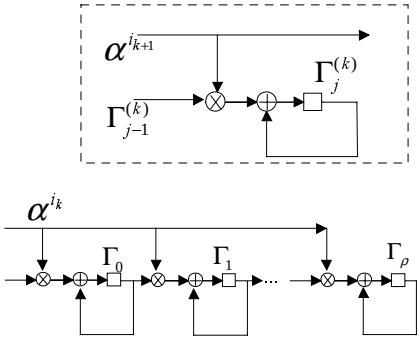

Figure 3— Erasure polynomial computation.

5 Hardware implementation of the decoder

In the following sections the design architecture of the decoder will be described. The first thing to explain is the pipeline that has been implemented in the circuit, since it will determine the minimum clock period that can be employed.

Two factors will limit the clock frequency: the longest combinational path between two registers and the memory access time. The data-path has been pipelined into different stages which, in the worst case, will include a multiplication and two additions of elements of the field $GF(2^8)$. This means that a multiplication and two additions must be performed in one clock cycle. Anyhow, the clock cycle has been limited by the memory access (40ns). Thus a working frequency of 20Mhz (50 ns) has been finally chosen.

Since the field is $GF(2^8)$ each register and bus that appears in the following pictures will be 8 bits wide. It is important to remember that the (128,124) Reed-Solomon code will have the following parameter values $t = 2$, $\rho = 4$. Next the different stages of the decoding process will be described.

6 Computation of the error and erasure locations.

6.1 Syndrome generation.

The first stage of the decoder computes the syndromes of the received word. The following equations show the operations to perform.

$$\begin{aligned} &S(x) = S_1 + S_2 x + S_3 x^2 + \ldots + S_{2t} x^{2t-1} \\ &S_j = \sum_{i=0}^{n-1} v_i \alpha^{i(b_0+j-1)} \quad j = 1,\ldots 2t \\ &\Rightarrow S_1 = v_0 + v_1 \alpha^{b_0} + \ldots + v_n \alpha^{n b_0} \\ &\quad S_2 = v_0 + v_1 \alpha^{(b_0+1)} + \ldots + v_n \alpha^{n(b_0+1)} \\ &\quad \ldots\ldots \end{aligned} \tag{4}$$

They can be implemented with the pipeline circuit of Figure 2. Initially the registers must be set to zero. This stage spends 128 cycles, since to generate a syndrome the 128 symbols of the code-word must be loaded.

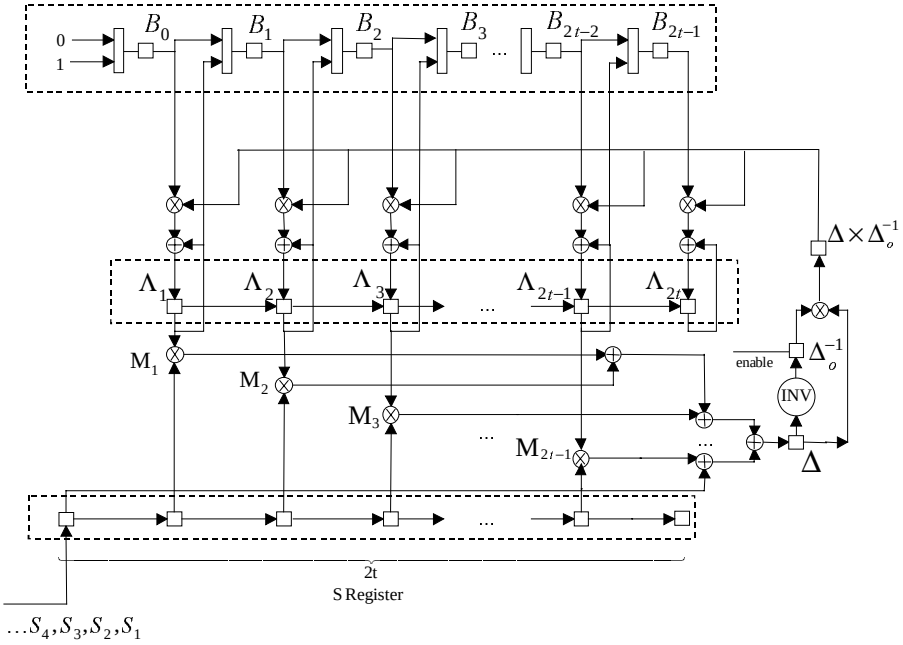

Figure 4— Berlekamp-Massey implementation.

6.2 Erasure polynomial computation

If there have been erasures in the transmission it is necessary to obtain the erasure polynomial. The equation is:

$$\Gamma(x) = (1-\alpha^{i_1}x)(1-\alpha^{i_2}x)\cdots(1-\alpha^{i_\rho}x) \qquad (5)$$

which can be recursively expressed as:

$$\begin{aligned}\Gamma^{(k+1)}(x) &= \Gamma^{(k)}(x)(1-\alpha^{i_{k+1}}x) = \\ &= \Gamma^{(k)}(x) - \Gamma^{(k)}(x)\alpha^{i_{k+1}}x \\ \Rightarrow \Gamma_i^{(k+1)} &= \Gamma_i^{(k)} - \Gamma_{i-1}^{(k)}\alpha^{j_{k+1}}\end{aligned} \qquad (6)$$

To implement those equations the pipeline of Figure 3 has been constructed. This stage will add a delay of 4 clock cycles.

6.3 Berlekamp-Massey algorithm.

Once the erasure polynomial and all the syndromes have been computed, the location of the errors can be found by means of the Berlekamp-Massey algorithm, which can be stated as follows:

Inital conditions

$$\Lambda^{(0)}(x) = \text{erasure polynomial}$$
$$\text{or } 1 \text{ if there were no erasures.} \qquad (7.1)$$
$$\mathrm{B}^{(0)}(x) = 1, L_0 = 0$$

Iteration counter, r

$$r = 1, \ldots, 2t$$

Discrepancy, Δ_r

$$\Delta_r = \sum_{j=0}^{n-1} \Lambda_j^{(r-1)} S_{r-j} \qquad (7.2)$$

Control variables updating, δ_r and L_r

$$\text{if } (\Delta_r \neq 0 \text{ and } 2L_{r-1} \leq r-1) \Rightarrow \delta_r = 1$$
$$\text{else } \Rightarrow \delta_r = 0 \qquad (7.3)$$

$$L_r = \delta_r(r - L_{r-1}) + (1-\delta_r)L_{r-1} \qquad (7.4)$$

Polynomials updating, $\Lambda^{(r)}$ and $B^{(r)}$

$$\begin{aligned}\Lambda^{(r)}(x) &= \Lambda^{(r-1)}(x) - \Delta_r x B^{(r-1)}(x) \\ B^{(r)}(x) &= \Delta_r^{-1}\delta_r \Lambda^{(r-1)}(x) + (1-\delta_r)xB^{(r-1)}(x)\end{aligned} \qquad (7.5)$$

Figure 4 shows the circuit employed, based in the architecture presented in [9]. The pipeline has been implemented with the criterion explained in section 5. Thus in each clock cycle there will be a maximum of a multiplication and two additions.

The algorithm latency is $2t$ cycles. Since $t = 2$, it will last 4 cycles. However, each one of these cycles has been partitioned into four clock cycles. Hence the final number is 16 clock cycles. The circuit operates in the following way. In the active clock edge a new syndrome symbol is introduced into the shift register S, beginning by S_1. Then the discrepancy, Δ_r, is computed by means of the multipliers M_1 to M_{2t} (eq. (**7.2**)). The result is stored in the Δ register during the next cycle (cycle 2).

In the third cycle the value $\Delta \times \Delta_0^{-1}$ is loaded and the tests of equation (**7.3**) are executed to decide the next path of the algorithm. The variables L and δ are updated (eq. (**7.3**) and (**7.4**)).

In the fourth cycle the registers B and Λ are updated as equation (**7.5**) indicates. Besides, if $\delta = 1$ the register Δ_0^{-1} will load the inverse of Δ.

7 Error values computation.

7.1 Error value polynomial computation, Ω(x).

Once the error location polynomial $\Lambda(x)$ has been obtained, the next step is to compute the error value polynomial. It is a polynomial multiplication as the

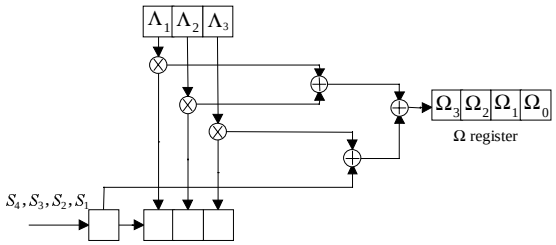

Figure 5— Ω(x) computation.

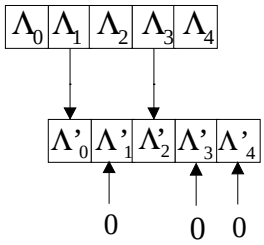

Figure 6— Computation of Λ'(*x*).

following equations show.

$$\Omega(x) = S(x)\Lambda(x) (\mathrm{mod}\ 2t)$$
$$S(x) = S_1 + S_2 x + \cdots S_4 x^3$$
$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \cdots \Lambda_3 x^3 \Lambda_3 S_1 \quad (8)$$

$$\Omega_0 = \Lambda_0 S_1$$
$$\Omega_1 = \Lambda_0 S_2 + \Lambda_1 S_1$$
$$\Omega_2 = \Lambda_0 S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 \quad (9)$$
$$\Omega_3 = \Lambda_0 S_4 + \Lambda_1 S_3 + \Lambda_2 S_2 + \Lambda_3 S_1$$

Since the multiplication is performed modulo 4 it is only necessary to compute up to the Ω_3 component. Figure 5 shows the implemented circuit.

In each cycle the syndromes are introduced in register S and the corresponding value of component Ω_i is loaded in the Ω register during the next cycle.

To save area the same registers and multipliers that were employed in the previous stage (Berlekamp-Massey circuit) are used again.

7.2 Derivative of the error location polynomial.

The next step is the computation of $\Lambda'(x)$, the derivative of $\Lambda(x)$, which is easily implemented with a wired circuit. The equations are:

$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3 + \Lambda_4 x^4$$
$$\Lambda'(x) = \Lambda_1 + 2\times\Lambda_2 x + 3\times\Lambda_3 x^2 + 4\times\Lambda_4 x^3 \quad (10)$$

and since $2 \equiv 0 \bmod 2$, $3 \equiv 1 \bmod 2$ and $4 \equiv 0 \bmod 2$ it results:

$$\Lambda'(x) = \Lambda_1 + 3\times\Lambda_3 x^2 = \Lambda_1 + \Lambda_3 x^2 \quad (11)$$

Thus the hardware implementation would be that of Figure 6.

7.3 Polynomial evaluation of *Ω*(*x*), Λ(*x*) y Λ'(*x*).

This stage consists in evaluating all the obtained polynomials for the finite field elements $\alpha^{-127}, \alpha^{-126}, \ldots \alpha^{-1}, 1$. If one of these elements is a root of the error location polynomial it will mean that an error exists in the position which corresponds to that exponent (with the opposite sign).

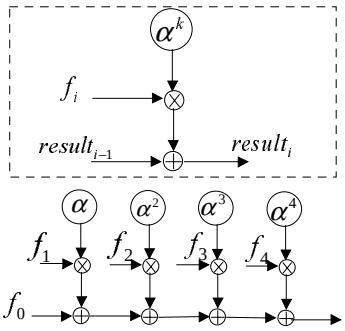

Figure 7— Polynomial evaluation.

The operations used for evaluating the polynomials are stated in the following equations:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \ldots + f_{2t} x^{2t}$$
$$f(\alpha^{-s+i}) = f_0 + f_1 \alpha^{-s+i} + f_2 \alpha^{2(-s+i)} +$$
$$+ \ldots + f_{2t} \alpha^{2t(-s+i)} \qquad s = 128;\ i = 1,2,\ldots s$$

$$f(\alpha^{-s+1}) = f_0 + f_1 \alpha^{-s+1} + f_2 \alpha^{2(-s+1)} +$$
$$+ \ldots + f_{2t} \alpha^{2t(-s+1)} \quad (12)$$
$$f(\alpha^{-s+2}) = f_0 + f_1 \alpha^{-s+2} + f_2 \alpha^{2(-s+2)} +$$
$$+ \ldots + f_{2t} \alpha^{2t(-s+2)}$$
$$\ldots$$
$$f(\alpha) = f_0 + f_1 \alpha + f_2 \alpha^2 + \ldots + f_{2t} \alpha^{2t}$$
$$f(1) = f_0 + f_1 + f_2 + \ldots + f_{2t}$$

As can be seen, the root search is exhaustive. However, since words only have 128 symbols there is no need to test the elements $\alpha^{-254}, \alpha^{-253}, \ldots, \alpha^{-128}$ of $GF(2^8)$ (since there can be neither errors nor erasures in those positions).

The circuit has been implemented in Figure 7. The elements symbolized as $\alpha, \alpha^2, \ldots, \alpha^4$ in the picture represent multipliers that multiply its content by $\alpha, \alpha^2, \ldots, \alpha^4$ respectively. They must be initialized to 1 at the beginning of the process. These multipliers can be easily implemented with an eight bit register and wired logic [10]. The full process spends 128 clock cycles.

7.4 Forney algorithm.

This is the last decoding phase. It consist in applying the Forney algorithm in those symbols which present an error or erasure. This happens when the $\Lambda(x)$ polynomial has a root in $\alpha^{-(\text{error position})}$.

The Forney algorithm is :

$$e_{i_l} = \frac{-\Omega(\alpha^{-i_l})\alpha^{-(b_0-1)i_l}}{\Lambda'(\alpha^{-i_l})} \qquad l = 1, \ldots, t \quad (13)$$

The circuit employed appears in Figure 8.

The value of the root α^{-i_l} is exponentiated to the power of $b_0 - 1$ in the *exponentiator* (which is implemented by means of a look-up table) and it is introduced in the $\exp\_f$ register. Besides,the inverse

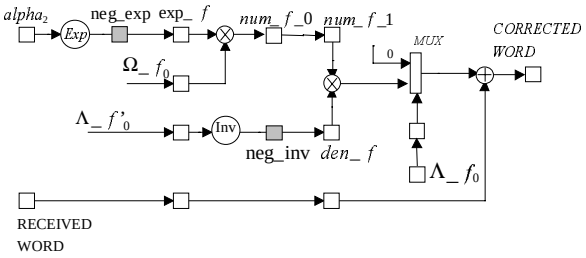

Figure 8— Forney algorithm.

of $\Lambda'(\alpha^{-i_l})$ is computed and introduced first in register *neg_inv* and afterwards in register den_f. Finally, it is decided if the symbol must be corrected depending on the value of the Λ_f_0 register. This register will content the results of evaluating $\Lambda(x)$ in the element α^{-i_l}.

This stage adds three more cycles of delay.

8 Circuit pipelining.

The decoding stages could be grouped in three phases:

- Phase 1 : syndrome computation. This operation lasts 128 clock cycles.
- Phase 2 : which would be composed of:
  1. Erasure polynomial computation: 4 cycles.
  2. Berlekamp-Massey algorithm: 16 cycles.
  3. Ω(x) and Λ'(x) computation : 4 cycles.

  Since the erasure computation is a part of the Berlekamp-Massey (initialization) its duration must not be taken into account. Thus this phase will spend 20 cycles.
- Phase 3: is composed of the polynomial evaluation and the frame correction by means of the Forney algorithm. They require 128+3=131 cycles.

Phase 1 and Phase 3 are the most expensive phases and have a similar duration. If they are implemented with different resources they will be allowed to overlap. If there were no overlapping the decoding process would require 128+20+131=279 cycles.

On the other hand when the pipeline is implemented, after a latency period of a decoding word (279 cycles), the decoding process would only take the length of Phase 3, which is 131 clock cycles. With this implementation a 128-symbol word can be decoded in 131 clock cycles. This implies that if we use a frequency of 20 MHz the throughput would be:

$$\frac{128\times 8}{131\times 50\cdot 10^{-9}} = 156\,\text{Mbps}$$

Thus the decoder can be used in hard-constrained applications such as video processing.

The proposed circuit has been modeled in Verilog HDL and has been synthesized with Synopsys tools into a 0,7 μm technology. The results are the following:

	Area (mm^2)	Equivalent gate number
Syndromes	0,678414	1785
Erasures	1,030693	2712
Berl.-Mass.	1,897561	4993
Pol. evaluation + Forney	2,817947	7415
Control	0,067955	179
TOTAL COUNT	6,492570	17084

Clock Frequency	20 Mhz
Throughput	156Mbps

9 Conclusions.

In this paper a RS decoder has been implemented. It presents an adequate performance for application in ATM networks. Future work should focus on the use of other decoder algorithms to improve the slowest decoding phases. Research on architectures suitable for parameter configuration would be also of great interest.

Acknowledgments.

We would like to thank María Luisa López Vallejo for her insightful comments and suggestions and her careful evaluation of this work.

References.

1. Yousef R. Shayan and Tho Le-Ngoc. "A cellular structure for a versatile Reed-Solomon decoder". IEEE Transactions on Computers, January 1997.
2. Keiichi Iwamura, Yasunori Dohi and Hideki Imai. "A design of Reed-Solomon decoder with systolic-array structure". IEEE Transactions on computers, January 1995.
3. Yousef R. Shayan, Tho Le-Ngoc and Vijay K. Bhargava. "A versatile Time-Domain Reed-Solomon Decoder" IEEE Journal on Selected Areas in Communications", October 1990.
4. Howard M. Saho and Irving S. Reed. "On the VLSI design of a pipeline Reed-Solomon decoder using systolic arrays". IEEE Transactions on computers, October 1988.
5. Howard M. Shao, T.K. Truong, Leslie J. Deutsch, Joseph H. Yuen and Irving S. Reed. "A VLSI design of a pipeline Reed-Solomon decoder". IEEE Transactions on computers, May 1985.
6. I. S. Hsu, T. K. Truong, L.J. Deutsch, and I. S. Reed, "A comparison of VLSI architecture of finite field multipliers using dual, normal or standard bases" , I.E.E.E. Transactions on Computers, June 1988.
7. P. A. Scott, S. E. Tarvares an L. E. Peppard, "A fast multiplier for $GF(2^m)$" , I.E.E.E. J. Select. Areas Commun., January 1986.
8. C.C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architecture for computing multiplications and inverses in $GF(2^m)$", I.E.E.E. Transactions on Computers, August 1985.
9. Kuang Yung Liu. "Architecture for VLSI Design of Reed-Solomon decoders", I.E.E.E. Transactions on computers, February 1984.
10. Richard E. Blahut. "Theory and practice of error control codes".Addison Wesley, 1984.
11. Antonio G. Lomeña. "Modelado, simulación, síntesis y fabricación de un codificador/ decodificador Reed-Solomon para control de errores". Master 's thesis, E.T.S.I.T., Universidad Politécnica de Madrid, 1998.