Implementation of a Hermitian Decoder IC in $0.35\mu m$ CMOS

Jonathan B. Ashbrook[1], Naresh R. Shanbhag[2], Ralf Koetter[2], and Richard E. Blahut[2]

[1]International Business Machines
Microelectronics Division
1000 River Rd., Essex Junction, VT 05452
ashbrook@us.ibm.com

[2]Coordinated Science Lab
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL 61801
shanbhag@uiuc.edu, [koetter,blahut]@csl.uiuc.edu

Abstract

This paper presents the first integrated circuit implementation of a Hermitian decoder thereby proving its practical viability. Hermitian codes provide much larger block lengths ($n = 4080$) compared to that of the popular Reed-Solomon (RS) codes ($n = 256$) over the same field ($GF(256)$). This translates to a coding gain of 0.6 dB for the same rate. However, Hermitian codes were deemed to be too complex to implement until the emergence of a recent algorithmic breakthrough which made the complexity of Hermitian decoders comparable to that of RS codes. Based on Koetter's decoding algorithm, the chip architecture consists of an array of sixteen interdependent Berlekamp-Massey algorithm (BMA) blocks. Thus, the same IC can be used for decoding RS codes as well. The decoder IC is designed in a $3.3V$, $0.35\,\mu$m, four-metal CMOS process and can correct up to $t = 60$ errors per block of $n = 4080$ words at a rate of 400 Mb/s. The IC prototype consumes 3.0 W with a 50 MHz clock.

I Introduction

Reed-Solomon (RS) codes have been the most popular block code since the mid-1970s and are found in most communication and magnetic storage systems. Extensive research has been performed in the optimization of decoding algorithms and architectures for RS codes. An inherent limitation of RS codes is that the block length n is constrained by the field over which the code is defined. In particular, $n = 2^{\ell}$ for the RS code defined over the field $GF(2^{\ell})$. Larger block lengths are desirable because it increases the code rate $r = k/n$, where k is the number of information symbols, for the same error correction capability $t = (n-k)/2$.

Hermitian codes provide much larger block lengths ($n = 4080$) compared to that of the popular RS codes ($n = 256$) over the same field ($GF(256)$). For the same rate (obtained by taking RS codes over $GF(2^{12})$), the Hermitian code provides a coding gain of up to 0.6 dB at a rate of 0.6 (see Fig. 1). Figure 1 also shows that the effectiveness of Hermitian codes improves as the code rate reduces. However, Hermitian codes were deemed to be too complex to implement until the recent emergence of a breakthrough in the form of Koetter's algorithm [1] which made the complexity of Hermitian decoders comparable to that of RS decoders. Hermitian codes require considerable memory and r times the basic computational complexity of RS codes. However, in order to achieve equivalent block sizes, RS codes require larger fields and thus longer symbols. Thus, the overall complexity for identical block sizes for the two types of codes scales at the same rate. RS [2] as well as Hermitian decoding require four major steps: 1.) syndrome calculation, 2.) error-locator polynomial computation, 3.) error-evaluator computation, and 4.) error-correction.

This work was supported by NSF grant CCR98-05080.

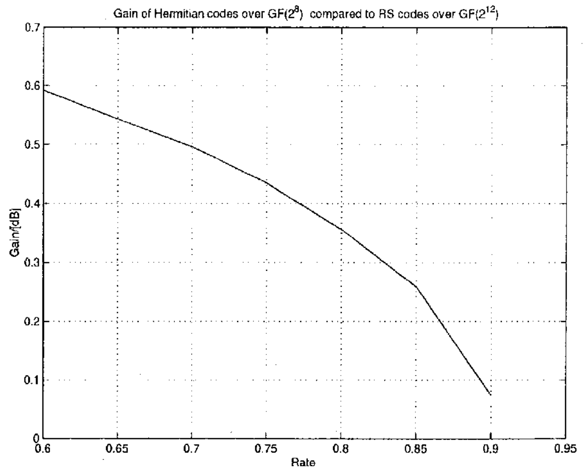

Fig. 1. Gain comparison of Hermitian and RS codes.

In RS decoding, steps 2 and 3 are often combined but these are consciously kept separate in case of Hermitian decoding for complexity reasons. Further, amongst all four steps, it is step 2 (computation of the error-locator polynomial) that is most complex and creates a throughput bottleneck. This research is concerned with an efficient implementation of step 2, which generates a polynomial that describes the locations of errors in the received block.

RS code symbols over $GF(256)$ can be obtained by evaluating the data polynomial of degree $k-1$ (also defined over $GF(256)$) over all $n = 256$ points in $GF(256)$, i.e., points on a line. Hermitian codes, on the other hand, are obtained by evaluating the data polynomial over all points on a curve (Hermitian curve) in $GF(256)$. Thus, Hermitian codes can be visualized as a kind of punctured two-dimensional RS codes. However, due to complexity reasons, implementation of decoders for Hermitian codes has not been attempted before. With dramatic reductions in decoding complexity afforded by [1], it has now become possible to implement such a decoder. In fact, the Koetter algorithm [1] is based upon the well-known Berlekamp-Massey Algorithm (BMA) used for decoding RS codes. The goal of research described in this paper is to prove that Hermitian decoders can be implemented in present day technology with reasonable cost. The interested reader is referred to [3] for details of the algorithm, architecture, and circuit.

The paper is organized as follows: sec. II consists of a comparative description of the Koetter and BMA computations. Section III describes the overall chip architecture while sec. IV discusses circuit implementation issues.

II The Koetter Algorithm

As mentioned earlier, the Hermitian code is based on 8-bit finite field elements. Thus, the field $GF(256)$ defines the potential symbols of the Hermitian code. Whereas an RS code has a block size of 256, the Hermitian code in $GF(256)$ uses a block size of 4080 where $\sqrt{256}$ equals the number of BMA blocks. This decoder can be programmed to correct up to $t = 60$ errors in each block of $n = 4080$ symbols referred to as the *codeword*. An important parameter in the code is derived as $m = 2t + 2g - 1$ where t is the number of errors to correct and the *genus* $g = 120$ is a constant dependent on the Hermitian code's base curve. The number of syndromes and the number of iterations are both functions of m.

The BMA is a popular algorithm for finding the error-locator polynomial of a RS codeword [2]. The Koetter algorithm [1] determines the error-locator polynomial for Hermitian codes and in fact employs sixteen parallel BMA blocks that operate in lock-step. Each of these sixteen blocks maintains one of the code's sixteen error-locator polynomials which are computed iteratively over $m+g+1$ iterations.

Each iteration of the error-locator algorithm breaks down into three main steps: 1.) compute the discrepancy Δ, 2.) calculate the missing syndrome, and 3.) update the error-locator $\sigma(z)$ and the scratch $\lambda(z)$ polynomials according to (1). The second step occurs only in the final 120 iterations as shown in Fig.2.

For the sake of comparison, equations (1) and (2) show the update operations for the Hermitian and RS decoders, respectively. The variable j in (1) represents the shifting of

```
**for** $i = 0$ to $r - 1$ **do**
   vote[i] = 0
   **for** $k = 0$ to $r - 1$ **do**
      **if** $\Delta^{(i)} = \Delta^{(k)}$ and $a_\sigma^{(k)} - a_\lambda^{(j^k)} > 0$ **then**
         vote[i] = vote[i] + $a_\sigma^{(k)} - a_\lambda^{(j^k)}$
      **end if**
   **end for**
**end for**
maxidx = i  s.t.  vote[i] > vote[k] $\forall$ k
$S_{new} = \Delta^{(\mathrm{maxidx})}$
$\Delta^{(k)} = \Delta^{(k)} \oplus S_{new} \ \forall\, k$
```

Fig. 2. Majority voting algorithm.

the $\lambda(z)$ polynomial from block to block in the algorithm.

$$\begin{bmatrix} \sigma^{(i)}(z) \\ \lambda^{j^{(i)}}(z) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta^{(i)} \\ \delta^{(i)}(\Delta^{(i)})^{-1}z & (1-\delta^{(i)})z \end{bmatrix} \begin{bmatrix} \sigma^{(i)}(z) \\ \lambda^{j^{(i)}}(z) \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} \sigma(z) \\ \lambda(z) \end{bmatrix} = \begin{bmatrix} 1 & \Delta z \\ \delta(\Delta^{-1}) & (1-\delta)z \end{bmatrix} \begin{bmatrix} \sigma(z) \\ \lambda(z) \end{bmatrix} \tag{2}$$

III The Decoder Architecture

The Hermitian decoder employs a modified version of the BMA block for computing the error-locator polynomials in RS codes. For this decoder chip, the symbol serial BMA implementation [4] is the basis for the modified BMA block. The sixteen modified BMA blocks are linked together such that the $\lambda(z)$ and a_λ move from one block to the next after each iteration. The syndrome register feeds the same syndromes to all sixteen blocks. Each block is linked to the majority voting unit by a global bus.

The architecture iteratively determines sixteen separate error-locator polynomials in parallel. The algorithm requires $m + g + 1$ iterations through a loop. As each iteration requires a new syndrome, the algorithm itself requires $m + g + 1$ syndromes per codeword. However, assuming a pipelined decoder where each stage comprises one of the previously mentioned four decoding stages, the syndrome computation stage would not be able to generate syndromes fast enough to keep up with the other pipeline stages. Consequently, we compute only $m + 1$ syndromes in the syndrome computation block and compute the remaining g syndromes while computing the error-locator polynomial [1].

A. *Modified BMA block*

The modified BMA architecture is the key to the Koetter algorithm. Fig. 3 shows the data-flow for this block. The shaded blocks are the extra hardware in the modified BMA data path. The delay in dashed lines is present for only RS implementations. The first two steps of the algorithm dictate two additions to the architecture: an extra syndrome input, and the multiplexer/adder to enable the use of the majority vote. The extra syndrome input allows a different syndrome to be used in certain cycles. The two

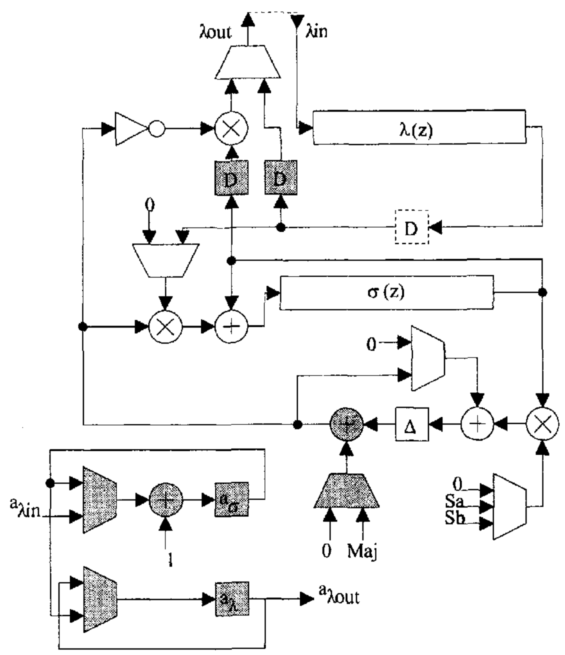

Fig. 3. Modified Berlekamp-Massey architecture.

signed integers a_σ and a_λ require two update and storage hardware. The circuitry for these integers is not normally required for RS decoding. RS decoding typically employs the one integer L. Finally, the $\lambda(z)$ polynomial and a_λ integer are shifted block to block during the polynomial update stage of each iteration.

Once the $m+g+1$ iterations are complete, the $\sigma(z)$ polynomial in each block holds a valid error-locator polynomial. However, it has been shown [1] that at least one of these polynomials will have a degree of not more than $t+g$. Only one of the sixteen polynomials is required in the succeeding decoding stages. Thus, the polynomial of smallest degree is found and shifted out to the next stage. The a_σ values from all sixteen blocks can be compared to find the $\sigma(z)$ polynomial of minimum degree. Since the error-locator computation is the focus of this work, the chip's output is this chosen $\sigma(z)$ polynomial.

B. The Majority Voting Unit

Large amounts of data need to be transferred from each block to the majority voting unit. For example, each block must transmit its 8-bit discrepancy value, as well as variables a_σ and a_λ, which are both 10-bit signed integers. In order to minimize routing constraints, a global bus is used for such transfers and all off-chip communication. Additionally, this information is transferred between the block and the majority voting unit one block at a time. The majority voting block is synthesized as a series of rotating shift registers. Computation is only done on the first and second positions from the sixteen possible for each.

$$[\sigma_0\ g_1 \cdots g_{15}\ |\ \sigma_1\ \sigma_2\ g_1 \cdots g_{14}\ |\ \cdots\ |\ \sigma_{120} \cdots \sigma_{135}]$$

Fig. 4. Gap sequence structure in a Hermitian polynomial. g_i represents a gap, or a coefficient that is permanently zero. σ_i is a nontrivial coefficient.

C. Gap sequence

The *gap sequence property* of Hermitian codes considerably reduces the amount of processing required on each iteration of the algorithm. This property, illustrated in Fig.4, states that certain coefficients in known positions of any polynomial will always be a *gap*, i.e., a coefficient that is permanently zero. Specifically, of the first $r^2 = 256$ coefficients, exactly $g = 120$ will always be zero. This property allows for two circuit-level optimizations. The first is the removal of 120 symbols of storage from the syndrome register. If the controller is aware of the gap sequence, it does not need to compute iterations corresponding to the gap sequence. A virtual addressing system has been employed in order to reduce the syndrome register memory. This block maps virtual addresses from the global controller into physical addresses in the syndrome register. A greater impact of the gap sequence is in the number of cycles required to complete the algorithm. After the first 256 iterations, the gap sequence allows for 120 cycles to be saved per iteration by skipping the gaps. This optimization saves 27000 clock cycles, or approximately 10% when correcting 60 errors.

D. Control and Pipelining

The algorithm contains some variables that are common to all sixteen modified BMA blocks, and others that are specific to each block. Control of the architecture is partitioned to represent these divisions. Each modified BMA block has a local controller to track its state for use during the discrepancy calculation and the polynomial update. The global controller is a finite state machine that keeps the local controllers in synch via the use of five state bits. This partitioning between local and global control reduces the complexity of the controller and minimizes wire routing between the global and local controllers.

Due to its complexity and distribution, the control logic requires special construction in order to meet the chip cycle time. The chip is partitioned into three pipeline stages. The global controller makes up the first stage, the local controller the second, and the actual datapath makes up the final pipeline stage. This results in the datapath running two clock cycles behind the global controller. Pipeline stalls are guaranteed not to occur as the global controller receives no direct feedback from the datapath.

E. Test Hardware

Several blocks were added to the design in order to facilitate testing. The added hardware allows read/write access to any memory location within the registers, as well as read access to many other signals throughout the chip. The test hardware is designed to be nondestructive, allowing decoding to complete correctly after a test operation. Although full scan-enabled test hardware would have been most desirable, time constraints dictated an ad hoc approach.

IV Circuit Implementation

The chip was designed using a synthesis-based standard cell design methodology. A simple standard cell library was developed and characterized in a 0.35 μm CMOS process. The library contained basic and complex logic gates with single drive capability. Inverters and buffers with multiple drive levels were added to enable efficient driving of various sized loads. All required library formats for the cells were created for use with Synopsys's Design Compiler for logic synthesis and static timing analysis, Cadence's Silicon Ensemble for cell placement and routing, and Cadence's Virtuoso layout editor for final layout preparation.

In addition to the standard cell library, two custom dual-port SRAM blocks were designed using a typical full-custom design methodology described in [5]. The SRAM base cell was reused from previous research [6]. These SRAM blocks were designed to be used as area efficient implementations of the polynomial registers. One block serves as the syndrome register, the other block serves as both the $\sigma(z)$ and $\lambda(z)$ registers in each BMA block. In addition to acting as area-efficient registers, the SRAM implementations allow for easy access to all coefficients in order to facilitate testing and characterization. Black box descriptions of these SRAM blocks were created for use in the standard cell methodology.

Nearly all of the blocks in the design were implemented by synthesizing VHDL behavioral descriptions. This approach was taken in order to minimize design time and make algorithmic adjustments easier to implement. The top-level chip description and the modified BMA blocks were written in structural VHDL in order to ensure connectivity of lower-level blocks. The blocks were then synthesized at various levels of the hierarchy in order to obtain the final logic. Functional verification was performed before and after synthesis in order to ensure correct logic synthesis from the behavioral descriptions.

After logic synthesis, Silicon Ensemble was used to place and route the standard cells. An iterative methodology was employed to obtain timing closure between synthesis and physical design.

The Hermitian decoder IC was fabricated in a 3.3V, 0.35 μm 4-metal CMOS process. Table 1 shows some of the critical details of the final design. Measurements on the test bench indicate that the decoder IC consumes 3.0W with a 50 MHz clock.

Table 1. Error-locator computer chip details.

Chip Dimensions	6.30 mm × 6.30 mm
Core Dimensions	6.69 mm × 6.69 mm
Total Transistors	1 242 126
Logic Transistors	334 966
Memory Transistors	907 160
Clock Cycle	20 ns
Signal Pins	16
Supply Pins	32

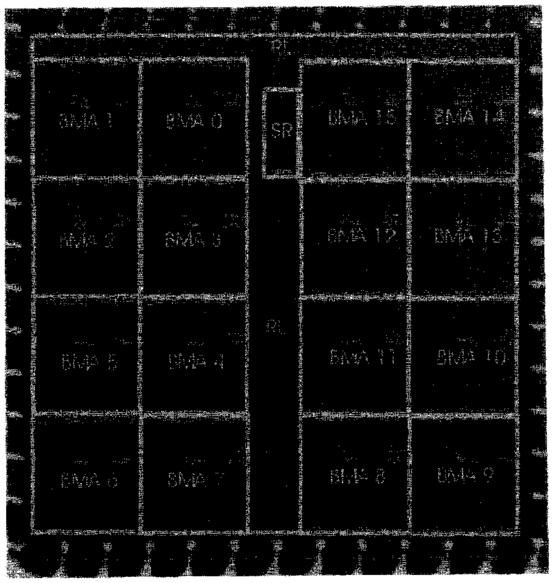

Fig. 5. Microphotograph of the final die showing the modified BMA blocks (BMA), the syndrome register (SR), and the global random logic (RL).

V Conclusion

This paper has described the first IC implementation of a Hermitian decoder thereby bringing this powerful generalization of RS codes into the practical domain. Further research is necessary to reduce the power and improve speed through the design of algorithmically pipelined Hermitian decoder architectures.

References

[1] R. Koetter, "A fast parallel implementation of a Berlekamp-Massey algorithm for algebraic-geometric codes," *IEEE Transactions on Information Theory*, 1998.

[2] V. Bhargava and S. Wicker, *Reed-Solomon Codes and Their Applications*. IEEE Press, 1994.

[3] J. Ashbrook, "Design of an error locator computer for a hermitian decoder," M.S. thesis, University of Illinois, Urbana, IL, 2000.

[4] R. E. Blahut, *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1983.

[5] J. Tschanz, "Design of a 51.84-mb/s very high speed digital subscriber loop reconfigurable receiver core," M.S. thesis, University of Illinois, Urbana, IL, 1999.

[6] M. Anders, "Adaptive distributed arithmetic filtering in the presence of deep submicron noise," M.S. thesis, University of Illinois, Urbana, IL, 1999.