Soft IP Compiler for a Reed-Solomon Decoder

Jong Kang Park and Jong Tae Kim

In this paper, we present a soft IP compiler for the Reed-Solomon decoder that generates a fully synthesizable VHDL core exploiting characteristic parameters and design constraints that we newly classify for the soft IP. It produces a structural design with an estimable regular architecture based on a finite state machine with a datapath (FSMD). Since characteristic parameters provide different design points on the design space, using one of two simple procedures called the constructive search with area increment (CSAI) and constructive search with speed decrement (CSSD) for design space exploration, the core compiler makes it possible for an IP user to create the Reed-Solomon decoder with appropriate sub-architectures without synthesizing many models. Experimental results show that the IP compiler can apply to several industry standards.

Keywords: Soft IP compiler, Reed-Solomon decoder.

I. Introduction

These days, as designs of integrated circuits increase in density and complexity, an intellectual property (IP), often referred to as a core or macro, plays an important role in the system-on-a-chip design for improved design productivity, and its concept is very close to component-based development in software engineering [1]. By definition, a hard IP is one that is delivered to the integrator as a GDSII file, and a soft IP is one that is delivered to the integrator as synthesizable register transfer level (RTL) code [2]. A soft IP is a technology-independent design that has more flexibility than a hard one, even though it is restricted to optimal performance targeted to a specific silicon process and IP protection.

Commonly, soft IPs are fully parameterized with the proper degree of the requirements in their application area. When an IP user selects the exact parameter set within the system requirement, the generation process merely instantiates the target model as modifying statements, such as "generic" and "constant" in VHDL, from the baseline design. Increasing functionality and reusability of a parameterized soft macro, however, especially extending the parameter set in the IP core, often makes it difficult for design refinements to meet design constraints managed by a checklist [3]. Since system designers may not know the precise characteristics of an IP block that they place into a silicon wafer, when a parameterized IP has extended parameters for more flexibility, it is never an easy problem to search for an optimal parameter set within the design space.

In this paper, we introduce a soft IP compiler for the Reed-Solomon (RS) decoder with several efficient sub-block architectures that cover a wide range of design space and application-specific parameters. The RS decoder has been widely used in today's numerous communication systems for forward error correction, for example, in compact disk players, digital audio tapes, hard disks, digital versatile disks, digital

Manuscript received Jan. 15, 2003; revised July 1, 2003.

This work is supported by grant No. R05-2001-000-00884-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

Jong Tae Kim is the corresponding author. Jong Kang Park (email: candy@ece.skku.ac.kr) and Jong Tae Kim (phone: +82 31 290 7130, email: jtkim@skku.edu) are with School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea.

video broadcasting, digital satellite communication, etc. Thus, the core compiler can support a broad scope of these applications through full parameterization of RS code specification. In addition, since past research has proposed several efficient algorithms and sub-architectures for the RS decoder, we can exploit them as parameters for an IP compiler for increasing precision and efficiency of the area and speed in generated IP models. To accelerate the search for the best solution, instead of doing manual design refinement using a checklist from a user guideline, we automated it as design space exploration with two procedures called the constructive search with area increment (CSAI) and the constructive search with speed decrement (CSSD), including area and speed estimation using a simple synthetic library. For a compiling directive, the CSAI and CSSD perform a selection of parameter sets for speed-optimum and area-optimum, respectively. In addition, the core compiler can provide a testbench that is fully parameterized and has an automatic validation process for generated sub-blocks.

This paper is organized as follows. In section II, we surveyed recent work on IPs of RS decoders that have been commercially distributed or researched. Section III briefly describes the structural and algorithmic characteristics of the target architecture for the core compiler. In section IV, several parameters used in RS compilers are introduced. Section V explains the main procedure of our compiler including the design-space exploration mechanism. Section VI reports our experimental results for commercially-used specifications. Finally, the conclusion is presented in section VII.

II. Related Work

Many studies have focused on RS decoders as IPs. Some of them, such as [4], [5], have a fixed RS code specification and cannot be integrated in other systems with different specifications. Other systems, like common IPs, do not have the form of an IP generator, so the customer must provide detailed information in advance when writing orders to the provider [6], [7]. Basically, these IPs provide all possible parameters for high reusability, such as a primitive polynomial of the Galois field (GF), a symbol length corresponding to the GF, a generator polynomial used in data encoding, data block size, parity size, and erasure support. In some special cases, the needed throughput and area of the IP is required in ordering information for more efficiency in a specific application. However, this exchange of formalities is a disadvantage when the component specification changes for design refinements or other necessary procedures. Clearly, it will increase the design time of the total project cycle.

Smith et al. proposed a design automation method for Reed-Solomon codecs using VHDL [8]. This parameterized RS decoder also has parameters for the code specification for common IPs. Since its algorithm in the key equation solver is fixed with the Berlekamp-Massey (BM) algorithm performing on a frequency domain including bit-serial GF multipliers, it cannot create an efficient architecture for various applications every time, because of the specific properties of the BM algorithm in terms of cost and speed.

The Altera RS compiler [9] supports many RS codes including a feature conforming to the Consultative Committee for Space Data Systems' (CCSDS) recommendations for telemetry channel coding. Because of the advantages of a generator, a user can create the design for any specification in a short time, if he/she needs to make changes in a design plan. Moreover, it not only satisfies full functionality for RS decoding, but it also provides an additional parameter, called "keysize," which creates two models with different characteristics. In this way, a user can always obtain two styles of architectures for a required specification. However, even if an IP user can estimate the throughput of the design for a given specification on the generation program, since there is no mechanism to estimate the number of cells counted on an Altera field programmable gate array (FPGA), the user can synthesize many designs until a generated design satisfies the system requirements.

The Xilinx RS decoder v.2.0 [10] is another good example of an IP generator, similar to Altera's. This core compiler also gives a parameter set to a user through a core graphic user interface (GUI) and supports CCSDS, shortened RS codes, and several interface types. However, its sub-architectures are fixed, so users can support only one model for a given specification, regardless of the system requirements.

III. Target Architectures

In general, simple and regular structures are inherently easier to design, code, verify, and synthesize than more complex designs. An IP, as a reusable design integrated on any system, should be kept as simple as possible and still meet its functionality and performance goals [2]. Hence, the observance of the register transfer level (RTL) coding guideline, such as in [2] and [11], especially for soft IPs, makes them more reliable and robust. Additionally, since basic very large scale integration (VLSI) architectures used in the IP compiler should be predictable for area and speed (throughput), many subdivisions of the RS decoder into subordinate blocks help these estimations, and is relatively clear at the level of a leaf cell. Although we chose primitive elements in the synthetic library based on two-input logic gates and we discounted what we expected in the logic minimizations and technology mapping

by a synthesis tool like the Synopsys Design Compiler, we can ignore the gap for performance degradation and the extra number of cells, considering the principal operations of the RS decoder are only GF addition and multiplication. Next, we discuss selected sub-architectures constructed as a basis of this structured RTL coding.

1. Architecture Template

The finite state machine with a datapath (FSMD) model is a good candidate for regularity in complex VLSI designs. Each sub-block of the RS decoder shown in Fig. 1, which requires at least two states, was constructed on the FSMD model. All datapaths generate single-bit control signals fed into an FSM, and the FSM also transmits single-bit control signals to datapaths (Fig. 2) [12]. As today's many synthesis tools have FSM synthesizers to encode FSM tables as memory elements, we can easily get the baseline design for each FSM through them, because all controls must be constant for any specification. Moreover, all FSMs for sub-architectures must support hand-shaking controls with other sub-blocks. The significance of localized control in the architecture template allows the flexibility of adding sub-blocks into the parameter set. For example, we can easily add a key equation solver of a different characteristic to the design space.

2. VLSI Implementation for the Syndrome Calculator

Traditionally, the VLSI architecture of the syndrome calculator is constructed by applying Horner's rule [13]. The *i*-th syndrome coefficient is given by

$$\begin{aligned} S_i &= r_0(\alpha^i)^0 + \Lambda + r_{n-2}(\alpha^i)^{n-2} + r_{n-1}(\alpha^i)^{n-1} \\ &= (\Lambda\ ((r_{n-1}\cdot\alpha^i + r_{n-2})\cdot\alpha^i + r_{n-3})\cdot\alpha^i + \Lambda)\cdot\alpha^i + r_0. \end{aligned} \quad (1)$$

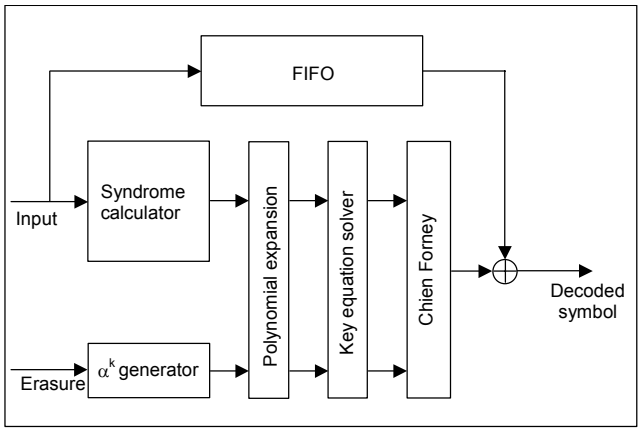

Fig. 1. A block diagram of the Reed-Solomon decoder.

We define the *constructive syndrome calculator* (*CSC*) as in the upper line of (1), and the *recursive syndrome calculator* (*RSC*) as in the second line of (1). In Fig. 3, we propose the new architecture for the CSC. The circled operators are variable GF multipliers that have two variable operands. The GF multiplier of the conventional RSC that was used in [13] can be implemented by the constant GF multiplier, since its one operand is a constant of α. This greatly reduces the number of gates in the RSC. However, because its computation performs recursively starting from the last symbols of the data block as r_{n-1} in (7), the serially-inputted symbols in the data block must be reversed before the syndrome calculation or after the errata evaluation in the last stage of the data decoding. No matter what methods we apply for the data reverse and while we should maintain the pipeline, we must insert an additional pipeline stage at the minimum that increases the complexity of the RS decoder. One efficient way to do the data reverse while using only one pipeline stage is not to arrange symbols for reverse order directly, but to store

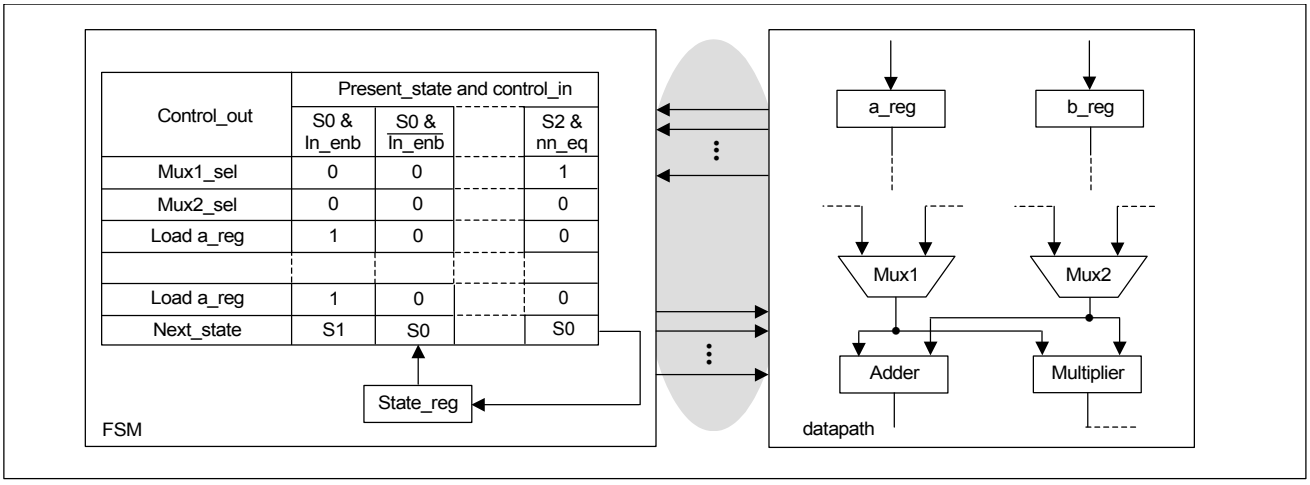

Fig. 2. Baseline template as FSMD.

locations and values of errors during the errata evaluation. Practically, corrupted symbols are corrected by a new sub-block, called *the corrector*.

Table 1 summarizes the characteristics of the VLSI architectures for the CSC and the RSC, with n and k denoting the single data block size and information block size, respectively. The CSC itself requires more logic gates than the RSC, and the total pipeline stages of the RS decoder is smaller than when the RSC is used alone, and it doesn't require the corrector. When a large data block has a small parity size, the CSC is the best choice for the design space.

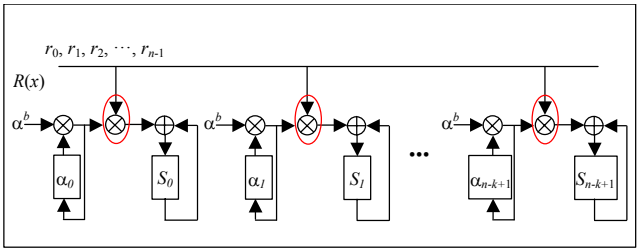

Fig. 3. Proposed CSC architecture.

Table 1. Comparisons between CSC and RSC.

Architecture	Regs.	Constant multipliers	Variable multipliers	FIFO stages	
				Error only	Erasure support
RSC	n–k	n–k	0	4	5
CSC	2(n–k)	n–k	n–k	3	4

3. VLSI Architectures for the Key Equation Solver

Traditionally, to solve a key equation, there have been three algorithms: the BM, Euclidean, and continued fraction.

The efficient modified Euclidean algorithm (MEA) and its VLSI architecture were proposed in [14]. The MEA block consists of only four GF multipliers without GF division, which causes large hardware overhead and many computing cycles. From [15], Lee et al. introduced the Euclidean cell (EC), which reduced the number of computing cycles, which affects the pipeline architecture in the following subsection without much hardware overhead. This is accomplished through proper algorithmic initialization and insertion of register files instead of shift registers as used in [14]. Thus, we selected the MEA architectures introduced in [15] for both error-only decoding and erasure correction.

The inversionless Berlekamp-Massey algorithm (iBMA) [16] also calculates the key equation without GF division. Even though it has a larger number of GF variable multipliers than the MEA, its number of iterations over the algorithm is much smaller than the MEA. Hence, we added it to the architectural parameter set since it could suggest a different design point in contrast to the MEA. Moreover, [17] greatly reduced the critical path delay of the iBMA by an algorithmic modification. We refer to it as the reformulation inversionless Berlekamp-Massey algorithm (RiBMA) as in [17] and also added it to the parameter set.

For an erasure correction in the iBMA or RiBMA, using the algorithmic modification in [18], we can reconstruct Fig. 1 as Fig. 4 or Fig. 5, where we remove one pipeline stage and expansion block for the syndrome calculation. Table 2 shows the summarized results for the architectural characteristics of the MEA, iBMA and RiBMA with the worst case, where t denotes the correctable symbol size in a single data block; T_{mult} and T_{add} are the operation time for a GF multiplier and GF adder, respectively; and T_{tri_buf} and T_{mux} in the MEA are the accessing time for the register files.

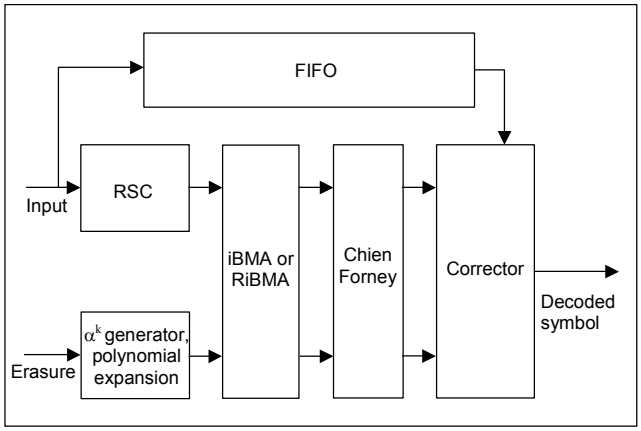

Fig. 4. Proposed RS decoder for the iBMA and RiBMA using the RSC.

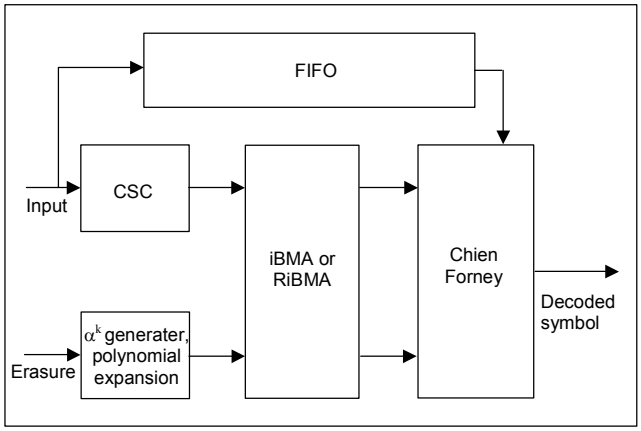

Fig. 5. Proposed RS decoder for the iBMA and RiBMA using the CSC.

Table 2. Comparisons between the MEA, iBMA, and RiBMA.

t=(n–k)/2					
Solver	Mode	Regs.	Multi-pliers	Iteration, Stages with RSC	Critical path delay
MEA	Error	8t	4	$3t^2$–2t+2, 4	$T_{tri\_buf}+T_{mult}+T_{add}+T_{mux}$
	Erasure	8(t+1)	4	$3t^2$–2t+2, 5	
iBMA	Error	4t	3t+3	3t+1, 4	$2T_{mult}+(1+\log_2(t+1))\cdot T_{add}$
	Erasure	4t	4t+3	5t+1, 4	
RiBMA	Error	6t	6t+2	2t+1, 4	$T_{mult}+T_{add}$
	Erasure	10t	8t+2	4t+1, 4	

4. Pipeline Strategies for the Reed-Solomon Decoder

Lee et al. introduced the multiplexed cell architecture for full pipelining [14]. When the bottleneck of computation cycles over the total sub-blocks is N+2, its one extra cycle denotes the loading cycles for parallel data, such as syndrome polynomials and error locator polynomials inserted into each pipeline stage, and the remaining cycle is needed in the Chien and Forney block, presented in [14], for partitioning the critical paths of these blocks. We chose a design with a fixed critical path, which is the critical path delay of the key equation solver. There will be no other sub-block with a larger critical path delay than that in most technologies. Next, we define the number of cells for the key equation solver as

$$N_{cell} = \left\lceil \frac{C_{chien}}{C_{key\_eq}} \right\rceil = \left\lceil \frac{C_{synd\_cal}}{C_{key\_eq}} \right\rceil, \quad (2)$$

where C_{chien}, C_{synd_cal}, and C_{key_eq} are computation cycles of the Chien and Forney block, the syndrome calculator, and the key equation solver, respectively.

In most cases, the multiplexed cell method for pipelining offers an efficient architecture with a reduced area. However, for small differences in computing cycles for each sub-block, it is clear that adding shift registers in front of the syndrome calculator is more efficient than adding an extra cell for the key equation solver. All the symbols, input-valid signal, and erasure flag are delayed by the shift registers. We combined two sub-architectures for supporting an efficient pipeline architecture for the RS decoder.

IV. Parameterization for the Soft IP

Junchao et al. classified parameters for a soft IP in parameterized IP core designs into two types: static parameters and dynamic parameters. A static parameter is used to instantiate the IP model by configuring HDL statements. A dynamic parameter exists as an input pin of the IP module and gives operation modes on-the-fly. Since applied parameters in the RS compiler are fixed for the applications and system requirements during the operation, we used only a static parameter set in the core design. At this time, we define those parameters for a new classification as follows.

- *Functional parameter* indicates the core functionality with a fixed value in the specific application, such as a code specification of the block code.
- *Characteristic parameter* is an algorithmic/architectural parameter that gives a different area/speed of the design on the same functionality.
- *Design constraints* decide boundaries on the design space, combine with functional parameters, and help the core compiler search for more efficient designs over the design space.
- *Simulation-management parameter* denotes a parameter for the testbench, which is also fully reusable as in [20]. Due to the importance of verifying the IP models, generated testbenches validate the design by checking internal registers with a high code coverage.

All parameters in the RS compiler are listed in Table 3. As described in section II, many soft IPs for the RS decoder have high reusability related to the code specification. In addition to the functional parameter set, we added the target architectures that we introduced in the previous section to the characteristic parameters. To explore the design space, after the compiler estimates the speed and number of cells of the designs based on the simple synthetic library, which consists of the technology-dependent components shown in Table 4, it picks out the best design within the boundary made by A_{max} or S_{min} on the design space. The stream size (the number of symbols) for continuous decoding is also added to the design constraints. It estimates the speed of the designs to conform to the system requirements by

$$S_{thruput} = \frac{d \times m}{\{(n \times m + 1) \times N_{pipe} + N_{shr}\} \times T_{clk} + (d-1) \times T_{clk} \times PL}, \quad (3)$$

where N_{pipe} denotes the number of actual pipeline stages, T_{clk} denotes the clock period time computed by the critical path

Table 3. Classified parameters in the RS compiler.

Category	Parameter name	Description
Functional parameters	n	Data block size
	k	Information size
	b	Generator start
	m	Symbol size
	p	Primitive polynomial
	E	Erasure support
Characteristic parameters	RSC, CSC	Syndrome calculator
	MEA, iBMA RiBMA	Key equation solver
	N_{cell}	# of cells for pipelining
	N_{shr}	# of shift registers for pipelining
Design constraints	d	Decodable stream size as unit of a symbol
	C_{pl}	Pipeline latency (input latency)
	A_{max}	Maximum allowable area
	S_{min}	Minimum required throughput
Simulation-management parameters	ClkStartVal	Start value of clock
	ClkDelay	Delay time of clock
	ClkWidth	Time of keeping logic '1' in clock
	ClkPeriod	Period of clock
	ClkStop	Stop time of clock
	repeat	Repeat time of input stimuli
	CheckTimeInClk	Delay time of input stimuli in clock
	StartCycle	Start cycle of input stimuli

delay of the design, and PL indicates the pipeline latency of the design, while C_{pl} is a design constraint for the user requirement. Initialization cycles for the RS decoder considering pipeline stages affect the throughput of the decoder on a small d in (3). The parameters for the simulation-management used in the RS compiler are mainly divided into two parts. One is a clock generator, and the other is an input stimuli generator as in [20]. It is possible to generate the desired testbench for any case. Basically, since input vectors of the test vehicle are randomly generated by the compiler, the RS compiler should support all the functions for RS encoding and decoding. The representation of each parameter will be used in the rest of the paper.

V. The Core Compiler Design

Figure 6 shows the block diagram of the primary functions

Table 4. Leaf cells of the synthetic library in the RS compiler.

Component name	Description
DFF_r	D-filpflop with reset pin
DFF_{rl}	D-flipflop with reset and load pin
MUX	1-bit 2:1 multiplexer
INV	inverter
NOR	2-input NOR
OR	2-input OR
NAND	2-input NAND
AND	2-input AND
XOR	2-input exclusive-OR
XNOR	2-input exclusive-NOR
TRI_BUF	three-state buffer
HA	half adder
FIFO	indicates FIFO RAM as a unit of a bit
ROM	indicates ROM as a unit of a bit

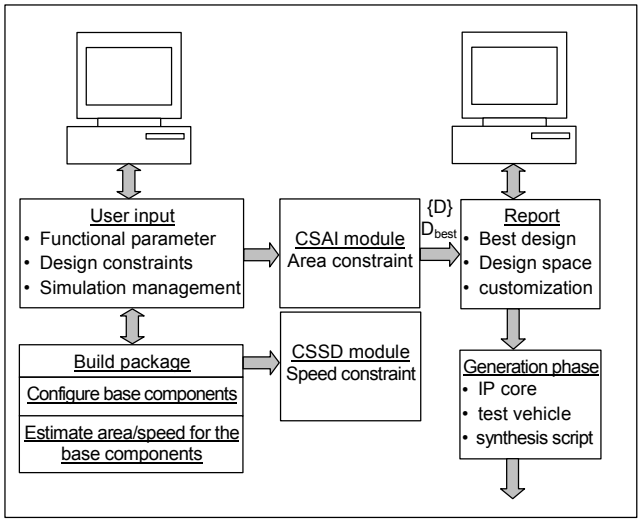

Fig. 6. Generation phases for RS compiler.

in the core compiler. If a user inputs functional parameters and design constraints with the synthetic library targeted to the technology process, the compiler will build the package and several base components, such as the register files, ring counters, binary counters, and muxes for symbols. It also estimates the number of cells and the propagation delay for each base component simultaneously. In our compiler, for simplicity, the interconnection delay and extrinsic propagation delay arising from fan-out loads are not considered. The detailed delay model can be used for accurate estimation of the critical path delay.

Let CP={cp_0,cp_1,cp_2,cp_3,…} be the characteristic design

point represented by a positive integer (i.e., cp_0=0 indicates the CSC architecture). Let BP={cp_0,cp_1,cp_2,cp_3,…} be the boundary where the compiler doesn't need to calculate the area and speed on the iteration for the CP, when the value of each cp is initially mapped to the ordered architecture by the area or speed. Thus, if the CP={0,1,1,2} is proven to be out of range in the design space for the cell size, the BP is set to {0,1,1,2}. The computation on the iteration for CP={0,2,2,2} is then unnecessary, because all elements of CP are not less than that of BP. It helps the core compiler not to explore the design space exhaustively.

At this point, the procedure called the constructive search with area increment (CSAI) is executed (Fig. 7) when the user gives an area constraint. The variable BC is a structure for storing information about the estimated gate count and critical path delay in the preprocessing stage of Fig. 6. The characteristic design point as represented by CP is described as the pair of (syndrome calculator, key equation solver, N_{cell}, N_{shr}) indicated by an integer. First, the algorithm initializes the order of the area for each element based on the base components in the BC structure. Basically, the algorithm, started with CP={0,0,1,1}, performs many iterations by increasing the low degree of CP on four "for" statements. However, by the function of update_bound() and check_bound(), many design points are removed on account of the diminishing boundaries as explained above. The function of get_max_cell(), calc_pl(), area_of(), and speed_of() are the maximum number of key equation cells for a given C_{pl} and characteristic parameters, the calculation function for a pipeline latency, the calculation of the number of cells of the design, and estimating the throughput of a design, respectively. The best design point D_{best} is determined by taking the fastest design within the area constraint.

Similarly, if the user inputs speed constraint S_{min}, another procedure, the constructive search with speed decrement (CSSD), searches for the best design with an initialization order

```
Procedure CSAI
Input : BC,n,k,m,E,d,Cpl,Amax
Return : {D}, Dbest
{D} = Ø;
Smax = 0;
CP = { cp0,cp1,cp2,cp3 };
initialize_order(BC,n,k,m,d,PL);
For cp0=0 step 1 until 1 do
  For cp1=0 step 1 until 2 do
  Nmax_cell = get_max_cell(n,k,Cpl);
    For cp2=0 step 1 until Nmax_cell do
      For cp3=0 step 1 until PL=Cpl do
        PL=calc_pl(CP,n,k,E);
        Atp=area_of(CP,BC,n,k,E);
        If (Atp > Amax) then
          BP = update_bound(BP,CP);
        Else
          Stp = speed_of(CP,BC,n,m,d,PL);
          {D} = {D} + {CP, Atp,Stp};
          If (Stp >= Smax) then
            Dbest = {CP, Atp,Stp};
            Smax = Stp;
          endIf
        endIf
        CP=check_bound(BP,CP);
      endFor
    endFor
  endFor
endFor
If (Dbest = Ø) then
  no_feasible_designs;
endIf
```

Fig. 7. The CSAI procedure.

Table 5. An example of design space exploration.

n = 15, k= 9, m = 8, b = 0, error only mode,
$p(x) = x^8 + x^4 + x^3 + x^2 + 1$,
d = 150, C_{pl} = 4, **A_{max} = 17000** or **S_{min} = 70**
synthetic library from Hyundai 0.65 μm,
cell size for ROM and FIFO per bit = 2.5 cells / bit

{D}	PL
(RSC, MEA, 1, 1, 14498, 55)	6
(RSC, MEA, 1, 2, 14588, 63)	5
(RSC, MEA, 1, 3, 14678, 72)	4
(RSC, MEA, 1, 4, 14768, 86)	3
(RSC, MEA, 1, 5, 14858, 105)	2
(RSC, MEA, 1, 6, 14968, 135)	1
(RSC, MEA, 2, 1, 21269, 135)	1
(RSC, iBMA, 1, 1, 15567, 116)	1
(RSC, RiBMA, 1, 1, 21128, 156)	1
(CSC, MEA, 1, 1, 15526, 59)	6
(CSC, MEA, 1, 2, 15616, 68)	5
(CSC, MEA, 1, 3, 15706, 80)	4
(CSC, MEA, 1, 4, 15796, 96)	3
(CSC, .MEA, 1, 5, 15886, 121)	2
(CSC, MEA, 1, 6, 15976, 165)	1
(CSC, iBMA, 1, 1, 16595, 141)	1
(CSC., RiBMA, 1, 1, 22156, 191)	1

of CP for decreasing the throughput. However, because of the existence of a continuous stream size, we cannot definitively discriminate speed variations for N_{shr} and for a key equation solver directly when it supports the erasure correction. Instead, it requires a simple computation from (3), and the compiler can get a basis of ordering for the speed decrement. This procedure also suggests the best design point, which has a minimal cell size within the speed constraint.

Finally, after the step of customized selection by the user, the RS compiler generates a target model and a test vehicle with a full parameter set.

VI. Experimental Results

In Table 5, we suggest a simple example of a design space exploration for a full parameter set with different constraints. The gray rows are the best design points within the design constraints. The corresponding design space explorations are shown in Figs. 8 and 9. When there is an area constraint with 17,000 gates, the CSAI selects the design D_{best}={CSC, MEA, 1, 6, 15976, 165}, which is the fastest design of all feasible designs (Fig. 8), while the required pipeline latency is 4. On the other hand, with the same constraint for pipeline latency, for S_{min}=70 the CSSD determines the area-minimized parameter set, that is D_{best}={RSC, MEA, 1, 3, 14678, 72} (Fig. 9).

Tables 6 and 7 show the optimal parameter sets with area and speed constraints for several industry standards of the RS codes. We assumed that the required design has a continuous decoding capability [9]; hence, we set parameter d as a large number. As the results show, the output data rates are fixed for a key equation solver, regardless of their pipeline depth. Thus, the compiler always selects the RSC rather than the CSC. The small stream size will lead the CSC to be superior in speed to

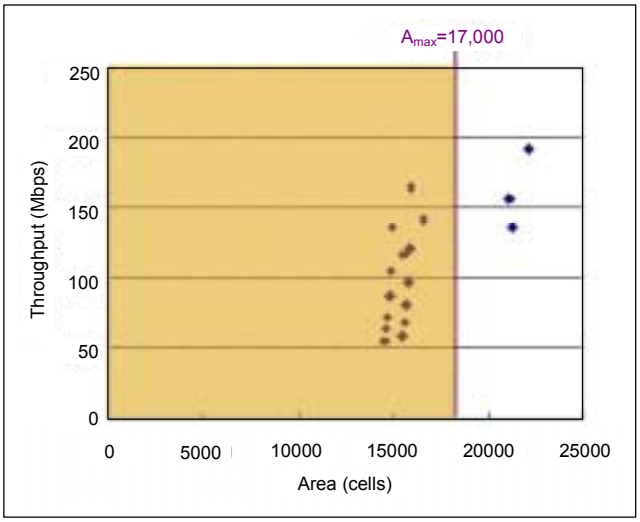

Fig. 8. Design space exploration for A_{max} = 17,000.

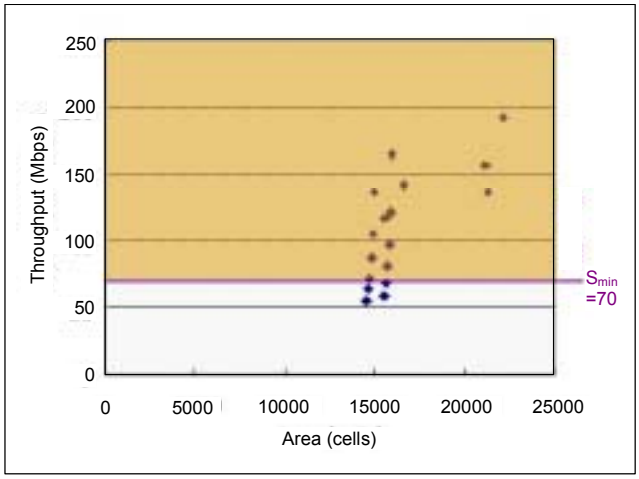

Fig. 9. Design space exploration for S_{min} = 70.

Table 6. Experimental results for industry standards.

b = 0, error only mode,
C_{pl} = 5, d= 100000,
synthetic library from Hyundai 0.65 μm,
cell size for ROM and FIFO per bit = 2.5 cells / bit

standard	(n,k,m)	A_{max} (cells)	D_{best}
CCSDS	(255,223,8)	90000	(RSC,RiBMA,1,1,83557,714)
DVD	(208,192,8)	35000	(RSC,MEA,1,1,30715,605)
DVB	(204,188,8)	35000	(RSC,MEA,1,1,32211,605)
VSB	(208,188,8)	50000	(RSC,MEA,1,73,49039,605)
IEEE 802.14B	(128,122,7)	35000	(RSC,MEA,1,1,22843,975)

Table 7. Experimental results for industry standards.

b = 0, error only mode,
C_{pl} = 5, d= 100000,
synthetic library from Hyundai 0.65 μm,
cell size for ROM and FIFO per bit = 2.5 cells/bit

standard	(n,k,m)	S_{min} (Mbps)	D_{best}
CCSDS	(255,223,8)	450	(RSC,iBMA,1,1,59619,495)
DVD	(208,192,8)	550	(RSC,MEA,1,1,30715,605)
DVB	(204,188,8)	550	(RSC,MEA,1,1,32211,605)
VSB	(208,188,8)	450	(RSC,iBMA,1,1,46859,495)
IEEE 802.14B	(128,122,7)	900	(RSC,MEA,1,1,22843,975)

the RSC. The MEA is the best choice when a single EC is enough to support those code specifications because of its architectural advantage. However, in the other cases as in CCSDS and VSB, the iBMA or RiBMA can be the best choice on the design space.

Consequently, if the IP users input any specifications including industry standards, they easily obtain optimal characteristic parameters with area and speed in a short time. Adding more sub-architectures to the characteristic parameters can suggest more capacity for covering a wide range over the design space.

VII. Conclusion

In this paper, we presented a soft IP compiler for the Reed-Solomon decoder. The considered parameter set of the core compiler includes application specific parameters as well as characteristic parameters, which give flexible sub-architectures on the design space. Moreover, since base designs have estimable regular structures, applying design constraints for a given code specification, an IP user can get a desired core without much effort. Through experiments, we verified the function of the core compiler with several industry standards. The results indicate that the RS compiler can apply to any code specification as it provides efficient architectures within its architectural capability. In the future, if efficient models are developed for estimating characteristics of soft IPs, such as power consumption and interconnection delay, this method of parameterization and design space exploration as in the RS compiler can help any soft-core compiler to create proper functional blocks at the early decision stage for the system specification.

References

[1] Woo-Jin Lee, Oh-Cheon Kwon, Min-Jung Kim, and Gyu-Sang Shin, "A Method and Tool for Identifying Domain Components Using Object Usage Information," *ETRI J.*, vol. 25, no. 2, Apr. 2003, pp. 121-132.

[2] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*, Kluwer Academics, USA, 1999.

[3] M.F. Jacome and H.P. Peixoto, "A Survey of Digital Design Reuse," *IEEE Des. & Test. Comput.*, vol. 18, no. 3, 2001, pp. 98-107.

[4] Tality Corp., *Reed Solomon Decoder (D10337) Datasheet*, www.design-reuse.com, 2001.

[5] Mentor Inventra, *RSDECD1 Reed-Solomon Decoder*, Altera MegaCore, 2000.

[6] Memec design, *MC-XIL-RSDEC Reed-Solomon Decoder*, Xlinx AllianceCORE™, 2002.

[7] ISS, *Reed-Solomon Decoder*, Xilinx AllianceCORETM, 2000.

[8] S. Smith, D. Taylor, and M. Benaissa, "Design Automation of Reed-Solomon Codecs Using VHDL," *The Microelectronics J.*, vol. 29, no. 12, 1998, pp. 977-982.

[9] Altera Corp., *Reed-Solomon Compiler User Guide*, Altera MegaCore, 2001.

[10] Xilinx Inc., *Reed-Solomon Decoder V. 2.0*, Xilinx LogiCore, 2001.

[11] SIPAC, *VHDL Coding Guideline V. 1.0*, www.sipac.org, 2002.

[12] D.D. Gajski, N.D. Dutt, A.C. Wu, and S.Y. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academics, 1992.

[13] A. Raghupathy and K.J.R. Liu, "Algorithm Based Low-Power/High-Speed Reed Solomon Decoder Design," *IEEE Trans. Circuits Syst. II: Analog and Digital Signal Processing*, vol. 47, no. 11, 2000, pp. 1254-1270.

[14] H.M. Shao and I.S. Reed, "On the VLSI Design of a Pipeline Reed-Solomon Decoder Using Systolic Arrays," *IEEE Trans. Comput.*, vol. 37, no. 10, 1988, pp .1273-1280.

[15] D.H. Lee and J.T. Kim, "Efficient Recursive Cell Architecture for the Reed-Solomon Decoder," *JKPS*, vol. 40, no. 1, 2002, pp. 82-86.

[16] I.S. Reed, M.T. Shih, and T.K. Truong, "VLSI Design of Inverse-Free Berlekamp-Massey Algorithm," *IEE Proc. of Computers and Digital Techniques*, vol. 138, no. 5, 1991, pp. 295-298.

[17] D.V. Sarwate and N.R. Shanbhag, "High-Speed Architectures for Reed-Solomon Decoders," *IEEE Trans. VLSI Syst.*, vol. 9, no. 5, 2001, pp. 641-655.

[18] J.H. Jeng and T.K. Truong, "On Decoding of Both Errors and Erasures of a Reed-Solomon Code Using an Inverse-Free Berlekamp-Massey Algorithm," *IEEE Trans. Commun.*, vol. 47, no. 10, 1999, pp. 1488-1494.

[19] Z. Junchao, C. Weiliang, and W. Shaojun, "Parameterized IP Core Design," *Proc. of 4th Int'l Conf. on ASIC*, 2001, pp. 744-747.

[20] M. Schutz, "How to Efficiently Build VHDL Testbenches," *Proc. of EURO-DAC'95*, 1995, pp. 554-559.

Jong Kang Park received BS and MS degrees in electric, electronics, and computer engineering from Sungkyunkwan University in Korea in 2001 and 2003. He is now a PhD candidate of the Electric and Electronics Engineering Department of Sungkyunkwan University. His current research interests include IP-based system platform and electronic system level design on the object-oriented environment.

Jong Tae Kim received the BS degree in electronics engineering from Sungkyunkwan University in Korea in 1982 and the MS and PhD degrees in electrical and computer engineering at the University of California, Irvine, in 1987 and 1992. He is currently an Associate Professor at the School of Electrical and Computer Engineering, Sungkyunkwan University, where he has been since 1995. From 1991 to 1993 he was with the Aerospace Corporation in Elsegundo, California. He was a full-time lecturer at Chunbuk National University in Korea from 1993 to 1995. His research interests include system-on-a-chip design, embedded systems, and computer architecture.