CoreRSENC v3.4 Handbook

Microsemi

Table of Contents

Introduction

Core Overview

CoreRSENC is an register transfer level (RTL) generator that produces a Microsemi® system-on-chip (SoC) Products Group FPGA–optimized Reed-Solomon (RS) encoder core based on user-defined parameters.

RS codes are a class of error-correcting codes used to detect and correct errors that might be introduced into digital data when it is transmitted or stored. Error-correcting codes incorporate redundancy in data. With this redundancy, only a subset of all possible transmissions contains valid messages. This means the valid codes are separated from each other so errors are not likely to corrupt one valid code into another. The encoded data can then be transmitted or stored.

When recovering data, a decoder first determines if a received message is valid. This step is called error detection. If an error is detected, the decoder finds the valid message “closest” to the received one. Provided the number of corrupted words (symbols) does not exceed a specified range, the message found is the one that was transmitted. Thus, the decoder conducts error correction.

The number of errors the code can correct depends on the amount of redundancy added. In other words, the more errors are expected to occur, the more redundant symbols need to be added. Note that the number of redundant symbols directly impacts the complexity of the RS codec (encoder and especially decoder).

The RS encoder and decoder do not necessarily have to be coupled. Both encoder and decoder operate over an RS code that you have defined via core configuration parameters. Once the same RS code parameters are defined, the encoder/decoder can communicate to a different decoder/encoder at logic level. You may need to provide physical-level converters and minimal handshaking logic, if necessary. Obviously, the Microsemi RS encoder and decoder work with each other with no extra logic or converters needed.

You can configure the RS encoder via a configuration user interface that controls the encoder parameter values. For a detailed description of the configuration parameters, refer to "Configuration Parameters" on page 14.

An example of a digital communication system utilizing an RS codec is shown in Figure 1. Data gets encoded then modulated and transmitted via a communication channel that may introduce one or more errors. At the receiver end, a demodulated message gets decoded with erroneous symbol(s) corrected. The recovered data goes to its destination.

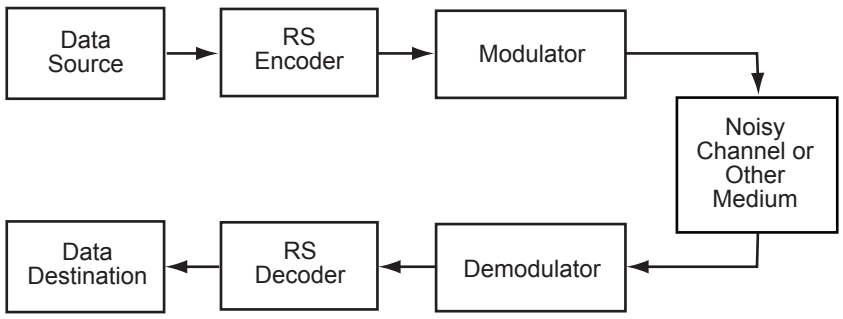

***Figure 1* • An Example of a Digital Communication System**

Key Features

CoreRSENC is a highly configurable core and has the following features:

- Parameterizable RS Encoder generator
- Supports arbitrary time intervals between the output code words, including zero interval
- Symbol widths from 3 to 8 bits
- Supports shortened code
- Supports any valid primitive polynomial for a given symbol width
- Supports CCSDS-16 and CCSDS-8 encoding
- In the CCSDS mode supports data encoding presented in conventional or dual basis

Device Utilization and Performance

CoreRSENC has been implemented in several Microsemi FPGA families. A summary of the utilization and performance data for CoreRSENC is given in Table 1.

Table 1 • **CoreRSENC Device Utilization and Performance**

FPGA Family and Device	Configuration[2]	Cells or Tiles			Utilization %	Device Speed Grade	Clock Rate MHz	Throughput Mbps
		Comb.	Seq.	Total				
IGLOO® AGL125V5	1	338	225	563	18.30 %	STD	101	730
	2	425	321	746	24.30 %	STD	89	623
ProASIC®3/E A3P250	1	342	221	563	9.2 %	-2	156	1,127
	2	451	319	770	12.5 %	-2	146	1,021
SmartFusion® A2F200M	1	320	202	543	11.78 %	-1	120	867
	2	399	298	718	15.58 %	-1	107	748
Fusion AFS250	1	344	221	565	9.20 %	-2	150	1,060
	2	443	319	762	12.40 %	-2	140	950
ProASIC PLUS® APA300	1	460	205	665	8.1 %	STD	78	564
	2	552	301	853	10.4 %	STD	82	574
Axcelerator® AX250	1	376	204	580	13.7 %	-2	155	1,120
	2	424	300	724	17.1 %	-2	151	1,056
RTAX-S RTAX250S	1	376	204	580	13.7 %	-1	112	784
	2	424	300	724	17.1 %	-1	99	715
SmartFusion®2 M2S005	1	243	200	443	3.60 %	-1	329	2400
	2	348	296	644	5.31 %	-1	282	2004
IGLOO®2 M2GL005	1	243	200	443	3.60 %	-1	332	2421
	2	348	296	644	5.31 %	-1	284	2018
RTG4™ RT4G150	1	303	206	509	0.30 %	-1	227	1656
	2	424	301	525	0.31 %	-1	175	1243

Notes:

1. *Data in this table is gathered using typical synthesis and layout settings. Throughput is computed as follows: (Bit width / Number of cycles) × Clock Rate (Performance).*
2. *The configuration details are described in Table 2.*

CoreRSENC configuration parameters were set as shown in Table 2.

Table 2 • **RS Encoder Test Configurations**

Parameter		Configuration	
Name	**Description**	**1**	**2**
MM	Symbol width, bits	8	8
NN	Codeword length, symbols	207	255
TT	Number of corrupted symbols the RS code can correct	10	16
first_root	First root of the primitive polynomial	0	112
Prim_poly	Primitive polynomial	285	391

1 – Functional Description

Theory of Operation

Properties of Reed-Solomon Codes

An RS code is a block code generally designated as RS(*n*, *k*) with *m*-bit symbols, where *k* is the number of data symbols per block, *n* is the number of symbols the encoded message contains, and the symbol size *s* can be in a range from one to several bits. The encoded message called codeword has $n - k$ redundant parity symbols. The code can correct up to $t = (n - k) / 2$ symbols.

The RS code is also a systematic one, since the encoder simply appends the parity symbols to the otherwise unchanged original data sequence. Figure 1-1 shows the RS code structure.

The RS code is a linear code. In practice, this means that every possible *m*-bit word is a valid symbol. For instance, with 8-bit RS symbols, any 8-bit word can be transmitted directly in the data part of a codeword (Figure 1-1), so the encoder does not care what the nature of the data is, whether it is a binary stream separated into blocks of *k* 8-bit symbols, ASCII codes, etc. Given a symbol size *m*, the maximum RS codeword length is $n_{max} = 2^s - 1$.

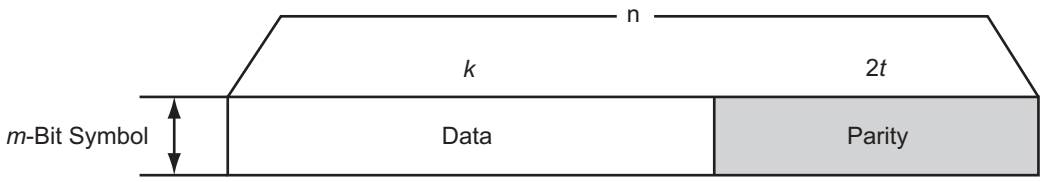

***Figure 1-1* • The RS Code Structure**

Consider an RS(255, 223) code with 8-bit symbols utilized by many standards. Here each codeword contains 223 data bytes and 32 parity bytes, totaling a 255-byte codeword. The code is capable of correcting up to 16 corrupted symbols. Parameters of the code are as follows:

- $n = 255$
- $k = 223$
- $m = 8$
- $t = (255 - 223) / 2 = 16$

A corrupted symbol can have one or more (up to *m*) erroneous bits. In the above example, the RS code can correct up to 16 symbol errors when every erroneous symbol has one to eight corrupted bits. This property makes RS a powerful tool for protecting data impacted by burst errors.

Galois Field Math

RS codes are based on Galois fields (GFs), also called finite fields. Rules of the GF arithmetic are different from the usual arithmetic rules. For instance, GFs are finite fields. This means that any field element, as well as a result of the element addition and multiplication, can be presented by a fixed-length binary word. To generate and decode an RS code of *m*-bit symbols, an *m*-bit-wide GF is used. References 1 and 2 in "References" on page 18 provide a gentle introduction to GF math. Here only a few notes on GF are discussed—those that help configure CoreRSENC and CoreRSDec.

A GF used to generate an RS code is defined by RS symbol size *s* and a primitive polynomial. The polynomial has binary coefficients, for example, either 0 or 1. For instance,

$$1 \cdot x^8 + 0 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1$$

EQ 1-1

Depending on the size *s*, there might be one or more valid primitive polynomials. Different polynomials generate different GFs and thus different RS codes. Usually, particular standards—for example, 802.11—define the primitive polynomial to be used in the RS encoder/decoder. The Microsemi RS cores support any user-defined polynomial valid with any symbol size *s*. Polynomials are entered as decimal numbers. The bits of this number's binary image correspond to the polynomial coefficients. For example,

$$1 \cdot x^8 + 0 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1 \Rightarrow 100011101 = 285$$

EQ 1-2

The core also checks the validity of the polynomial entered. If the polynomial is invalid, the core replaces it with a default primitive polynomial and issues an appropriate warning. Default polynomials are listed in Table 1-1.

***Table 1-1* • Default Primitive Polynomials**

Symbol Size, *s*	Default Polynomial	Decimal Form
3	$x^3 + x + 1$	11
4	$x^4 + x + 1$	19
5	$x^5 + x^2 + 1$	37
6	$x^6 + x + 1$	67
7	$x^7 + x^3 + 1$	137
8	$x^8 + x^4 + x^3 + x^2 + 1$	285

There is another important polynomial an RS codec directly utilizes: a Generator polynomial. This is derived from the primitive polynomial based on the first root of the Generator polynomial. The particular standards often define the first root value to be used in the RS codec. Most common first root values are 0 or 1, but the Microsemi RS encoder supports any value in the range from 0 to $n - 1$.

Shortened Codes

A shortened codeword contains fewer symbols than the maximum $n_{max} = 2^m - 1$. The shortened codeword keeps the same number of parity symbols, $2t$, to correct up to t errors. Therefore, the number of data symbols in the shortened code is reduced by the same amount as the overall codeword length. For instance, RS(204, 188) is the shortened code of RS(255, 239). Both codes have a symbol width of 8 and use the same number of parity symbols, 16.

Conceptually, shortening a codeword is done by assuming initial extra data symbols of the maximum-length codeword are set to 0. Though efficiency of the shortened code is less than that of the maximum-length code, some standards require the RS codec to use it.

RS Encoder Block Diagram

Figure 1-2 shows a block diagram of the RS(*n*, *k*) encoder. For *k* clock cycles, it accepts *m*-bit input symbols. Since the RS code is systematic, the unaltered input data pass through the encoder via an output multiplexer. Once the *k* data symbols enter the encoder, the feedback shift registers reg0 – reg(*n* – *k* – 1) finish calculating the parity symbols. These are appended to the data symbols with the help of the output multiplexer to form a complete codeword.

The feedback shift register generates parity symbols in such a way as to make the complete codeword a whole multiple of the RS Generator polynomial. At the decoder end, the codeword is divided by the Generator polynomial. If the remainder is zero, no errors are detected. Otherwise, there are errors in the received codeword. All calculations are performed using finite field arithmetic.

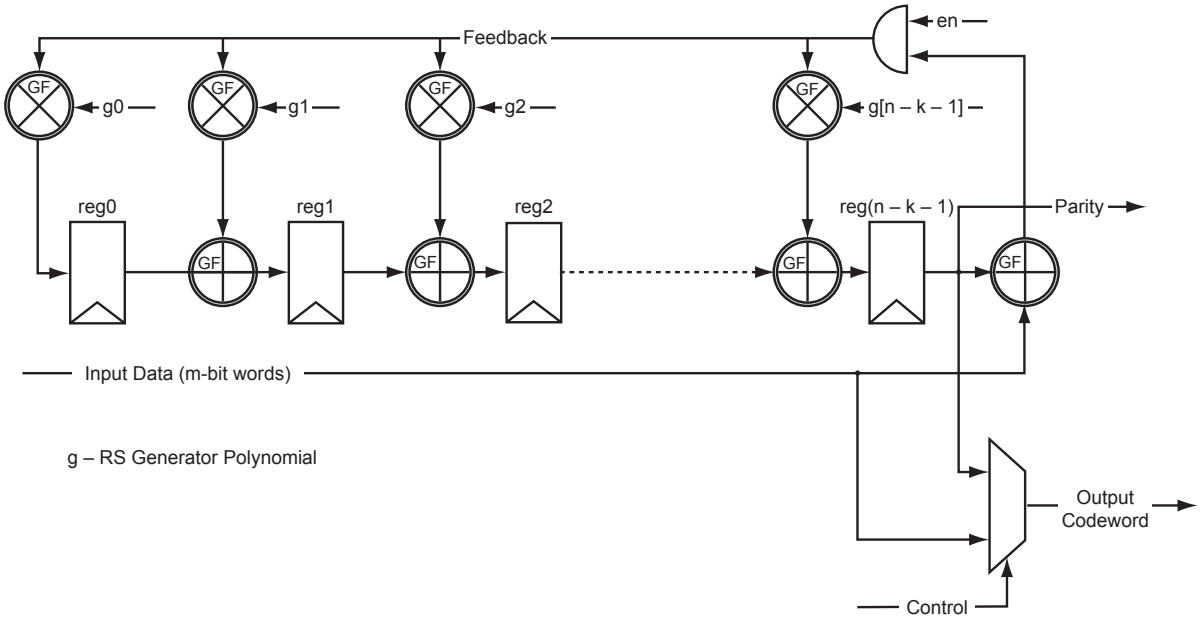

***Figure 1-2* • RS Encoder Block Diagram**

CCSDS Compatibility

The core optionally complies with the CCSDS 131.0-B-1 standard. It supports dual basis encoding with the following RS parameters:

- Symbol size MM — 8 bits
- Primitive (field generator) polynomial — 391 ($x^8 + x^7 + x^2 + x + 1$)
- Codeword size NN — 255 symbols
- Primitive element — 11
- First root — 112
- Error correction capacity — 16 or 8

In the CCSDS-16 or CCSDS-8 modes that correspond to the error correction capacity of 16 or 8 respectively, the other RS parameters automatically take the above values.

Note: In the Conventional (non-CCSDS) mode you can select values for the other parameters.

The core implements the CCSDS compatibility by using a dual-to-conventional basis data converter infront of the conventional encoder and a conventional-to-dual basis data converter at the conventional encoder output. The input dual-to-conventional basis converter is optional and should be used only if the input data is present in dual basis. The CCSDS_CONV parameter controls the input converter inference. Once the parameter is set, the core automatically instantiates the converter.

Encoder Latency

The conventional encoder latency equals 2 clock cycles. Thus in the non-CCSDS mode the latency = 2. Every converter mentioned in the CCSDS Compatibility section introduces additional one- clock latency. If the CCSDS input data to be encoded are present in the conventional basis, only the output converter is used and the overall latency = 3. When the data to be encoded is present in the dual basis both converters are instantiated and the overall latency = 4.

2 – Tool Flows

Licensing

CoreRSENC is provided with a full RTL license, as outlined below.

RTL

Complete RTL source code is provided for the core and testbenches.

SmartDesign

CoreRSENC is available for download to the Libero IP catalog via the web repository. Once it is listed in the catalog, the core can be instantiated using the SmartDesign flow. For more information on using SmartDesign to instantiate, configure, connect, and generate cores, refer to the Libero online help.

Figure 2-1 below depicts the CoreRSENC configuration window. In the Conventional mode you can select the configuration parameters either from drop-down menus or typing in the desired parameter values. In both CCSDS modes the parameters are pre-selected to comply with the standard. Once the CCSDS mode is selected, the parameter values show up on the configuration window but cannot be changed.

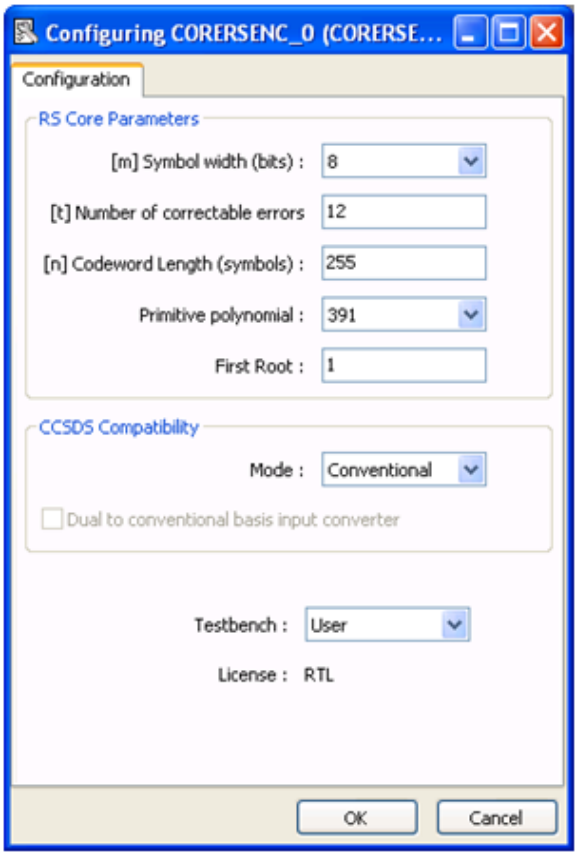

Figure 2-1 • **CoreRSENC Configuration Window**

Simulation Flows

To run simulations, in the core configuration window, select **User** from the **Testbench** drop-down list as shown in Figure 2-1. After generating the core, Libero installs the pre-synthesis testbench HDL files.

Consider an example of instantiating CoreRSENC as a part of the SmartDesign design sd. To run the testbench, set the Libero design root to the core instance sd_CORERSENC_0_CORERSENC and run Pre-Synthesis design simulation.

Synthesis in Libero

Set Synplify Pro to use the Verilog 2001 standard, if Verilog is being used. To run synthesis on the core, set the design root to the SmartDesign instance sd and run synthesis tool from Libero Design Flow pane.

Place-and-Route in Libero

After the design has been synthesized, run **Compile** and then **Place-and-Route** tools.

3 – Interface Descriptions

Ports

The port signals for CoreRSENC are defined in Table 3-1 and illustrated in Figure 3-1.

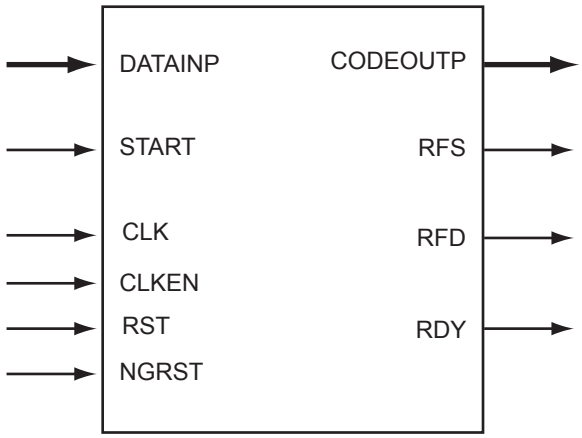

***Figure 3-1* • CoreRSENC I/O Signals**

***Table 3-1* • I/O Signal Descriptions**

Signal	Direction	Description
DATAINP[MM-1:0]	Input	Data input to be encoded. The input bus is *m* bits wide.
START	Input	This signal starts a new codeword cycle. It informs the encoder that at the next clock interval the first *m*-bit data symbol of a *k*-symbol sequence appears at DATAINP.
CLK	Input	Encoder clock signal
CLKEN	Input	Encoder clock enable signal
RST	Input	Synchronous reset
NGRST	Input	Asynchronous reset; active low
CODEOUTP[MM-1:0]	Output	Encoded data (codeword) output. The output is *m* bits wide.
RFS	Output	Ready For Start. This signal is active when the encoder is ready to accept a new START signal.
RFD	Output	Ready For Data. This signal is active when the encoder is ready to accept a new data portion of *k* symbols on DATAINP.
RDY	Output	(Output RS Code) Ready. This signals that a valid codeword is present at the core output.

I/O Signal Functionality

NGRST, RST Inputs

Both reset signals reset all registers of the RS encoder to bring it to an initial state. In the initial state, the feedback registers reg0 – reg(*n* – *k* – 1) (Figure 1-2 on page 8) are set to zero, signals RFS and RFD are active, and the RDY signal is inactive. The RS encoder is ready to accept fresh input data. NGRST is an asynchronous signal (active low) and RST (active high) is synchronous to rising edge of the clock signal. One or both must be used to prevent the very first codeword from being incorrect. If this is not a problem, the reset signals are optional.

CLKEN Input

When CLKEN is inactive (Low), the core is frozen. All inputs except NGRST are ignored and the core retains its current state.

START Input

This signal starts a new codeword cycle. It informs the encoder that at the next clock interval the first *m*-bit data symbol DATAINP0 of a *k*-symbol sequence appears at the DATAINP bus (Figure 3-2). It is assumed that the CLKEN signal is active in Figure 3-2.

The START signal can be set at any time to launch another codeword generation. Normally, a data source is supposed to issue START once the RFS signal goes High. If the data source wishes to issue a START before the completion of the current codeword, it must first issue a reset by either asserting the RST signal or deasserting the NGRST signal. Otherwise, the subsequent codeword will be corrupted and must be discarded.

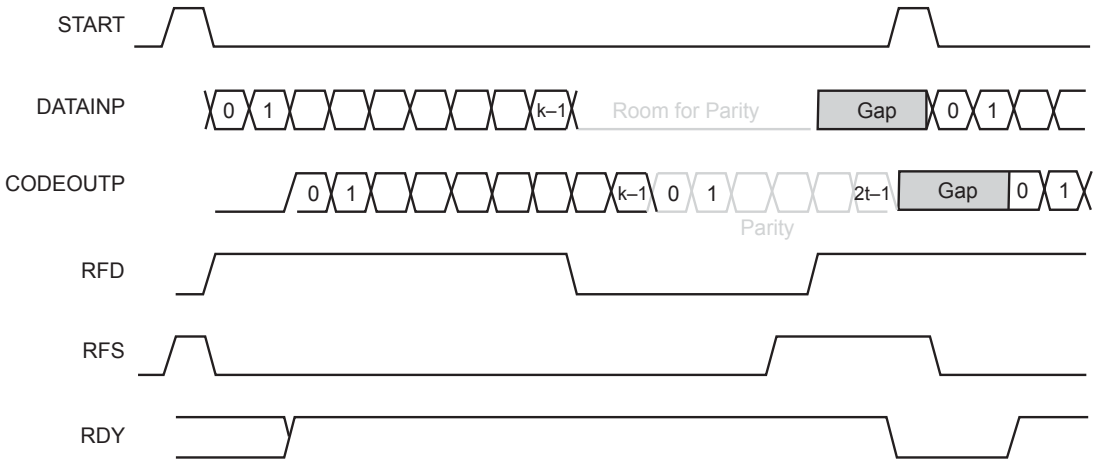

***Figure 3-2* • RS Encoder timing**

It can be seen in Figure 3-2, that START can be asserted as early as to repeat the RFS signal, or can otherwise be delayed arbitrarily with regard to a positive edge of RFS. Figure 3-2 shows an example of a three-clock-cycle delay.

RFS Output

The core asserts this output when it is ready to accept another START signal. With normal RS encoder functionality, the data source should wait for RFS to go High to issue the START signal. Logically, RFS signifies completion of the current codeword generation, but it gets asserted three clock cycles prior to actual codeword termination (Figure 3-2) due to the RS encoder latency.

RFD Output

This optional output signal is asserted when the RS encoder core is ready for fresh input data. Once the core fetches *k* symbols of data, RFD goes LOW, thus blocking input data when the core generates redundant parity symbols (Figure 3-2).

RDY Output

The optional RDY signal marks an interval of time when a codeword is present at the RS encoder output CODEOUTP (Figure 3-2 on page 13).

Configuration Parameters

CoreRSENC generates the RS encoder engine RTL code based on parameters that you have set. CoreRSENC supports the variations specified in Table 3-2.

***Table 3-2* • CoreRSENC Configuration Parameters**

Name	Valid Values	Description
MM	3 to 8	Symbol width, bits (m)
NN	5 to 2^m – 1	Codeword length, symbols (n)
TT	1 to 16 as long as t < n / 2 – 1	Number of correctable errors (t)
Prim_poly	Arbitrary valid polynomial selectable from a drop-down menu	Primitive polynomial identifying Galois field
first_root	0 to n–1	First root of the primitive polynomial
FPGA Family	IGLOO, IGLOOPLUS, IGLOOe, ProASIC3/E/L, SmartFusion, Fusion, ProASIC^{PLUS}, Axcelerator, RTAX-S, SmartFusion2, IGLOO2, RTG4	Identifies family of FPGA devices. The parameter value sets automatically after a Libero project is configured for a specific family. Does not require any action.
CCSDS	0 (Conventional mode), 1 (CCSDS-16), 2 (CCSDS-8)	CCSDS mode. Value of 0 defines Conventional mode, values of 1 and 2 set the CCSDS-compatible modes with 16 or 8 correctable errors respectively
CCSDS_CONV	0, 1	Input dual-to-conventional basis converter. If the parameter value = 1, the core automatically instantiates the converter, thus enabling the encoder to accept dual basis data. Otherwise the encoder accepts conventional basis data.

4 – Testbench Operation and Modification

User Testbench

Included with the releases of CoreRSENC is a user testbench that verifies operation of the CoreRSENC engine. A simplified block diagram of the user testbench is shown in Figure 4-1. The user testbench instantiates the RS encoder engine that you have configured, as well as behavioral, non-synthesizable models of an input test vector generator, a golden codeword generator, a comparator, and a signal generator that provides necessary clock, reset, and other signals. The testbench compares the actual RS encoder output codeword and the golden codeword vector. CoreRSENC automatically generates Verilog or VHDL testbench behavioral code based on the selected core language.

The same testbench can be used for pre-synthesis and post-synthesis simulation. A simulation tool displays the verification result.

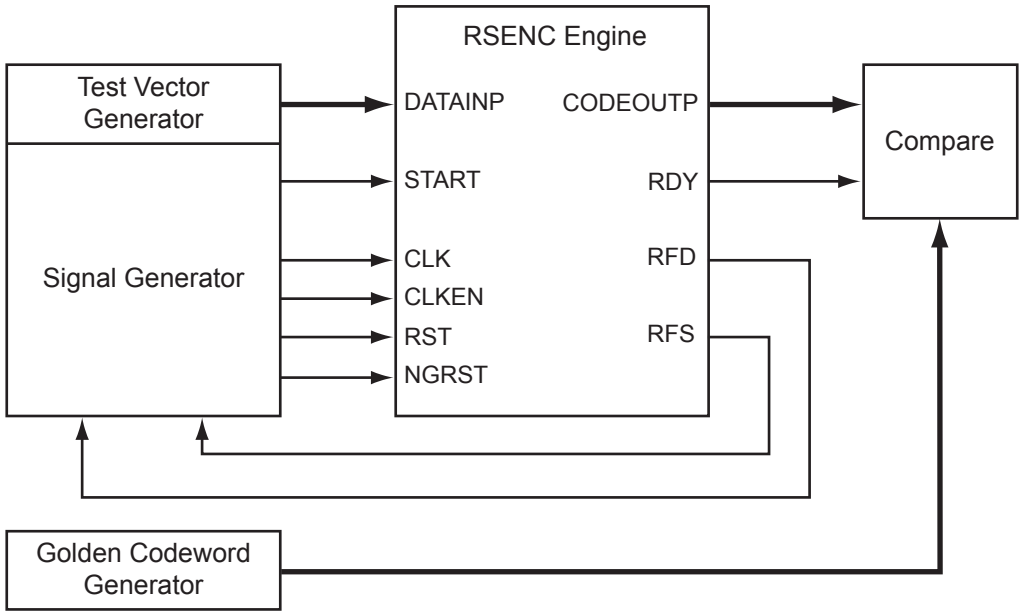

***Figure 4-1* • CoreRSENC User Testbench**

5 – Ordering Information

Ordering Codes

CoreRSENC can be ordered through your local Microsemi sales representative. It should be ordered using the following number scheme: CoreRSENC-XX, where XX is listed in Table 5-1.

***Table 5-1* • Ordering Codes**

XX	Description
RM	RTL for RTL source — multi-use license

6 – List of Changes

The following table shows important changes made in this document for each revision.

Revision	Changes	Page
Revision 5 (October 2015)	The core version was updated from v3.3 to v3.4.	NA
Revision 4 (April 2015)	The core version was updated from v3.2 to v3.3.	NA
Revision 3 (February 2015)	The core version was updated from v3.1 to v3.2 (SAR 57402).	NA
Revision 2 (April 2012)	The core version was updated from v3.0 to v3.1 (SAR 37415).	NA
	The "Key Features" section was updated (SAR 37415).	4
	The "Device Utilization and Performance" section was updated (SAR 37415).	4
	The "CCSDS Compatibility" section was added (SAR 37415).	8
	The "Encoder Latency" section was added (SAR 37415).	9
	The "Tool Flows" section was updated (SAR 37415).	10
	The Table 3-2 • CoreRSENC Configuration Parameters was updated (SAR 37415).	14
Revision 1 (February 2011)	The core version was updated from v2.0 to v3.0. SmartFusion numbers were added to Table 1 • CoreRSENC Device Utilization and Performance.	4
	The parameter names were changed in Table 2 • RS Encoder Test Configurations.	5
	The *s*-bit was changed to *M*-bit in Figure 1-1 • The RS Code Structure and the accompanying text.	6
	The "Tool Flows" section was replaced.	10
	[MM-1:0] was added to the signal names DATAINP and CODEOUTP in Table 3-1 • I/O Signal Descriptions.	12
	Port names in Figure 3-1 • CoreRSENC I/O Signals and Figure 3-2 • RS Encoder timing were changed from lower case to upper case. Occurrences of these port names throughout the document were revised accordingly.	12
	The "START Input" section was revised to explain necessary steps if the START signal will be initiated before completion of the current codeword.	13
	The names and descriptions of configuration parameters were revised in Table 3-2 • CoreRSENC Configuration Parameters. IGLOO PLUS and SmartFusion were added as valid FPGA families and ProASIC3/E was changed to ProASIC3/E/L. The HDL selection parameter was removed.	14
	The terminology "verification testbench" was changed to "user testbench" in the "Testbench Operation and Modification" chapter.	15

A – References

Rorabaugh, C. Britton. *Error Coding Cookbook*. McGraw-Hill, 1995.

Sweeney, Peter. *Error Control Coding*. John Wiley & Sons, 2002.

B – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call 800.262.1060
From the rest of the world, call 650.318.4460
Fax, from anywhere in the world, 408.643.6913

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

For Microsemi SoC Products Support, visit

http://www.microsemi.com/products/fpga-soc/designsupport/fpga-soc-support

Website

You can browse a variety of technical and non-technical information on the SoC home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office.Visit About Us for sales office listings and corporate contacts.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Index

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 3,600 employees globally. Learn more at **www.microsemi.com**.

50200106-5/10.15