HB0249

CoreRSDEC v3.6 Handbook

12 2016

Microsemi
Power Matters.™

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center 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; enterprise storage and communication solutions; 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 4,800 employees globally. Learn more at www.microsemi.com.

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
www.microsemi.com

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 5.0

Updated changes related to CoreRSDEC v3.6.

1.2 Revision 4.0

Updated changes related to CoreRSDEC v3.5.

1.3 Revision 3.0

Updated changes related to CoreRSDEC v3.4.

1.4 Revision 2.0

Updated changes related to CoreRSDEC v3.1.

1.5 Revision 1.0

Revision 1.0 was the first publication of this document. Created for CoreRSDEC v3.0

Contents

List of Figures

List of Tables

2 Introduction

2.1 Overview

CoreRSDEC is a register transfer level (RTL) generator that produces a Microsemi® fabric programmable gate array (FPGA)–optimized Reed-Solomon (RS) decoder core based on user-defined parameters.

RS code is 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 a valid one. This step is called error detection. Once any error is detected, the decoder finds a 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.

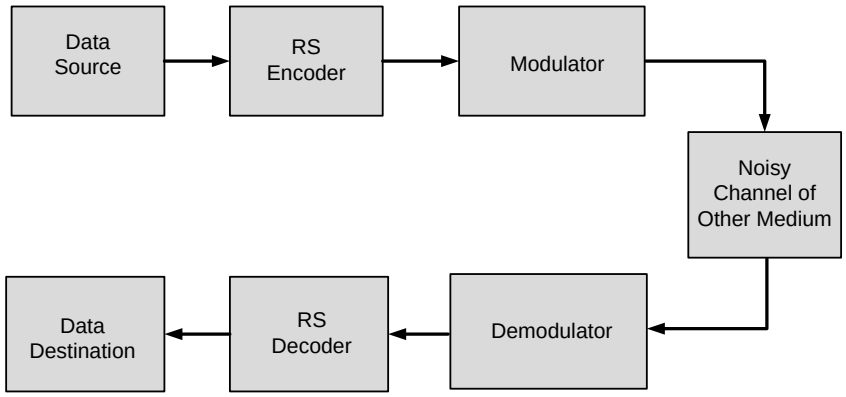
The number of errors the code can correct depends on the amount of redundancy added. In other words, if more errors are expected to occur, more redundant symbols need to be added. The number of redundant symbols directly impacts the complexity of the Reed-Solomon codec (encoder and especially decoder).

The RS encoder and decoder do not necessarily have to be coupled. Both encoder and decoder operate over an RS code that is entirely defined by a user through the core configuration parameters. Once the same RS code parameters are defined, the encoder/decoder can communicate to a different decoder/encoder at logical level. A physical level converters and minimal handshaking logic are required to be provided if necessary. The RS encoders and decoders work with each other with no extra logic or converters necessary.

The CoreRSDEC is configured through the Configurator GUI. Only the desired parameter values need to be set. For a detailed description of the configuration parameters, refer to the Configuration Parameters section.

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

Figure 1 An Example of a Digital Communication System



2.2 Features

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

- Parameterizable CoreRSDEC generator
- Symbol widths from 3 to 8 bits
- Supports shortened code in conventional mode decoding
- Supports CCSDS-16 and CCSDS-8 decoding
- CCSDS mode supports data decoding presented in dual basis

2.3 Core Version

This handbook is for CoreRSDEC version 3.6.

2.4 Supported Families

- SmartFusion®2
- SmartFusion®
- Axcelerator®
- RTAX™-S
- ProASICPLUS®
- ProASIC®3
- ProASIC3E
- ProASIC3L
- Fusion®
- IGLOO®
- IGLOOe
- IGLOOPLUS
- IGLOO®2
- RTG4™
- PolarFire

2.5 Device Utilization and Performance

CoreRSDEC has been implemented in several Microsemi FPGA families. A summary of the data for CoreRSDEC is listed in Table 1.

Table 1 CoreRSDEC Device Utilization and Performance

FPGA Family and Device	Config	Logic Elements			Utilization %	RAM Blocks	Device SpeedGrade	Clock Rate (MHz)
		Comb	Seq	Total				
Fusion® AFS600	1	4,721	1,201	5,922	42.84	1	-2	54.9
	2	4,760	1,229	5,989	43.32	1	-2	56.8
	3	7,582	2,075	9,657	69.86	1	-2	51.7
IGLOO® AGL600V5	1	4,720	1,203	5,923	42.85	1	STD	35.1
	2	4,788	1,228	6,016	43.52	1	STD	35.3
	3	7,628	2,073	9,701	70.18	1	STD	31.5
ProASIC®3 A3P600	1	4,728	1,202	5,930	42.90	1	-2	54.9
	2	4,760	1,229	5,989	43.32	1	-2	56.8
	3	7,582	2,075	9,657	69.86	1	-2	51.7
ProASICPLUS® APA1000	1	7,027	1,268	8,295	14.7	2	STD	31.9
	2	7,412	1,295	8,437	15	2	STD	31
	3	11,830	2,157	13,987	24.8	2	STD	30
Axcelerator® AX1000	1	4,092	1,268	5,360	29.54	1	-2	63.1
	2	4,116	1,347	5,463	30.11	1	-2	65.7
	3	7,128	2,225	9,353	51.55	1	-2	59.5
RTAX™-S RTAX1000	1	4,027	1,305	5,332	29.39	1	-1	71.5
	2	4,090	1,333	5,423	29.89	1	-1	66.5
	3	7,126	2,215	9,341	51.48	1	-1	67.5
SmartFusion®2 M2SO50T	1	3,218	1,157	4,365	8.99	1	-1	115.8
	2	3,230	1,184	4,414	9.07	1	-1	121.8
	3	5,334	2,022	7,356	15.11	1	-1	116
IGLOO®2 M2GL005	1	3,275	1,198	4,473	36.90	1	-1	124.0
	2	3,499	1,207	4,706	38.82	1	-1	116.6
	3	5,486	2,039	7,525	62.08	1	-1	116.7
RTG4™ RT4G150	1	5,751	2,232	7,893	3.85	1	-1	93.6
	2	3,623	1,364	4,987	1.64	1	-1	90.5
	3	5,896	2,267	8,163	2.68	1	-1	85.2
PolarFire MPF300T_ES	1	5,217	2,072	7,289	2.43	1	STD	119.6
	2	3,422	1,230	4,652	1.55	1	STD	122.6
	3	5,316	2,083	7,399	2.47	1	STD	114.7

Note: 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).

CoreRSDEC configuration parameters are set as listed in Table 2: Conventional, CCSDS-8, and CCSDS-16 respectively.

Table 2 CoreRSDEC Test Configurations

Parameters		Configuration		
Name	**Description**	**1**	**2(CCSDS-8)**	**3(CCSDS-16)**
m	Symbol width, bits.	8	8	8
n	Codeword length, symbols.	255	255	255
t	Number of correctable symbols.	16	8	16
B0	First root of the Primitive polynomial.	112	120	112
prim_poly	Primitive polynomial.	391	391	391
Enable erasure	Erasure enabled.	No	No	No
No Error Flag	Error Flag enabled.	No	No	No
Enable Tag	Tag enabled.	No	No	No

3 Functional Description

3.1 Theory of Operation

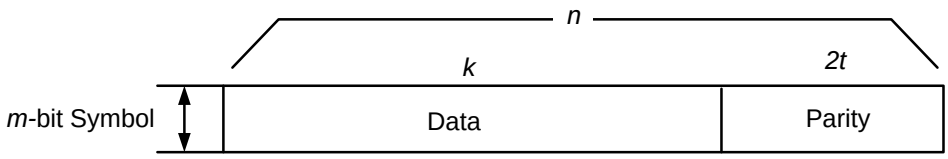
3.1.1 Properties of Reed-Solomon Codes

An RS 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 m can be in a range from one to several bits [2, 4]. Obviously, the encoded message, called a 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 appends the parity symbols to the otherwise unchanged original data sequence. Figure 2 shows the RS code structure.

The RS code is a linear code. In practice, this means 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 2), 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, and so on. Given a symbol size m, the maximum RS codeword length is $n_{max} = 2^m - 1$.

Figure 2 The RS Code Structure



An RS (255, 223) code with 8-bit symbols utilized by many standards, should be considered. Each codeword contains 223 data bytes and 32 parity bytes, a total of 255-byte codeword. The code is capable of correcting up to 16 corrupted symbols.

Parameters of the code are as following:

- 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 while every erroneous symbol has one to eight corrupted bits. This property makes RS codes a powerful tool for protecting data impacted by burst errors.

3.1.2 Galois Field Math

RS codes are based on Galois fields (GFs), also called finite fields. The rules of GF arithmetic are different from the usual arithmetic rules. For instance, GFs are finite fields. To generate and decode RS code of m-bit symbols, an m-bit wide Galois field is used. References section (Rorabaugh and Sweeney) provides a gentle introduction to the GF math. Only a few notes on GFs are discussed (those that help configure CoreRSENC and CoreRSDEC).

A Galois field used to generate RS code is defined by RS symbol size m and a primitive polynomial. The polynomial has binary coefficients—that is., either 0 or 1.

For instance:

1 * x8 + 0 * x7 + 0 * x6 + 0 * x5 + 1 * x4 + 1 * x3 + 1 * x2 + 0 * x + 1

Depending on the size m, 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 an RS encoder/decoder. The Microsemi RS cores support any user-defined polynomial valid with any symbol size m. Polynomials are entered as decimal numbers. The bits of this number's binary image correspond to the polynomial coefficients.

For example:

1 * x8 + 0 * x7 + 0 * x6 + 0 * x5 + 1 * x4 + 1 * x3 + 1 * x2 + 0 * x + 1 => 100011101 = 285

Configurator provides a drop-down menu that lists all valid primitive polynomials. In case the user does not select a specific polynomial, the core uses a default primitive polynomial. Default polynomials are listed in Table 3.

Table 3 Default Primitive Polynomials

Symbol Size, m	Default Polynomial	Decimal Form
3	x3 + x + 1	11
4	x4 + x + 1	19
5	x5 + x2 + 1	37
6	x6 + x + 1	67
7	x7 + x3 + 1	137
8	x8 + x4 + x3 + x2 + 1	285

Another important polynomial an RS codec directly utilizes is a generator polynomial. This is derived from the primitive polynomial based on the first root of the generator polynomial. Again, 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 CoreRSDEC supports any value in the range from 0 to n – 1.

3.1.3 Shortened Codes

A shortened codeword contains fewer symbols than the maximum nmax = 2m – 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 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 lesser than the maximum-length code, some standards require the RS codec to use it.

3.1.4 Erasures

Normally, the CoreRSDEC detects and corrects errors based solely on the n – k redundant parity symbols. Additional data is not needed to perform data detection/correction.

Erasure is an instance when CoreRSDEC knows an incoming symbol is likely to be an error. This knowledge comes from outside rather than from decoding the RS code and detecting the error inside CoreRSDEC. For example, a receiver can have a threshold detector that decides whether an input signal level carries a bit value of 0 or 1. It is reasonable to set the thresholds in such way that there are thresholds where the receiver is certain it receives 0 or 1, and an uncertainty threshold between the first two where the receiver refuses to make a decision. Such a receiver produces a three-value bit: 0, 1, or X (uncertain). Erasure is an instance when an uncertain symbol gets into CoreRSDEC.

Erasure locations (which symbols are uncertain) are known to the decoder beforehand.

CoreRSDEC can work with a mixture of errors and erasures. If the number of erasures in a codeword is e and the number of errors in the same codeword is r, the following relation holds for the codeword that includes 2t parity symbols (*Morelos-Zaragoza*, References): 2t > 2r + e.

Erasure mode can either be enabled or disabled when configuring the core. Enabling Erasure mode substantially increases the FPGA resource utilization.

For example: in case of mixture of error and erasure in conventional mode of decoding.

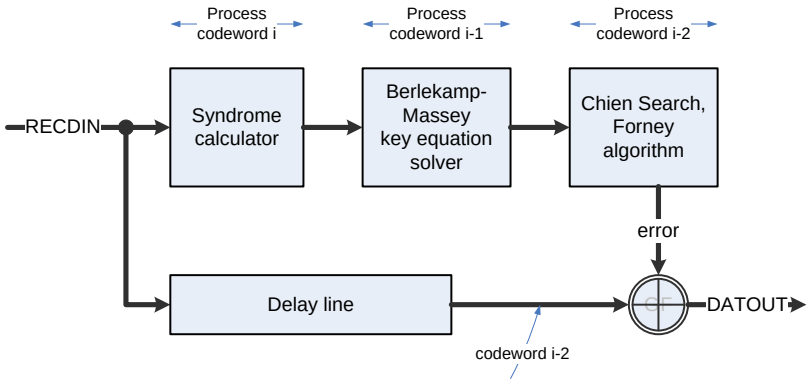
if tt=8 then 16 > 2 * r + e . Specific case is 16 > 2 * 7 + 1 in that case 7 errors and 1 erasure mixture can be correctable.

3.1.5 CoreRSDEC Block Diagram

Figure 3 shows a simplified block diagram of the CoreRSDEC. A received codeword comes at the input recdIn of the CoreRSDEC. A Syndrome calculator calculates a set of Syndromes for every received codeword. Next, a key equation solver determines an error location and any error value polynomials. The key equation solver implements the Berlekamp-Massey algorithm. These polynomials are used by the Chien search and Forney algorithm to determine the error locations and values. The Chien-Forney block puts out errors detected. An actual error correction is happening at the m-bit-wide XOR gates.

The three major blocks work concurrently: when the Syndrome calculator processes the codeword (i), the key equation solver processes data relevant to the previous codeword (i – 1), and the Chien-Forney block processes the data relevant to the yet older codeword (i – 2). The delay line serves to balance the processing delay the three major blocks introduce, so the erroneous input symbol and the corresponding error detected get to the symbol-wide XOR gate at the same time.

Figure 3 CoreRSDEC Block Diagram

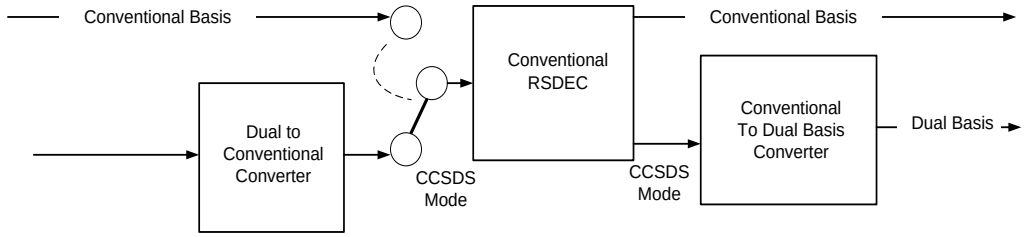


CCSDS Compliance

CoreRSDEC works in Conventional mode or in CCSDS mode. In conventional mode, the decoder parameters supported are carried forward from the previous release. Refer to Table 5.

For making CoreRSDEC to CCSDS-compliant, a dual-to-conventional converter should be optionally used in front of the existing traditional decoder.

Figure 4 CoreRSDEC in CCSDS /Conventional Usage Block Diagram



If the dual basis code comes from a communications channel/encoder, a dual basis needs to be applied to conventional converter. Then the decoder will always face conventional code only.

CoreRSDEC provides the dual basis output for the Dual basis input which is referred CCSDS mode.

CoreRSDEC provides the conventional basis output for the conventional basis input, which is referred as conventional mode.

3.1.6 CoreRSDEC Timing

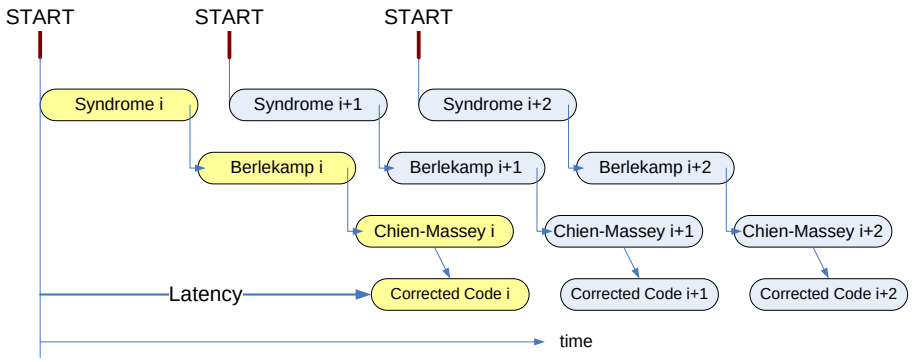
Latency

Figure 5 shows a sequence of the decoding process. The Syndrome calculator analyzes incoming codeword symbols in real time. It keeps doing this while a codeword is entering the decoder. Ovals called Syndrome i, i + 1, and i + 2 in Figure 5 shows time intervals when the Syndrome calculator is busy processing incoming codewords i, i + 1, and i + 2, respectively. Once the codeword is over, the Syndrome calculator transfers the syndromes to the Berlekamp block. The latter takes a certain time to compute the necessary polynomials. The ovals Berlekamp i, i + 1, and i + 2 in Figure 5 show the time intervals the Berlekamp block takes to perform appropriate computations. Once the Berlekamp results are ready, they are transferred to the Chien-Massey block. This produces the final result of the decoder with a small latency of a few clock periods. The ovals Corrected Code i, i + 1, and i + 2 reflect the time intervals when the corrected code is being put out. The overall delay from a START signal to the moment the decoder starts generating the final result is shown in Figure 5 as the latency.

As seen in Figure 5, the latency does not depend on the time interval between the incoming codewords, as the Berlekamp computation immediately follows Syndrome completion, and then Chien-Massey starts right after the Berlekamp is over.

In most practical cases, the latency equals approximately twice the codeword length. In some cases, namely when the parameters are set so that n < 9t + 8, the latency equals approximately twice the Berlekamp processing time. The Decoder Processing Cycle section provides a detailed explanation of the difference. The precise latency value can be measured as a time interval between the input START signal and the output RDY signal. The core generates the RDY signal to mark the time interval when the corrected code is coming out.

Figure 5 CoreRSDEC Latency



3.1.7 Decoder Processing Cycle

In Figure 5, Syndrome calculation, Berlekamp, and Chien-Massey times are shown equal for simplicity. In reality, this is not a common case. Syndrome and Chien-Massey calculations take equal time intervals of n clock cycles, but the Berlekamp algorithm time does not depend on the codeword length of n but rather on the code correction capability of t. The Berlekamp block takes 9t + 8 clock cycles to compute the necessary polynomials. The larger of those two times, n and 9t + 8, determines the decoder processing cycle.

For certain n and t parameter selections, the decoder might not be ready for the next codeword if it comes immediately after the previous one. Depending on actual values of n and t, two distinct situations are possible. It is important to know which situation the decoder faces based on parameter selection.

Figure 6 shows a practical case when the codeword length is larger than the Berlekamp computation time. In this example, the Syndrome calculation and Chien-Massey time intervals each are equal to the codeword length of n clock cycles. The Berlekamp block is busy only a fraction of the time; the rest of the time it is idle. The decoder processing cycle that determines the minimum interval between two consecutive START signals equals n clock cycles. This means the codewords can come without gaps between.

Figure 6 Codeword Length Determines Minimum Inter-Start Interval

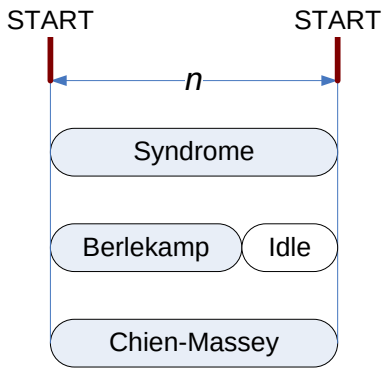
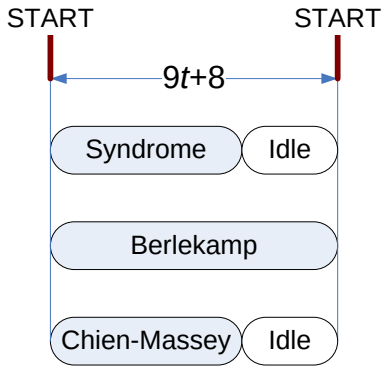


Figure 6 shows a different situation where the Berlekamp computation takes longer than the codeword length. Such a situation occurs when the correction capacity t is relatively large and the codeword length n is relatively small. In Figure 6, the Berlekamp computation determines the decoder processing cycle while the Syndrome calculator and Chien-Massey block are idle during a fraction of the cycle. The CoreRSDEC is not ready to accept another codeword in n clock cycles, even though the Syndrome calculation is over, because the Berlekamp block is still busy processing the data of the previous codeword.

Figure 7 Berlekamp Stage Determines Minimum Inter-Start Interval

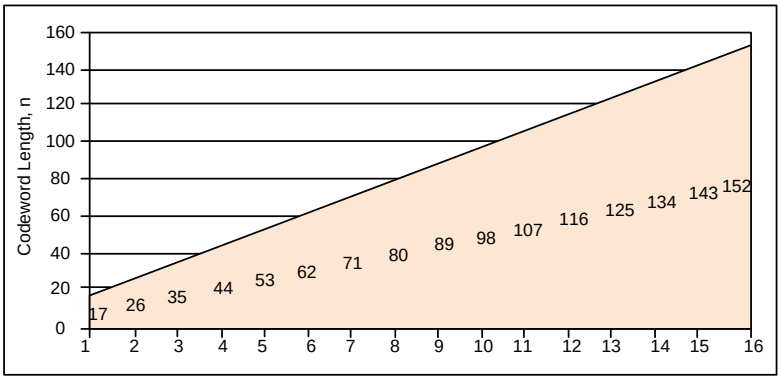


If parameters are selected so that 9t + 8 > n, the decoder is in the situation depicted in Figure 7. There needs to be a gap between incoming codewords of at least n – 9t – 8 clock cycles. Otherwise, the decoder is in the situation of Figure 7, where there is no need for the gap.

Figure 7 shows the Berlekamp processing time depending on the selected parameter t.

For example, at t = 3, the Berlekamp computation takes 35 clock cycles. The decoder can accept any codewords longer than 35 symbols with no gaps in between. Once a crossing point of the selected parameter values of t and n, falls in the white area of the Figure 7, a gap is not needed between the incoming codewords.

Figure 8 Berlekamp Computation Time vs t



The same Figure 8 can be used to determine approximate decoder latency. If the above crossing point falls in the white area, the latency is 2n, otherwise the latency is 18t + 16.

Note: When both ERA and CCSDS-16 parameters are enabled(that is, ERA = 1 and CCSDS = 2), then the favourable decoder latency lies in the range between: 680 to 780 (2.6n to 3n) clock cycles.

4 Interface

4.1 Ports

The port signals for CoreRSDEC are described in Table 4 and as shown in Figure 9.

Figure 9 CoreRSDEC I/O Signals

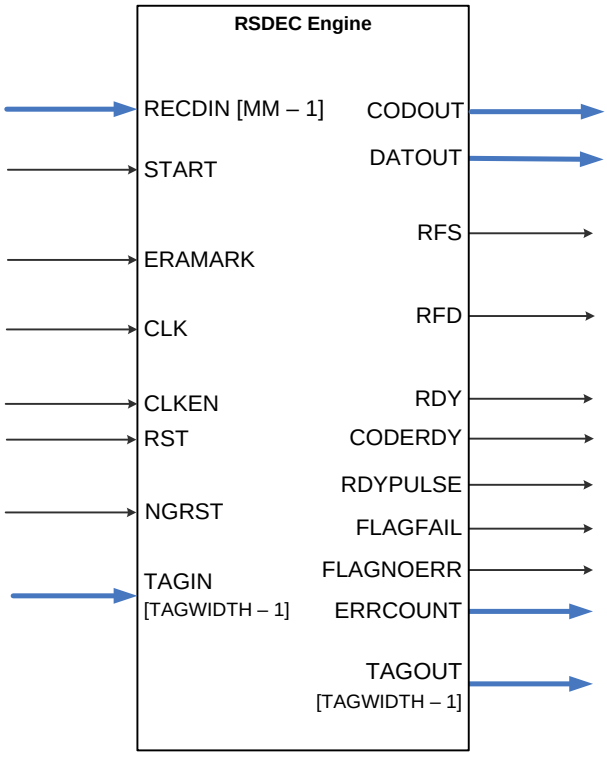


Table 4 I/O Signal Description

Signal	Direction	Description
RECDIN	Input	(Received) codeword input to be decoded. The input bus is m bits wide.
START	Input	Starts a new codeword cycle. It informs the decoder that, at the next clock interval, the first m-bit symbol of the n-symbol codeword appears at RECDIN.
ERAMARK	Input	One-clock-wide erasure pulse marks symbols that are found to be corrupted prior to entering CoreRSDEC.
CLK	Input	Decoder clock signal.
CLKEN	Input	Decoder clock enable signal.
RST	Input	Synchronous reset.
NGRST	Input	Asynchronous reset (active-low).
TAGIN	Input	Optional tag bits the core attaches to every input symbol. Bit width of the tag can be parameterized. Serves to help identify which delayed output symbol or codeword corresponds to an input symbol or codeword.
DATOUT	Output	Corrected data output. The output is m bits wide.
CODOUT	Output	Corrected codeword output. In addition to corrected data, it contains the corrected parity symbols. The output is m bits wide.
RFS	Output	Ready for Start. This signal is active when the decoder is ready to accept a new START signal.
RFD	Output	Ready for Data. This signal is active when the decoder is ready to accept a new codeword at the RECDIN input.
RDY	Output	Corrected Data Ready. This signals that valid, corrected data is present at the core output.
CODERDY	Output	Corrected Codeword Ready. This signals that a valid, corrected codeword—that is, data plus parity symbols—is present at the core output.
RDYPULSE	Output	A clock-wide pulse preceding the RDY signal.
FLAGFAIL	Output	Failure flag. This signals that the CoreRSDEC has failed to correct a codeword.
FLAGNOERR	Output	No Error Flag. This signals that there were no erroneous symbols in a codeword.
ERRCOUNT	Output	Number of erroneous symbols detected in a codeword, up on FLAGFAIL, ERRCOUNT is set to “0” or invalid to consider.
TAGOUT	Output	Output of the tag bits the core attaches to every input symbol. Bit width of the tag can be parameterized. Serves to help identify which delayed output symbol or codeword corresponds to an input symbol or codeword.

4.2 Configuration Parameters

CoreRSDEC generates the CoreRSDEC engine RTL code based on parameters set by the user. CoreRSDEC supports the variations specified in Table 5.

Table 5 CoreRSDEC Configuration Parameters

Name	Valid Values	Description
m	3 to 8	Symbol width, bits.
n	5 to $2^m - 1$	Codeword length, symbols.
t	1 to 16 as long as t < *n* / 2 – 1	Number of corrupted symbols the RS code can correct.
Primitive Polynomial	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 (B0).
Enable Erasure	On, Off	Enables/disables erasure support. Erasure support being enabled substantially increases FPGA resource utilization. Correcting mixture of error and erasure in CCSDS mode. In case of CCSDS-8 maximum of 5 (6 > 2r + e) mixture of error and erasure can be correctable. In case of CCSDS-16 maximum of 11 (12 > 2r + e) mixture of error and erasure can be correctable. In case of CCSDS-8 or CCSDS-16 and Erasure enable but no erasures found can correct tt number errors.
No Error Flag	On, Off	Enables/disables the Error flag.
Error Count	On, Off	Enables/disables Error Count at the output.
Enable Tag	On , Off	Enables / disables the tag support.
Tag width	1 to 10	Bit width of the tag.
Use Micro SRAM	On, Off	For SmartFusion2 FPGA Family On: use the micro static random access memory (uSRAM) Off: use the large SRAM (LSRAM)
CCSDS compatibility	Mode	Conventional CCSDS: 8 (Error correction capacity of 8) CCSDS: 16 (Error correction capacity of 16)
Dual to conventional basis output converter	Enable/Disable	Enable : Provide the conventional basis output for the for CCSDS mode dual basis input Disable: Provide the dual basis output for the for CCSDS mode dual basis input
Family	Family	SmartFusion®2 (19) SmartFusion® (18) Axcelerator® (11) RTAX™-S (12) ProASICPLUS® (14) ProASIC®3 (15) ProASIC3E (16) ProASIC3L (22) Fusion® (17) IGLOO® (20) IGLOOe (21) IGLOOPLUS (23) IGLOO®2 (24) RTG4™ (25) PolarFire(26)

5 Timing Diagrams

5.1 I/O Signal Functionality

5.1.1 NGRST, RST Input

Both signals reset all registers of CoreRSDEC to bring it to an initial state. In the initial state, signals RFS and RFD are active, and the RDY signal is inactive. CoreRSDEC 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.

5.1.2 CLK, CLKEN Input

Clock signal CLK is active on the rising edge.

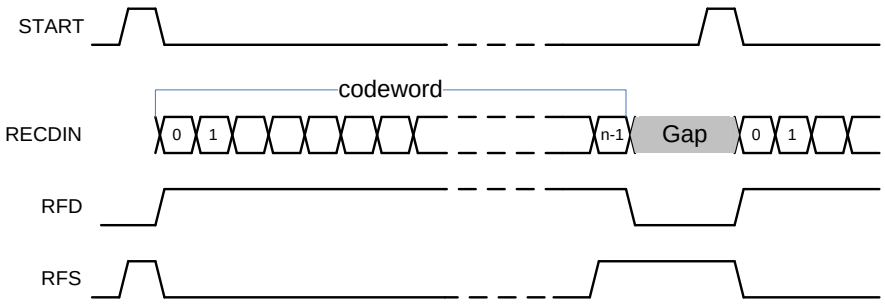
When CLKEN is inactive (LOW), the core is frozen. When core is frozen all inputs except NGRST are ignored and the core retains its current decoding state. The CLKEN going inactive for a cycle is valid and makes the core frozen. When CLKEN returning to active has to be stable for 2 cycles, if not, the core may not retain its current state.

5.1.3 START Input

This signal starts a new codeword cycle. It informs the decoder that, at the next clock interval, the first m-bit data symbol RECDIN 0 of an n-symbol codeword appears on the RECDIN bus (Figure 10). It is assumed that the CLKEN signal is active in Figure 10.

Normally, a codeword source is supposed to issue the START signal once the RFS (Ready for Start) signal goes active. If START is asserted prior to completion of the current codeword—that is, when RFS is still inactive—it will be ignored.

Figure 10 RS Decoder Timing



As shown in Figure 10, the START can be asserted early to repeat the RFS signal. Figure 10 also shows an example of a gap between the incoming codewords. The gap is caused by the START signal being issued later than RFS turned active.

5.1.4 RFS Output

The core asserts this output when it is ready to process another codeword—that is, to accept another START signal. With normal CoreRSDEC functionality, the data source should wait for RFS to go active to issue another START signal. If START is asserted prior to completion of the current codeword—that is when RFS is still inactive—it will be ignored.

When using CoreRSDEC, it is common to send the codewords side-to-side with no gaps between them, which requires connecting the RFS output to the START input of the core.

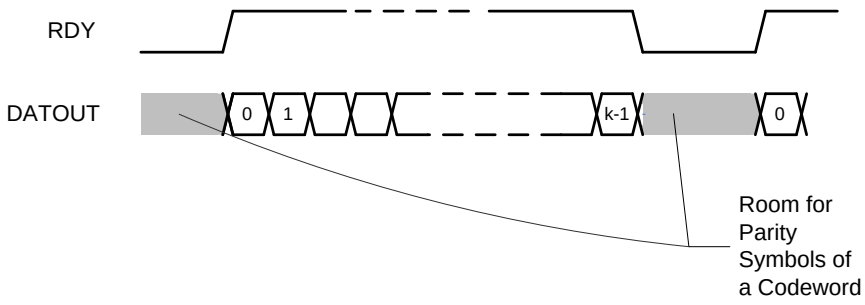
5.1.5 RFD Output

This optional output signal is asserted when CoreRSDEC is ready for a fresh input codeword. Once the core fetches n symbols of a codeword, RFD goes low, thus blocking input data when there is a gap between the input codewords Figure 10. If there is no gap between the incoming codewords, the RFD signal is permanently active.

5.1.6 RDY Output

The optional RDY signal marks an interval of time when decoded, corrected data is present at the CoreRSDEC output DATOUT (Figure 11). Obviously, there are gaps between the output data that, prior to decoding, were used to fit parity symbols.

Figure 11 RDY Signal Accompanies Corrected Output Data



5.1.7 RECDIN Input

The m-bit symbols of the input codewords are supposed to come to this input when the signal RFD is active. The symbols come at every clock without gaps between symbols belonging to the same codeword. Gaps are allowed between codewords only.

5.1.8 DATOUT Output

The corrected data symbols appear one-by-one at the m-bit output. A new output symbol emerges each clock period until all k data symbols of a codeword come out. The signal RDY accompanies the k-symbol sequence.

5.1.9 CODOUT Output

Optional output m-bit bus similar to DATOUT. The CODOUT signal differs from DATOUT in that the former contains corrected parity symbols in addition to the corrected data symbols. In other words, the core puts out the whole corrected codeword through the CODOUT output. The signal CODERDY accompanies the n-symbol sequence. If the input codewords come without gaps in between, the output codewords follow the same pattern.

There is a delay between an input and output codeword. The same delay separates input and output codeword data portions. The delay is termed the Decoder Latency and is explained in the Latency section.

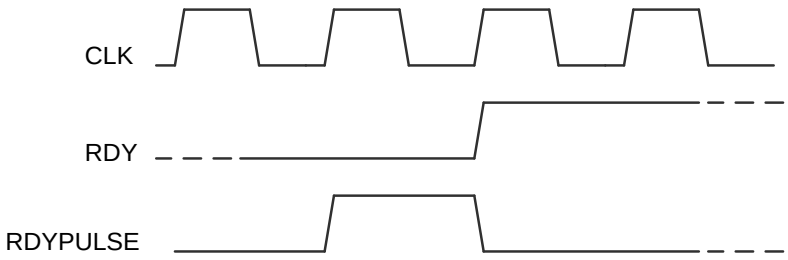
5.1.10 CODERDY Output

The optional CODERDY signal marks an interval of time when a decoded, corrected codeword is present at the CoreRSDEC output CODOUT. Once the input and consequently output codewords come without gaps between them, the CODERDY signal is permanently active.

5.1.11 RDYPULSE Output

This is an optional, short, one-clock signal that immediately precedes the RDY signal (Figure 12).

Figure 12 RDYPULSE Signal



5.1.12 FLAGFAIL Output

This is an optional one-bit flag that alerts that the decoder has detected more errors in a codeword than it could correct.

Usually, it is possible for CoreRSDEC to detect such a situation, but in some cases it is not. The FLAGFAIL signal, as well as other flags, follows an output data portion or a codeword, as shown in Figure 13, which shows the precise timing for the flags the core optionally generates. It can be seen that the flags become valid one clock period after CoreRSDEC completes putting out another codeword. The flags stay valid until the flags relevant to the next codeword become valid.

Figure 13 Flags Refer to the Last Output Data Portion or Codeword

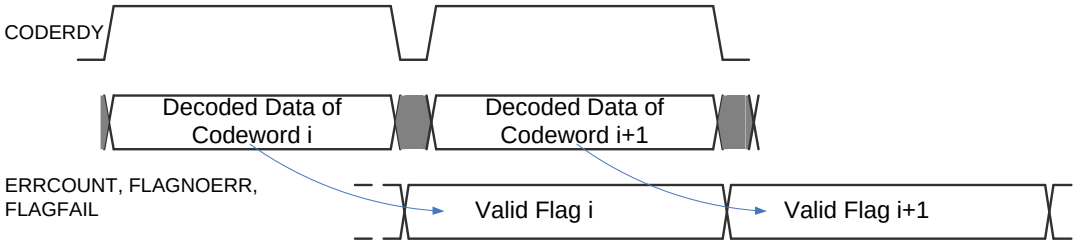
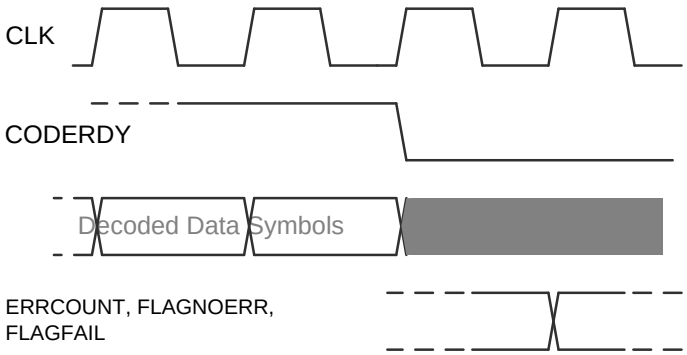


Figure 14 Precise Timing for the Flags



5.1.13 FLAGNOERR Output

This flag goes active if the just-processed codeword does not contain any errors. Erasures do not influence the flag.

5.1.14 ERRCOUNT Output

This flag contains an error count for the just-processed codeword. Erasures do not influence the error count.

5.1.15 ERAMARK Input

This one-bit input marks the positions of the symbols that are known to be erroneous prior to entering CoreRSDEC.

5.1.16 TAGIN, TAGOUT

This optional tag gets attached to arbitrary symbols or codewords selected by a user. The number of TAGIN bits can be parameterized and equals the number of TAGOUT bits. TAGIN gets delayed exactly the same amount of time as the data entering CoreRSDEC. As a result, it is easy to locate any symbol or codeword when it appears at the decoder output.

The tag feature being enabled virtually does not consume FPGA resources if the tag bit width does not exceed 9 – m; otherwise, it may consume an extra on-chip RAM block.

6 Tool Flow

6.1 License

CoreRSDEC requires a RTL license to be used and instantiated. Complete source code is provided for the core.

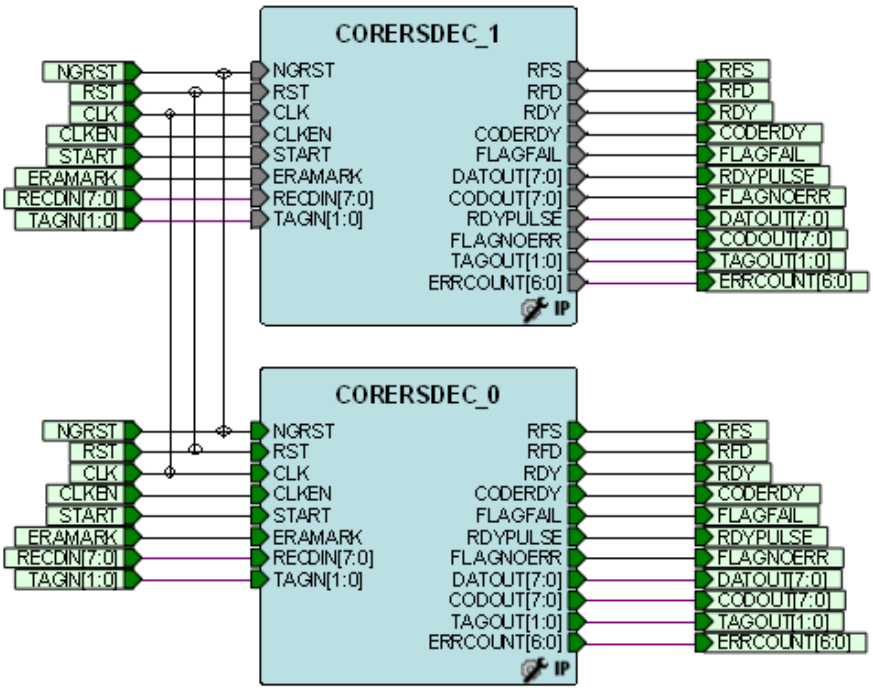
6.2 SmartDesign

CoreRSDEC is available for download in the Libero IP catalog through the web repository. Once it is listed in the catalog, the core can be instantiated using the SmartDesign flow. For information on using SmartDesign to configure, connect, and generate cores, refer to the Libero online help. An example instantiated view is shown in Figure 15.

After configuring and generating the core instance, basic functionality can be simulated using the testbench supplied with the core. The testbench parameters automatically adjust to the core configuration. The core can be instantiated as a component of a larger design. Figure 15 shows an example of a larger design that instantiates two instances of CoreRSDEC. Every instance is configured separately.

Note: CoreRSDEC is compatible with both Libero integrated design environment (IDE) and Libero System-on-Chip (SoC). Unless specified otherwise this document uses the common name Libero to identify Libero IDE and Libero SoC.

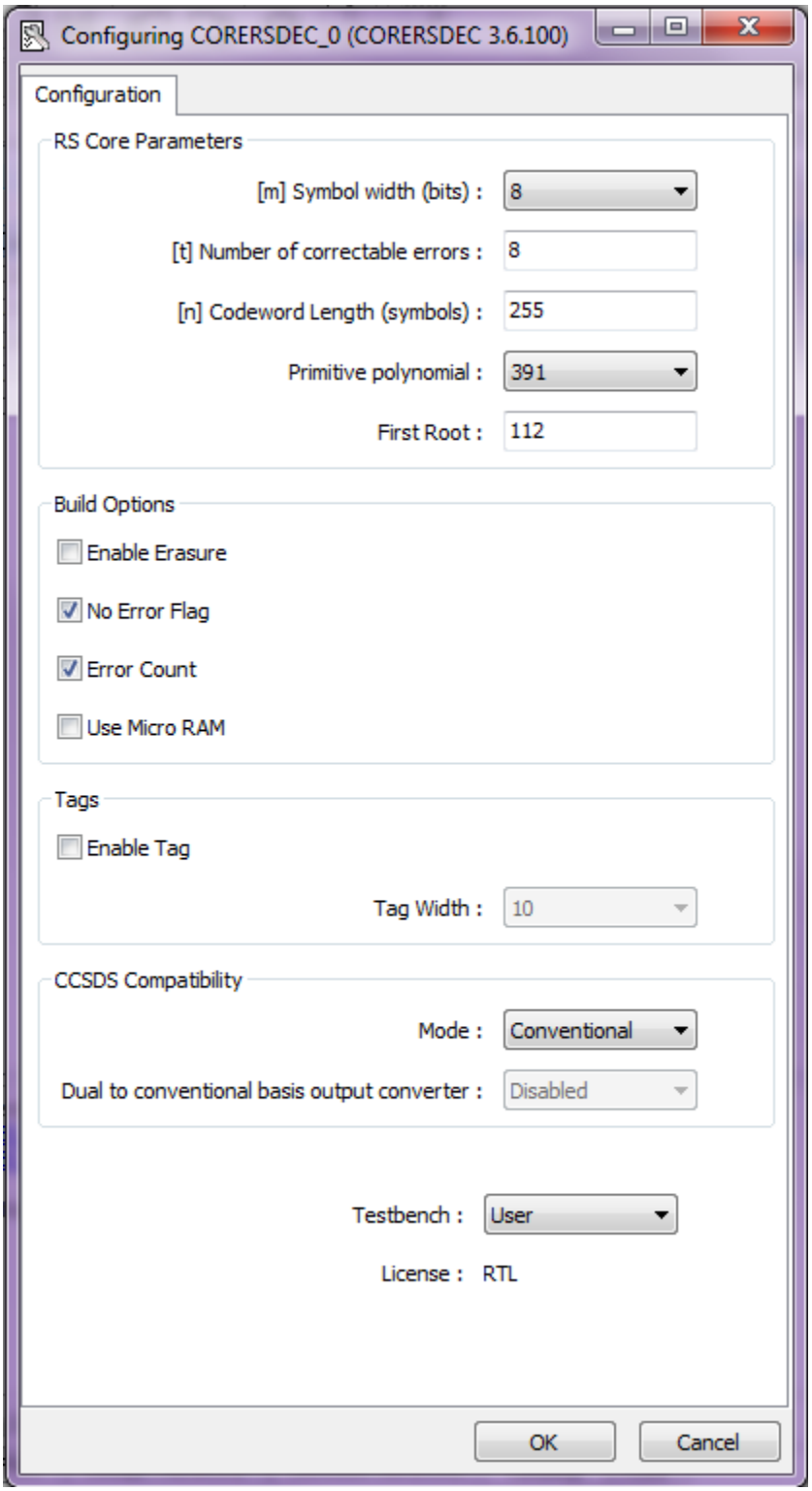
Figure 15 SmartDesign CoreRSDEC Instance View



Note: For RTG4, asynchronous reset ports NGRST must be either tied to a single top-level reset net or tied high so that only synchronous resets RST are used.

6.3 Configuring CoreRSDEC in SmartDesign

Figure 16 Configuring CoreRSDEC in SmartDesign



6.4 Simulation Flows

To run simulations, select the user testbench in the core configuration window. After generating the core, the pre-synthesis testbench hardware description language (HDL) files are installed in Libero. Consider an example of instantiating CoreRSDEC as an IP component named ***top_rsdec***. To run the testbench, set the Libero design root to the core instance **top_rsdec_CORERSDEC_0_CORERSDEC** and run Pre-Synthesized design simulation.

6.5 Synthesis in Libero

To run synthesis on the core, set the design root to the IP component instance ***top_rsdec*** and run the synthesis tool from the Libero Design Flow pane.

6.6 Place-and-Route in Libero

After the design is synthesized, run the compilation and then place-and-route the tools.

7 Testbench

A unified testbench is used to verify and test CoreRSDEC. It is called a user testbench.

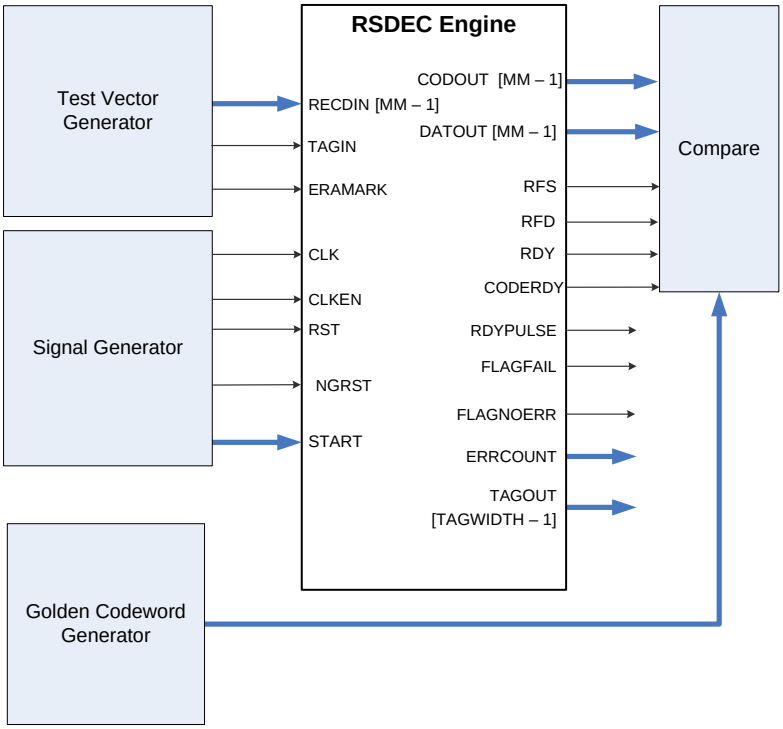
7.1 User Test-bench

Included with the releases of CoreRSDEC is a user testbench that verifies operation of the CoreRSDEC engine.

A simplified block diagram of the user testbench is as shown in Figure 17. The user testbench instantiates the CoreRSDEC engine configured by the user, 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 CoreRSDEC output codeword against the golden codeword vector. Data output of the decoder present on the datOut bus is not tested since it is a part of the output codeword present on the codOut bus. CoreRSDEC automatically generates Verilog or very high speed integrated circuit (VHSIC) HDL—also known as VHDL testbench behavioral code based on the user selection of the core language. Optional outputs such as RDYPULSE, FLAGFAIL, FLAGNOERR, and ERRCOUNT are verified by visual inspection of the simulated waveforms.

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

Figure 17 CoreRSDEC User Testbench



7.2 References

1. Rorabaugh, C. Britton. Error Coding Cookbook. McGraw-Hill, 1995.
2. Sweeney, Peter. Error Control Coding. John Wiley & Sons, 2002.
3. Morelos-Zaragoza, Robert H. The Art of Error Correcting Coding. John Wiley & Sons, 2002.
4. Lin, Shu and Daniel J. Costello. Error Control Coding. Prentice Hall, 2004.

8 System Integration

This IP core is a generic design component to use in a system level design.

9 Ordering Information

9.1 Ordering Codes

CoreRSDEC can be ordered through your local Sales Representative. It should be ordered using the following number scheme: CoreRSDEC-XX, where XX is listed in Table 6.

Table 6·Ordering Codes

XX	Description
RM	Available as Verilog and VHDL RTL source code