
Chipsmall
深圳市津利帝科技有限公司
Geelead Technology Limited
Chipsmall
采购部

Lattice Semiconductor Corporation

LatticeCORE™

# Dynamic Block Reed-Solomon Decoder User's Guide

December 2010
IPUG52_01.6

Lattice Semiconductor Corporation

# Table of Contents

Chapter 1:

# Introduction

Reed-Solomon codes are widely used in various communications and storage applications for forward error correction. Reed-Solomon codes are well suited for burst error correction and are frequently used as outer codes in communication systems. A Reed-Solomon Decoder performs detection and correction of the encoded data at the receiver. Lattice's Dynamic Block Reed-Solomon Decoder (RS Decoder) IP core is compliant with several industry standards including the more recent IEEE 802.16-2004 and can be custom configured to support other non-standard applications as well. The RS Decoder supports a wide range of symbol widths and allows the user to define the field polynomial, generator polynomial and several other parameters.

The newer standards like IEEE 802.16-2004 require the use of Reed-Solomon codes with dynamically varying block sizes. Lattice's RS Decoder IP core provides an ideal solution that meets such needs of today's forward error correction world. This core allows the block size and number of check symbols to be varied dynamically through input ports. Lattice's RS Decoder IP can be used with Lattice's RS Decoder for a complete Reed-Solomon code based forward error correction application. For more information on these and other IP products for forward error correction, refer to the Lattice web site at www.latticesemi.com/products/intellectualproperty.

## Quick Facts

Table 1-1 through Table 1-9 give quick facts about the RS Decoder IP core for LatticeEC™, LatticeECP™, LattceECP2™, LattticeSC™, LatticeSCM™, LatticeXP™, LatticeECP2M™, LatticeXP2™, and LatticeECP3™ devices.

*Table 1-1. RS Decoder IP core for LatticeEC Devices Quick Facts*

| | | **RS Decoder IP Configuration** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **OC-192** | **CCSDS** | **DVB** | **ATSC** | **IEEE 802.16-2004 SCa/OFDM** | **IEEE 802.16-2004 SC** |
| **Core Requirements** | FPGA Families Supported | LatticeEC | | | | | |
| | Minimal Device Needed | LFEC1E | LFEC3E | LFEC1E | LFEC3E | LFEC3E | LFEC3E |
| **Resource Utilization** | Targeted Device | LFEC20E-5F672C | | | | | |
| | LUTs | 1100 | 2000 | 1200 | 1500 | 1900 | 2100 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1100 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond™ 1.0 or ispLEVER® 8.1 | | | | | |
| | Synthesis | Synopsys® Synplify™ Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec® Active-HDL™ 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics® ModelSim™ SE 6.3F | | | | | |

**Table 1-2. RS Decoder IP core for LatticeECP Devices Quick Facts**

| | | RS Decoder IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | OC-192 | CCSDS | DVB | ATSC | IEEE 802.16-2004 SCa/OFDM | IEEE 802.16-2004 SC |
| **Core Requirements** | FPGA Families Supported | LatticeECP | | | | | |
| | Minimal Device Needed | LFECP6E | | | | | |
| **Resource Utilization** | Targeted Device | LFECP20E-5F672C | | | | | |
| | LUTs | 1100 | 2000 | 1200 | 1500 | 1900 | 2100 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1100 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

**Table 1-3. RS Decoder IP core for LatticeECP2 Devices Quick Facts**

| | | RS Decoder IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | OC-192 | CCSDS | DVB | ATSC | IEEE 802.16-2004 SCa/OFDM | IEEE 802.16-2004 SC |
| **Core Requirements** | FPGA Families Supported | LatticeECP2 | | | | | |
| | Minimal Device Needed | LFE2-6E | | | | | |
| **Resource Utilization** | Targeted Device | LFE2-50E-7F672C | | | | | |
| | LUTs | 1100 | 2000 | 1200 | 1500 | 1800 | 2100 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1100 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-4. RS Decoder IP core for LatticeSC Devices Quick Facts*

| | | RS Decoder IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | **OC-192** | **CCSDS** | **DVB** | **ATSC** | **IEEE 802.16-2004 SCa/OFDM** | **IEEE 802.16-2004 SC** |
| **Core Requirements** | FPGA Families Supported | LatticeSC | | | | | |
| | Minimal Device Needed | LFSC3GA15E | | | | | |
| **Resource Utilization** | Targeted Device | LFSC3GA25E-7F900C | | | | | |
| | LUTs | 1200 | 2000 | 1200 | 1600 | 1900 | 2200 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1000 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-5. RS Decoder IP core for LatticeSCM Devices Quick Facts*

| | | RS Decoder IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | **OC-192** | **CCSDS** | **DVB** | **ATSC** | **IEEE 802.16-2004 SCa/OFDM** | **IEEE 802.16-2004 SC** |
| **Core Requirements** | FPGA Families Supported | Lattice SCM | | | | | |
| | Minimal Device Needed | LFSCM3GA15EP1 | | | | | |
| **Resource Utilization** | Targeted Device | LFSCM3GA25EP1-7F900C | | | | | |
| | LUTs | 1200 | 2000 | 1200 | 1600 | 1900 | 2200 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1000 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-6. RS Decoder IP core for LatticeXP Devices Quick Facts*

| | | RS Decoder IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | OC-192 | CCSDS | DVB | ATSC | IEEE 802.16-2004 SCa/OFDM | IEEE 802.16-2004 SC |
| **Core Requirements** | FPGA Families Supported | Lattice XP | | | | | |
| | Minimal Device Needed | LFXP3E | | | | | |
| **Resource Utilization** | Targeted Device | LFXP20E-5F484C | | | | | |
| | LUTs | 1100 | 2000 | 1200 | 1500 | 1900 | 2100 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1100 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-7. RS Decoder IP core for LatticeECP2M Devices Quick Facts*

| | | RS Decoderr IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | OC-192 | CCSDS | DVB | ATSC | IEEE 802.16-2004 SCa/OFDM | IEEE 802.16-2004 SC |
| **Core Requirements** | FPGA Families Supported | Lattice ECP2M | | | | | |
| | Minimal Device Needed | LFE2M20E | | | | | |
| **Resource Utilization** | Targeted Device | LFE2M35E-7F484C | | | | | |
| | LUTs | 1100 | 2000 | 1200 | 1500 | 1800 | 2100 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1100 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-8. RS Decoder IP core for LatticeXP2 Devices Quick Facts*

| | | RS Decoder IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | **OC-192** | **CCSDS** | **DVB** | **ATSC** | **IEEE 802.16-2004 SCa/OFDM** | **IEEE 802.16-2004 SC** |
| **Core Requirements** | FPGA Families Supported | Lattice XP2 | | | | | |
| | Minimal Device Needed | LFXP2-5E | | | | | |
| **Resource Utilization** | Targeted Device | LFXP2-17E-7FT256C | | | | | |
| | LUTs | 1100 | 2000 | 1200 | 1500 | 1800 | 2100 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1100 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

*Table 1-9. RS Decoder IP core for LatticeECP3 Devices Quick Facts*

| | | RS Decoder IP Configuration | | | | | |
|---|---|---|---|---|---|---|---|
| | | **OC-192** | **CCSDS** | **DVB** | **ATSC** | **IEEE 802.16-2004 SCa/OFDM** | **IEEE 802.16-2004 SC** |
| **Core Requirements** | FPGA Families Supported | Lattice ECP3 | | | | | |
| | Minimal Device Needed | LFE3-35EA | | | | | |
| **Resource Utilization** | Targeted Device | LFE3-95E-8FN672CES | | | | | |
| | LUTs | 1100 | 2000 | 1200 | 1500 | 1800 | 2100 |
| | sysMEM EBRs | 2 | 2 | 2 | 2 | 3 | 3 |
| | Registers | 900 | 1500 | 900 | 1100 | 1400 | 1600 |
| **Design Tool Support** | Lattice Implementation | Lattice Diamond 1.0 or ispLEVER 8.1 | | | | | |
| | Synthesis | Synopsys Synplify Pro for Lattice D-2009.12L-1 | | | | | |
| | Simulation | Aldec Active-HDL 8.2 Lattice Edition | | | | | |
| | | Mentor Graphics ModelSim SE 6.3F | | | | | |

## Features

- 3- to 12-Bit Symbol Width
- Configurable Field Polynomial
- Configurable Generator Polynomial: Starting Root and Root Spacing
- User-defined Codewords
  - Maximum of 4095 symbols
  - Maximum of 256 check symbols
  - Shortened codes
- Off-the-shelf Support for the Following Communication Standards:
  - OC-192
  - DVB

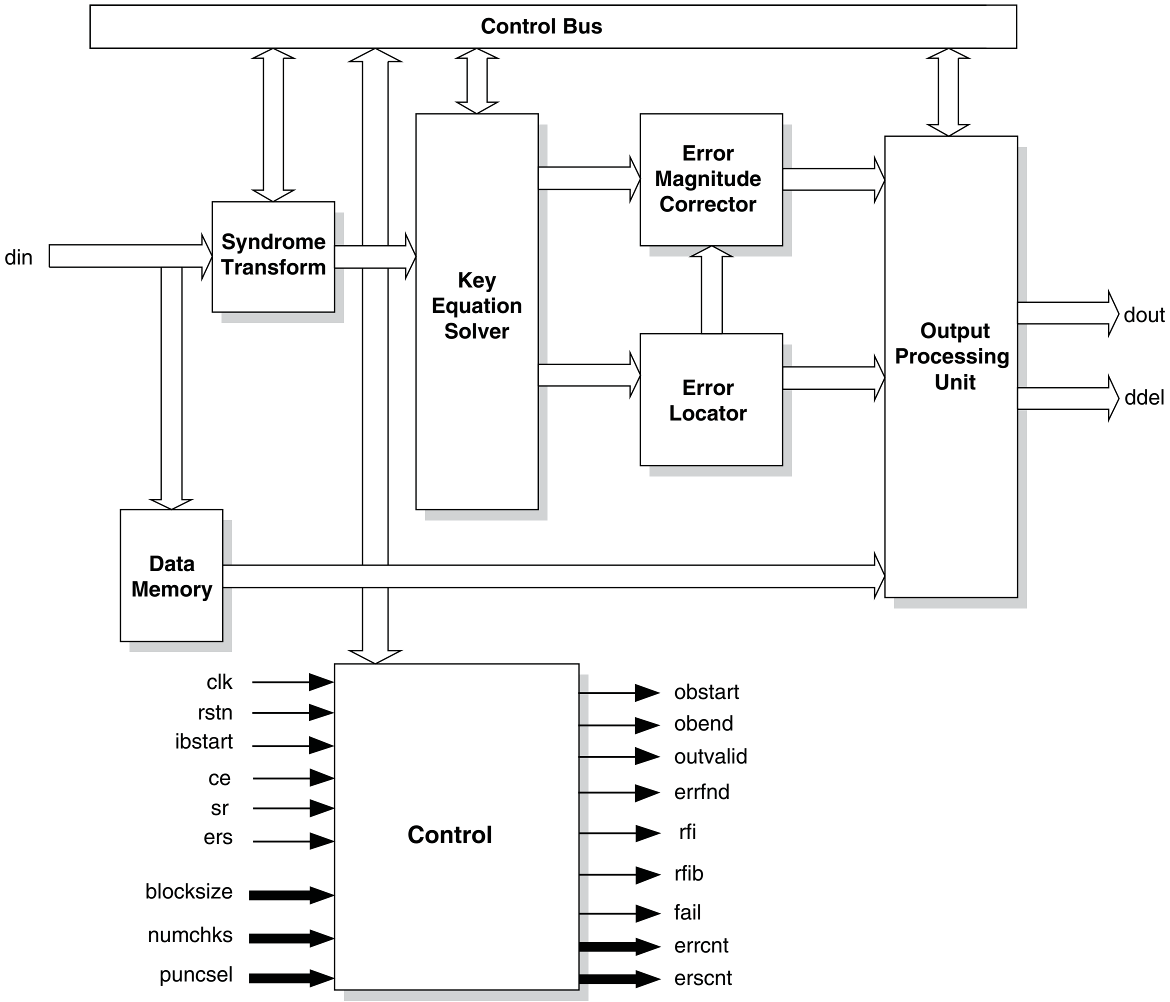
  - CCSDS
  - ATSC
  - IEEE 802.16-2004 WirelessMAN-SCa/OFDM
  - IEEE 802.16-2004 WirelessMAN-SC
- Fully Synchronous
- Systematic Decoder
- Full Handshaking Capability
- Dynamically Variable Block Size
- Dynamically Variable Check Symbols
- Error, Erasure and Puncturing Modes
- Error Measurement Information

# Functional Description

A block diagram of the RS Decoder is shown in Figure 2-1. The RS Decoder IP is comprised of the Syndrome Transform, Key Equation Solver, Error Locator, Error Magnitude Corrector, Data Memory and Output Processing blocks.

## Block Diagram

*Figure 2-1. RS Decoder Block Diagram*

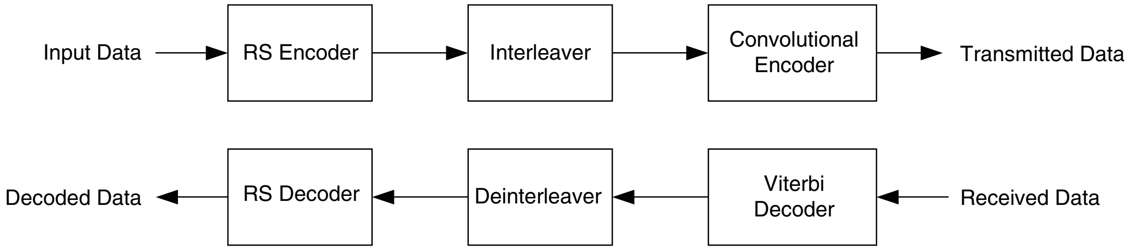


## General Description

Reed-Solomon codes are used to perform Forward Error Correction (FEC). FEC introduces controlled redundancy in the data before it is transmitted to allow error correction at the receiver. The redundant data (check symbols) are transmitted with the original data to the receiver. An RS Decoder is used in the receiver to correct any transmission errors. This type of error correction is widely used in data communications applications such as Digital Video Broadcasting (DVB) and Optical Carriers (i.e. OC-192).

Reed-Solomon codes are written in the format RS(`n`,`k`) where `k` is the number of information symbols and `n` is the total number of symbols in a codeword or block. Each symbol in the codeword is `wsymb` bits wide. The RS Decoder performs detection and correction of encoded data available at the receiver after demodulation. The RS encoded data is then processed to determine whether any errors have occurred during transmission. Once the number of errors is determined, the decoder decides if they are within the range of correction. After determining this, the decoder corrects the errors in the received data. A typical application of space signal processing is shown in Figure 2-2.

*Figure 2-2. Application of Reed-Solomon Code in a Space Communication System*



Reed-Solomon codes are defined on a finite field known as Galois field. The size of the field is determined by the symbol width, wsymb, and is equal to 2wsymb. When n is less than its maximum value of 2wsymb-1, the corresponding code RS(`n`,`k`) is referred to as a shortened code.

Reed-Solomon codes are characterized by two polynomials: the generator polynomial and the field polynomial. The field polynomial defines the Galois field where the information and check symbols belong. The generator polynomial determines the check symbol generation and it is a prime polynomial for all codewords (i.e. all codewords are exactly divisible by the generator polynomial). Both the field and the generator polynomials are user configurable.

## Field Polynomial

The field polynomial is defined by its decimal value (f). The decimal value of a field polynomial is obtained by setting x = 2 in the polynomial. For example, the polynomial $x^2 + x + 1$ in decimal value is $2^2 + 2 + 1 = 7$. The field polynomial can be specified as any prime polynomial with decimal value up to $2^{wsymb+1} - 1$.

## Generator Polynomial

The generator polynomial determines the value of the check symbols. The generator polynomial can be defined by the parameters starting root (`gstart`) and root spacing (`rootspace`). The general form of the generator polynomial is given by:

$$g(x) = \mathop{\pi}\limits_{i=0}^{n-k-1} (x - \alpha^{rootspace \cdot (gstart + i)}) \qquad (1)$$

where $\alpha$ is called the primitive element of the field polynomial. For a binary Galois field GF(2), $\alpha$ is equal to 2.

## Shortened Codes

When the size of the Reed-Solomon codewords, `n`, is less than the maximum possible size, $2^{wsymb-1}$, they are called shortened codes. For example, RS (204,188) when `wsymb` = 8 is a shortened code.

## Systematic Decoder

The decoder can only decode data encoded by a systematic Reed-Solomon Encoder. In a systematic encoder, the information symbols are unchanged and are followed by check symbols in the output.

## Decoding Modes

The decoder can support Error, Erasure and Puncturing modes. In the error mode no information is available about the symbols in error. In this mode the decoder needs to compute both position and magnitude of the error symbols. In the erasure mode the user can dynamically indicate the erased symbols using the input port `ers`. Erased symbols are those symbols in error whose positions are known in advance. Error mode can be thought of a special case of Erasure mode, when number of erased symbols is zero. Therefore it is not necessary to identify all correctable errors as erasures through the input port `ers` in the erasure mode and combinations of errors and erasures can be used. If erased symbols are known and the position of the erased symbols can be dynamically indicated using `ers`, then erasure mode is useful. The symbol correction capability of the decoder increases since the position of the symbol in error is already known and only the magnitude needs to be computed. Generally erasure support substantially increases the decoder latency and resource utilization. Puncturing mode is an optimized version of the erasure mode and can be used when the position of the erased symbols is known in advance. The user can define a maximum of (`n` - `k`) puncture patterns and can dynamically select one of these patterns using the input port `puncsel`. The format for the puncture pattern file is explained in a later section. The decoder will be able to correct the errors and erasures successfully if the following conditions are satisfied.

For the Error mode, the number of correctable errors $E_{err}$ is given by

$E_{err}$ = (`n` - `k`) /2, when `Block size type` is constant.

$E_{err}$ = (`Number of check symbols`) /2, when `Block size type` is variable and `Variable check symbols` is not defined.

$E_{err}$ = (value on `numchks` port) /2, when `Variable check symbols` is defined.

For the Erasure and Puncturing modes, the number of correctable errors $E_{err}$ and the number of correctable erasures $E_{ers}$ (given through the input port `ers`) are bound by the following relations

(2 * $E_{err}$ + $E_{ers}$ ð (`n` - `k`)) and ($E_{ers}$ ((`n` - `k` - 2)), when `Block size type` is constant.

(2 * $E_{err}$ + $E_{ers}$ ð (`Number of check symbols`)) and ($E_{ers}$ ð (`Number of check symbols` - 2)), when `Block size type` is variable.

# Functional Description

A block diagram of the RS Decoder is shown in Figure 2-1. The RS Decoder is comprised of the Syndrome Transform, Key Equation Solver, Error Locator, Error Magnitude Corrector, Data Memory and Output Processing blocks.

The data received by the RS Decoder is Reed-Solomon encoded data. This data is a representation of a polynomial in a Galois Field. If there are no errors in the received data, the data polynomial will evaluate to zero at the roots of the generator polynomial. This result is obtained because the roots of the generator polynomial and the received data polynomial are the same when there are no errors. If the received data has been corrupted during the transmission, the polynomial will not evaluate to zero. The RS Decoder can construct the syndrome polynomial by evaluating the received polynomial at all the roots of the generator polynomial. Once the syndrome polynomial has been constructed, it can be used to solve the Error Locator polynomial and Error Evaluator polynomial. Using these two polynomials, the decoder can find the error locations and magnitudes. Finally, the decoder can correct the errors in the received data, provided the errors are in the range of correctable errors (determined by the level of encoding that has been performed).

If there are errors in the received codeword, it can be expressed as follows:

$r(x) = c(x) + e(x)$

where:

$c(x)$ is the Transmitted codeword

$r(x)$ is the Received codeword

$e(x)$ is the Error polynomial

The syndrome polynomial $S(x)$ is obtained by evaluating the received word at each root of the generator polynomial. The Error Locator polynomial $\Lambda(x)$ is orthogonal to the syndrome polynomial in the Galois field. This can be represented as:

$S(x)\Lambda(x) =$ ¾(x) mod $x^{2t}$

where:

¾(x) is the Error Evaluator polynomial.

2t is the number of check symbols introduced in the encoder.

The following sections describe the function of each block of the RS Decoder.

## Syndrome Transform

The Syndrome Transform (also called Syndrome Generation) block evaluates the received codeword of the generator polynomial. If the received data contains an error, the syndrome polynomial generated will be non-zero. If the received data has no error, the syndrome polynomial is zero, and the data is passed out of the decoder without any error correction.

## Key Equation Solver

This is the heart of the RS Decoder. This block generates the Error Locator polynomial $\Lambda(x)$ (also known as the “Key Equation” as it is the key to solve the decoding problem). After the Error Locator polynomial has been determined, it is used to compute the Error Evaluator polynomial ¾(x)

## Error Locator

This block is implemented using the Chien-search method. Essentially, this method evaluates the Error Locator polynomial at all the elements in the Galois Field. The Error Locator polynomial evaluates to zero at its roots. The Chien-search takes up to `m` cycles, where `m` is the number of elements in the Galois Field, to determine all the roots. If the roots are determined before `m` cycles are over, the search is terminated early.

## Error Magnitude Corrector

Once the location of the error has been determined, the Error Magnitude Corrector evaluates the evaluator polynomial at that root. It uses the result to calculate the value of the error at the given location. Once this has been determined, the value is added to the received word to recover the original data. The addition occurs only when the Error Locator polynomial evaluates to zero.

## Control Unit

The control unit handles the interface, pipelining and handshaking communication between the various blocks and the I/O ports. The control circuit moves the data without processing it through the decoder when no error is detected. Similarly, when the number of errors exceeds the maximum range of correction, the control circuit stops all data processing activities. The control circuit interacts with the other blocks to generate the status signals like `obstart`, `obend`, `outvalid`, `rfib`, `errfnd`, `errcnt`, `erscnt` and `fail`. Once the block has been processed, the control circuit sends out the rfi signal to the output to start the processing of the next data block.

## Basis Conversion Modules

When core configuration is selected as CCSDS, then two additional Basis Conversion modules are added to the RS Decoder. These modules comply with the CCSDS specification. Dual-basis to normal polynomial-basis conversion module is added after the din input port and normal polynomial-basis to dual-basis conversion module is added before the dout output port.

## Variable Block Size

In the constant `Block size type` option, the block size value and number of information symbols are provided as constant values through the RS Decoder GUI before core generation. For variable `Block size type` option, the block size value is provided dynamically through the input port `blocksize`. The number of the information symbols is calculated from the block size value provided through the input port and the number of check symbols. The number of check symbols can be either constant and defined in the GUI or variable and given through the input port `numchks`, depending on the parameter Variable check symbols. Once block size value, number of check symbols and number of information symbols are known then the core operates in the same way as when block size was constant.

### Variable Check Symbols

This option can be used when there is requirement for variable error correction capability. One example of this type of application is IEEE 802.16-2004 WirelessMAN-SC. In this option the number of check symbols value is provided dynamically through the input port numchks. Dynamically `Variable check symbols` option is available only for the Error Decoding mode.

## Puncturing Pattern File Format

This file contains the pre-defined puncture patterns that are selected using the `puncsel` signal. This file is necessary when `Decoding mode` is selected as Puncturing. This file should have the ".cfg" extension. The RS Decoder IP core GUI requires this file during core configuration. The format and a sample of this file is given below, followed by a brief explanation.

Format:

| | |
|---|---|
| <n1> <n2> <n3> ....<nN> | First line lists the number of punctured symbols in each pattern. N is the total number of puncture patterns. There should be N lines following this line, one for each pattern |
| <p1> <p2> .... <pn1><br><p1> <p2> ... <pn2><br>... | These are the actual puncture patterns. Each line contains one pattern. p1, p2, etc. in each pattern are the positions of the punctured symbols from the end of the block for that pattern. |
| ... | |
| <p1> <p2> ... <pnN> | |

The number of punctured symbols in the first line, <ni>, can be set to zero to indicate there is no puncturing for that pattern.

Sample content:

```
0 4 8 12
0
0 1 2 3
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8 9 10 11
```

In the sample content above, the first line contains the number of punctured symbols for each puncture pattern. 0 indicates that for the first puncture pattern there are 0 symbols punctured. 4 indicates that for the second puncture pattern there are 4 symbols punctured. 8 indicates that for the third puncture pattern there are 8 symbols punctured. 12 indicates that for the fourth puncture pattern there are 12 symbols punctured. The first puncture pattern is defined on line 2 and the second puncture pattern is defined on line 3, and so on. Each puncture pattern lists the position of the punctured symbol starting from the end of the block. This means that the last symbol is numbered as 0 and the next to the last symbol is numbered as 1, and so on. Therefore, for the second puncture pattern defined on line 3, number 0 indicates the last symbol in the block is punctured and number 1 indicates the second from the last symbol is punctured. The values are entered in decimal format.

### Default Field Polynomials

The default field polynomials used in the GUI for different symbol widths are given in Table 2-1. The user, however, can enter any valid polynomial.

*Table 2-1. Default Field Polynomials*

| Symbol Width | Default Field Polynomial | Decimal Value |
|---|---|---|
| 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 |
| 9 | $x^9 + x^4 + 1$ | 529 |
| 10 | $x^{10} + x^3 + 1$ | 1033 |
| 11 | $x^{11} + x^2 + 1$ | 2053 |
| 12 | $x^{12} + x^6 + x^4 + x + 1$ | 4179 |

## Signal Descriptions

Table 2-2 shows the definitions of the interface signals available with the RS Decoder IP Core.

*Table 2-2. Interface Signal Descriptions*

| Port | Bits | I/O | Description |
|---|---|---|---|
| **All Configurations** | | | |
| `clk` | 1 | I | System clock. This is the reference clock for input and output data. |
| `rstn` | 1 | I | System wide asynchronous active-low reset signal. |
| `ibstart` | 1 | I | Indicates that the data on `din` is the first information symbol of a new codeword. |
| `din` | 3 - 12 | I | Input data port. The `wsymb` parameter defines the port width of this signal. |
| `dout` | 3 - 12 | O | Output data port. The `wsymb` parameter defines the port width of this signal. |
| `obstart` | 1 | O | Output block start. Indicates the first output data of the codeword on the `dout` port. |
| `obend` | 1 | O | Output block end. Indicates the last output data of the codeword on the `dout` port. |
| `outvalid` | 1 | O | Output data valid. Indicates valid data is present on `dout`. |
| `errfnd` | 1 | O | Error found indicator. Asserted at the same time `obend` is asserted if the block has at least one symbol in error. |
| `rfi` | 1 | O | Ready for input. Indicates the decoder is ready to receive input data. Typically, this signal is high when the core is ready to read input symbols. This signal is low when the decoder is busy processing a previous block of data and cannot accept new block of data. |
| **For Variable Check Symbols (When the Parameter `Variable check symbols` is Yes)** | | | |
| `rfib` | 1 | O | Ready for input block. Indicates that the decoder is ready to receive the first information symbol in the block. |
| `numchks` | 2 - 9 | I | This port is used to provide the variable number of check symbols value. The width of this port is defined as the number of bits required to represent the `Max. number of check symbols` parameter value provided by the user. The width of this port is defined as follows: ceil(log2(`Max. number of check symbols`)). The value at this port is read only when `ibstart` is high. The operator ceil() stands for the next higher integer. |
| **For Variable Block Size Type Only (When the Parameter `Variable block size` is "Yes")** | | | |
| `blocksize` | 3 - 12 | I | Variable block size value. The value at this port is read only when `ibstart` is high. The `wsymb` parameter defines the port width of this signal. |

*Table 2-2. Interface Signal Descriptions (Continued)*

| Port | Bits | I/O | Description |
|---|---|---|---|
| **For Puncturing Mode (when `Decoding mode` is "Puncturing" and `Number of puncture patterns` is More than 1)** | | | |
| `puncsel` | 1 - 8 | I | Puncture pattern select signal. The value on this port selects the puncturing pattern from the number of predefined patterns for the current block of data. The width of this port is defined as ceil(log2(`Number of puncture patterns`)). The value at this port is read only when `ibstart` is high. |
| **For Erasure mode** | | | |
| `ers` | 1 | I | Erasure. Asserted to indicate the input data symbol at the `din` port is erased. |
| **Optional I/Os** | | | |
| `ce` | 1 | O | Clock enable. While this is de-asserted, the decoder will ignore all other synchronous inputs and maintain its current state. |
| `sr` | 1 | O | Synchronous reset. Asserted for at least one symbol duration in order to reinitialize the decoder state. Input data symbols sampled before `sr` is asserted are not given at the output. |
| `ddel` | 1 | O | Original uncorrected data output. A delayed copy of the input data block. Data is presented on `ddel` concurrently with the decoded block on `dout`. The `wsymb` parameter defines the port width of this signal. |
| `errcnt` | 1-8 | O | Error Counter. Provides the number of corrected errors in the most recent output block. The bus width `errwidth` is equal to the number of bits required to represent the maximum possible number of correctable errors, as given in the following equation:<br>When `Block size type` is Constant, `errwidth` is defined as `errwidth` = ciel(log2((n-k+1)/2)) when Error Decoding mode is selected.<br>`errwidth` = ciel(log2(n-k-0.5)) when Erasure or Puncturing Decoding mode is selected.<br>When `Block size type` is Variable and `Variable check symbols` is No, `errwidth` is defined as `errwidth` = ciel(log2((Number of check symbols+1)/2)) when Error Decoding mode is selected.<br>`errwidth` = ciel(log2(Number of check symbols-0.5)) when Erasure or Puncturing Decoding mode is selected.<br>When Variable check symbols is Yes, `errwidth` is defined as `errwidth` = ciel(log2((Max. number of check symbols+1)/2)) The operator ciel() stands for the next higher integer. |
| `erscnt` | 1-8 | O | Erasure Counter. Provides a count of the number of erasures fed into the decoder in the most recent input data block. The bus width `erswidth` is equal to the number of bits required to represent the maximum possible number of correctable erasures, as given in the following equation:<br>When `Block size type` is Constant, `erswidth` is defined as `erswidth` = ciel(log2(n-k-1.5)).<br>When `Block size type` is Variable, `erswidth` is defined as `erswidth` = ciel(log2(Number of check symbols -1.5)) The operator ciel() stands for the next higher integer. |
| `fail` | 1 | O | Decoding failure indicator. Asserted at the same time `obend` is asserted to indicate that the block has more errors than the decoder can correct. |

## Timing Specifications

The decoder receives the data in blocks. The assertion of signal `ibstart` indicates the first symbol of the new block of data at the input of the decoder. The `ibstart` signal should be asserted only during the first clock cycle of a data block. The `ibstart` signal should not be re-asserted until the decoder is ready to receive the next block of data as indicated by `rfib` going high. The signal `rfib` can be used to generate the `ibstart` signal. If a new block of input data has to be applied before the decoder is ready for a new block, the decoder operation should be reset using the synchronous reset signal `sr`.

Figure 2-3 shows the I/O signals' status after asynchronous reset `rstn` is asserted at the beginning of the block. The output `rfi` goes high after reset to indicate the core is ready to receive input data. After `ibstart` signal is asserted, the decoder reads in the data block sequentially and starts the decoding process. When the decoded data is given at the output, `obstart` is asserted for one clock cycle during the first decoded output symbol. The output `obend` is asserted for one clock cycle when the last decoded symbol is given at the `dout` port.

***Figure 2-3. RS Decoder Normal Operation Timing Diagram***

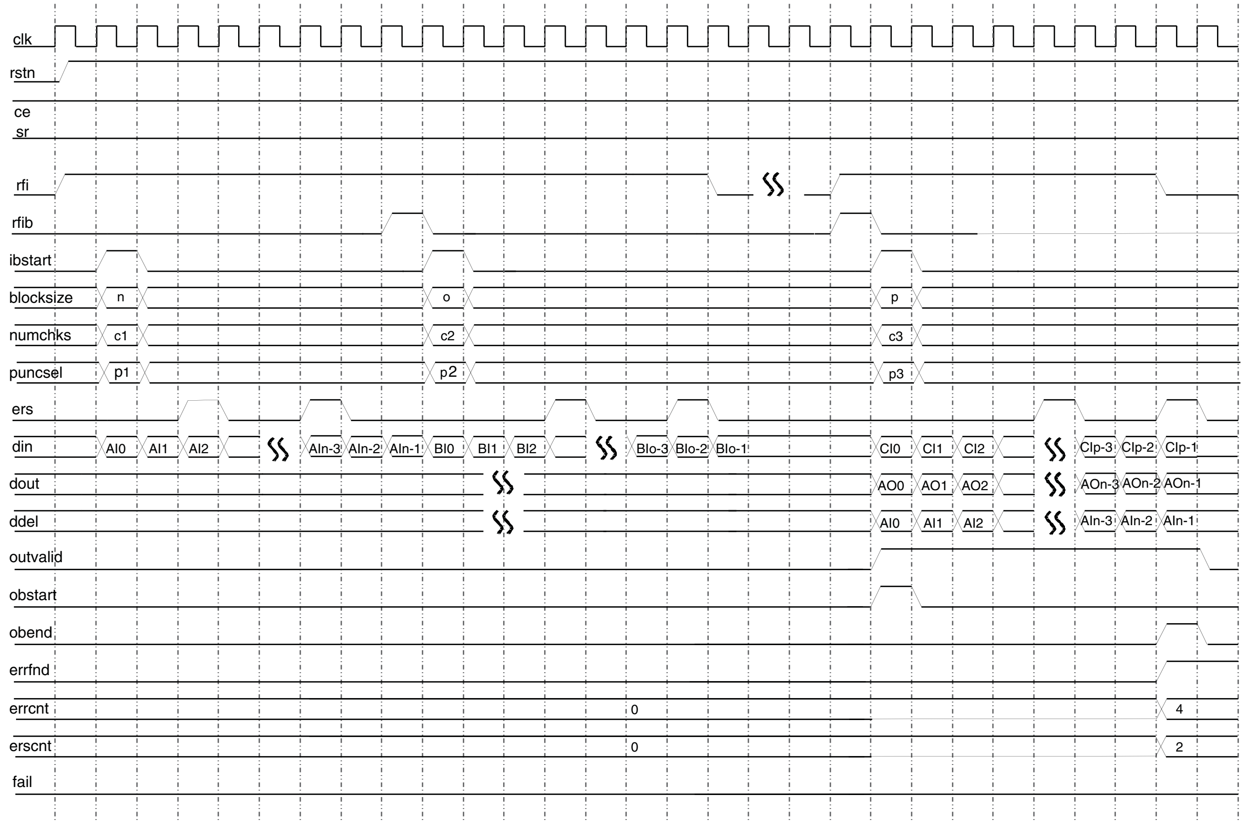


Figure 2-4 illustrates the output status when `sr` is asserted. When `sr` is asserted during the decoding process, it reinitializes the decoder state similar to power on reset state. The output data stops appearing at the output. The decoder operation can be started again by asserting the `ibstart` signal.

***Figure 2-4. Effect of Synchronous Reset on the Output Data from the Decoder***

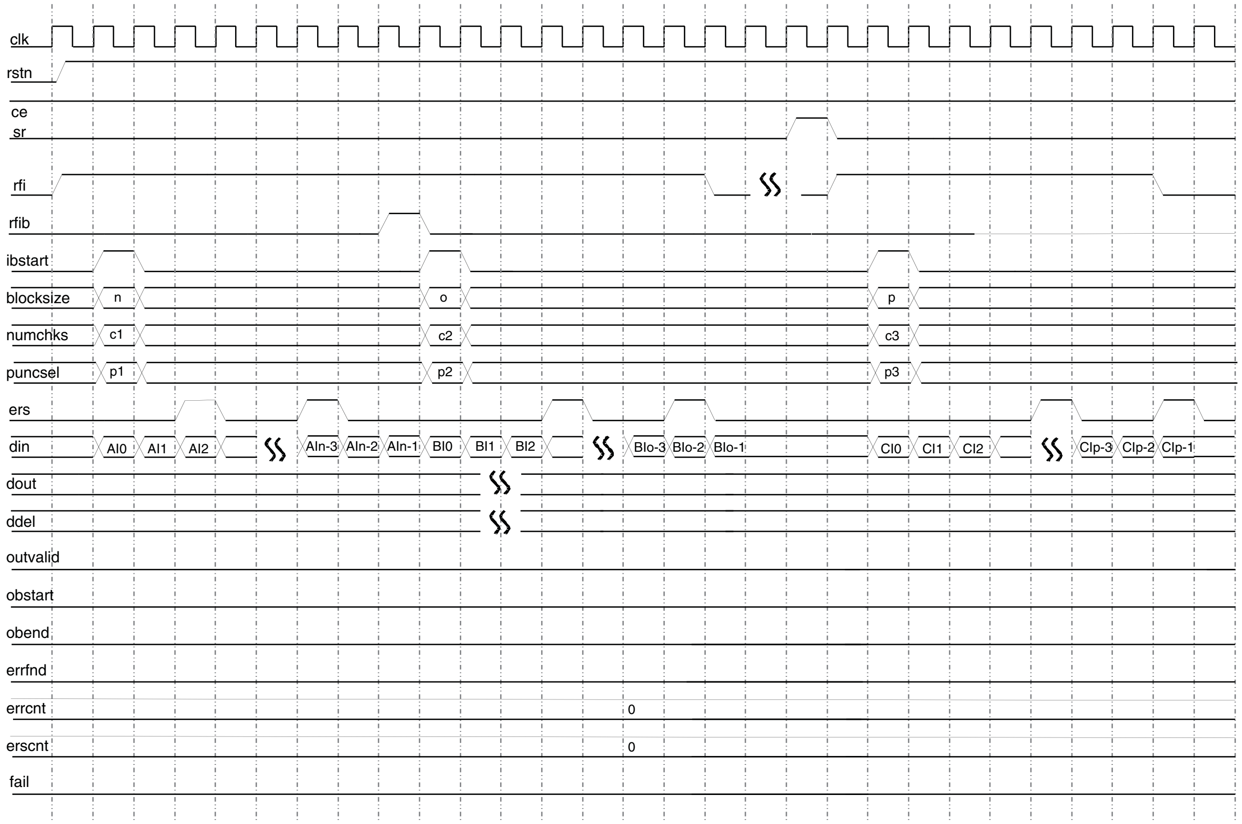
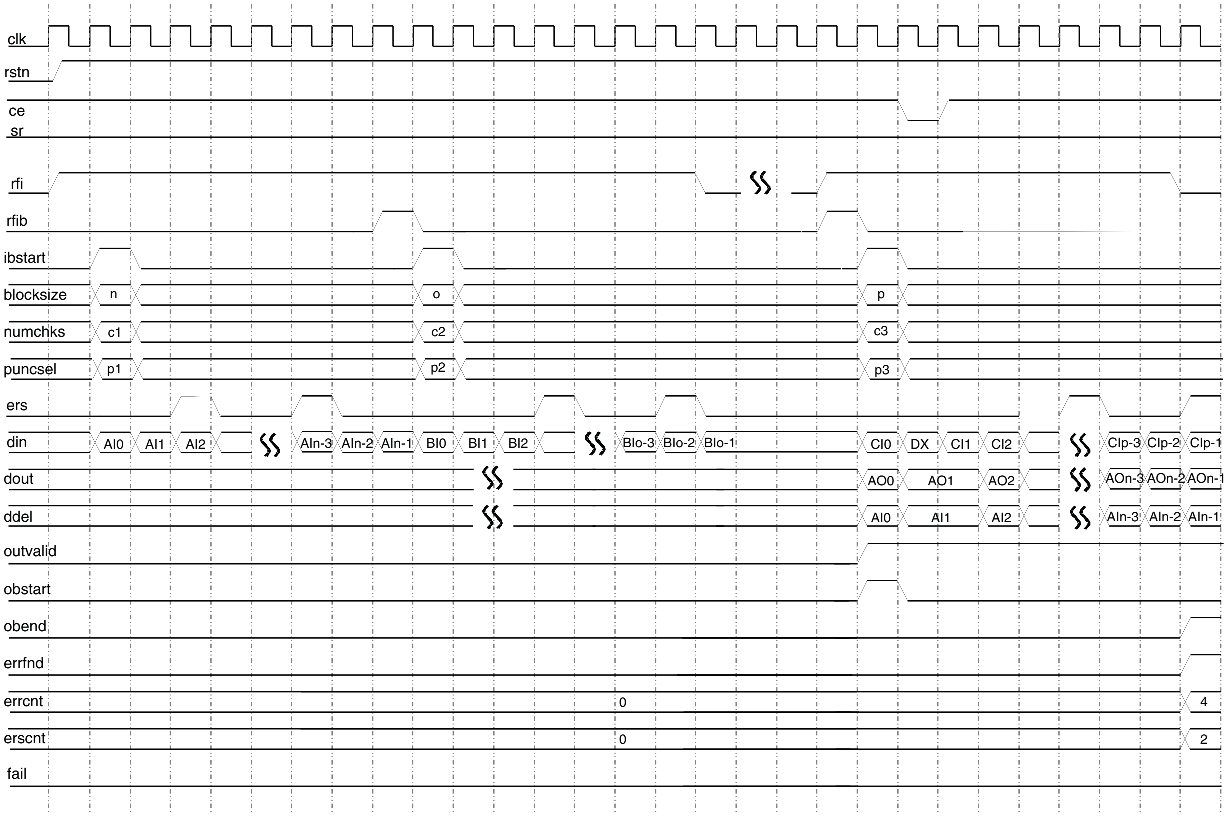


Figure 2-5 illustrates the effect of clock enable (`ce`) on the output data from RS Decoder. The decoder ignores all other synchronous inputs and remains in its current state when `ce` is de-asserted. When `ce` is asserted, the decoder goes back to the normal decoding process. In the figure, the data DX at `din` (that occurs during ce going low) is not recognized by the decoder.

***Figure 2-5. Effect of Clock Enable on the Output Data from Decoder***



## Chapter 3:

# Parameter Settings

The IPexpress™ tool is used to create IP and architectural modules in the Diamond and ispLEVER software. Refer to “IP Core Generation” on page 24 for a description on how to generate the IP.

The RS Decoder IP core Configuration GUI allows the user to create a custom configuration or to select one of the standard configurations: OC-192, CCSDS, DVB, ATSC, IEEE 802.16-2004 WirelessMAN-SCa/OFDM and IEEE 802.16-2004 WirelessMAN-SC. Table 3-1 provides the list of user configurable parameters for the RS Decoder IP core.

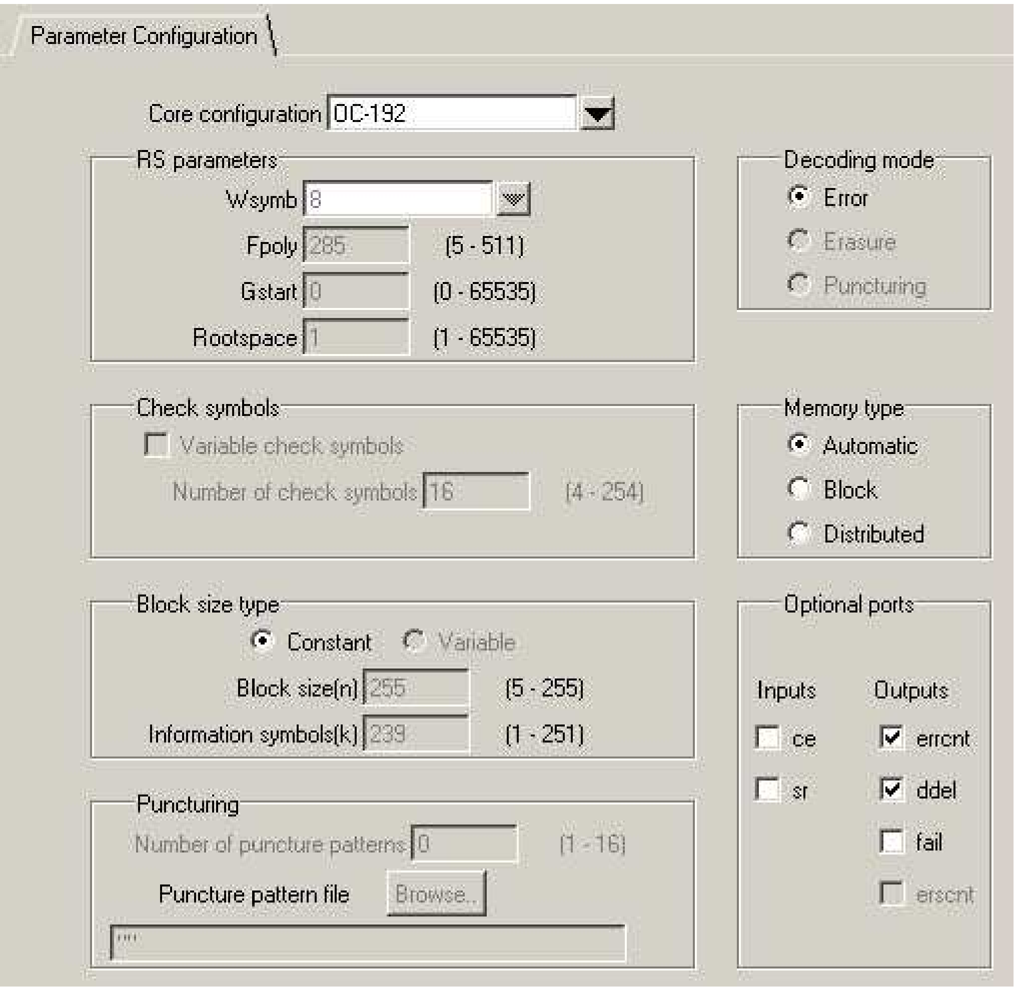
***Table 3-1. User Configurable Parameters***

| Parameter | Range/Options | Default |
|---|---|---|
| **Core Configuration** | | |
| Core configuration | Custom, OC-192, CCSDS, DVB, ATSC, IEEE 802.16-2004 SCa, IEEE 802.16-2004 SC | OC-192 |
| Connect reset port to GSR | Yes/No | Yes |
| **RS Parameters** | | |
| wsymb | 3 - 12 bits | 8 bits |
| fpoly | 5 - 8191 | 285 |
| gstart | 0 - 65535 | 0 |
| rootspace | 1 - 65535 | 1 |
| **Check Symbols** | | |
| Variable check symbols | Yes/No | No |
| Number of check symbols | 4 - $((2^{wsymb})-2)$. Maximum value is limited to 256. | 16 |
| Max. number of check symbols | 4 - $((2^{wsymb})-2)$. Maximum value is limited to 128. | 32 |
| **Block Size Type** | | |
| Block size type | Variable} if Variable check symbols is checked. {Constant, Variable} if Variable check symbols is not checked. {Constant} if wsymb is 3 and Decoding mode is Erasure. | Constant |
| Block size(n) | 5 - $((2^{wsymb})-1)$ Default value is $((2^{wsymb})-1)$ | 255 |
| Information symbols(k) | 1 - (n-4) | 239 |
| **Puncturing** | | |
| Number of puncture patterns | 1 - (n-k) when Block size is Constant. 1 - (Number of check symbols) when Block size is Variable. | 1 |
| Puncture pattern file | Edit field to enter the file name directly or indirectly by using the browse button. | |
| **Decoding Mode** | | |
| Decoding mode | {Error, Erasure, Puncturing} | Error |
| **Memory Type** | | |
| Memory type | {Automatic, Block, Distributed} | Automatic |
| **Optional Ports** | | |
| ce | Yes/No | No |
| sr | Yes/No | No |
| errcnt | Yes/No | Yes |
| ddel | Yes/No | Yes |
| fail | Yes/No | No |
| erscnt | Yes/No | |

## RS Decoder Configuration GUI

Figure 3-1 shows the contents of the RS Decoder IP core Configuration GUI.

***Figure 3-1. RS Decoder IP core Configuration GUI***



Parameter Configuration

Core configuration OC-192

RS parameters
Wsymb 8
Fpoly 285 (5 - 511)
Gstart 0 (0 - 65535)
Rootspace 1 (1 - 65535)

Decoding mode
● Error
○ Erasure
○ Puncturing

Check symbols
☐ Variable check symbols
Number of check symbols 16 (4 - 254)

Memory type
● Automatic
○ Block
○ Distributed

Block size type
● Constant ○ Variable
Block size(n) 255 (5 - 255)
Information symbols(k) 239 (1 - 251)

Optional ports
Inputs
☐ ce
☐ sr
Outputs
☑ errcnt
☑ ddel
☐ fail
☐ erscnt

Puncturing
Number of puncture patterns 0 (1 - 16)
Puncture pattern file Browse...
""

### Core Configuration

Selects between custom and pre-defined standard configurations. Table A-1 on page 33 defines the fixed parameter values for different standard configurations.

### RS Parameters

**Wsymb**
This parameter sets symbol width.

**Fpoly**
This parameter sets the decimal value of the field polynomial. Table 2-1 on page 15 gives the default field polynomial values for different symbol widths.

**Gstart**
This parameter sets the offset value of the generator polynomial. The starting value for the first root of the generator polynomial is calculated as rootspace * gstart.

### Rootspace

This parameter sets the root spacing of the generator polynomial. The value of rootspace must satisfy the following equation: GCD(rootspace, 2wsymb-1) = 1. GCD is Greatest Common Divisor.

## Check Symbols

### Variable Check Symbols

This option allows the number of check symbols to be varied through the port in addition to varying the block size dynamically. In this case, the number of check symbols is defined through the input port numchks. This option is available only when Block size type is Variable and Decoding mode is Error.

### Number of Check Symbols

Constant value for the Number of check symbols in the codeword. This parameter is available when Block size type is selected as Variable and Variable check symbols is not checked.

### Max. Number of Check Symbols

Maximum value for number of check symbols provided through the input port numchks. This parameter selection is available only when Variable check symbols is checked.

## Block Size Type

This parameter specifies whether block size is provided as a constant value or varied through the input port. If Block size type is selected as Variable, the block size is read from the input port blocksize. Options depend on Variable check symbols.

### Block Size(n)

This parameter specifies the total number of symbols in the codeword. Defined only if Block size type is Constant.

### Information Symbols(k)

This parameter specifies the number of information symbols in the codeword. Defined only if Block size type is Constant. The value of k also depends on n as the maximum value of (n-k) is limited to 256

## Puncturing

### Number of Puncture Patterns

This is the number of pre-defined puncture patterns that can be dynamically selected using puncsel. This parameter is enabled when Decoding mode is selected as Puncturing.

### Puncture Pattern File

This is the file containing the pre-defined puncture patterns. The format of the puncture pattern file is described in the Puncture Pattern File Format section of this document. The browse button to load the puncture pattern file is enabled when Decoding mode is selected as Puncturing. The file should have a .cfg extension.

## Decoding Mode

Selects between different decoding modes. The selection of this parameter depends on the application requirements.

## Memory Type

Specifies the type of memory used for storing input data. If Memory type is selected as Block, then EBR memory is used. If Memory type is selected as Distributed then distributed memory is used. If Memory type is selected as Automatic then memory will be selected in a most optimized way depending on the other parameters selected.

## Optional Ports

**ce**
Determines whether the input port ce (clock enable) is present.

**sr**
Determines whether the input port sr (synchronous reset) is present.

**errcnt**
Determines whether the output port errcnt (error count) is present.

**ddel**
Determines whether the output port ddel (delayed data) is present.

**fail**
Determines whether the output port fail (decoding failure) is present.

**erscnt**
Determines whether the output port erscnt (erasure count) is present.

Chapter 4:

# IP Core Generation

This chapter provides information on licensing the RS Decoder IP core, generating the core using the Diamond or ispLEVER software IPexpress tool, running functional simulation, and including the core in a top-level design. The Lattice RS Decoder IP core can be used in LatticeECP3, LatticeECP2/M, LatticeECP, LatticeSC/M, LatticeXP, and LatticeXP2 device families.

## Licensing the IP Core

An IP license is required to enable full, unrestricted use of the RS Decoder IP core in a complete, top-level design. An IP license that specifies the IP core and device family is required to enable full use of the core in Lattice devices. Instructions on how to obtain licenses for Lattice IP cores are given at:

http://www.latticesemi.com/products/intellectualproperty/aboutip/isplevercoreonlinepurchas.cfm

Users may download and generate the IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The RS Decoder IP core core also supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (approximately four hours) without requiring an IP license (see "Instantiating the Core" on page 28 for further details). However, a license is required to enable timing simulation, to open the design in the Diamond or ispLEVER EPIC tool, and to generate bitstreams that do not include the hardware evaluation timeout limitation.

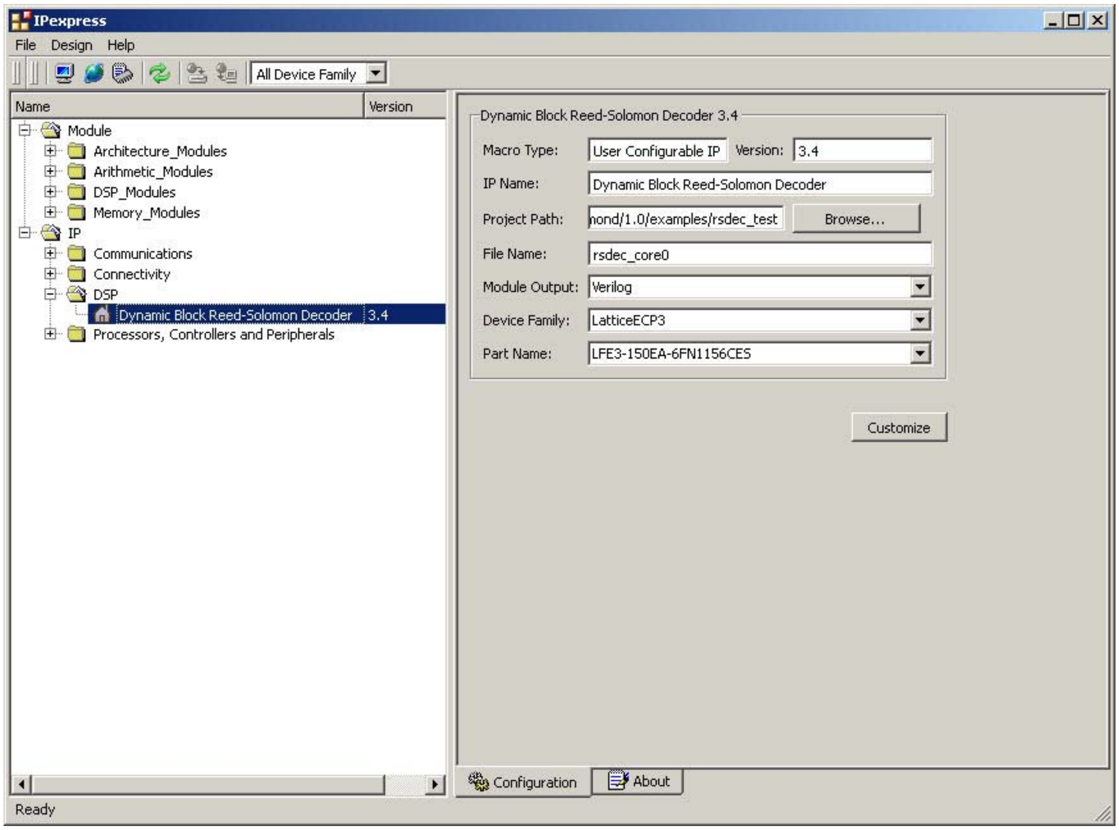
## Getting Started

The RS Decoder IP core is available for download from the Lattice IP Server using the IPexpress tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the IPexpress GUI dialog box shown in Figure 4-1.

The IPexpress tool GUI dialog box for the RS Decoder IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **Project Path** – Path to the directory where the generated IP files will be located.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **(Diamond) Module Output** – Verilog or VHDL.
- **(ispLEVER) Design Entry Type** – Verilog HDL or VHDL.
- **Device Family** – Device family to which IP is to be targeted (e.g. LatticeSCM, Lattice ECP2M, LatticeECP3, etc.). Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family.

*Figure 4-1. IPexpress Tool Dialog Box (Diamond Version)*



Dynamic Block Reed-Solomon Decoder 3.4

Macro Type: User Configurable IP Version: 3.4

IP Name: Dynamic Block Reed-Solomon Decoder

Project Path: nond/1.0/examples/rsdec_test

File Name: rsdec_core0

Module Output: Verilog

Device Family: LatticeECP3

Part Name: LFE3-150EA-6FN1156CES

Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output (Design Entry in ispLEVER), Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration, the user clicks the **Customize** button in the IPexpress tool dialog box to display the RS Decoder IP core Configuration GUI, as shown in Figure 4-2. From this dialog box, the user can select the IP parameter options specific to their application. Refer to "Parameter Settings" on page 16for more information on the parameter settings.