Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Dynamic Block Reed-Solomon Encoder User's Guide

August 2010
IPUG40_03.6

Table of Contents

Introduction

Lattice's Dynamic Block Reed-Solomon Encoder IP core can be used for forward error correction in many terrestrial communication, space communication, data storage, and data retrieval systems. The encoder is compliant with several industrial standards including the more recent IEEE 802.16-2004. The Reed-Solomon Encoder IP core provides a customizable solution allowing forward error correction in other non-standard applications as well.

The encoder supports both a fixed, as well as a variable number of total symbols (block) and check symbols. In the variable configurations, either the block size or both the block size and check symbols can be dynamically varied through ports. The core allows dynamic output check symbols puncturing in the fixed check symbols configurations.

This user's guide describes the functionality and implementation of the Reed-Solomon Encoder. Lattice also offers a Reed-Solomon Decoder core that can serve as a complementary pair for decoding. For more information on Lattice products, refer to the Lattice web site at www.latticesemi.com.

Quick Facts

Table 1-1 through Table 1-9 give quick facts about the Dynamic Block Reed-Solomon Encoder IP core for LatticeEC™, LatticeECP™, LatticeECP2™, LattceECP2M™, LattticeSC™, LatticeSCM™, LatticeXP™, LatticeXP2™, and LatticeECP3™ devices.

Table 1-1. Dynamic Block Reed-Solomon Encoder IP core for LatticeEC Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice EC					
	Minimal Device Needed	LFEC1E	LFEC1E	LFEC1E	LFEC1E	LFEC1E	LFEC3E
Resource Utilization	Targeted Device	LFEC20E-5F672C					
	LUTs	300	500	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	600
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeECP Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice ECP					
	Minimal Device Needed	LFECP6E					
Resource Utilization	Targeted Device	LFECP20E-5F672C					
	LUTs	300	500	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	600
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeECP2 Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice ECP2					
	Minimal Device Needed	LFE2-6E					
Resource Utilization	Targeted Device	LFE2-50E-7F484C					
	LUTs	300	400	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	500
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeECP2M Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice ECP2M					
	Minimal Device Needed	LFE2M20E					
Resource Utilization	Targeted Device	LFE2M35E-7F484C					
	LUTs	300	400	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	500
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeSC Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice SC					
	Minimal Device Needed	LFSC3GA15E					
Resource Utilization	Targeted Device	LFSC3GA25E-7F900C					
	LUTs	300	500	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	500
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeSCM Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice SCM					
	Minimal Device Needed	LFSCM3GA15EP1					
Resource Utilization	Targeted Device	LFSCM3GA25EP1-7F900C					
	LUTs	300	500	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	500
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeXP Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice XP					
	Minimal Device Needed	LFXP3E					
Resource Utilization	Targeted Device	LFXP20E-5F256C					
	LUTs	300	500	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	600
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeXP2 Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**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	300	400	300	400	400	2700
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	500
Design Tool Support	Lattice Implementation	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. Dynamic Block Reed-Solomon Encoder IP core for LatticeECP3 Devices Quick Facts

		Dynamic Block Reed-Solomon Encoder IP Configuration					
		OC-192	**CCSDS**	**DVB**	**ATSC**	**IEEE 802.16-2004 SCa**	**IEEE 802.16-2004 SC**
Core Requirements	FPGA Families Supported	Lattice ECP3					
	Minimal Device Needed	LFE3-35EA					
Resource Utilization	Targeted Device	LFE3-95E-8FN484CES					
	LUTs	300	500	300	400	400	2800
	sysMEM EBRs	0	0	0	0	0	0
	Registers	300	400	300	300	300	500
Design Tool Support	Lattice Implementation	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					

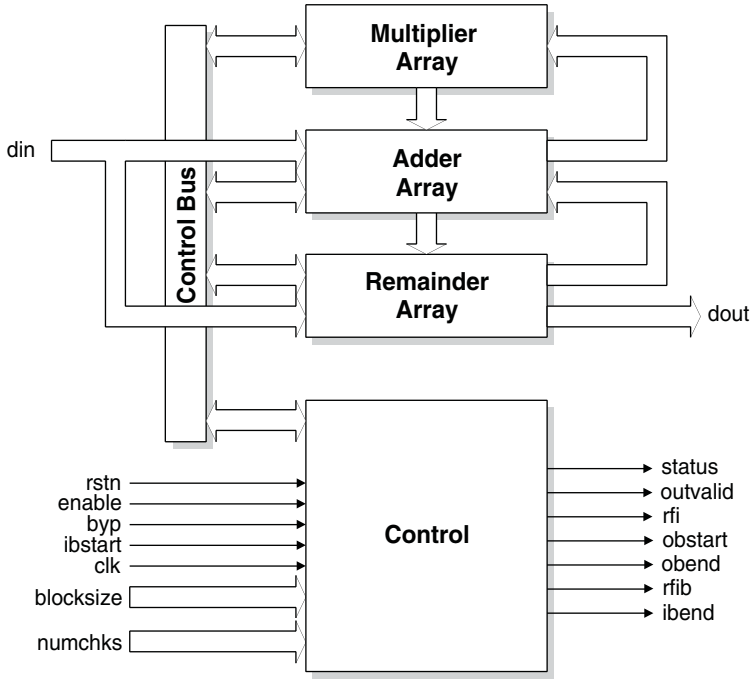
Features

- 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

- Selectable Reed-Solomon standards
  - OC-192
  - DVB
  - CCSDS
  - ATSC
  - IEEE 802.16-2004 WirelessMAN-SCa/OFDM
  - IEEE 802.16-2004 WirelessMAN-SC
- Fully synchronous
- Registered input selection
- Systematic encoder
- Full handshaking capability
- Dynamically variable block size
- Dynamically variable check symbols
- Dynamically variable check symbols puncturing

Figure 2-1 illustrates the operation of a systematic encoder.

Figure 2-1. Reed-Solomon Encoder 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. A Reed-Solomon 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 Broadcast (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 first k symbols in the Reed-Solomon Encoder output are information symbols and the last n-k symbols are check symbols. This type of encoder, where the information symbols are unchanged and are followed by check symbols in the output, is called a systematic encoder. Figure 2-1 illustrates the operation of a systematic encoder.

Reed-Solomon codes are defined on a finite field known as a Galois field. The size of the field is determined by the symbol width, wsymb, and is equal to 2^{wsymb}. When *n* is less than its maximum value of 2^{wsymb} - 1, it 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 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:

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{rootspace \cdot (gstart + i)}) \quad (1)$$

where a (alpha) is called the primitive element of the field polynomial. For a binary Galois field GF(2), a (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. Only the non-zero data is transmitted to the encoder (i.e., 188 in the above example). The encoder then generates the required check symbols from the non-zero data.

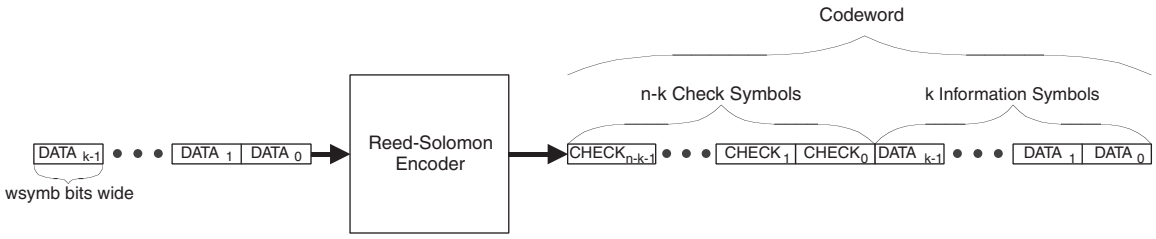
Output Latency

Output Latency for the Reed-Solomon Encoder core is defined as the number of clock cycles between the sampling of the first input data and the availability of the first data at the output port. It is three clock cycles when the inputs are registered and two clock cycles otherwise.

Functional Description

The Reed-Solomon Encoder utilizes Multiplier, Adder and Remainder arrays to perform finite field arithmetic. A block diagram of the Reed-Solomon Encoder is shown in Figure 2-2. The following section explains the operation of the arrays and the Control block.

Figure 2-2. Systematic Reed-Solomon Encoder



Multiplier Array

The Multiplier array performs the Galois field multiplication between the generator coefficients and the modulo-2 sum of input data and feedback data. This multiplication is optimized during the processing of the core.

Adder Array

The Adder array performs modulo-2 addition on the data from the previous element of the Remainder array and the result of the corresponding Galois field multiplication from the Multiplier array. The outputs from the Adder array are latched into the Remainder array on each clock cycle.

Remainder Array

The Remainder array is a shift register array. It stores the remainder polynomial after the polynomial division. The remainder polynomial becomes the check symbols once all information symbols have been processed. The Remainder array shifts-in the data from the Adder array while the information symbols are processed. When all the information symbols have been processed, the polynomial multiplication stops and the contents of the Remainder array are output to `dout`.

Control

The Control block generates all control signals and determines the state of the Reed-Solomon Encoder. The `rstn`, `enable`, `byp`, `ibstart`, and `clk` inputs control the state of the encoder. The Control block uses these inputs to control the state of the Multiplier, Adder and Remainder arrays as well as to generate the `rfi`, `status`, `outvalid`, `ibend`, `obstart` and `obend` outputs.

Basis Conversion Modules

When core type is selected as CCSDS, then two additional Basis Conversion modules are added to the Reed-Solomon Encoder. These modules comply to 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.

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 Reed-Solomon encoder 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.
`enable`	1	I	Enables the encoder to process data on `din`. When low, the input data is ignored and `dout` holds its state.
`byp`	1	I	When asserted, the data at the input `din` is passed directly to the output `dout` after the pipeline latency of the core.
`ibstart`	1	I	Indicates that the data on `din` is the first information symbol of a new codeword. This signal is ignored if `byp` is high or `enable` is low.
`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.
`status`	1	O	Indicates the information symbols are present on `dout` or `byp` is asserted.

Table 2-2. Interface Signal Descriptions (Continued)

Port	Bits	I/O	Description
For Variable Check Symbols or Punctured Check Symbols			
numchks	2-9	I	This signal is used for two functions. 1) When the parameter Variable check symbols is "Yes", this port is used to provide the number of check symbols value. The width of this port is equal to ceil(log2(Max. number of check symbols)) 2) When the parameter Puncture check symbols is "Yes", this port is used to indicate the number of transmitted check symbols out of total of Number of check symbols. Only the first numchks number of check symbols are given out at the dout port. The width of this port is equal to ceil(log2(Number of check symbols)). In both cases, the value at this port is read only when ibstart is high.
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.
Optional I/Os			
outvalid	1	O	Output data valid. This indicates valid data is present on dout.
rfi	1	O	Ready for input. This indicates the encoder is ready to receive input data. Typically, this signal is high when the core is ready to read information symbols and turns low when check symbols are being calculated.
rfib	1	O	Ready for input block. This indicates that the encoder is ready to receive the first information symbol in the block.
ibend	1	O	Input block end. This indicates that the encoder is sampling the last information symbol on the data input port din.
obstart	1	O	Output block start. This indicates first output data of the codeword on the dout port.
obend	1	O	Output block end. This indicates last output data of the codeword on the dout port.

Timing Specifications

Figure 2-3 illustrates the timing of an RS (7,3) double pipelined encoder during normal operation. The diagram shows a typical behavior for the handshake signals status, rfi and outvalid.

Figure 2-3. Timing of an RS (7,3) Double Pipelined Encoder

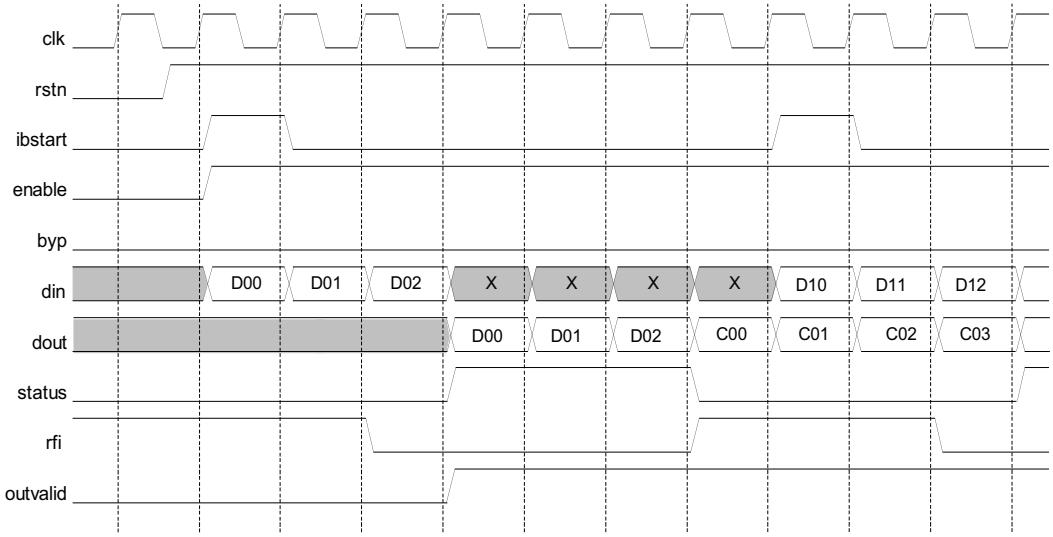


Figure 2-4 shows the timing of an RS (7,3) double pipelined encoder with `byp` asserted during the operation of the encoder. The handshake signals are identical to normal operation, but the output is shifted due to the extra bypass data.

Figure 2-4. Timing of an RS (7,3) Double Pipelined Encoder with byp Asserted

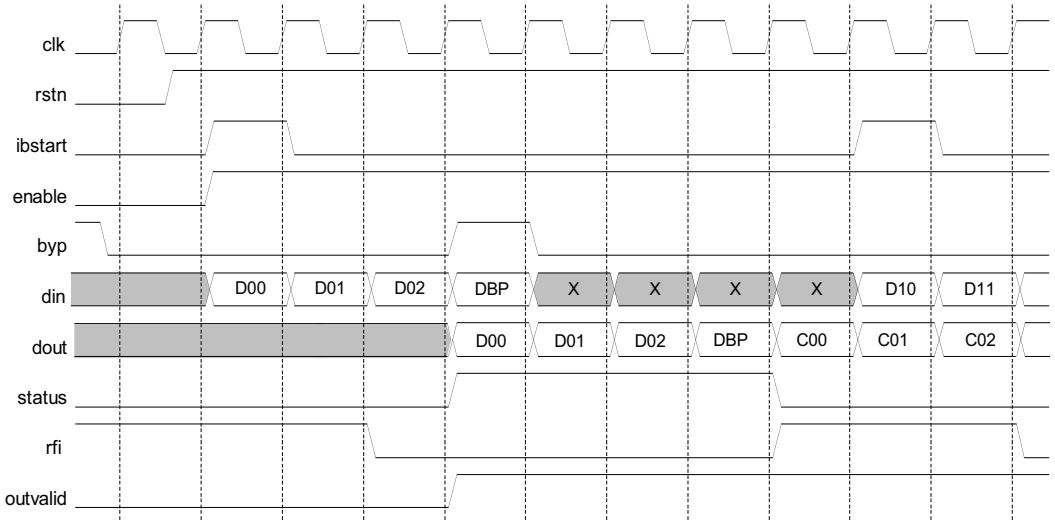


Figure 2-5 shows the timing of an RS (7,3) double pipelined encoder with `enable` de-asserted during the operation of the encoder. The de-assertion of `enable` results in corresponding invalid outputs happening after a few cycles determined by the output latency and indicated by `outvalid` going low. When `outvalid` is low, the output handshake signals maintain their last state.

Figure 2-5. Timing of an RS (7,3) Double Pipelined Encoder with enable De-asserted

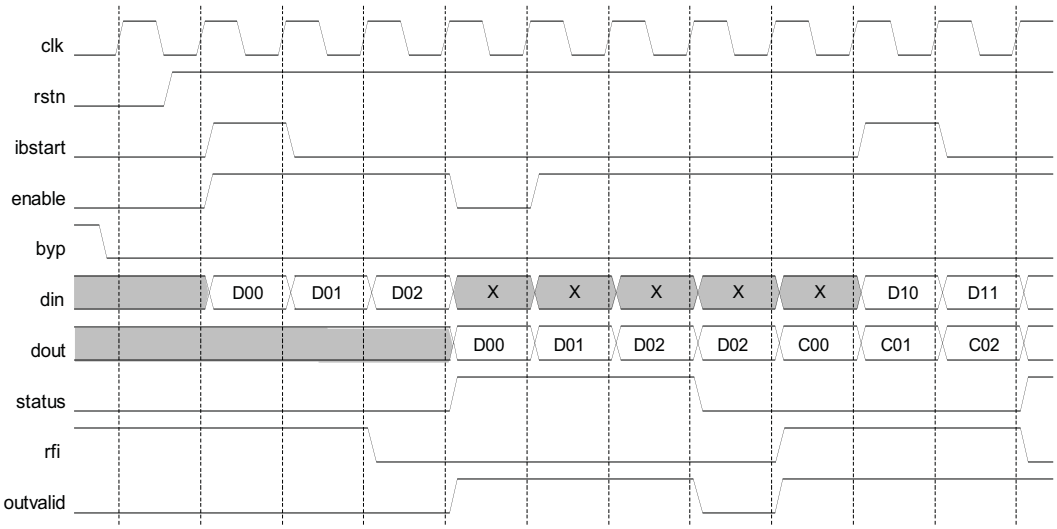


Figure 2-6 shows the timing of an RS (7,3) double pipelined encoder with `ibstart` re-asserted during the operation of the encoder. The handshake signal `rfi` goes high to indicate the encoder is ready to receive a new set of data when `ibstart` is re-asserted during encoding.

Figure 2-6. Timing of an RS (7,3) Double Pipelined Encoder with ibstart Re-asserted

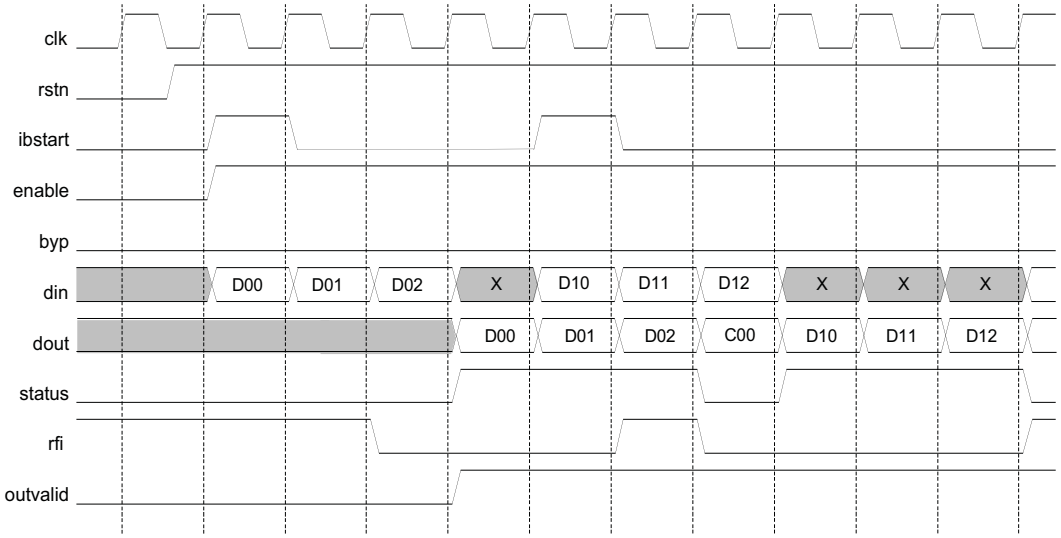
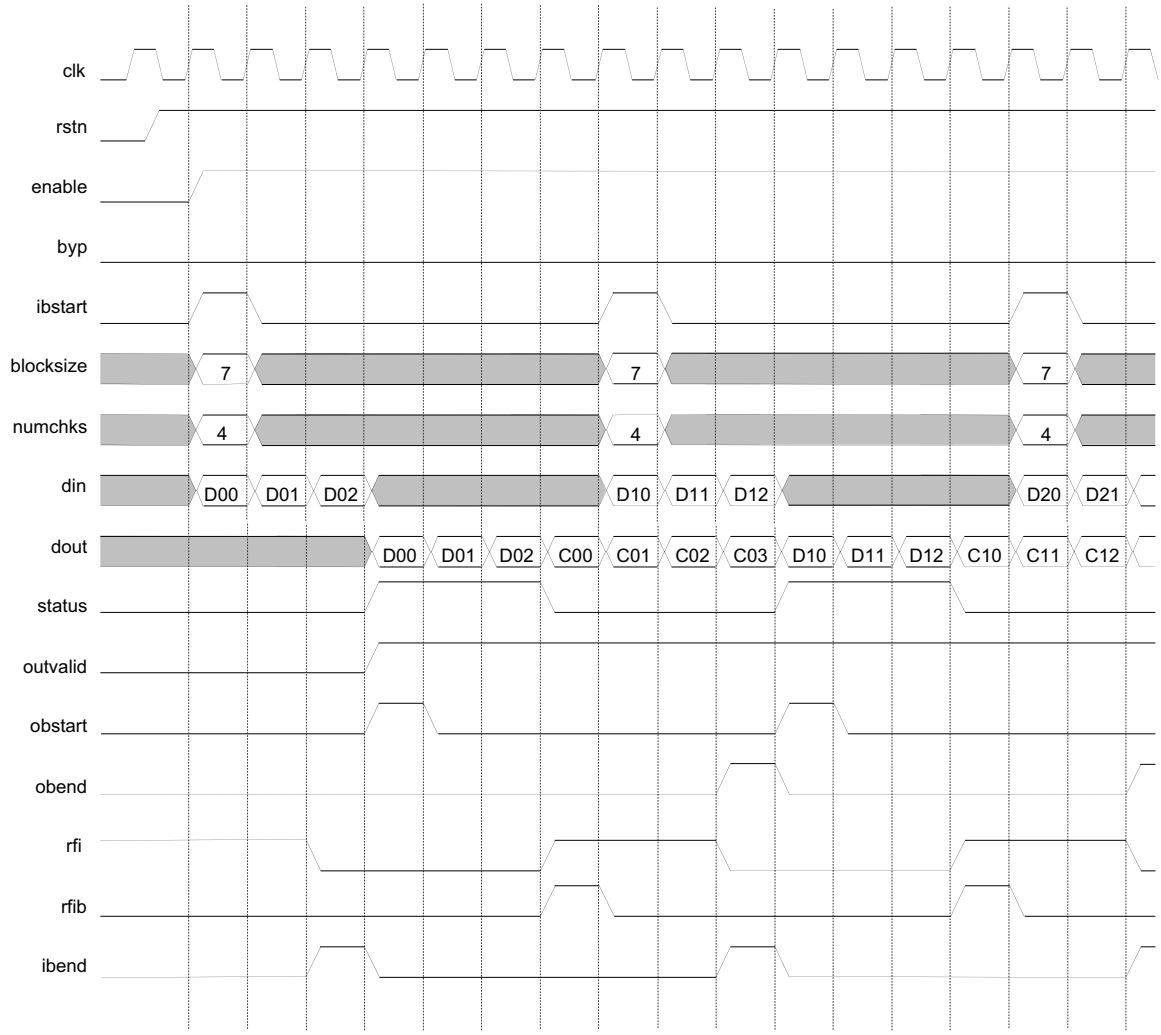


Figure 2-7 explains the timing of an RS (7,3) double pipelined encoder with variable block size and variable check symbols. The figure also shows the timing of the optional output ports `outvalid`, `obstart`, `obend`, `rfi`, `rfib` and `ibend`.

Figure 2-7. Timing of an RS (7,3) Encoder with Variable Block Size and Variable Check Symbols



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 Dynamic Block Reed-Solomon Encoder IP core 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 Reed-Solomon Encoder IP core.

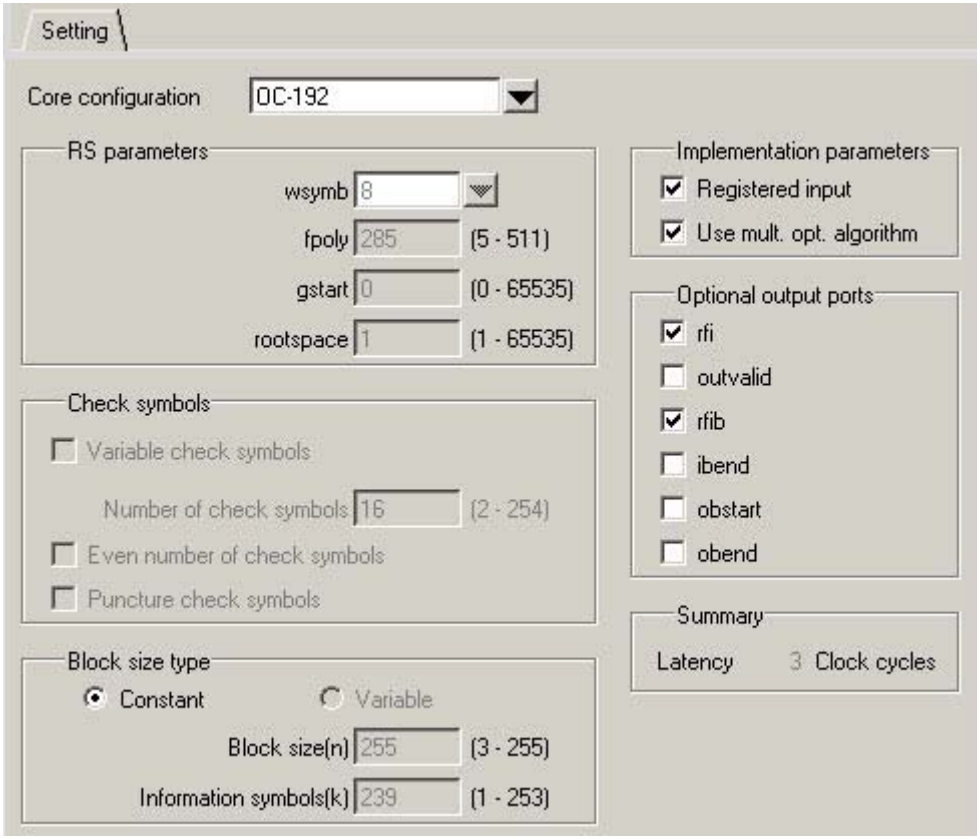
Table 3-1. User Configurable Parameters

Parameter	Range	Default
Core Type		
Core type	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
Block Size Type		
Block size type	{Constant, Variable} if "Variable check symbols" is not checked. {Variable} if "Variable check symbols" is checked.	Constant
n	3 - 4095	255
k	1 - 4093	239
Check Symbols		
Variable check symbols	Yes,/No	No
Max. number of check symbols	3 - 128	32
Even number of check symbols	Yes,/No	No
Number of check symbols	2 - 256	16
Puncture check symbols	Yes,/No	No
Implementation Parameters		
Registered input	Yes,/No	Yes
Use mult. opt. algorithm	Yes,/No	Yes
Optional Output Ports		
rfi	Yes,/No	No
outvalid	Yes,/No	No
rfib	Yes,/No	No
ibend	Yes,/No	No
obstart	Yes,/No	No
obend	Yes,/No	No

Reed-Solomon Encoder Configuration GUI

Figure 3-1 shows the contents of the Reed-Solomon Encoder IP core Configuration GUI.

Figure 3-1. Reed-Solomon Encoder IP core Configuration GUI



Core Configuration

This parameter selects between custom and pre-defined standard configurations. The Parameter Settings of the Standard Configurations table in the Dynamic Block Reed-Solomon Encoder User's Guide defines the fixed parameter values for different standard configurations.

RS Parameters

wsymb
This parameter specifies the symbol width.

fpoly
This parameter specifies the decimal value of the field polynomial. Table 2-1 on page 12 defines the default field polynomial parameter values for different symbol widths.

gstart
This parameter specifies 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 specifies 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
Specifies whether the number of check symbols is variable through the input port.

Number of Check Symbols
This parameter specifies the maximum value for number of check symbols provided through the input port numchks. This parameter selection is available when Variable check symbols is checked.

Even Number of Check Symbols
If this is checked, only even values should be provided on the input port numchks. If an odd value is given, it is internally set to the next lower even value. If this parameter is checked the core area is smaller and performance is better. This parameter selection is available when Variable check symbols is checked.

Pucture Check Symbols
This paramete enables the puncturing of check symbols, whereby only the first few check symbols are transmitted. The number of transmitted check symbols value is provided through the input port numchks. This parameter selection is available when Block size type is selected as “Variable” and Variable check symbols is not checked.

Block Size Type

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.

Block Size(n)
This parameter specifies the total number of symbols in the codeword. This parameter can be defined only when block size is constant.

Information Symbols(k)
This parameter specifies the number of information symbols in the codeword. This parameter can be defined only when block size is constant

Implementation Parameters

Registered Input
This parameter specifies whether the inputs are registered. Having registered inputs improves the performance of the core, but the latency will increase by one.

Use Mult. Opt. Algorithm
This parameter enables Galois field multiplication optimization algorithm to be used before synthesis. If this option is not checked, the optimization is left to the synthesis tool. In most cases, using an optimization algorithm results in improved performance and reduced area.

Optional Output Ports

rfi
Determines whether the output port rfi (ready for input) is present.

outvalid
Determines whether the output port outvalid (output valid) is present.

rfib
Determines whether the output port rfib (ready for input block or first data in the codeword) is present.

ibend
Determines whether the output port ibend (input block end or last information symbol in the codeword) is present.

obstart
Determines whether the output port obstart (output block start) is present.

obend
Determines whether the output port obend (output block end) is present.

Summary

The Summary entry in Figure 3-1 shows the output latency of the IP core based on the specified parameters. Output Latency for the Reed-Solomon Encoder IP core is defined as the number of clock cycles between the sampling of the first input data and the availability of the first data at the output port. It is three clock cycles when the inputs are registered and two clock cycles otherwise.

Chapter 4:

IP Core Generation

This chapter provides information on licensing the Dynamic Block Reed-Solomon Encoder 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 Dynamic Block Reed-Solomon Encoder 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 Dynamic Block Reed-Solomon Encoder 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 Dynamic Block Reed-Solomon Encoder IP 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 25 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.

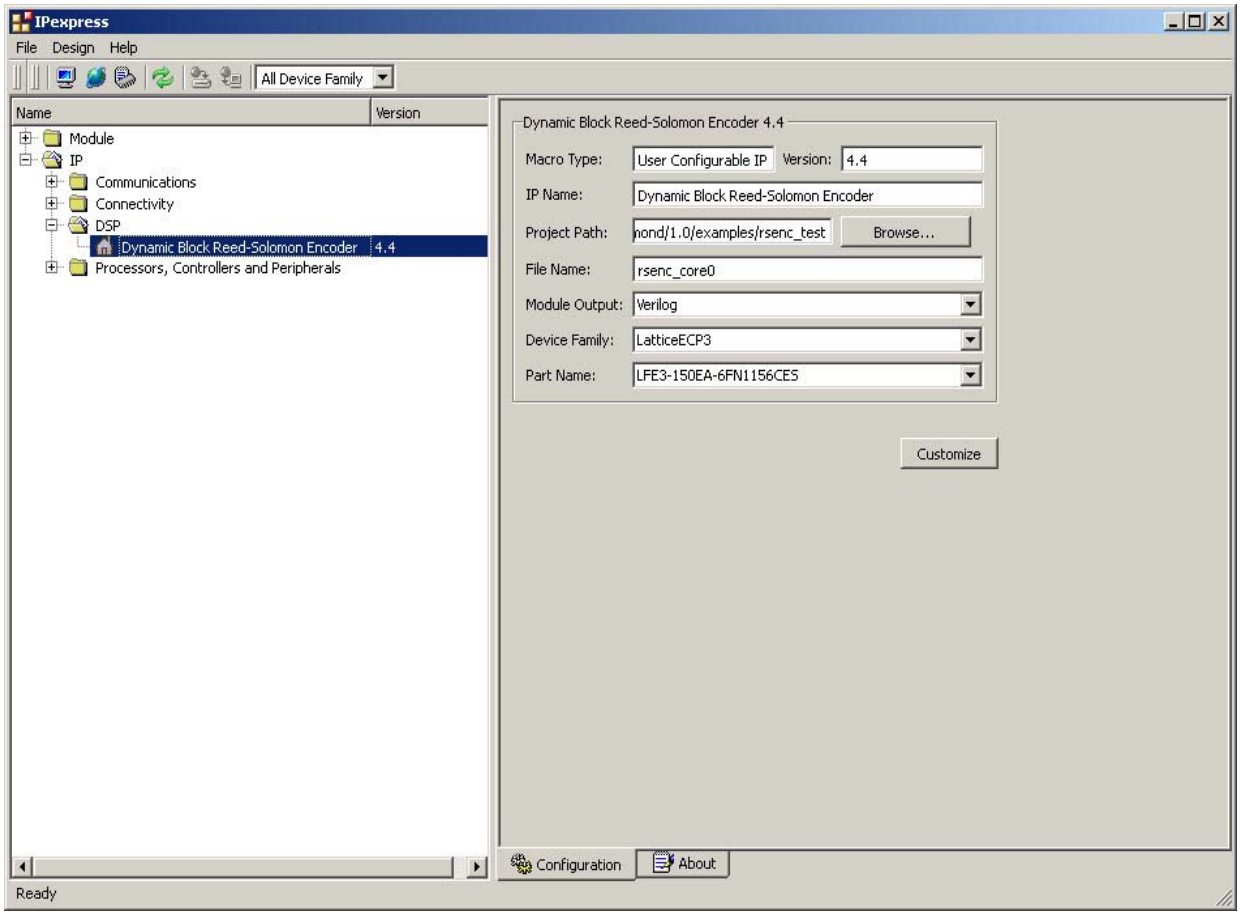
Getting Started

The Dynamic Block Reed-Solomon Encoder 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 Dynamic Block Reed-Solomon Encoder IP 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 loaded.
- **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.

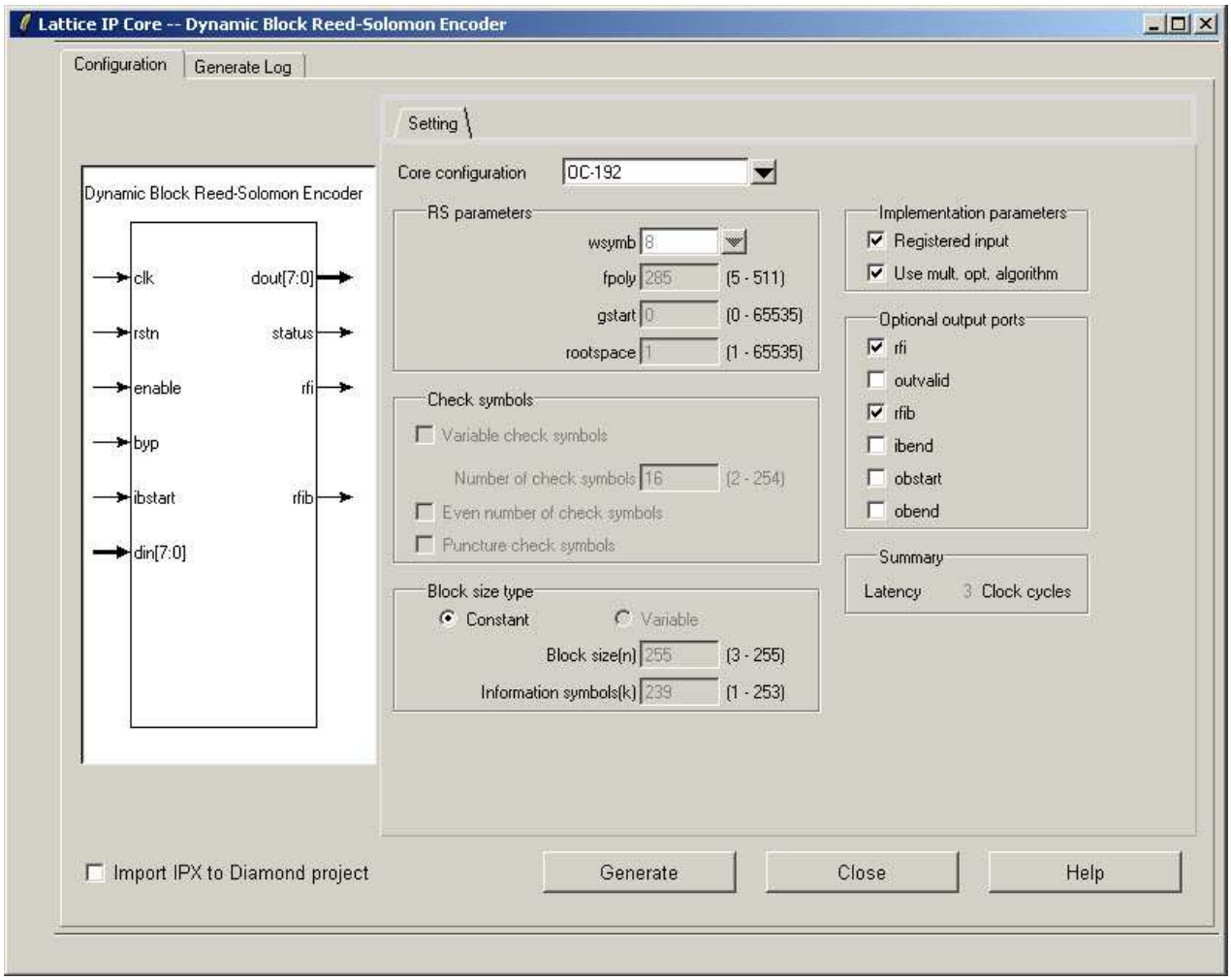
Figure 4-1. IPexpress Dialog Box (Diamond Version)



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 Dynamic Block Reed-Solomon Encoder 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 17 for more information on the parameter settings.

Figure 4-2. Configuration Dialog Box (Diamond Version)



IPexpress-Created Files and Top Level Directory Structure

When the user clicks the **Generate** button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified “Project Path” directory. The directory structure of the generated files is shown in Figure 4-3.

Figure 4-3. Lattice Dynamic Block Reed-Solomon Encoder IP core Directory Structure

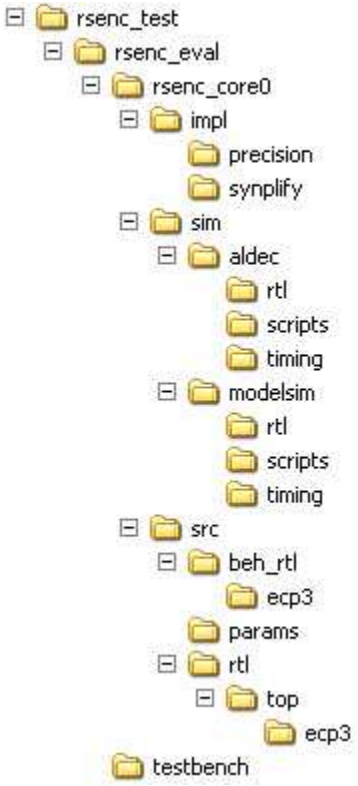


Table 4-1 provides a list of key files created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool.

Table 4-1. File List

File	Description
<*username*>_inst.v	This file provides an instance template for the IP.
<*username*>_bb.v	This file provides the synthesis black box for the user's synthesis.
<*username*>.ngo	The ngo files provide the synthesized IP core used by Diamond or ispLEVER. This file needs to be pointed to by the Build step by using the search path property.
<*username*>.lpc	This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool.
<*username*>.ipx	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
<*username*>_top.[v,vhd]	This file provides a module which instantiates the RS Encoder core. This file can be easily modified for the user's instance of the RS Encoder core. This file is located in the `rsenc_eval/<username>/src/rtl/top/` directory.
<*username*>_generate.tcl	This file is created when GUI "Generate" button is pushed and generation is invoked. This file may be run from command line.
<*username*>_generate.log	This is the IPexpress scripts log file.
<*username*>_gen.log	This is the IPexpress IP generation log file.

Instantiating the Core

The \<rsenc_eval> and subtending directories provide files supporting Dynamic Block Reed-Solomon Encoder IP core evaluation. The \<rsenc_eval> directory shown in Figure 4-3 contains files and folders with content that is constant for all con-figurations of the Dynamic Block Reed-Solomon Encoder. The \<*username*> subfolder (\resnc_core0 in this example) contains files and folders with content specific to the username configuration. The \rsenc_eval directory is created by IPexpress the first time the core is generated and updated each time the core is regenerated. A \<*username*> directory is created by IPexpress each time the core is generated and regenerated each time the core with the same file name is regenerated. A separate \<*username*> directory is generated for cores with different names, e.g. rsenc_core0, rsenc_core1, etc.

Running Functional Simulation

Simulation support for the Dynamic Block Reed-Solomon Encoder IP core is provided for Aldec Active-HDL (Verilog and VHDL) simulator, Mentor Graphics ModelSim simulator. The functional simulation includes a configuration-specific behavioral model of the Dynamic Block Reed-Solomon Encoder IP core. The test bench sources stimulus to the core, and monitors output from the core. The generated IP core package includes the configuration-specific behavior model (<username>_beh.v) for func-tional simulation in the “Project Path” root directory. The simulation scripts supporting ModelSim evaluation simulation is provided in \<*project_dir*>\rsenc_eval\<*username*>\sim\modelsim\scripts. The simulation script supporting Aldec evaluation simulation is provided in \<*project_dir*>\rsenc_eval\<*username*>\sim\aldec\scripts. Both Modelsim and Aldec simulation is supported via test bench files provided in \<*project_dir*>\rsenc_eval\testbench. Models required for simulation are provided in the corresponding \models folder. Users may run the Aldec evaluation simulation by doing the following:

1. Open Active-HDL.
2. Under the Tools tab, select **Execute Macro**.
3. Browse to folder \<project_dir>\rsenc_eval\<username>\sim\aldec\scripts and execute one of the "do" scripts shown.

Users may run the Modelsim evaluation simulation by doing the following:

1. Open ModelSim.
2. Under the File tab, select **Change Directory** and choose the folder \<*project_dir*>\rsenc_eval\<*username*>\sim\modelsim\scripts.
3. 3. Under the Tools tab, select **Execute Macro** and execute the ModelSim “do” script shown.

Note: *When the simulation completes, a pop-up window will appear asking “Are you sure you want to finish?” Answer* ***No*** *to analyze the results (answering* ***Yes*** *closes ModelSim).*

Synthesizing and Implementing the Core in a Top-Level Design

Synthesis support for the Dynamic Block Reed-Solomon Encoder IP core is provided for Mentor Graphics Precision or Synopsys Synplify. The Dynamic Block Reed-Solomon Encoder IP core itself is synthesized and is provided in NGO format when the core is generated in IPexpress. Users may synthesize the core in their own top-level design by instantiating the core in their top-level as described previously and then synthesizing the entire design with either Synplify or Precision RTL Synthesis. The following text describes the evaluation implementation flow for Windows platforms. The flow for Linux and UNIX platforms is described in the Readme file included with the IP core. The top-level files <*username*>_top.v are provided in \<*project_dir*>\rsenc_eval\<*username*>\src\rtl\top. Push-button implementation of the reference design is supported via Diamond or ispLEVER project files, <*username*>.syn, located in the following directory: \<*project_dir*>\rsenc_eval\<*username*>\impl\(synplify or precision).