LogiCORE IP Reed-Solomon Encoder v7.1

DS251 March 1, 2011 **Product Specification**

Features

- Implements many different Reed-Solomon coding standards, including all ITU-T J.83 and CCSDS codes
- Automatically configured by user-entered parameters
- Efficiently handles multiple channels
- Fully synchronous design using a single clock
- Supports continuous output data with no gap between code blocks
- Symbol width from 3 bits to 12 bits
- Code block length variable up to 4095 symbols with up to 256 check symbols
- Block length can be changed in real time
- The number of check symbols can be changed in real time
- Supports shortened codes
- Supports any primitive field polynomial for a given symbol width
- User-configured generator polynomial
- Easy-to-use interface with handshaking signals
- Use with Xilinx CORE Generator™ software and Xilinx System Generator for DSP v13.1

Applications

The Reed-Solomon Encoder is used in many Forward Error Correction (FEC) applications and in systems where data are transmitted and subject to errors before reception, for example, communications systems, disk drives, and so on.

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family(1)	Virtex-7 and Kintex-7, Virtex-6, Virtex-5, Virtex-4, Spartan-6, Spartan-3, Spartan-3E,Spartan-3A/3AN/3A DSP				
Supported User Interfaces	Not Applicable				
	Resources(2)				**Frequency**
Configuration	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs(3)**	**Max. Freq.(4)**
DVB	193	186	0	0/0	583
G.709	193	186	0	0/0	596
CCSDS	330	316	0	1/0	316
Provided with Core					
Documentation	Product Specification				
Design Files	Netlist				
Example Design	Not Provided				
Test Bench	Not Provided				
Constraints File	Not Applicable				
Simulation Model	VHDL, Verilog				
Tested Design Tools					
Design Entry Tools	CORE Generator tool 13.1 System Generator for DSP 13.1				
Simulation	Mentor Graphics ModelSim 6.6d Cadence Incisive Enterprise Simulator (IES) 10.2 Synopsys VCS and VCS MX 2010.06 ISIM 13.1				
Synthesis Tools	N/A				
Support					
Provided by Xilinx, Inc.					

1. For a complete listing of supported devices, see the release notes for this core.
2. Resources listed here are for Virtex-6 (-3) devices. For more complete device performance numbers, see Table 5.
3. Based on 18K/36K block RAMs.
4. Performance numbers listed are for Virtex-6 (-3) FPGAs. For more complete performance data, see Performance Characteristics, page 14,

Pinout

Port names for the core module are shown in Figure 1 and described in Table 1.

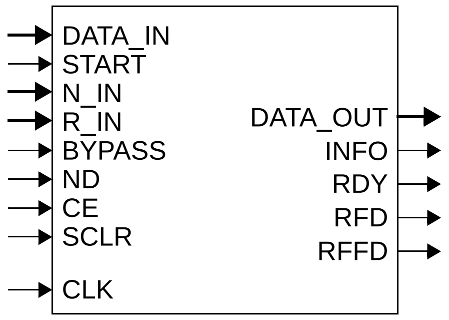

Figure 1: **Core Schematic Symbol**

Table 1: **Core Signal Pinout**

Signal	Direction	Description
DATA_IN	INPUT	**Data Input**: This signal is the data to be encoded.
N_IN	INPUT	**Block Size (N) Input**: This signal is used when block size is variable.
R_IN	INPUT	**Check Symbols (R) Input**: This signal is used when number of check symbols is variable.
SCLR	INPUT	**Synchronous Clear**: This signal is an Active high synchronous reset.
CE	INPUT	**Clock Enable**: Core is frozen when low.
ND	INPUT	**New Data**: Input signals are sampled when high.
BYPASS	INPUT	**Bypass**: When high, DATA_IN is passed to DATA_OUT without affecting the generated check symbols.
START	INPUT	**Start**: Active high informs the encoder that the first symbol of a new block is on DATA_IN.
CLK	INPUT	**Clock**: This signal is Active on rising edge.
INFO	OUTPUT	**Information**: This signal is high when there is information on DATA_OUT.
RDY	OUTPUT	**Ready**: This signal is high when there is valid data on DATA_OUT.
RFD	OUTPUT	**Ready for Data**: This signal is high when the core is ready to accept new data on DATA_IN.
RFFD	OUTPUT	**Ready for First Data**: This signal is high when the core is ready to accept a new pulse on START.
DATA_OUT	OUTPUT	**Data Output**: This is the data output.

Functional Description

Reed-Solomon codes are usually referred to as (*n*,*k*) codes, where *n* is the total number of symbols in a code block and *k* is the number of information or data symbols. This core generates systematic code blocks where the complete code block is formed from the *k* information symbols, followed by the *(n-k)* check symbols. The maximum number of symbol errors in a block that can be guaranteed to be correct is $t = (n-k)/2$. A symbol error can contain any number of bit errors.

Normally $n = 2^{(Symbol\_Width)}-1$. If *n* is less than this, the code is referred to as a "shortened code." The encoder core handles both full-length and shortened codes.

The parameters *n*, *k*, and (*n-k*) are optionally variable from block to block. The current blockHeading2 code settings for *n*, *k*, and (*n-k*) are *n_block*, *k_block,* and *r_block,* respectively.

A Reed-Solomon code is also characterized by two polynomials: the field polynomial and the generator polynomial. The field polynomial defines the Galois field, of which the symbols are members. The generator polynomial defines how the check symbols are generated. Both of these polynomials are usually defined in the specification for any particular Reed-Solomon code. The core GUI allows both of these polynomials to be configured.

The core synchronous input control signals (`START`, `ND`, `BYPASS`, `CE`) are not registered inside the core. It is assumed these are registered external to the core if required.

The following discussion and timing diagrams assume that the latency equals 3. In reality, the latency is dependent on the number of channels selected and the selected spec. The actual latency is reported in the core GUI after the parameters have been entered.

CE Input

All control signals are sampled on rising clock edges. `CE` has the highest priority. When `CE` is deasserted (low), all the other synchronous inputs are ignored and the core remains in its current state.

The core is frozen when `CE` is low. This is illustrated in Figure 2. Nothing changes state within the core.

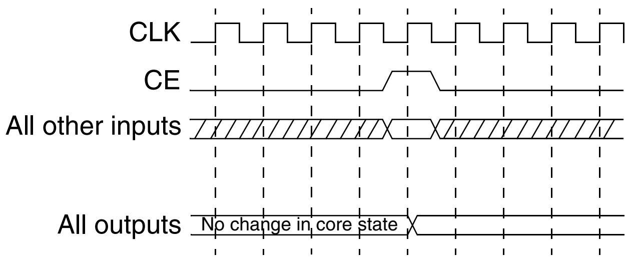

Figure 2: **Clock Enable Timing**

SCLR Input

When `SCLR` is asserted (high), the core is synchronously initialized. `SCLR` can be asserted for one or more clock cycles. Normal operation resumes as soon as `SCLR` is deasserted. `SCLR` is ignored if `CE` is low. `SCLR` is an optional pin, as it is not required for normal operation. The core always starts a new block when `START` is asserted, even if it is in the middle of processing an old block.

Bypass Input

BYPASS has the next highest priority after CE and SCLR. If BYPASS is asserted (high) at a particular rising clock edge, DATA_IN at that clock edge is passed straight through to DATA_OUT with the standard latency. This symbol has no effect on the generated check symbols. BYPASS may be asserted at any time.

In Figure 3, data symbol D, on DATA_IN, is passed straight to DATA_OUT without affecting the state of the code generator. During the other clock cycles, DATA_OUT is either a delayed version of DATA_IN or one of the code block check symbols. When the core is not encoding a block, DATA_IN is automatically bypassed to DATA_OUT.

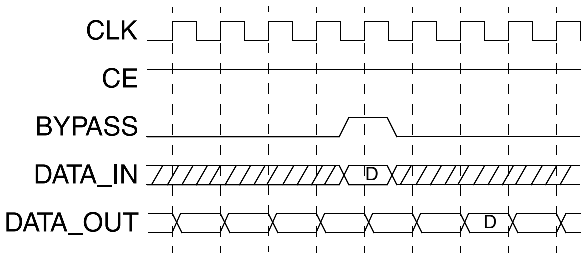

Figure 3: **Bypass Timing**

Start Input

If START is asserted (high) at a particular rising clock edge, it is assumed that the symbol on DATA_IN at that time is the first symbol of a new code block. START can be asserted at any time. It is ignored if CE or ND is deasserted or BYPASS is asserted. In Figure 4, D1 is taken to be the first symbol of a new code block. This figure also shows the optional N_IN input being used to set the block length, *n_block*, to N1 and the optional R_IN input being used to set the blockHeading2 number of check symbols, *r_block*, to R1.

This example block code settings are *n_block* = N1, *k_block* = N1 - R1, and *r_block* = R1.

The core samples *k_block* information symbols and immediately follows the last information symbol on DATA_OUT with *r_block* check symbols. In Figure 4 the latency is 3. If START is sampled high more than *latency*-2 clock edges before the clock edge that outputs the last check symbol, the core abandons the current code block and starts afresh with a new one.

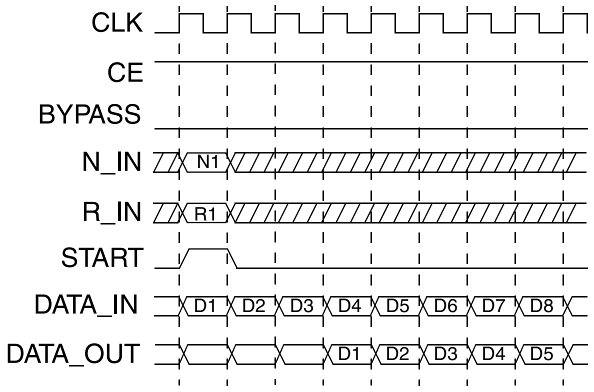

Figure 4: **Start Timing**

Figure 5 illustrates the timing at the end of a code block. This shows the earliest time that START can be reasserted, if all the previous check symbols are to be shifted out. $r = r\_block$ and C_r is the *r_block*th check symbol to be shifted out for this code block. The symbol on DATA_IN immediately prior to D1 is the *n_block*th symbol from the start of the previous block. The final *r_block* symbols of a block on DATA_IN are ignored.

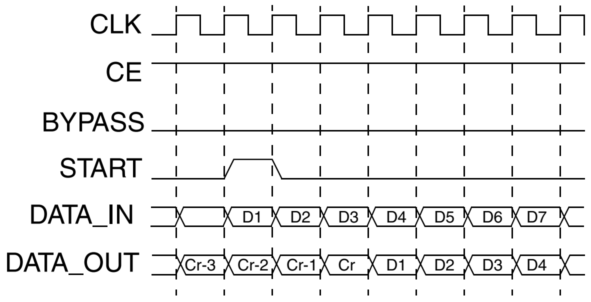

Figure 5: **Consecutive Code Blocks**

ND Input

ND is an optional input that can be used to signal *New Data* on DATA_IN. The core only samples inputs on DATA_IN, as part of a block to be encoded, when ND is high. Symbols sampled when ND is deasserted are automatically bypassed to DATA_OUT and do not affect the generation of the check symbols.

If an input is sampled and ND is then brought low, the core continues to process the last input as far as it can without further inputs. Thus, ND differs from CE. It does not freeze the state of the core when it is low.

ND operation is illustrated in the four check symbol example of Figure 7. In this case, the symbol X1 is sampled and passed to DATA_OUT, but it has no effect on the generation of the check symbols C1 to C4.

One example of the use of ND would be where a new symbol is available on DATA_IN only once every two clock cycles. ND is asserted when a new symbol is on DATA_IN and brought low for the next clock cycle. After the last symbol of the block is sampled, ND is ignored by the core until it is ready to start a new block again. This is because it does not need to sample any new symbols on DATA_IN to output the check symbols. A new check symbol is output every clock cycle. Contrasting this to CE, if CE was brought low every other cycle, then it would take twice as long to output the check symbols.

N_IN Input

N_IN is an optional input that is provided when a variable block length is required.

N_IN is sampled at the start of each block. The new block length, *n_block*, is set to N_IN sampled. N_IN sampled must be in the range given for *n_block* in Figure 4. N_IN sampled also affects the new blockHeading2 number of data symbols, *k_block*. See Block Code Settings, page 12, for further details on *n_block* and *k_block*.

N_IN may vary from block to block.

N_IN operation is illustrated in Figure 4.

R_IN Input

R_IN is an optional input that is provided when a variable number of check symbols is required.

R_IN is sampled at the start of each block. The new block's number of check symbols, *r_block*, is set to R_IN sampled. R_IN sampled must be in the range given for *r_block* in Table 4. R_IN sampled also affects the new block number of data symbols, *k_block*. See Block Code Settings, page 12, for further details on *k_block* and *r_block*.

R_IN may vary from block to block.

R_IN operation is illustrated in Figure 4.

Info Output

The INFO output is low when there are check symbols on DATA_OUT and high at all other times. This is illustrated in Figure 6. INFO is also high when the symbols on DATA_OUT were input with BYPASS asserted. Figure 6 shows the point where DATA_OUT changes from outputting DATA_IN to outputting the first check symbol. D_k is the last information symbol of the block. The core counts the information symbols and determines when to start outputting check symbols. It automatically takes account of cycles where BYPASS was asserted.

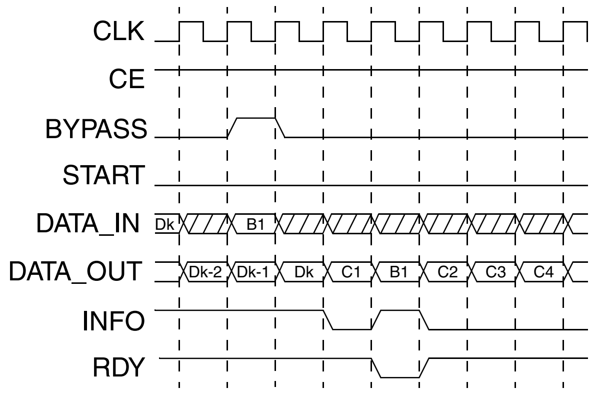

Figure 6: **Info Timing**

RDY Output

This optional output shows when valid data is present on DATA_OUT. Valid data is either a symbol sampled on DATA_IN as part of a block to be encoded, or a check symbol. If ND is deasserted, then RDY goes low when the symbols sampled with ND low are being output. This is shown in Figure 7. Notice that RDY is deasserted when X1 is output.

RDY also goes low when BYPASS symbols are being output. This is illustrated in Figure 6.

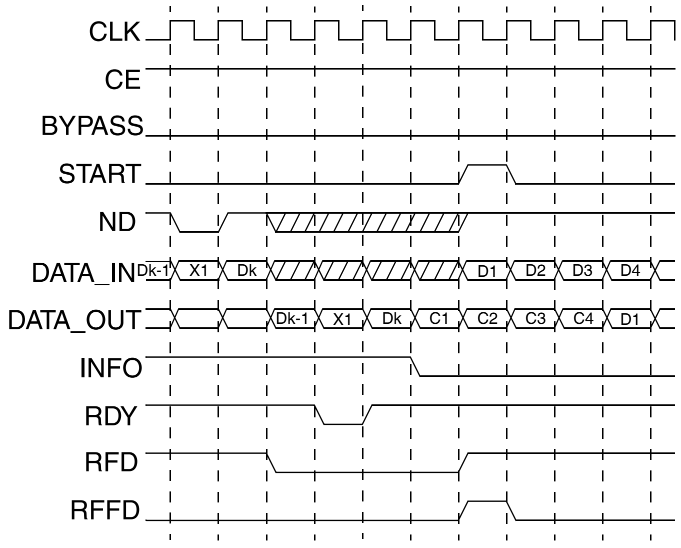

Figure 7: **Handshaking Signal Timing**

RFD Output

The *Ready for Data* optional output is asserted high when the core is ready to sample symbols to be encoded on DATA_IN. It is deasserted after *k_block* symbols have been sampled and reasserted when the core is ready for a new start pulse. This is illustrated in Figure 7.

RFFD Output

The *Ready for First Data* output is asserted high when the core is ready to accept a new start pulse. It is deasserted as soon as a valid start pulse has been sampled. This is illustrated in Figure 7.

If START is asserted when RFFD is low, the current block in progress is abandoned and a new block started.

Parameters

The core GUI provides a number of parameter values for several common Reed-Solomon standards. It also allows the user to define the following parameters:

Symbol Width

This is the bus-width of N_IN, DATA_IN and DATA_OUT.

Field Polynomial

This is used to generate the Galois field for the code. It is entered as a decimal number where the bits of the binary equivalent correspond to the polynomial coefficients. For example,

$$x8 + x4 + x3 + x2 + 1 => 100011101 => 285$$

A value of zero causes the default polynomial for the given Symbol_Width to be selected. If Field_Polynomial is not primitive, the core GUI highlights it in red. Table 2 shows the default field polynomial.

Table 2: **Default Polynomials**

Symbol Width	Default Polynomial	Decimal Representation
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

GeneratorStart

This is the Galois field logarithm of the first root of the generator polynomial.

Normally, GeneratorStart is 0 or 1; however, it can be any positive integer up to 1023.

$$g(x) = \prod_{i=0}^{n-k-1} (x - \alpha^{h \times (GeneratorStart + i)})$$

n

The number of symbols in a fixed length code block. If this is a shortened code, *n* should be the shortened number.

When a variable block length is required, the *n* parameter is defaulted to $2^{(\text{Symbol\_Width})}$-1 and the block length, *n_block*, is set to the value sampled on N_IN.

k

The number of information or data symbols in a fixed length code block.

When a variable block length and a fixed number of check symbols are required, the block's number of data symbols, *k_block*, is set to the value sampled on N_IN minus parameter (*n-k*).

When a variable number of check symbols is required, the block's number of data symbols, *k_block*, is set to the value sampled on N_IN minus the value sampled on R_IN.

h

The scaling factor for the generator polynomial root index. Normally, *h* is 1.

To ensure correct operation of the Reed-Solomon decoder, the value of *h* must be chosen so that the greatest common divisor of *h* and $2^{(Symbol\_Width)}-1$ is 1, that is, *h* and $2^{(Symbol\_Width)}-1$ must be relative primes.

Variable Block Length

The N_IN port is added when a variable block length is required.

- Whenever a block is started, the value on N_IN is sampled. The new block length, *n_block*, is set to the value sampled on N_IN.
- The number of check symbols in the new block is independent of the block length, so varying N_IN also changes the block's number of data symbols, *k_block*, by the same amount.
- The *n* parameter must be set to $2^{(Symbol\_Width)}-1$, and the *k* parameter should be set such that (*n-k)* equals the number of check symbols required.
- For multi-channel implementations, *n_block* is the same for all channels.
- The value sampled on N_IN must be in the range given for *n_block* in Table 4.

For full details on the variable block code settings *n_block* and *k_block*, see Block Code Settings, page 12.

Variable Number of Check Symbols

The R_IN port is added when a variable number of check symbols is required.

- Whenever a block is started, the value on R_IN is sampled. The new block's number of check symbols, *r_block*, is set to the value sampled on R_IN.
- The number of check symbols in the new block is independent of the block length, so varying R_IN also changes the number of data symbols in the block, *k_block*, by negative the same amount.
- The *n* parameter must be set to $2^{(Symbol\_Width)}-1$. The *k* parameter should be set such that $(n\text{-}k)$ equals the maximum number of check symbols required. The width of R_IN port is the number of bits required to represent $(n\text{-}k)$ in unsigned binary format.
- A multi-channel implementation is not available if a variable number of check symbols is required.
- The value sampled on R_IN must be in the range given for *r_block* in Table 4.

For full details on the variable block code settings *k_block* and *r_block*, see Block Code Settings, page 12.

Memory Style

If the target device architecture supports block memory, the following options are available:

- *Distributed* – Core should not use any block memories if possible. This is useful if they are required elsewhere in the design.
- *Block* – Core should use block memories wherever possible. This keeps the number of CLBs used to a minimum, but might use block memory wastefully.
- *Automatic* – This allows the core to use the most appropriate style of memory for each case, based on required memory depth.

Check Symbol Generator

If a variable number of check symbols is not required, then Check Symbol Generator must be set to *Fixed Architecture*.

- *Fixed Architecture* – The check symbol generator is implemented using a highly efficient fixed architecture.

If a variable number of check symbols is required, the Check Symbol Generator must be set to one of the following:

- *Optimized for Flexibility* – The check symbol generator implementation is optimized to maximize the range of input, N_IN.
- *Optimized for Area* – The check symbol generator implementation is optimized for area and speed efficiency. The range of input, N_IN, is reduced.

Code Specification

The GUI aids creation of cores for a number of common Reed-Solomon specifications. After a particular specification has been chosen, the GUI automatically selects the values necessary to meet the specification.

Most of the standards listed just result in particular values being set and then greyed out for most of the parameters in the GUI. However, some of the standards result in additional logic being added to the core. These are described in the following sections.

CCSDS

When implementing the *CCSDS* specification, the core automatically implements the dual-basis conversions defined in the *CCSDS* specification. This is illustrated in Figure 8. If the dual-basis conversions are not required, select *custom* specification instead of *CCSDS* and enter all the code parameters manually. Selecting *CCSDS* increases the latency of the encoder by 2.

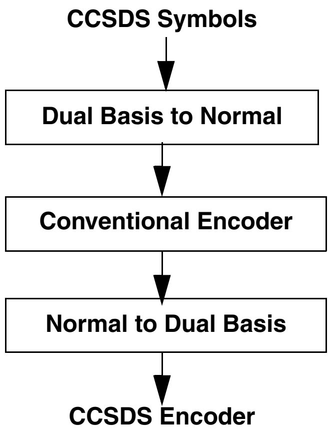

Figure 8: **CCSDS Encoder**

ITU J.83 Annex B

This standard is unusual in that it calls for a (128,122) code. This suggests that *n* is greater than $2^{(Symbol\_Width)}-1$, as the symbol width is only 7 bits. However, the standard specifies that the first 127 symbols are generated as a normal RS code. A special 128th symbol is then appended to the end of the block. If *ITU J.83 Annex B* is selected, then the core includes the logic required to generate this 128th symbol.

Selecting *ITU J.83 Annex B* increases the latency of the encoder by 1.

The other RS codes specified in the ITU J.83 standard do not require this additional symbol, and the *custom* code specification should be selected for them.

Number of Channels

The core can process multiple input channels simultaneously with only a small increase in area. This is much more efficient than instantiating multiple RS Encoder cores.

When a new block is started for one channel, a new block is started for all the other channels as well. The code settings *n_block*, *k_block* and *r_block* are the same for all channels. The multi-channel configuration is not available when a variable number of check symbols is required.

The latency is increased by 1 for each additional channel.

With multiple channels, there is still only one `DATA_IN` port. Incoming symbols for the channels are interlaced so the core samples the first symbol of channel 1 on the first rising clock edge, then the first symbol of channel 2 on the second rising clock edge, and so on. Symbols (both information and check) are output on `DATA_OUT` in the same sequence. An example with three channels, *k*=4 and *r*=2, is shown in Figure 9.

`START` is asserted only for a single clock cycle. This starts a new block for all three channels. A1, B1, and C1 are the first symbols for the new block for channels A, B, and C.

ND can be deasserted at any time. In this example, symbol X1 is passed to DATA_OUT (between A3 and B3) without affecting check symbol generation for any of the channels. The check symbols are output immediately after the information symbols in the normal way. They are also interlaced, so in the example the sequence is:

A_{C1}, B_{C1}, C_{C1}, A_{C2}, B_{C2}, C_{C2}, where A_{C1} is the first check symbol for channel A.

A new block could be started at the point when X2 is sampled on DATA_IN if required.

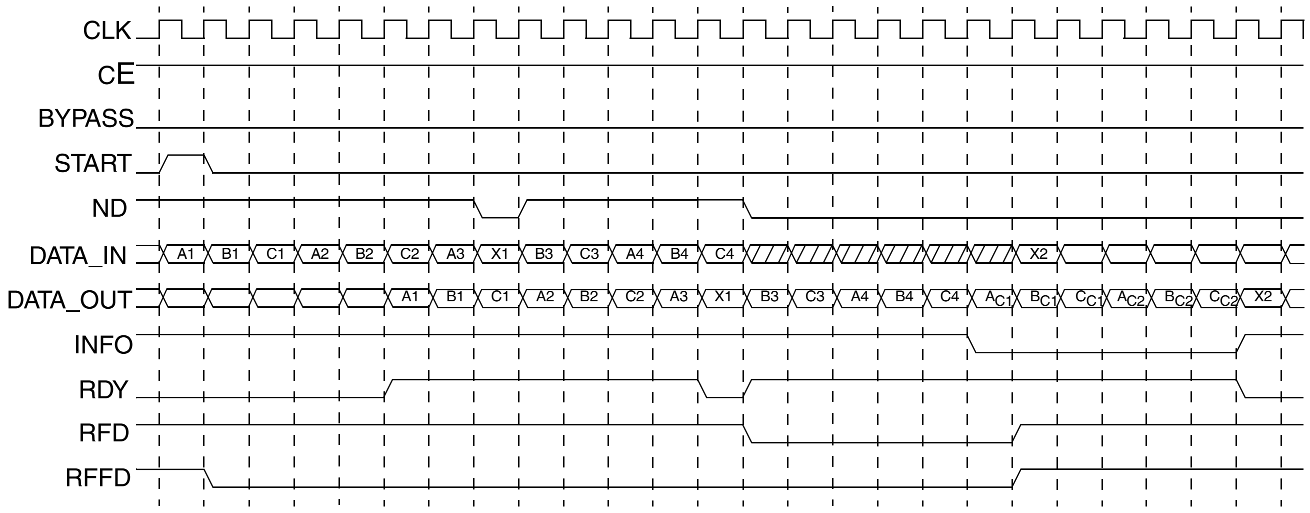
CLK
CE
BYPASS
START
ND
DATA_IN
A1 B1 C1 A2 B2 C2 A3 X1 B3 C3 A4 B4 C4 X2
DATA_OUT
A1 B1 C1 A2 B2 C2 A3 X1 B3 C3 A4 B4 C4 AC1 BC1 CC1 AC2 BC2 CC2 X2
INFO
RDY
RFD
RFFD

Figure 9: **Multi-Channel Operation**

Table 3: **Valid Parameter Ranges**

Parameter	Range	Max
Symbol Width	3	12
n	4	$2^{(Symbol\_Width)}-1$
k [1]	2	$2^{(Symbol\_Width)}-2$
r = n–k	2	min(*n-k*, 256)
GeneratorStart	0	1023
h	1	$2^{(16)}-1$
Polynomial	0	$2^{(13)}-1$
Number of Channels	1	128

Notes:

1. The maximum value for *k* is $2^{(Symbol\_Width)}-3$ if the Code Specification is set to CCSDS.

Block Code Settings

The RS Encoder generates a systematic (*n_block*, *k_block*) block code, where the output block is *n_block* symbols long, comprised from *k_block* data symbols followed by *r_block* check symbols. The block code settings *n_block*, *k_block* and *r_block* are optionally variable on a block-by-block basis. For multi-channel configurations, all channels have the same settings for *n_block*, *k_block* and *r_block*. See Table 4.

Table 4: **Block Code Settings – Value and Range**

Block Code Settings	Value	Range Min	Range Max
Fixed Block Length			
n_block	*n*	4	$2^{(Symbol\_Width)}-1$
k_block[1]	*k*	2	$2^{(Symbol\_Width)}-2$
r_block	*(n-k)*	2	min(*n-k*, 256)
Variable Block Length. Fixed Number of Check Symbols			
n_block	N_IN	4	$2^{(Symbol\_Width)}-1$
k_block[1]	N_IN - (*n-k*)	2	$2^{(Symbol\_Width)}-2$
r_block	(*n-k*)	2	min(*n-k*, 256)
Variable Number of Check Symbols (optimized for flexibility)			
n_block	N_IN	5	$2^{(Symbol\_Width)}-1$
k_block[1]	N_IN - R_IN	3	$2^{(Symbol\_Width)}-2$
r_block	R_IN	2	min(*n-k*, 128)
Variable Number of Check Symbols (optimized for area)			
n_block	N_IN	2*(*n-k*)	$2^{(Symbol\_Width)}-1$
k_block[1]	N_IN - R_IN	3	$2^{(Symbol\_Width)}-2$
r_block	R_IN	2	min(*n-k*, 128)

Notes:

1. The maximum value for *k* is $2^{(Symbol\_Width)}-3$ if the Code Specification is set to CCSDS.

n_block

The block code setting *n_block* specifies the total number of symbols in the current code block.

- When a variable block length is not required, *n_block* is set to the parameter *n* for every code block.
- When a variable block length is required, *n_block* is set at the start pulse of each new block to the value sampled on `N_IN`.

k_block

The block code setting *k_block* specifies the number of data symbols in the current code block.

- When a variable block length is not required, *k_block* is set to the parameter *k* for every block.
- When a variable block length is required and a variable number of check symbols is not required, *k_block* is set at the start pulse of each new block to the value sampled on `N_IN` minus the parameter (*n-k*).
- When a variable number of check symbols is required, *k_block* is set at the start pulse of each new block to the value sampled on `N_IN` minus the value sampled on `R_IN`.

r_block

The block code setting *r_block* specifies the number of check symbols in the current code block.

- When a variable number of check symbols is not required, *r_block* is set to parameter (*n-k*) for every block.
- When a variable number of check symbols is required, *r_block* is set at the start pulse of each new block to the value sampled on `R_IN`.

Latency

For this core, *latency* is defined as the number of rising clock edges from sampling `DATA_IN` to the sampled value appearing on `DATA_OUT`. The basic latency of the core is (2 + number of channels). For example, the latency in Figure 9 is 5.

- Selecting *CCSDS* increases the latency of the encoder by 2.
- Selecting *ITU J.83 Annex B* increases the latency by 1.

System Generator for DSP Graphical User Interface

The Reed-Solomon Encoder core is available through Xilinx System Generator for DSP, a design tool that enables the use of the model-based design environment Simulink® for FPGA design. The Reed-Solomon Encoder core is one of the DSP building blocks provided in the Xilinx blockset for Simulink. The core can be found in the Xilinx Blockset in the Communication section. The block is called "Reed-Solomon Encoder 7.1." See the System Generator User Manual for more information.

The controls in the System Generator GUI work identically to those in the CORE Generator GUI, although the layout has changed slightly. See Parameters, page 7, for detailed information about all other parameters.

Core Resource Utilization

The area of the core increases with (*n-k*) and Symbol_Width. Some example implementations are shown in Table 5.

When a variable number of check symbols is not required, the check symbol generator is implemented efficiently as a fixed architecture. When a variable number of check symbols is required, the check symbol generator must be either o*ptimized for area*, where the implementation area is increased by a factor of 3, or o*ptimized for flexibility*, where the implementation area is increased by a factor of 5.

The option to map primary I/O registers into IOB flip-flops should be selected if the core I/Os are to be connected directly onto a PCB via the FPGA package pins. This gives lower output clock-to-out times and predictable setup and hold times. Remember the control signal inputs are used unregistered inside the core, so these should be registered external to the core.

Performance Characteristics

In general, performance increases as *n-k* and Symbol_Width decrease. The clock frequencies given in Table 5 can be achieved when the corresponding period constraint is specified for the core clock input.

The area and speed values were obtained with map and par effort level set to high. Apart from this, default implementation tools options were used. It may be possible to improve slightly on these values by trying different options for the place and route software.

An implementation where a variable number of check symbols is required results in a lower maximum speed.

Table 5 provides resource and performance data for Virtex-6 FPGAs. For other devices, the user should generate a core and consult a map report to determine device utilization. The Xilinx SmartXplorer utility may be used to determine the maximum achievable frequency for the configuration.

Table 5: **Example Implementations**

	DVB 1	DVB 2	ATSC	G.709	ETSI-BRAN	CCSDS	ITU J.83 Annex B
Symbol Width	8	8	8	8	8	8	7
Generator Start	0	0	0	0	0	112	1
h	1	1	1	1	1	11	1
k	188	188	187	239	239	223	122
n	204	204	207	255	255	255	127[1]
Field Polynomial	285	285	285	285	285	391	137
Number of Channels	1	16	1	1	1	1	1
Variable Block Length	No	No	No	No	Yes	No	No
Xilinx Part	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T
LUT/FF Pairs[2]	197	338	243	195	199	337	97
LUTs[3]	190	337	237	192	194	329	87
FFs	186	326	218	186	186	316	109
Block RAMs (36k)	0	0	0	0	0	0	0
Block RAMs (18k)	0	0	0	0	0	1	0
Latency	3	18	3	3	3	5	3
Max Clock Freq[2][4]	436/583	448/562	450/589	441/596	442/580	245/316	450/600

Notes:

1. There are actually 128 symbols per block, but *n* is set to 127. The core automatically generates the 128th symbol if the spec is set to J.83 Annex B.
2. Area and max clock frequencies are provided as a guide. They may vary with new releases of the Xilinx implementation tools, etc.
3. LUT count includes route-thrus and may vary when the core is packed with other logic. Resource information is for -1 speed grade.
4. Maximum clock frequencies are shown in MHz for -1/-3 parts. Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification. ISE speed file version used for -1 speed grade was “PRODUCTION 1.07a 2010-05-25.” ISE speed file version used for -3 speed grade was “PRELIMINARY 1.07a 2010-05-25.”

Evaluation

An evaluation license is available for this core. The evaluation version of the core operates in the same way as the full version for several hours, dependent on clock frequency. Operation is then disabled and the data output does not change. If you notice this behavior in hardware, it probably means you are using an evaluation version of the core. The Xilinx tools warn that an evaluation license is being used during netlist implementation. If a full license is installed for the core to run on hardware, delete the old XCO file and recreate the core from new.

Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY.*

Refer to the IP Release Notes Guide (XTP025) for further information on this core. On the first page there is a link to "All DSP IP." The relevant core can then be selected from the displayed list.

For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Bug Fixes
- Known Issues

Ordering Information

This Xilinx LogiCORE IP product is provided under the terms of the SignOnce IP Site License.

To evaluate this core in hardware, generate an evaluation license, which can be accessed from the Xilinx IP Evaluation page.

After purchasing the core, you will receive instructions for registering and generating a full core license. The full license can be requested and installed from the Xilinx IP Center for use with the Xilinx CORE Generator software v13.1. The CORE Generator software is bundled with the ISE® Design Suite software v13.1 at no additional charge.

Contact your local Xilinx sales representative for pricing and availability on Xilinx LogiCORE products and software.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
03/28/03	1.0	Revision History added to document.
03/16/04	2.0	Updated to version 5.0 standards.
07/13/06	3.0	Updated to version 6.0 standards.
05/17/07	3.1	Updated to version 6.1 standards.
06/24/09	4.0	Updated to version 7.0 standards.
07/23/10	4.1	Updated to version 7.1 standards.
03/01/11	4.2	Support added for Virtex-7 and Kintex-7. ISE Design Suite 13.1. Reset last document version from 7.1 to 4.1 because document version is independent of core version.

Notice of Disclaimer