Reed-Solomon Encoder

January 12, 2000 Product Specification

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124
Phone: +1 408-559-7778
Fax: +1 408-559-7114
URL: http://www.support.xilinx.com/support/techsup/tappinfo.htm

Features

- Reed-Solomon Encoder
- Implements many different Reed-Solomon coding standards
- Automatically configured by user entered parameters
- Fully synchronous design using a single clock
- Supports continuous output data with no gap between code blocks
- Symbol width from 3 to 12 bits
- Code block length variable up to 4095 symbols with up to 256 check symbols
- Supports shortened codes
- Supports any primitive field polynomial for a given symbol width
- User-configured generator polynomial
- Can be optimized for area or speed
- Available for all Virtex™, Virtex™-E, Spartan™-II, XC4000 and Spartan™ family members.

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 errors in a block which can be guaranteed to be corrected is *t* = (*n-k*)/2.

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

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 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.

Pinout

The schematic symbol, with the signal names, is shown below.

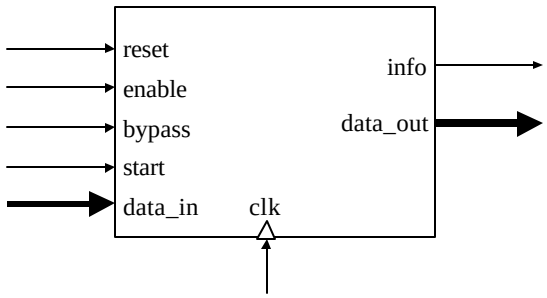

Figure 1 – Core Symbol

The following table summarizes the signal functions. The signals are described in more detail in the remainder of this section.

Signal	Signal Direction	Description
reset	Input	Active high asynchronous clear
clk	Input	Clock – active on rising edge
enable	Input	Input signals are sampled when high.
bypass	Input	When high, 'data_in' is passed to 'data_out' without affecting the generated check symbols.
start	Input	Active high. Informs encoder that first symbol of a new block is on 'data_in'.
data_in	Input	Input data
data_out	Output	Output data
info	Output	High when there is information on 'data_out'

Table 1 – Core Signal Pinout

Reset Input
All control signals are synchronous to the rising edge of 'clk' except 'reset'. When 'reset' is asserted (high) all the core flip-flops are asynchronously initialized. The core will remain in this state until 'reset' is de-asserted.

Enable Input
Of the remaining control signals, 'enable' has the highest priority. When 'enable' is de-asserted (low) all the other synchronous inputs are ignored and the core remains in its current state. The following discussion and timing diagrams assume an area-optimized core has been chosen, so all the synchronous inputs have a two clock cycle latency. Therefore if 'enable' changes state at one clock edge, 'data_out' will not be affected until another two rising clock edges have occurred (see Figure 2).

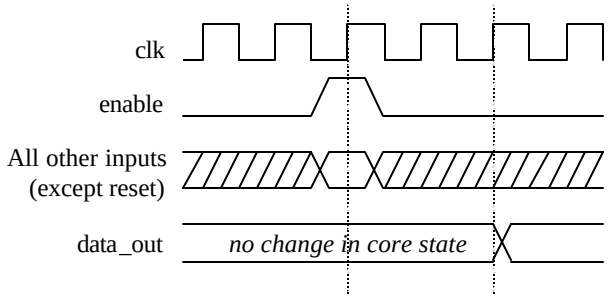

Figure 2 - Enable Timing

Bypass Input
'Bypass' has the next highest priority after 'reset' and 'enable'. If 'bypass' is asserted (high) at a particular rising clock edge, then 'data_in' at that clock edge is passed straight through to 'data_out' with a two clock cycle delay. This symbol will have no affect 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.

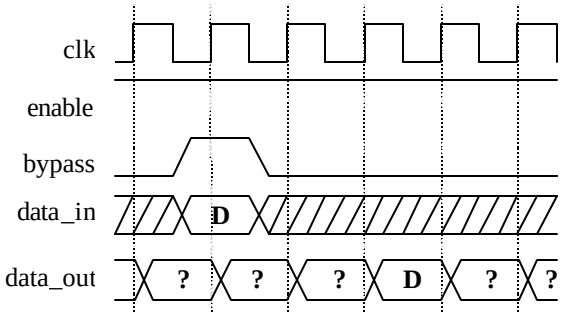

Figure 3 – Bypass Timing

Start Input
If 'start' is asserted (high) at a particular rising clock edge, then it is assumed that the symbol on 'data_in' at that time is the first symbol of a new code block. 'Start' may be asserted at any time. It is ignored if 'enable' is de-asserted or 'bypass' is asserted. In Figure 4, D1 is taken to be the first symbol of a new code block.

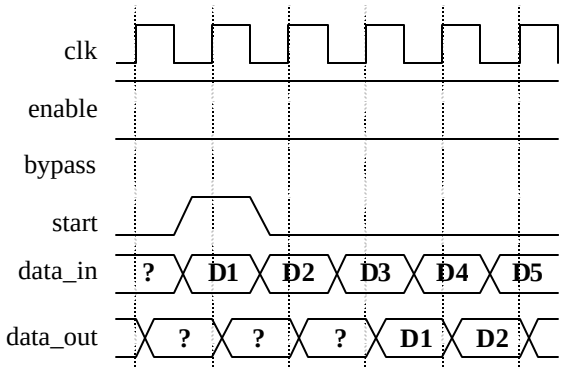

Figure 4 – Start Timing

The core will sample *k* information symbols and immediately follow the last information symbol on 'data_out' with the *n-k* check symbols. 'Start' has a two clock cycle latency. If 'start' is re-asserted more than two clock cycles before the last check symbol has been shifted out, the core will abandon the current code block and start afresh with a new one. The following figure illustrates the timing at the end of a code block. This shows the earliest time that 'start' may be reasserted, if all the previous check symbols are to be shifted out. R=*n-k* and C_R is the (*n-k*)th check symbol to be shifted out for this code block. The symbol on 'data_in' immediately prior to D1 is the *n*th symbol from the start of the previous block. The final *n-k* symbols of a block on 'data_in' are ignored.

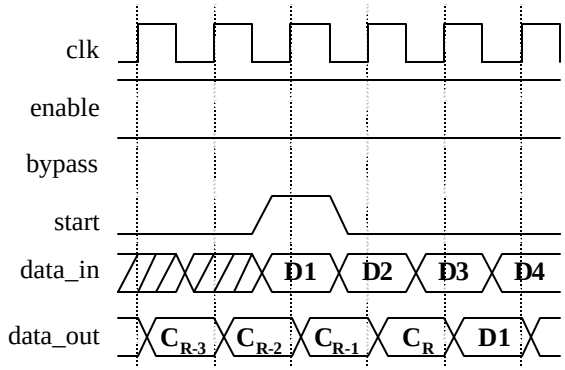

Figure 5 – Consecutive Code Blocks

Info Output

The 'info' output is high when there is information on 'data_out' and low when there are check symbols on 'data_out'. This is illustrated in Figure 6. 'Info' also goes high when 'bypass' is asserted (with a two clock cycle latency).

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 or 'enable' de-asserted.

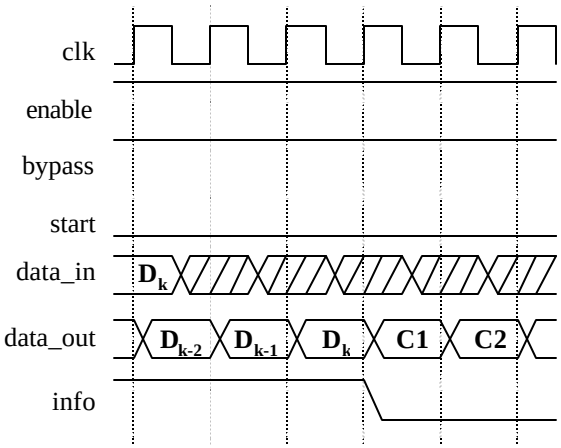

Figure 6 – Info Timing

Parameters

The core GUI provides a number of pre-set 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 'data_in' and 'data_out'.

- **Field Polynomial**

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

$$x^8+x^4+x^3+x^2+1 \Rightarrow 100011101 \Rightarrow 285$$

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.

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

Table 2 - Default Polynomials

- **Generator Start**

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

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

Normally *Generator_Start* is 0 or 1, however it can be any whole number.

- **n**

Number of symbols in an entire code block. If this is a shortened code, then *n* should be the shortened number.

- **k**

Number of information or data symbols in a code block.

- **h**

Scaling factor for the generator polynomial root index. Normally *h* is 1, however it can be any positive integer.

- **Optimization**

This parameter selects whether the core is optimized for area or speed. The area-optimized core has a two clock cycle latency while the speed optimized-core has a three clock cycle latency.
Selecting area optimization will only yield a smaller core for XC4000/Spartan target devices. For other device families the core will be the same size but slower than if speed optimization had been selected. The only point in selecting area optimization for non XC4000/Spartan devices is if a two clock cycle latency is required.

- **Create RPM**

When this option is selected, the core generator adds placement information to the core to create a relationally placed macro (RPM). RPMs can result in more predictable performance when used with other cores and user-defined logic, and usually require less time to place and route.

If a small target device is used, the core shape may not fit within the fixed rectangular CLB matrix of the FPGA. In this case the *Create_RPM* option should be de-selected.

Valid ranges for the parameters are given in Table 3.

Parameter	Min	Max
Symbol Width	3	12
n	3	$2^{(Symbol\_Width)}$-1
k	1	$2^{(Symbol\_Width)}$-3
r = n–k	2	min(n-k, 256)
Generator Start	0	-
h	1	-

Table 3 – Parameter Ranges

Latency

The latency of the core depends on whether the user has chosen to optimize it for area or speed. An area-optimized core has a two clock cycle latency, therefore 'data_out' reflects what was happening on the inputs (except 'reset') two rising clock edges previously. A speed-optimized core has a three clock cycle latency, therefore 'data_out' reflects what was happening on the inputs (except 'reset') three clock edges previously.

Core Resource Utilization

The area of the core increases with *n-k* and *Symbol_Width*. Some example implementations are shown in Table 4 to Table 7. 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 will give lower output clock-to-out times and predictable set up and hold times.

The results were obtained with the "–c 1" packfactor option applied during mapping. This causes the Xilinx mapper to pack as much logic as possible into each CLB.

Performance Characteristics

In general, performance increases as *n-k* and *Symbol_Width* decrease. The clock frequencies given in Table 4 to Table 7 can be comfortably achieved when the corresponding period constraint is specified for the core clock input. It may be possible to improve slightly on these values by trying different seed values for the place and route software. If necessary, performance can easily be increased by selecting a part with a faster speed grade.

Table 5 shows four possible ways of implementing the DVB Encoder standard (ETS 300 421) by varying the *Optimization* and *Create_RPM* parameters. Notice that de-selecting the *Create_RPM* parameter can result in higher maximum operating frequencies. However, this may not always be true if the core is used with other cores or user-defined logic. The RPM version of the core is more likely to maintain its performance characteristics when used with other cores and logic, and should also require less time to place and route.

Ordering Information

This core can only be obtained by agreeing to the terms of the Xilinx LogiCORE™ Reed-Solomon license. Please contact Xilinx for further information.

	ATSC	ATSC	CCSDS	CCSDS	Custom
Symbol Width	8	8	8	8	4
Field Polynomial	285	285	391	391	19
n	207	207	255	255	15
k	187	187	223	223	13
Generator Start	0	0	112	112	0
h	1	1	11	11	1
Optimization	Area	Area	Area	Speed	Speed
Create RPM	Yes	No	Yes	No	Yes
Xilinx Part	XCS10XL-5	XCS05XL-5	XC4013XLA-09	XC4013XLA-09	XC4013XLA-09
Use IOB Flip-Flops	Yes	Yes	Yes	Yes	Yes
Area (CLBs)	91	95	139	166	14
CLBs Remaining	105	5	437	410	562
Latency	2	2	2	3	3
Max. Clock Freq.[1]	34 MHz[3]	45 MHz[3]	44 MHz	67 MHz	104 MHz

Notes:
1. Higher frequencies may be attainable by setting the packfactor option on the mapper to "-c 100" rather than "-c 1".
2. Area and max clock frequencies are provided as a guide. They may vary with new releases of the Xilinx implementation tools, etc.
3. Higher frequencies are attainable by moving to the XC4000 or Virtex target device families.

Table 4 - Example XC4000/Spartan Encoder Implementations

	DVB#1	DVB#2	DVB#3	DVB#4
Symbol Width	8	8	8	8
Field Polynomial	285	285	285	285
n	204	204	204	204
k	188	188	188	188
Generator Start	0	0	0	0
h	1	1	1	1
Optimization	Area	Area	Speed	Speed
Create RPM	Yes	No	Yes	No
Xilinx Part	XC4013XLA-09	XC4013XLA-09	XC4013XLA-09	XC4013XLA-09
Use IOB Flip-Flops	Yes	Yes	Yes	Yes
Area (CLBs)	75	79	102	106
CLBs Remaining	501	497	474	470
Latency	2	2	3	3
Max. Clock Freq.[1]	47 MHz	54 MHz	62 MHz	71 MHz

Notes:
1. Higher frequencies may be attainable by setting the packfactor option on the mapper to "-c 100" rather than "-c 1".
2. Area and max clock frequencies are provided as a guide. They may vary with new releases of the Xilinx implementation tools, etc.

Table 5 - Example XC4000/Spartan DVB Encoder (ETS 300 421) Implementations

	ATSC	ATSC	CCSDS	Custom1	Custom2
Symbol Width	8	8	8	4	12
Field Polynomial	285	285	391	19	4179
n	207	207	255	15	512
k	187	187	223	13	448
Generator Start	0	0	112	0	1
h	1	1	11	1	3
Optimization	Speed	Speed	Speed	Speed	Speed
Create RPM	Yes	No	Yes	Yes	Yes
Xilinx Part	XCV50-6	XCV50-6	XCV50-6	XCV50-6	XCV50-6
Use IOB Flip-Flops	Yes	Yes	Yes	Yes	Yes
Area (Slices)	118	123	167	19	571
Slices Remaining	650	645	601	749	197
Latency	3	3	3	3	3
Max. Clock Freq.[1]	109 MHz	112 MHz	102 MHz	172 MHz	71 MHz

Notes:
1. Higher frequencies may be attainable by setting the packfactor option on the mapper to "-c 100" rather than "-c 1".
2. Area and max clock frequencies are provided as a guide. They may vary with new releases of the Xilinx implementation tools, etc.

Table 6 - Example Virtex Encoder Implementations

	DVB#1	DVB#2	DVB#3
Symbol Width	8	8	8
Field Polynomial	285	285	285
n	204	204	204
k	188	188	188
Generator Start	0	0	0
h	1	1	1
Optimization	Area[3]	Speed	Speed
Create RPM	Yes	Yes	No
Xilinx Part	XCV50-6	XCV50-6	XCV50-6
Use IOB Flip-Flops	Yes	Yes	Yes
Area (Slices)	107	107	112
Slices Remaining	661	661	656
Latency	2	3	3
Max. Clock Freq.[1]	81 MHz	110 MHz	113 MHz

Notes:
1. Higher frequencies may be attainable by setting the packfactor option on the mapper to "-c 100" rather than "-c 1".
2. Area and max clock frequencies are provided as a guide. They may vary with new releases of the Xilinx implementation tools, etc.
3. Selecting area optimization for Virtex core has no effect on the number of slices required but it does result in a latency of 2 rather than 3, at the expense of a reduced max clock frequency.

Table 7 - Example Virtex DVB Encoder (ETS 300 421) Implementations