Reed-Solomon Decoder

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

- High speed, compact Reed-Solomon Decoder
- Implements many different Reed-Solomon coding standards
- Fully synchronous design using a single clock
- Supports continuous input data with no gap between code blocks
- Automatically configured by user entered parameters
- Symbol size from 3 to 12 bits*
- Code block length variable up to 4095 symbols
- Supports shortened codes
- Supports error and erasure** decoding
- Parameterizable number of errors corrected
- Supports any primitive field polynomial for a given symbol size
- Counts number of errors corrected and flags failures
- User-selectable control signal behavior
- High performance and density due to use of Xilinx Relational Placed Macro (RPM) mapping and placement technology
- Available for all Virtex™, Virtex™-E, Spartan™-II, XC4000 and Spartan™ family members.

* Symbol sizes greater than 8 are not available for XC4000 and Spartan devices.
** See page 5 for limitations on erasure support.

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. In a systematic code the complete code block is formed from the k data symbols, followed by the n-k check symbols.

Normally $n = 2^{(\text{Symbol Width})}-1$. If n is less than this then the code is referred to as a “shortened code”. The Decoder 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 user-defined.

The Reed-Solomon Decoder samples the n symbols on the ‘data_in’ port and attempts to correct any errors. The corrected symbols are output on the ‘data_out’ port after a fixed latency.

The maximum number of errors in a block that can be guaranteed to be corrected is $t = (n\text{-}k)/2$. This is always rounded down to the nearest whole number. If a block is received with more than t errors then the Decoder will fail. Depending on the number of errors, the Decoder may or may not be able to determine if there was a failure.

Pinout

Some of the input pins are optional. The outputs that are not required should be left unconnected. The Xilinx mapping software will remove the logic driving them, ensuring FPGA resources are not wasted.

A representative symbol, with the signal names, is shown in Figure 1.

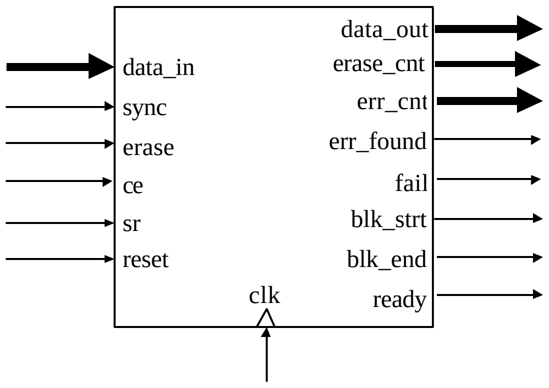

Figure 1 - Pinout

Table 1 summarizes the signal functions. They are described in more detail in the remainder of this section.

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.
‘Reset’ has a high fan-out and should be driven by a GSR buffer.

SR Input

The Synchronous Reset input is an optional pin. It can be used to re-initialize the Decoder at any time. ‘Sr’ needs to be asserted high for at least one symbol period to initialize the circuit. The Decoder becomes ready for normal operation as soon as ‘sr’ goes low. This pin should be selected with caution as it increases the size of the core and may reduce performance.

The timing for the ‘sr’ input is illustrated in Figure 2. Note that the ‘data_out’ output is not reset by ‘sr’. The Decoder’s symbol buffer is not cleared, so any symbols sampled prior to the synchronous reset will continue to be shifted out after the normal latency, albeit with no error correction.

Signal	Signal Direction	Description
clk	Input	Clock – active on rising edge
reset	Input	Active high asynchronous initialize
sr	Input	Synchronous reset (optional)
data_in	Input	Input data
ce	Input	Clock enable (optional)
sync	Input	Timing control input
erase	Input	Flag an input symbol as an erasure (optional)
data_out	Output	Corrected data output
err_cnt	Output	Number of errors detected in a block
erase_cnt	Output	Number of erasures flagged in a block (optional)
err_found	Output	High if Decoder found any errors in the block
fail	Output	High if Decoder failed to correct the block
blk_strt	Output	High to signal the start of a block on ‘data_out’
blk_end	Output	High to signal the end of a block on ‘data_out’
ready	Output	High when the Decoder is ready to accept symbols

Table 1 - I/O Ports

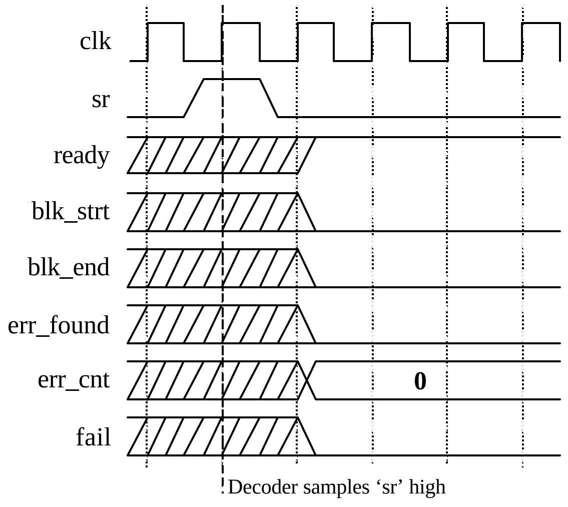

Figure 2 - Synchronous Reset Timing

Data_in Input
This is the input bus for the incoming Reed-Solomon coded data. The width of the bus is set by the symbol width parameter. Prior to a code block start being signaled with the 'sync' input, 'data_in' is passed through to 'data_out' with no error correction and latency as described on page 8.

CE Input
The Clock Enable input is another optional pin. It can be used to tell the Decoder to ignore some of the symbols coming in on 'data_in'. When 'ce' is de-asserted (low), all the other synchronous inputs are ignored and the core remains in its current state. This pin should only be used if it is genuinely required because it has a high fan out within the core and may result in lower performance.
An example of 'ce' operation is shown in Figure 3. In this case, the Decoder will ignore symbol D_4. Note that the Decoder still samples *n* symbols; D_4 will not be one of the *n*.
In this figure, the symbol period is the same as the clock period, with one symbol per clock cycle. It is possible to have more than one clock cycle per symbol period. This is explained in the processing delay section on page 6.

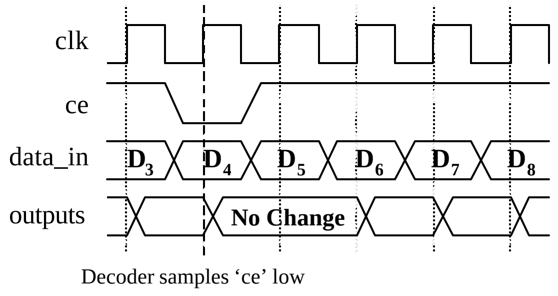

Figure 3 - Clock Enable Timing

Notice that the Decoder samples 'ce', as it is a synchronous input. The Decoder outputs will not change on the *following* rising 'clk' edge.

Sync Input
The timing of the core is controlled via the 'sync' input. The user can select between a number of control signal behaviors to control the timing. Currently two modes are defined:

1) Start Pulse
'Sync' input is high for one, and only one, symbol period when the first symbol of a code block is on 'data_in', low otherwise.

2) Data Symbol Enable
'Sync' input is high whilst the *k* data symbols are on 'data_in', low otherwise.

"Start Pulse" timing mode is illustrated in Figure 4. D_0 is the first symbol of a code block. Once started, the remaining symbols in the code block are sampled on consecutive symbol periods. It is impossible to start another code block before the first one has been completely sampled. If 'sync' is brought low and high again before the block is over, then the output for the current block will be corrupted and a further *n* cycles, at least, must elapse before a new block can be started.

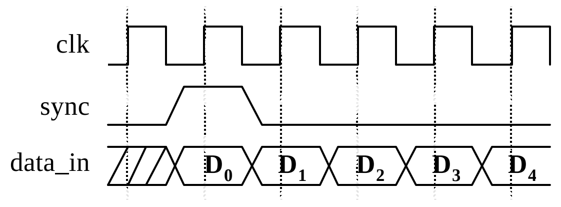

Figure 4 - Start Pulse Timing

"Data Symbol Enable" timing mode is illustrated in Figure 5 and Figure 6. In this case, the 'sync' input is held high until the last data symbol has been sampled. It is then held low until the start of the next code block.

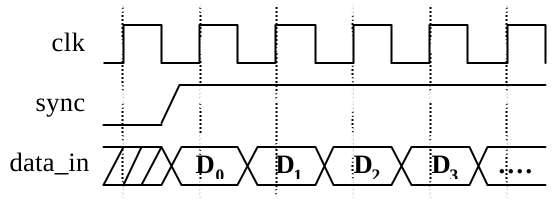

Figure 5 - Data Symbol Enable Timing

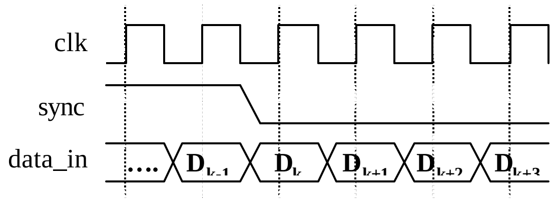

Figure 6 - Data Symbol Enable Timing

Erase Input
This optional input is only available when erasure support is required. Erasure handling is described on page 5.

Data_out Output
This is the output bus for the corrected symbols. This bus always has the same width as 'data_in'.
Corrected symbols start to appear a predefined number of symbol periods after the first symbol is sampled on 'data_in'. This delay is termed the "latency" of the Decoder and is fully explained in the Latency section on page 8.

Blk_strt Output
'Blk_strt' is pulsed high for one symbol period to indicate that the first symbol of a block is currently on 'data_out'. The timing for 'blk_strt' is illustrated in Figure 7.

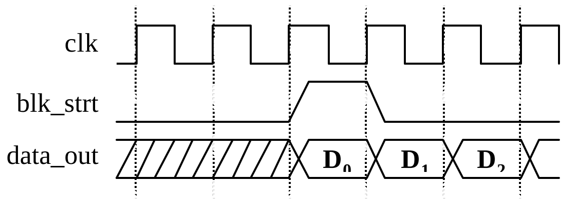

Figure 7 - 'blk_strt' Timing

Blk_end Output
'Blk_end' is pulsed high for one symbol period to indicate that the last symbol of a block is currently on 'data_out'. The timing for 'blk_end' is illustrated in Figure 8.

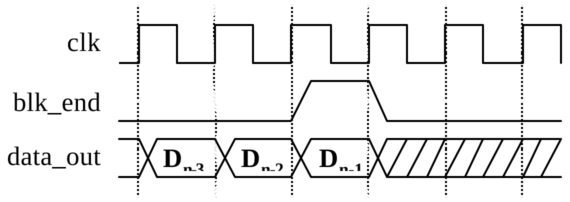

Figure 8 - 'blk_end' Timing

Err_found Output
This is one of a number of status outputs, which are set as the last symbol of a block is output on 'data_out'. 'Err_found' goes high at this time if the Decoder detected any errors in the code block. If no errors were found then 'err_found' goes low at this time. The status outputs retain their state until the end of the next code block or the core is reset. The timing for all the status outputs is illustrated in Figure 9.

Err_cnt Output
This is another of the status outputs. The 'err_cnt' bus gives the number of errors that were corrected in the block just output. The width of the bus depends on the input parameters *n* and *k*. The bus width is equal to the number of binary bits required to represent (*n*-*k*). If *n*-*k* = 16, for example, then the 'err_cnt' bus will be five bits wide.

Erase_cnt Output
This status output is only available when erasure support is required. The bus width is equal to the number of binary bits required to represent *n*. Erasure handling is described on page 5.

Fail Output
'Fail' is also a status output. The Decoder sets 'fail' high if it determines that there were more errors in the code block than it could correct.

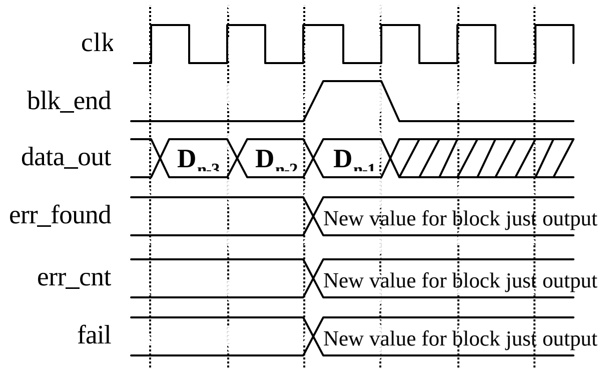
clk
blk_end
data_out
Dn-3
Dn-2
Dn-1
err_found
New value for block just output
err_cnt
New value for block just output
fail
New value for block just output

Figure 9 - Status Output Timing

Ready Output
The 'ready' output is high when the Decoder is ready to sample symbols on 'data_in'. If the processing

delay (see page 6) is greater than *n*, 'ready' will go low after the last symbol of a block is sampled. This is the case in Figure 10. It will remain low until the processing delay is over, as shown in Figure 11. If the processing delay is less than or equal to *n*, 'ready' will always be high.

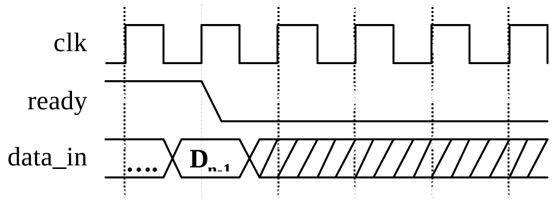

Figure 10 - 'ready' Timing

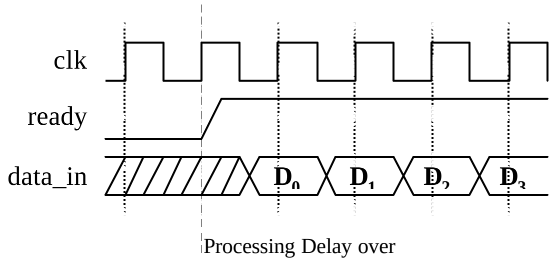

Figure 11 - 'ready' Timing

If the processing delay is greater than *n* then 'ready' always goes low one symbol period after the *second-last* symbol has been sampled. Under normal circumstances this will result in the expected behavior. However if the 'ce' input is brought low immediately after the second-last symbol has been sampled, 'ready' will go low despite the fact that the last symbol has still to be sampled. Thus in this case, 'ready' goes low a number of symbol periods earlier than expected. It will also go high again one symbol period earlier than expected. This is a feature the user must be aware of and take appropriate action if it is possible for this to occur in the system.

Erasure Decoding

An erased symbol is an input symbol that is known to be wrong. The symbol is flagged as being erased by asserting the 'erase' input high whilst the symbol is being sampled. In the example shown in Figure 12, D_2 is flagged as an erasure.

The Decoder will correct the code block if $2e + E \le n-k$, where e is the number of errors and E is the number of erasures.

The 'erase_cnt' output provides a count of the number of erasures that were flagged for the block just output. It is updated at the same time as 'err_cnt' and the other status outputs. If erasure decoding is selected then 'err_cnt' provides a count of the number of errors plus erasures that were corrected.

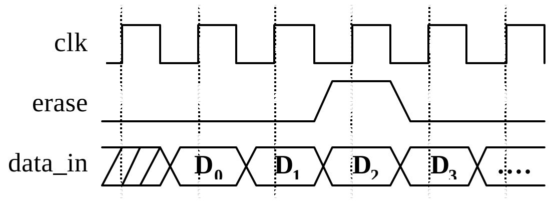

Figure 12 - 'erase' Timing

Erasure decoding increases the size of the core considerably. It should only be selected if it is essential and is only recommended for codes where *n-k* is less than eight. The core does support erasures for larger values of *n-k* but there will be a large area overhead compared to the same core without erasure support. See the example implementations in Table 5 on page 10.

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:

- **Generator Start**

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

Equation 1

$$g(x) = \prod_{i=generator\_start}^{generator\_start+n-k-1} (x - a^{i})$$

Normally *generator start* is 0 or 1, however the core will accept any positive integer.

- **k**

Number of information or data symbols in a code block.

- **n**

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

- **Field Polynomial**

This is the Galois Field polynomial, used to generate the Galois Field for the code. Polynomials are entered as decimal numbers. The bits of the binary equivalent correspond to the polynomial coefficients. For example,

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

A value of zero causes the default polynomial for the given symbol width to be selected.

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

- **Symbol Width**

This is the bus-width of 'data_in' and 'data_out'.

- **Clock Enable**

Select this if a clock enable input is required.

- **Synchronous Reset**

Select this if a synchronous reset input is required.

- **Erase**

Select this if erasure support is required. Note the provisos in the Erasure Decoding section on page 5.

- **Sync Mode**

This allows you to select the timing control mode, either "Start Pulse" or "Data Symbol Enable", as described in the "Sync Input" section on page 3.

- **Clock Periods Per Symbol**

Normally there is only one clock period per symbol. This may be increased to reduce the processing delay. This is described in the next section. The symbol period must always be a whole number of clock periods.

- **Target Device**

Select the required target device family from the list. The core functionality is independent of the target device. The maximum operating frequency and area are likely to differ between family selections.

- **Memory Style**

If the target device architecture supports block memory then 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. Note that for symbol widths of 8 and under, this option will result in no block memories being used. For symbol widths greater than 8, some will be used but their use will be kept to a minimum.

- *Block* - core should use block memories wherever possible. This will keep the number of CLBs used to a minimum but may use block memory wastefully.

- *Automatic* - allow the core to use the most appropriate style of memory for each case, based on required memory depth.

The parameter values for several common standards are given in Table 4 on page 10.

Valid ranges for the parameters are given in Table 3.

Parameter	Min	Max	Notes
n	3	$2^{(Symbol\_Width)}-1$	
k	1	$2^{(Symbol\_Width)}-3$	1
r=n-k	2	128	2
Symbol Width	3	12	3
Gen Start	0	-	
Clock Periods Per Symbol	1	65535	4

Table 3 - Parameter Ranges

Notes:
1. Max=n-r
2. In reality, r is limited by the maximum size of device available. If the core exceeds the device size due to r being large, and a larger FPGA cannot be selected, then the size of the core can be reduced by increasing the number of clock periods per symbol. If erasure support is enabled then r is limited to 64.
3. Symbol widths greater than 8 are not available for XC4000 and Spartan devices.
4. The practical limit for this parameter will usually be caused by the core maximum clock frequency being reached.

Processing Delay

For some parameter selections, the Decoder may not be ready to accept one code block immediately after another. This is because it is still processing the first block. The processing delay for a given *t*, assuming one clock cycle per symbol period, is shown in Figure 13. This is the minimum number of symbol periods from the start of the first symbol period of a code block, before another code block may be started. If the processing delay is greater than *n*, it is not possible to follow one code block immediately with another. Note that measures can be taken to reduce the processing delay.

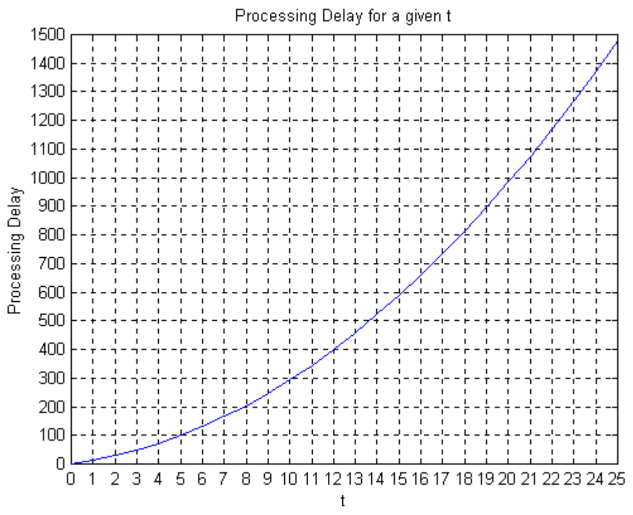

Figure 13 - Processing Delay vs. *t*

The number of symbol periods may be calculated using the following equation:

Equation 2

$$Processing\ Delay = t^2 + 2(3t + \sum_{i=1}^{t+1} i) + 2$$

If erasure decoding is enabled then the following equation should be used:

Equation 3

$$Processing\ Delay = 2(\sum_{i=1}^{n-k+1} i\,) + 3(n\text{-}k) + 3$$

If necessary, the processing delay can be reduced by increasing the number of clock periods per symbol.

In this case the processing delay (in symbol periods) may be calculated using the following expression:

Equation 4

$$1 + RoundDown(\mathrm{P/ClksPerSym})$$

"P" is the result from Equation 2 or Equation 3. "ClksPerSym" is the number of clock cycles per symbol. This may be increased until the maximum input clock frequency of the core is reached. Note that the processing delay is always rounded down to the nearest whole number. To calculate the processing delay in terms of clock cycles, multiply the above result by the number of clock periods per symbol.

If the number of clock periods per symbol is greater than one then all synchronous inputs must be synchronized to symbol periods. This is illustrated for four clock periods per symbol in Figure 14.

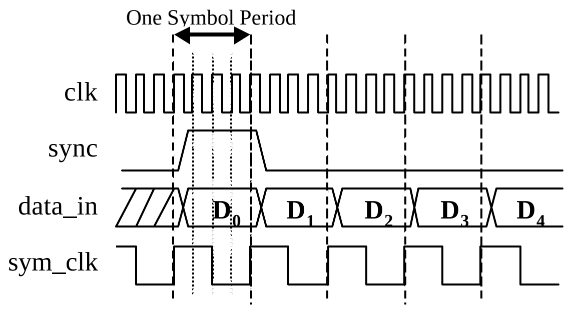

Figure 14 - Input Timing, 4 Periods Per Symbol

The Decoder samples its inputs on rising edges of 'clk' coincident with rising edges of the imaginary 'sym_clk' signal.

The outputs are also all synchronized to the symbol period. This is illustrated for three clock periods per symbol in Figure 15.

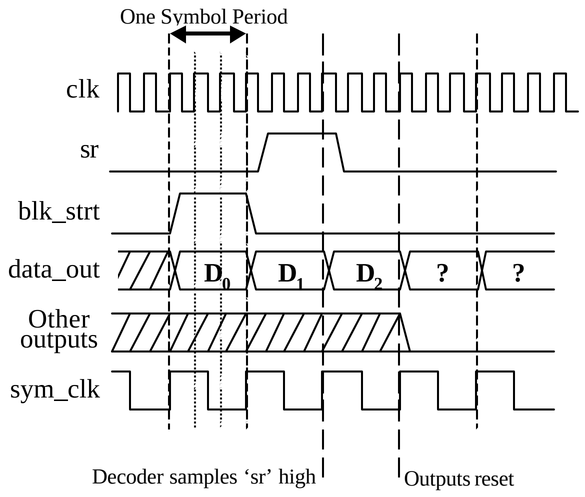

Figure 15 - Output Timing, 3 Periods Per Symbol

This figure also shows the timing for 'sr' when the number of clock periods per symbol is greater than one. 'Sr' must be pulsed for a whole number of symbol periods, just like all the other synchronous inputs. It is sampled high and the outputs are reset one symbol period later.

The Decoder always resynchronizes itself when it detects a rising edge on 'sync'. If the rising edge of 'sync' occurs less than two 'clk' periods before the rising edge of 'sym_clk', then the Decoder samples the first symbol twice. This situation is illustrated in Figure 16. The Decoder still operates correctly, with D_0 as the first symbol of the block and D_1 as the second. D_0 appears on 'data_out' 'latency' symbol periods after t1. The corrected value of D_0 appears 'latency' symbol periods after t2.

This may occur when 'sync' goes high at the start of the very first code block. It will not re-occur if the gaps between blocks are always a whole number of symbol periods.

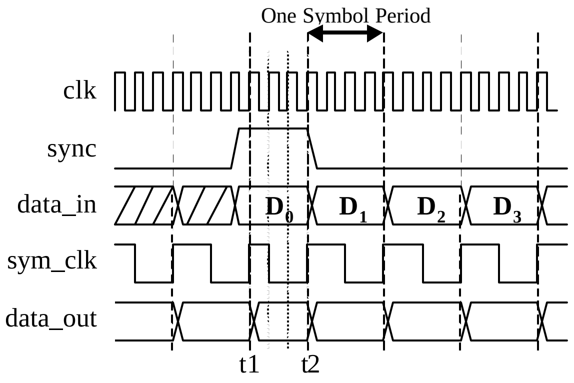

Figure 16 - Resynchronization using 'sync'

IMPORTANT - If a code block is started prior to corrected symbols from the previous block appearing on 'data_out', there must be an integer number of symbol periods between the last symbol of the previous block and the first symbol of the new block. This is illustrated in Figure 17. Figure 18 shows the case where there is a non-integer number of symbol periods between blocks A and B. This may cause the Decoder to lose track of block A.
It is only safe to start another block with 'sync' misaligned, relative to the symbol periods from the previous block, after the corrected symbols from the previous block have started to appear on 'data_out'.

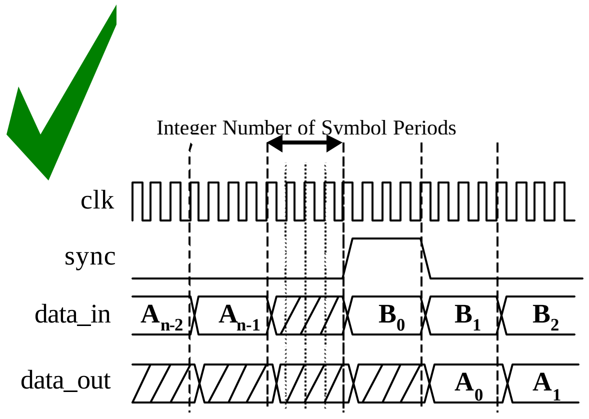

Figure 17 - Integer Number of Symbols between Blocks

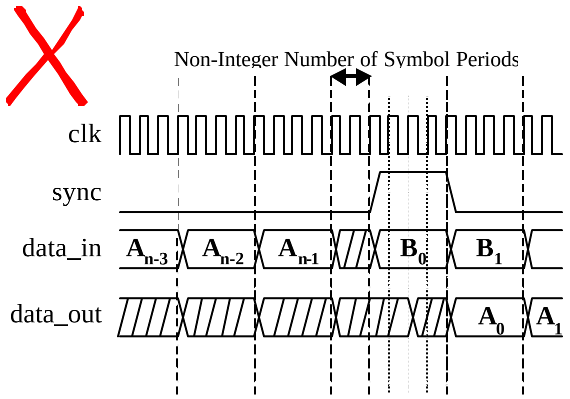

Figure 18 - Resynchronization before 'data_out' for Previous Block

Latency

The latency is the number of symbol periods from a symbol being sampled on 'data_in', to the corrected version of that symbol appearing on 'data_out'. This should not be confused with the processing delay; a subsequent code block may be started before the latency delay is over.

An example, with a latency of two symbol periods and one clock period per symbol, is shown in Figure 19. In reality, the latency will usually be much greater than this.

The latency is dependent on the values of *n* (the number of symbols in a code block) and *t* (the number of correctable errors). The total latency may be determined from the following equation:

Equation 5

$$Latency = n + m + ProcessingDelay - s$$

This gives the latency in symbol periods. The variables, *m* and *s*, are defined as follows:

Symbol Width	*m*
=8	6
≠8	5

Clock Periods Per Symbol	*s*
=1	0
>1	1

To calculate the latency in clock periods just multiply the above result by the number of clock periods per symbol.

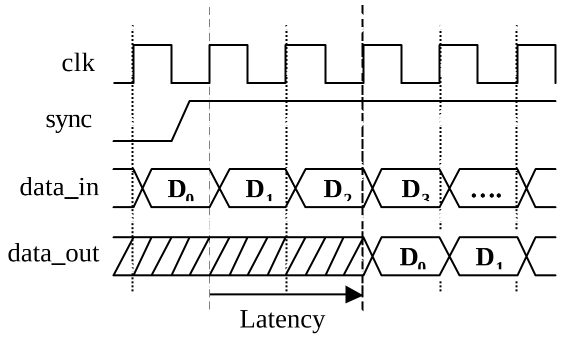

Figure 19 - Latency = 2

Core Resource Utilization

The area of the core increases with *n*, *n-k* and the symbol width. Some example configurations are shown in Table 4 to Table 7. In these tables, the 'ce' and 'sr' inputs were used, as were all the outputs. The CLB counts can be reduced slightly by selecting the option to map primary I/O registers into IOBs during placement. This option should certainly 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. This may reduce performance slightly.

Performance Characteristics

It is important to set a maximum period constraint on the core clock input. The figures in Table 4 to Table 7 show clock speeds that can be comfortably achieved when this is done. It may be possible to improve slightly on these values by trying different seed values for the place and route software or by setting the packfactor to give a less dense layout (use mapper option "-c 100"). If necessary, performance can easily be increased by selecting a part with a faster speed grade. Performance increases as *n*, *n-k* and the symbol width decrease.

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.

	DVB (ETS 300 421)	ATSC			
Generator Start	0	0	0	0	0
k	188	187	223	223	223
n	204	207	255	255	255
Polynomial	285	285	285	285	285
Symbol Width	8	8	8	8	8
Sync Mode	Data Sym En	Data Sym En	Start Pulse	Start Pulse	Start Pulse
Erasure Decoding	No	No	No	No	No
Clock Periods Per Symbol	1	1	1	8	20
Processing Delay[1]	204	294	660	83	34
Latency[1]	414	507	921	343	294
Xilinx Part	XC4020XLA-09	XC4028XLA-09	XC4036XLA-09	XC4036XLA-09	XC4036XLA-09
Use IOB Flip-Flops	No	No	Yes	Yes	Yes
Area (CLBs)	702	845	1095	953	943
CLBs Remaining	82	179	201	343	353
Max. Clock Freq.[2]	47 MHz	44 MHz	43 MHz	41 MHz	41 MHz

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

Table 4 - Example XC4000/Spartan Decoder Implementations

Generator Start	0	0	120	120
k	20	20	124	124
n	27	27	128	128
Polynomial	37	37	391	391
Symbol Width	5	5	8	8
Sync Mode	Start Pulse	Start Pulse	Start Pulse	Start Pulse
Erasure Decoding	No	Yes	No	Yes
Clock Periods Per Symbol	1	1	1	1
Processing Delay[1]	49	96	30	45
Latency[1]	81	128	164	179
Xilinx Part	XCS20XL-5	XCS30XL-5	XC4013XLA-09	XC4013XLA-09
Use IOB Flip-Flops	Yes	Yes	Yes	Yes
Area (CLBs)	235	427	376	547
CLBs Remaining	165	149	200	29
Max. Clock Freq.[2]	60 MHz	46 MHz	48 MHz	48 MHz

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

Table 5 - More Example XC4000/Spartan Decoder Implementations

	DVB (ETS 300 421)	DVB (ETS 300 421)			
Generator Start	0	0	0	0	0
k	188	188	223	223	223
n	204	204	255	255	255
Polynomial	285	285	285	285	285
Symbol Width	8	8	8	8	8
Sync Mode	Data Sym En	Data Sym En	Start Pulse	Start Pulse	Start Pulse
Erasure Decoding	No	No	No	No	No
Clock Periods Per Symbol	1	1	1	8	20
Processing Delay[1]	204	204	660	83	34
Latency[1]	414	414	921	343	294
Xilinx Part	XCV50-6	XCV50-6	XCV100-6	XCV100-6	XCV100-6
Use IOB Flip-Flops	No	No	Yes	Yes	Yes
Memory Style	Block	Distributed	Automatic	Automatic	Automatic
Block RAMs Used	2	0	3	2	2
Area (Slices)	637	764	1012	1006	1005
Slices Remaining	131	4	188	194	195
Max. Clock Freq.[2]	62 MHz	62 MHz	55 MHz	56 MHz	59 MHz

Notes:
1. Measured in symbol periods.
2. Higher frequencies may be attainable by setting the packfactor option on the mapper to “-c 100” rather than “-c 1”.
3. 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 Decoder Implementations

	ATSC				
Generator Start	0	0	0	120	120
k	187	20	20	124	124
n	207	27	27	128	128
Polynomial	285	37	37	391	391
Symbol Width	8	5	5	8	8
Sync Mode	Data Sym En	Start Pulse	Start Pulse	Start Pulse	Start Pulse
Erasure Decoding	No	No	Yes	No	Yes
Clock Periods Per Symbol	1	1	1	1	1
Processing Delay[1]	294	49	96	30	45
Latency[1]	507	81	128	164	179
Xilinx Part	XCV100-6	XCV50-6	XCV50-6	XCV50-6	XCV50-6
Use IOB Flip-Flops	No	Yes	Yes	Yes	Yes
Memory Style	Automatic	Automatic	Automatic	Automatic	Automatic
Block RAMs Used	2	1	1	2	2
Area (Slices)	775	214	393	335	530
Slices Remaining	425	554	375	433	238
Max. Clock Freq.[2]	63 MHz	75 MHz	74 MHz	60 MHz	60 MHz

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

Table 7 - More Example Virtex Decoder Implementations