Reed-Solomon II IP Core User Guide

Subscribe

Send Feedback

UG-01090
2016.05.02

101 Innovation Drive
San Jose, CA 95134
www.altera.com

ALTERA® now part of Intel

Contents

About the Reed-Solomon II IP Core 1

2016.05.02

UG-01090

Related Information

- **Reed-Solomon II IP Core Document Archives** on page 5-1
  Provides a list of user guides for previous versions of the Reed-Solomon II IP core.
- **Introduction to Altera IP Cores**
  Provides general information about all Altera IP cores, including parameterizing, generating, upgrading, and simulating IP.
- **Creating Version-Independent IP and Qsys Simulation Scripts**
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- **Project Management Best Practices**
  Guidelines for efficient management and portability of your project and IP files.
- **High-speed Reed-Solomon IP Core User Guide**
  The Altera High-speed Reed-Solomon IP Core uses a highly parallel architecture for large applications that require throughput of 100 Gbps and greater. The IP core is suitable for 10G (such as optical transport networks (OTN)) or 100G Ethernet (IEEE 802.3bj/bm) applications.

Altera DSP IP Core Features

- Avalon® Streaming (Avalon-ST) interfaces
- DSP Builder ready
- Testbenches to verify the IP core
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators

Reed-Solomon II IP Core Features

- High-performance encoder or decoder for error detection and correction:
  - Fully parameterizable:
  - Number of channels
  - Number of bits per symbol
  - Number of symbols per codeword
  - Number of check symbols per codeword
  - Field polynomial
- Erasures-supporting decoder—-the decoder can correct symbol errors up to the number of check symbols, if you give the location of the errors to the decoder
- Error symbol output—the decoder provides the error values
- Bit error output—either split count or full count
- Multiple channels for resource sharing

DSP IP Core Device Family Support

Altera® offers the following device support levels for Altera IP cores:

- Preliminary support—Altera verifies the IP core with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. You can use it in production designs with caution.
- Final support—Altera verifies the IP core with final timing models for this device family. The IP core meets all functional and timing requirements for the device family. You can use it in production designs.

Table 1-1: DSP IP Core Device Family Support

Device Family	Support
Arria® II GX	Final
Arria II GZ	Final
Arria V	Final
Arria 10	Final
Cyclone® IV	Final
Cyclone V	Final
MAX® 10 FPGA	Final
Stratix® IV GT	Final
Stratix IV GX/E	Final
Stratix V	Final
Other device families	No support

DSP IP Core Verification

Before releasing a version of an IP core, Altera runs comprehensive regression tests to verify its quality and correctness. Altera generates custom variations of the IP core to exercise the various parameter options and thoroughly simulates the resulting simulation models with the results verified against master simulation models.

Reed-Solomon II IP Core Release Information

Use the release information when licensing the IP core.

Table 1-2: Release Information

Item	Description
Version	16.0
Release Date	May 2016
Ordering Code	IP-RSCODECII (Primary License) IPR-RSCODECII (Renewal License)
Product ID	00E5 (Encoder/Decoder)
Vendor ID	6AF7

Altera verifies that the current version of the Quartus Prime software compiles the previous version of each IP core. Altera does not verify that the Quartus Prime software compiles IP core versions older than the previous version. The *Altera IP Release Notes* lists any exceptions.

Related Information

- **Altera IP Release Notes**
- **Errata for Reed-Solomon IP core in the Knowledge Base**

Reed-Solomon II IP Core Performance and Resource Utilization

Table 1-3: Performance and Resource Utilization

Typical expected performance for a Reed-Solomon II IP Core using the Quartus Prime software with the Arria V (5AGXFB3H4F40C4), Cyclone V (5CGXFC7D6F31C6), and Stratix V (5SGSMD4H2F35C2) devices.

Device	Parameters				ALM	Memory		Registers		fMAX (MHz)
	Type	Check Symbols	Bits Per Symbol	Bits Per Check Symbol		M10K	M20K	Primary	Secondary	
Arria V	Erasures decoder	16	8	204	1,687	1	--	1,765	291	217
Arria V	Erasures variable decoder	16	8	204	1,688	1	--	1,810	269	213
Arria V	Full error decoder	16	8	204	952	1	--	989	170	239
Arria V	Split error decoder	16	8	204	976	1	--	999	144	224
Arria V	Standard decoder large	32	8	255	1,628	1	--	1,751	285	215
Arria V	Standard decoder medium	16	8	204	944	1	--	974	178	225
Arria V	standard decoder small	6	4	15	201	1	--	272	23	315
Arria V	Standard encoder	16	8	204	87	0	--	164	0	422
Arria V	Variable decoder	16	8	204	964	1	--	1,019	174	209
Arria V	Variable encoder large	32	8	204	904	0	--	299	0	234
Arria V	Variable encoder small	16	8	204	444	0	--	169	0	259
Cyclone V	Erasures decoder	16	8	204	1,670	1	--	1,769	366	192
Cyclone V	Erasures variable decoder	16	8	204	1,683	1	--	1,812	342	196
Cyclone V	Full error decoder	16	8	204	953	1	--	989	232	215

Device	Parameters				ALM	Memory		Registers		fMAX (MHz)
	Type	Check Symbols	Bits Per Symbol	Bits Per Check Symbol		M10K	M20K	Primary	Secondary	
Cyclone V	Split error decoder	16	8	204	968	1	--	1,003	198	209
Cyclone V	Standard decoder large	32	8	255	1,631	1	--	1,752	409	193
Cyclone V	Standard decoder medium	16	8	204	938	1	--	972	227	222
Cyclone V	standard decoder small	6	4	15	200	1	--	272	56	275
Cyclone V	Standard encoder	16	8	204	87	0	--	164	0	372
Cyclone V	Variable decoder	16	8	204	968	1	--	1,016	241	220
Cyclone V	Variable encoder large	32	8	204	905	0	--	299	0	188
Cyclone V	Variable encoder small	16	8	204	444	0	--	169	0	217
Stratix V	Erasures decoder	16	8	204	1,648	--	1	1,765	423	367
Stratix V	Erasures variable decoder	16	8	204	1,664	--	1	1,802	405	368
Stratix V	Full error decoder	16	8	204	955	--	1	987	252	424
Stratix V	Split error decoder	16	8	204	969	--	1	1,003	248	424
Stratix V	Standard decoder large	32	8	255	1,624	--	1	1,749	432	404
Stratix V	Standard decoder medium	16	8	204	939	--	1	972	281	410

Device	Parameters				ALM	Memory		Registers		fMAX (MHz)
	Type	Check Symbols	Bits Per Symbol	Bits Per Check Symbol		M10K	M20K	Primary	Secondary	
Stratix V	standard decoder small	6	4	15	197	--	1	272	52	525
Stratix V	Standard encoder	16	8	204	87	--	0	164	0	610
Stratix V	Variable decoder	16	8	204	966	--	1	1,017	270	409
Stratix V	Variable encoder large	32	8	204	902	--	0	299	0	397
Stratix V	Variable encoder small	16	8	204	435	--	0	169	0	434

Reed-Solomon II IP Core Getting Started 2

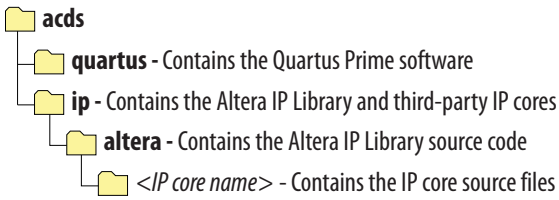
2016.05.02

UG-01090 Subscribe Send Feedback

Licensing IP Cores

The Altera IP Library provides many useful IP core functions for your production use without purchasing an additional license. Some Altera MegaCore® IP functions require that you purchase a separate license for production use. However, the OpenCore® feature allows evaluation of any Altera IP core in simulation and compilation in the Quartus® Prime software. After you are satisfied with functionality and performance, visit the Self Service Licensing Center to obtain a license number for any Altera product.

Figure 2-1: IP Core Installation Path



Note: The default IP installation directory on Windows is `<drive>:\altera\<version number>`; on Linux the IP installation directory is *<home directory>*`/altera/<version number>`.

OpenCore Plus IP Evaluation

Altera's free OpenCore Plus feature allows you to evaluate licensed MegaCore IP cores in simulation and hardware before purchase. You only need to purchase a license for MegaCore IP cores if you decide to take your design to production. OpenCore Plus supports the following evaluations:

- Simulate the behavior of a licensed IP core in your system.
- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware.

OpenCore Plus evaluation supports the following two operation modes:

- Untethered—run the design containing the licensed IP for a limited time.
- Tethered—run the design containing the licensed IP for a longer time or indefinitely. This requires a connection between your board and the host computer.

Note: All IP cores that use OpenCore Plus time out simultaneously when any IP core in the design times out.

Related Information

- **Altera Licensing Site**
- **Altera Software Installation and Licensing Manual**

Reed-Solomon II IP Core OpenCore Plus Timeout Behavior

All IP cores in a device time out simultaneously when the most restrictive evaluation time is reached. If a design has more than one IP core, the time-out behavior of the other IP cores may mask the time-out behavior of a specific IP core .

For IP cores, the untethered time-out is 1 hour; the tethered time-out value is indefinite. Your design stops working after the hardware evaluation time expires. The Quartus Prime software uses OpenCore Plus Files (`.ocp`) in your project directory to identify your use of the OpenCore Plus evaluation program. After you activate the feature, do not delete these files..

When the evaluation time expires, for encoders `out_data` goes low, `rst` goes high; for decoders, `data` goes low, `rst` goes high .

Related Information

AN 320: OpenCore Plus Evaluation of Megafunctions

IP Catalog and Parameter Editor

The IP Catalog displays the installed IP cores available for your design. Double-click any IP core to launch the parameter editor and generate files representing your IP variation. Use the following features to help you quickly locate and select an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, open the IP core's installation folder, and click links to IP documentation.
- Click **Search for Partner IP**, to access partner IP information on the Altera website.

The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Qsys system file (`.qsys`) or Quartus Prime IP file (`.qip`) representing the IP core in your project. You can also parameterize an IP variation without an open project.

The IP Catalog is also available in Qsys (**View** > **IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus Prime IP Catalog. For more information about using the Qsys IP Catalog, refer to *Creating a System with Qsys* in Volume 1 of the *Quartus Prime Handbook*.

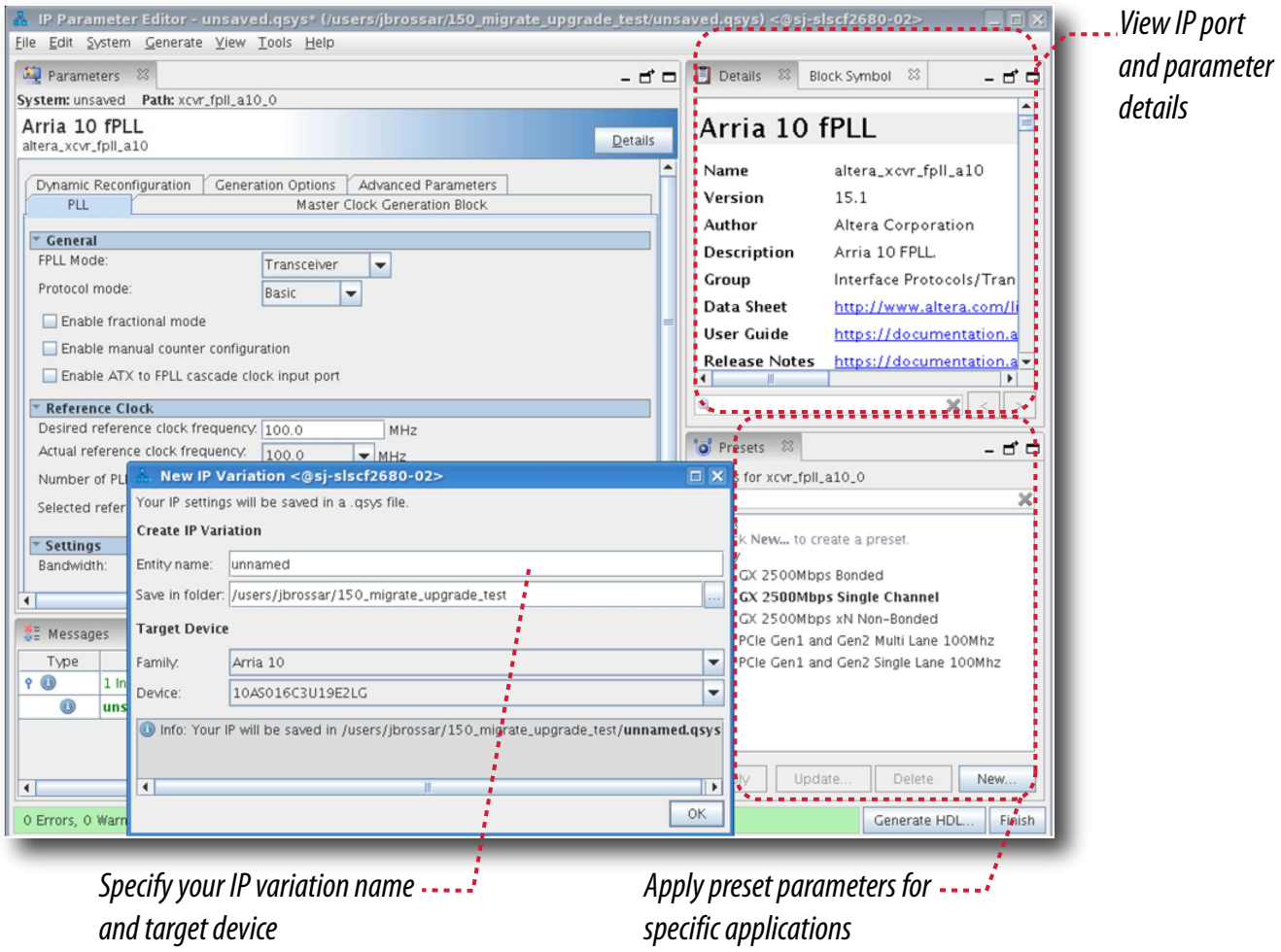
Related Information

Creating a System with Qsys

Generating IP Cores

You can quickly configure a custom IP variation in the parameter editor.
Use the following steps to specify IP core options and parameters in the parameter editor:

Figure 2-2: IP Parameter Editor



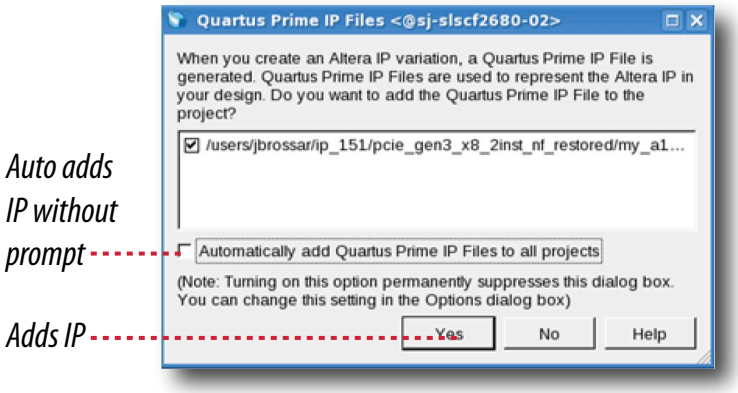
1. In the IP Catalog (**Tools** > **IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. The parameter editor saves the IP variation settings in a file named *<your_ip>*.qsys. Click **OK**. Do not include spaces in IP variation names or paths.
3. Specify the parameters and options for your IP variation in the parameter editor, including one or more of the following:

- Optionally select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
- Specify parameters defining the IP core functionality, port configurations, and device-specific features.
- Specify options for processing the IP core files in other EDA tools.

Note: Refer to your IP core user guide for information about specific IP core parameters.

4. Click **Generate HDL**. The **Generation** dialog box appears.
5. Specify output file generation options, and then click **Generate**. The IP variation files synthesis and/or simulation files generate according to your specifications.
6. To generate a simulation testbench, click **Generate** > **Generate Testbench System**. Specify testbench generation options, and then click **Generate**.
7. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate** > **Show Instantiation Template**.
8. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project. Optionally turn on the option to **Automatically add Quartus Prime IP Files to All Projects**. Click **Project** > **Add/Remove Files in Project** to add IP files at any time.

Figure 2-3: Adding IP Files to Project



Note: For Arria 10 devices, the generated `.qsys` file must be added to your project to represent IP and Qsys systems. For devices released prior to Arria 10 devices, the generated `.qip` and `.sip` files must be added to your project for IP and Qsys systems.

The generated `.qsys` file must be added to your project to represent IP and Qsys systems.

9. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

Note: Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

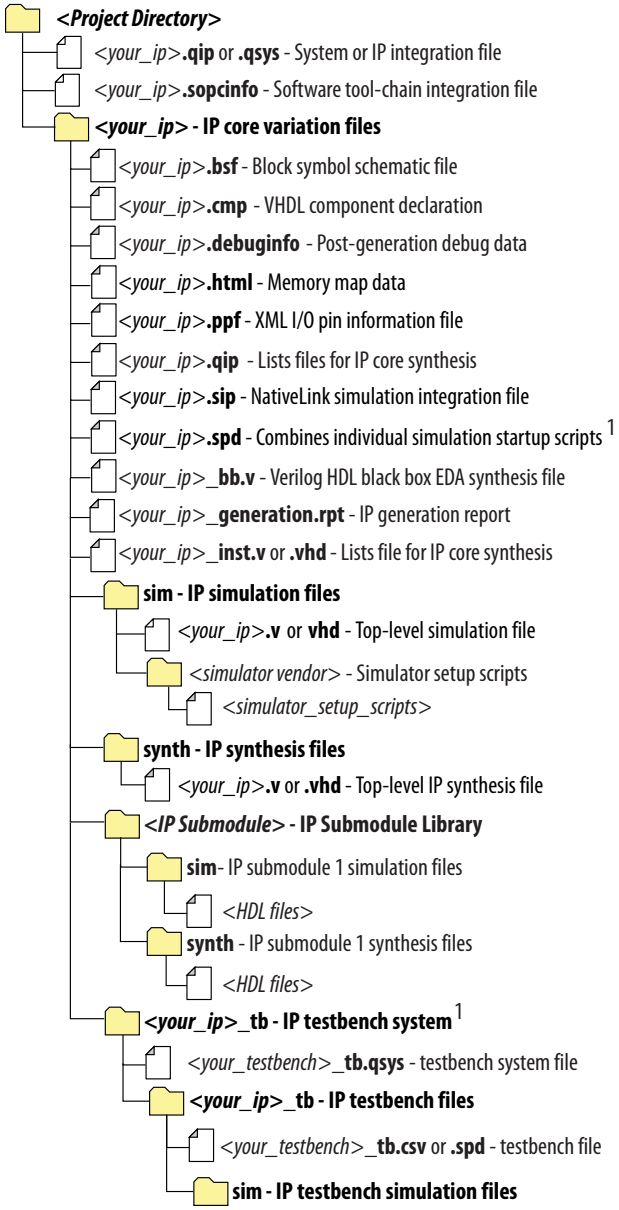
Related Information

- **IP User Guide Documentation**
- **Altera IP Release Notes**

Files Generated for Altera IP Cores and Qsys Systems

The Quartus Prime software generates the following output file structure for IP cores and Qsys systems. The generated `.qsys` file must be added to your project to represent IP and Qsys systems. For devices released prior to Arria 10 devices, the generated `.qip` and `.sip` files must be added to your Quartus Prime Standard Edition project to represent IP and Qsys systems

Figure 2-4: Files generated for IP cores and Qsys Systems



1. If supported and enabled for your IP core variation.

Table 2-1: IP Core and Qsys Simulation Generated Files

File Name	Description
`<my_ip>.qsys`	The Qsys system or top-level IP variation file.
`<system>.sopcinfo`	Describes the connections and IP component parameterizations in your Qsys system. You can parse the contents of this file to get requirements when you develop software drivers for IP components. Downstream tools such as the Nios II tool chain use this file. The `.sopcinfo` file and the `system.h` file generated for the Nios II tool chain include address map information for each slave relative to each master that accesses the slave. Different masters may have a different address map to access a particular slave component.
`<my_ip>.cmp`	The VHDL Component Declaration (**.cmp**) file is a text file that contains local generic and port definitions that you can use in VHDL design files.
`<my_ip>.html`	A report that contains connection information, a memory map showing the slave address with respect to each master that the slave connects to, and parameter assignments.
`<my_ip>_generation.rpt`	IP or Qsys generation log file. A summary of the messages during IP generation.
`<my_ip>.debuginfo`	Contains post-generation information. Passes System Console and Bus Analyzer Toolkit information about the Qsys interconnect. The Bus Analysis Toolkit uses this file to identify debug components in the Qsys interconnect.
`<my_ip>.qip`	Contains all the required information about the IP component to integrate and compile the IP component in the Quartus Prime software.
`<my_ip>.csv`	Contains information about the upgrade status of the IP component.
`<my_ip>.bsf`	A Block Symbol File (**.bsf**) representation of the IP variation for use in Quartus Prime Block Diagram Files (**.bdf**).
`<my_ip>.spd`	Required input file for `ip-make-simscript` to generate simulation scripts for supported simulators. The **.spd** file contains a list of files generated for simulation, along with information about memories that you can initialize.
`<my_ip>.ppf`	The Pin Planner File (**.ppf**) stores the port and node assignments for IP components created for use with the Pin Planner.
`<my_ip>_bb.v`	You can use the Verilog blackbox (**_bb.v**) file as an empty module declaration for use as a blackbox.

File Name	Description
`<my_ip>.sip`	Contains information required for NativeLink simulation of IP components. You must add the **.sip** file to your Quartus project to enable NativeLink for Arria II, Arria V, Cyclone IV, Cyclone V, MAX 10, MAX II, MAX V, Stratix IV, and Stratix V devices. The Quartus Prime Pro Edition does not support NativeLink simulation.
`<my_ip>_inst.v` or `_inst.vhd`	HDL example instantiation template. You can copy and paste the contents of this file into your HDL file to instantiate the IP variation.
`<my_ip>.regmap`	If the IP contains register information, the Quartus Prime software generates the .**regmap** fil. The **.regmap** file describes the register map information of master and slave interfaces. This file complements the .**sopcinfo** file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
`<my_ip>.svd`	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Qsys system. During synthesis, the Quartus Prime software stores the `.svd` files for slave interface visible to the System Console masters in the `.sof` file in the debug session. System Console reads this section, which Qsys can query for register map information. For system slaves, Qsys can access the registers by name.
`<my_ip>.v` `<my_ip>.vhd`	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
`mentor/`	Contains a ModelSim® script `msim_setup.tcl` to set up and run a simulation.
`aldec/`	Contains a Riviera-PRO script `rivierapro_setup.tcl` to setup and run a simulation.
`/synopsys/vcs` `/synopsys/vcsmx`	Contains a shell script `vcs_setup.sh` to set up and run a VCS® simulation. Contains a shell script `vcsmx_setup.sh` and `synopsys_sim.setup` file to set up and run a VCS MX® simulation.
`/cadence`	Contains a shell script `ncsim_setup.sh` and other setup files to set up and run an NCSIM simulation.
`/submodules`	Contains HDL files for the IP core submodule.
`<IP submodule>/`	For each generated IP submodule directory, Qsys generates `/synth` and `/sim` sub-directories.

Simulating Altera IP Cores

The Quartus Prime software supports RTL and gate-level simulation of Altera IP cores in supported EDA simulators. The Quartus Prime software generates simulation files for each IP core during IP generation,

including the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts for each IP core. You can use the functional simulation model and the testbench or example design generated with your IP core for simulation. The IP generation output also includes scripts to compile and run any testbench. The generated scripts list all models or libraries required to simulate your IP core.

The Quartus Prime software provides integration with your simulator and supports multiple simulation flows, including your own scripted and custom simulation flows. Whichever flow you chose, IP core simulation involves the following steps:

1. Generate simulation model, testbench (or example design), and simulator setup script files.
2. Set up your simulator environment and any simulation script(s).
3. Compile simulation model libraries.
4. Run your simulator.

The Quartus Prime software integrates with your preferred simulation environment. This section describes how to setup and run typical scripted and NativeLink simulation flows. The Quartus Prime Pro Edition software does not support NativeLink simulation.

Related Information

Simulating Altera Designs

DSP Builder Design Flow

DSP Builder shortens digital signal processing (DSP) design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

This IP core supports DSP Builder. Use the DSP Builder flow if you want to create a DSP Builder model that includes an IP core variation; use IP Catalog if you want to create an IP core variation that you can instantiate manually in your design.

Related Information

Using MegaCore Functions chapter in the DSP Builder Handbook.

Reed-Solomon II IP Core Functional Description

3

2016.05.02

UG-01090 Subscribe Send Feedback

This topic describes the IP core's architecture, interfaces, and signals.

Architecture

You can parameterize the Reed-Solomon II IP core as an encoder or a decoder. The encoder receives data packets and generates the check symbols; the decoder detects and corrects errors.

Reed-Solomon II Encoder

When the encoder receives data symbols, it generates check symbols for a given codeword and sends the input codeword together with the check symbols to the output interface. The encoder uses backpressure on the upstream component when it generates the check symbols.

Figure 3-1: Reed-Solomon II Codeword Encoding

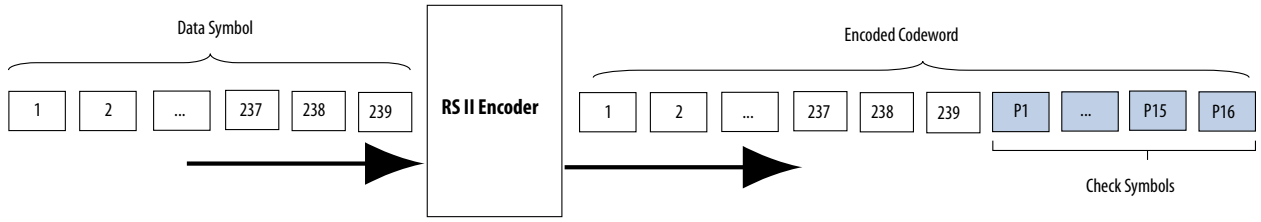
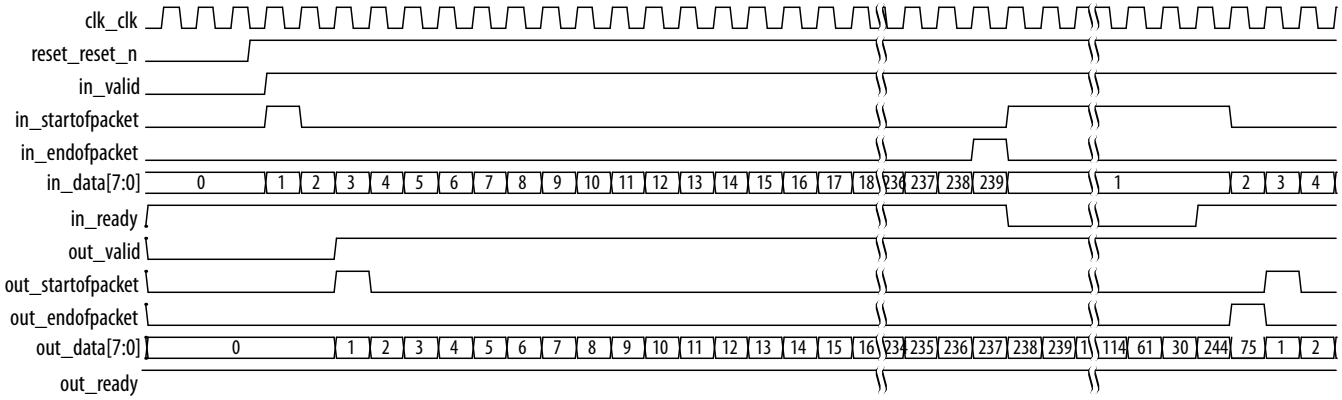


Figure 3-2: Encoder Timing—One Channel

Shows the timing diagram of the RS II encoder with one channel.



The `in_startofpacket` signal starts a codeword; the `in_endofpacket` signals its termination. An asserted `in_valid` signal indicates valid data. The `in_startofpacket` signal is only valid when you assert the `in_valid` signal. For a 1-channel codeword, assert the `in_startofpacket` and `in_endofpacket` signals for one clock cycle. The encoder uses backpressure by deasserting the `in_ready` signal when it receives the `in_endofpacket` signal. During this time, the encoder signals that it cannot accept more incoming symbols and generates the check symbols for the current codeword. The IP core does not verify if the number of symbols (N) exceeds the maximum symbols per codeword. You must ensure that the codeword sent to the core has a valid N. The `reset_reset_n` signal is active low and you can assert this signal asynchronously. However, you have to deassert the `reset_reset_n` signal synchronously with the `clk_clk` signal.

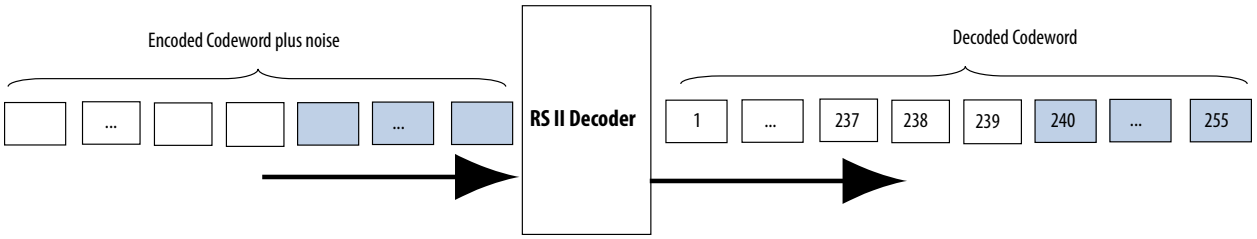
Shortened Codewords

The RS II IP core supports shortened codewords. A shortened codeword contains fewer symbols than the maximum value of *N*, which is $2M - 1$, where *N* is the total number of symbols per codeword and M is the number of bits per symbol. A shortened codeword is mathematically equivalent to a maximum-length code with the extra data symbols at the start of the codeword set to 0. For example, (204,188) is a shortened codeword of (255,239). Both of these codewords use the same number of check symbols, 16. To use shortened codewords with the decoder, use the parameter editor to set the codeword length to the correct value; for the encoder assert endofpacket once it generates enough symbols.

Reed-Solomon II Decoder

When the decoder receives the encoded codeword, it uses the check symbols to detect errors and correct them.

Figure 3-3: Codeword Decoding



The received encoded codeword may differ from the original codeword due to the noise in the channel. The decoder detects errors using several polynomials to locate the error location and the error value. When the decoder obtains the error location and value, the decoder corrects the errors in a codeword, and sends the codeword to the output. As the number of errors increases, the decoder gets to a stage where it can no longer correct but only detect errors, at which point the decoder asserts the `out_error` signal.

Table 3-1: Decoder Detection and Correction

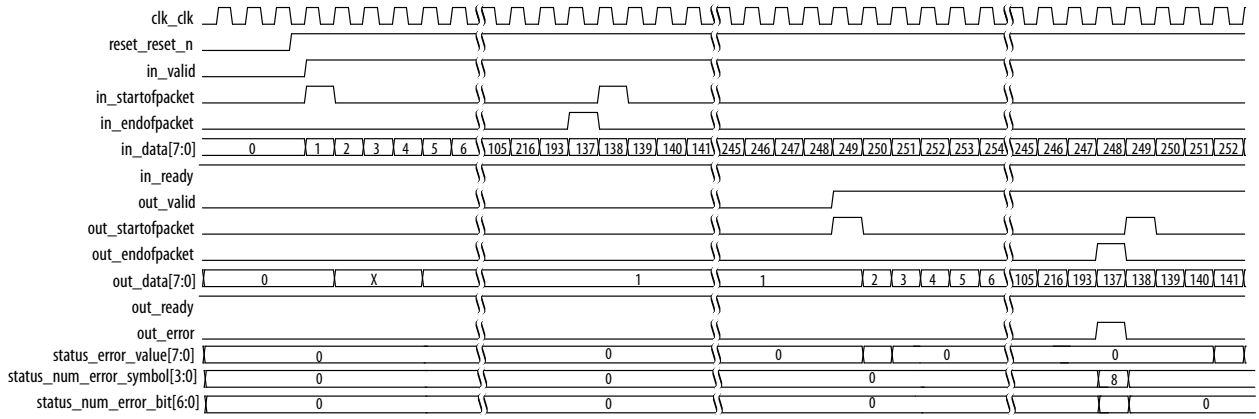
Lists how the decoder corrects and detects errors (e) depending on the number of check symbols (R).

Number of Errors	Description
$e \leq R/2$	Decoder detects and corrects errors.
$R/2 < e$	Decoder asserts error signal and decoding fails. The probability that the decoder might not assert the error signal is less than 1/t.

For small numbers of check symbols, `out_error` is not always reliable. RS codewords have at least d different symbols: $d = R + 1$. A received packet containing e errors can be either the transmitted codeword $t1$ with e errors, or another valid codeword $t2$ with $d–e$ errors (if $t2$ exists). When $e > R/2$, the received packet looks more like $t2$ than $t1$ (because $d - e < e$) so, the decoder outputs t2 and does not assert `out_error`. The probability that $t2$ exists is inferior or equal to the inverse of factorial of $R/2$. It decreases exponentially as R increases, but is nonetheless significant for small numbers of check symbols.

Figure 3-4: Decoder Timing—One Channel

shows the timing diagram of the RS II decoder with one channel.



The codeword starts when you assert the `in_valid` signal and the `in_startofpacket` signal.The decoder accepts the data at `in_data` as valid data. The codeword ends when you assert the `in_endofpacket` signal. For a 1-channel codeword, assert the `in_startofpacket` and `in_endofpacket` signals for one clock cycle. When the decoder deasserts the `in_ready` signal, the decoder cannot process any more data until it asserts the `in_ready` signal again.

At the output, the operation is identical. When the decoder asserts the `out_valid` signal and the `out_startofpacket` signal, the decoder provides valid data on `out_data`. The decoder asserts the `out_startofpacket` signal and the `out_endofpacket` signal to indicate the start and end of a codeword. The decoder automatically detects and corrects errors in a codeword and asserts the `out_error` signal when it encounters a non-correctable codeword. The decoder outputs the full codeword including the check symbols, which you should remove.

Variable Decoding

Under normal circumstances, the decoder allows variable decoding, so you can change the number of symbols per codeword (*N*) using `sink_eop`, but not the number of check symbols while decoding. However, you cannot change the length of the codeword, if you turn on the erasure-supporting option. If you turn on the variable option, you can vary:

- The number of symbols per codeword (using the `numn` signal) and the number of check symbols (using the `numcheck` signal), in real time.
- Both from their minimum allowable values up to their selected values, even with the erasures-supporting option turned on.

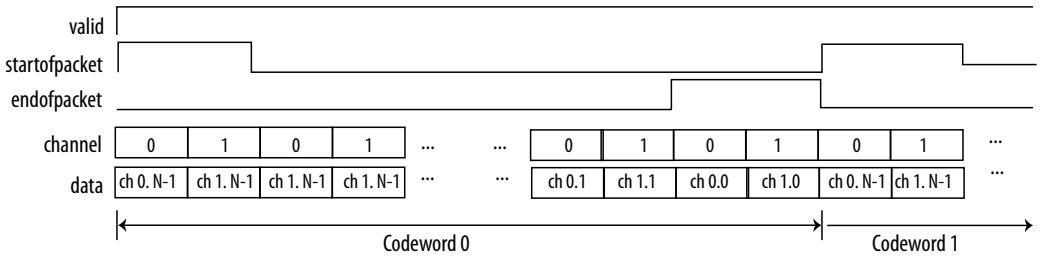
Multiple Input Channels

The RS II IP core processes multiple input channels simultaneously.

The IP core receives codewords in a fixed pattern. Symbols coming in through the channels are interleaved. The IP core samples the first symbol of channel one on the first rising clock edge, then the first symbol of channel two on the second rising clock edge, etc. Both information and check symbols are output in the same sequence.

Figure 3-5: Codeword for *C* Channels and *N* Symbols

The channel signal indicates the channel associated to the current symbol. The channel sequence is fixed. `startofpacket` indicates the first symbol of a codeword per channel. For a *C*-channel codeword, `startofpacket` must be high for *C* consecutive cycles. `endofpacket` indicates the last symbol of a codeword per channel. For a *C*-channel codeword, `endofpacket` must be high for *C* consecutive cycles.



Note: The `startofpacket` and `endofpacket` governs the number of symbols per codeword, *N*. The IP core does not verify if *N* exceeds the maximum symbols per codeword. The IP core also does not verify the channel or data pattern. You must ensure that the codeword sent to the IP core has a valid *N* and a valid pattern.

Figure 3-6: Encoder Timing—Two Channels

For a two-channel codeword, the encoder asserts the `in_startofpacket` and `in_endofpacket` signals for two consecutive cycles.

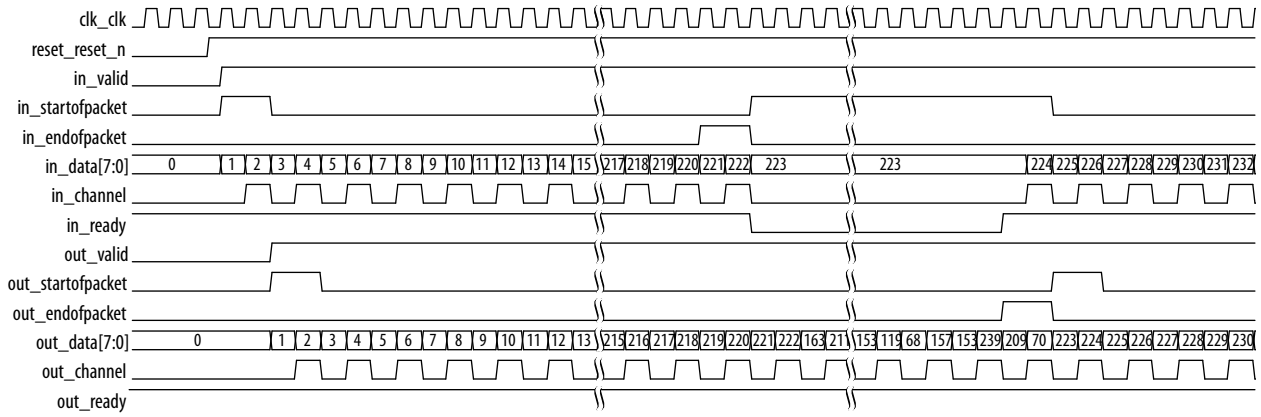
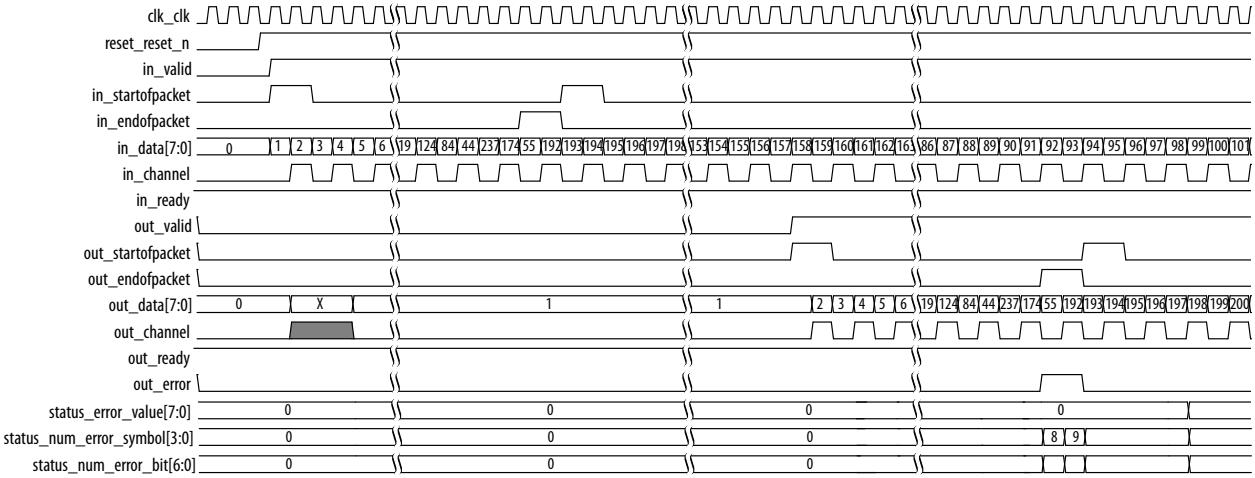


Figure 3-7: Decoder Timing—Two Channels

For a two-channel codeword, the decoder asserts the `in_startofpacket` and `in_endofpacket` signals for two consecutive cycles.



Reed-Solomon II IP Core Parameters

Table 3-2: Parameters

Parameter	Legal Values	Default Value	Description
Reed-Solomon	Encoder or Decoder	Encoder	Specifies an encoder or a decoder.
Parameters			
Number of channels	1 to 16	1	Specifies the number of input channels (*C*) to process. The channel pattern is fixed.
Number of bits per symbol	3 to 12	8	Specifies the number of bits per symbol (*M*).
Number of symbols per codeword	1 to 2*M*–1	255	Specifies the total number of symbols per codeword (*N*). The decoder accept a new symbol every clock cycle if 6.5*R* < N. If *N*> =6.5*R*+1, the decoder shows continuous behavior.
Number of check symbols per codeword	1 to *N*–1	16	Specifies the number of check symbols per codeword (*R*).
Field Polynomial	Any valid polynomial	285	Specifies the primitive polynomial defining the Galois field. The parameter editor allows you to select only legal values. If you cannot find your intended field polynomial, contact Intel Premier Support.

Parameter	Legal Values	Default Value	Description
Type of generator polynomial	Classical or CCSDS-like	Classical	Specifies the representation of the generator polynomial.
First root of the polynomial generator	1 to 2*M*–2	0	Specifies the first root of the generator polynomial.
Root spacing in the polynomial generator	Any primitive elements in the field	1	Specifies spacing between roots in the generator polynomial.
Options			
Erasures-supporting decoder	On or off	Off	Specifies the erasures-supporting decoder. This option substantially increases the logic resources the design uses.
Variable codeword length	On or off	Off	Specifies variable codeword length with `numn` signal. When off, the latency equation is: $L = N + 6.5R+8$
Control signal	sop or eop or in_numn	in_numn signal	Specifies how to pass the codeword length to the IP core
Optimization type	Latency or resource	Latency	Select resource, so the latency always corresponds to that of the largest possible codeword. Select latency, so the latency corresponds to the current codeword.
Variable number of check symbols	On or off	Off	Specifies check symbols with `numcheck` signal.
Decoder Status Signals			
Error symbol value	On or off	On	Specifies whether the decoder indicates the error symbols.
Error symbol count	On or off	On	Specifies whether the decoder indicates the number of error symbols per codeword.
Error bit count	On or off	On	Specifies whether the decoder indicates the number of error bits per codeword.
Error bits count format	Full or split	Full	Specifies full or split count: • With full count the decoder just counts the number of received error bit • With split count the decoder counts the number of received error bits with initial value "1" (then corrects to value "0") and outputs `num_error_bit1` . It also counts the number of received error bits with initial value "0" (then corrects to the value "1") and outputs `num_error_bit0`
CCSDS Options			

Parameter	Legal Values	Default Value	Description
Dual basis representation	Yes or no	No	In standard representation, elements are represented in the basis {α}={1,α,α2,...,αm-1} as: a = u0.α0+u1.α1+u2.α2+...um-1αm-1 where α is root of the field polynomial. More generally an element can be represented in the standard base of any primitive element β=αd. In the dual basis of a standard base {β}, elements are represented in the basis {λ0,λ1,λ2,...,λm-1} a = v0.λ0+v1.λ1+v2.λ2+...vm-1λm-1 such that: Trace(βi.λj=1 if i=j and 0 otherwise
Dual basis of the primitive element	alpha^1 to alpha^254	alpha^1	Dual basis of the primitive element input and output are represented in the trace-orthogonal basis.

Reed-Solomon II IP Core Interfaces and Signals

The RS II Avalon-ST interface supports backpressure, which is a flow control mechanism, where a sink can indicate to a source to stop sending data.

The ready latency on the Avalon-ST input interface is 0; the number of symbols per beat is fixed to 1.

The clock and reset interfaces drive or receive the clock and reset signal to synchronize the Avalon-ST interfaces and provide reset connectivity. The status interface is a conduit interface that consists of three error status signals for a codeword. The decoder obtains the error symbol value, number of error symbols, and number of error bits in a codeword from the status signals.

Avalon-ST Interfaces in DSP IP Cores

Avalon-ST interfaces define a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface.

The input interface is an Avalon-ST sink and the output interface is an Avalon-ST source. The Avalon-ST interface supports packet transfers with packets interleaved across multiple channels.

Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. Such interfaces typically contain data, ready, and valid signals. Avalon-ST interfaces can also support more complex protocols for burst and packet transfers with packets interleaved across multiple channels. The Avalon-ST interface inherently synchronizes multichannel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

Avalon-ST interfaces support backpressure, which is a flow control mechanism where a sink can signal to a source to stop sending data. The sink typically uses backpressure to stop the flow of data when its FIFO buffers are full or when it has congestion on its output.

Related Information

Avalon Interface Specifications

Reed-Solomon II IP Core Signals

Table 3-3: Clock and Reset Signals

Name	Avalon-ST Type	Direction	Description
clk_clk	clk	Input	The main system clock. The whole IP core operates on the rising edge of clk_clk .
reset_ reset_n	reset_n	Input	An active low signal that resets the entire system when asserted. You can assert this signal asynchronously. However, you must deassert it synchronous to the clk_clk signal. When the IP core recovers from reset, ensure that the data it receives is a complete packet.

Table 3-4: Avalon-ST Input and Output Interface Signals

Name	Avalon-ST Type	Direction	Description
in_ready	ready	Output	Data transfer ready signal to indicate that the sink is ready to accept data. The sink interface drives the in_ready signal to control the flow of data across the interface. The sink interface captures the data interface signals on the current clk rising edge.
in_valid	valid	Input	Data valid signal to indicate the validity of the data signals. When you assert the in_valid signal, the Avalon-ST data interface signals are valid. When you deassert the in_valid signal, the Avalon-ST data interface signals are invalid and must be disregarded. You can assert the in_valid signal whenever data is available. However, the sink only captures the data from the source when the IP core asserts the in_ready signal.
in_data[]	data	Input	Data input for each codeword, symbol by symbol. Valid only when you assert the in_valid signal. For Qsys systems, the in_data bus is: [in_numn,in_numcheck,data] If you have no variable check it is: [numn,data] For example, for a maximum codeword length of 255 corresponding to 8 bits: • in_data[7:0] = data • in_data[15:0] = numn
in_channel	channel	Input	Specifies the channel number for data the IP core transfers on the current cycle. The in_channel signal is available only when you configure the IP core to support multiple channels.

Name	Avalon-ST Type	Direction	Description
in_startof-packet	sop	Input	Start of packet (codeword) signal.
in_endofpacket	eop	Input	End of packet (codeword) signal.
in_error	error	Input	Error signal. Specifies if the input data symbol is an error and whether the decoder can consider it as an erasure. Erasures-supporting decoders only.
out_startof-packet	sop	Output	Start of packet (codeword) signal. This signal indicates the codeword boundaries on the in_data[] bus. When the IP core drives this signal high, it indicates that the start of packet is present on the in_data[] bus. The IP core asserts this signal on the first transfer of every codeword.
out_endofpacket	eop	Output	End of packet (codeword) signal. This signal indicates the packet boundaries on the in_data[] bus. When the IP core drives this signal high, it indicates that the end of packet is present on the in_data[] bus. The IP core asserts this signal on the last transfer of every packet.
out_ready	ready	Input	Data transfer ready signal to indicate that the downstream module is ready to accept data. The source provides new data (if available) when you assert the out_ready signal and stops providing new data when you deassert the out_ready signal. If the source is unable to provide new data, it deasserts out_valid for one or more clock cycles until it is prepared to drive valid data interface signals.
out_valid	valid	Output	Data valid signal. The IP core asserts the out_valid signal high, whenever a valid output is on out_data ; the IP core deasserts the signal when there is no valid output on out_data .
out_data	data	Output	Contains decoded output when the IP core asserts the out_valid signal. The corrected symbols are in the same order that they are entered.
out_channel	channel	Output	Specifies the channel whose result is presented at out_data . Available only when you configure the IP core to support multiple channels.
out_error	error	Output	Indicates non-correctable codeword (decoder only). Valid when the IP core asserts out_endofpacket .

Table 3-5: Configuation Signals

Name	Direction	Description
num_check	Output	Sets the variable number of check symbols up to a maximum value set by the parameter *R* (variable option only).
numn	Output	Variable value of *N*. Can be any value from the minimum allowable value of *N* up to the selected value of *N* (variable and erasures-supporting option only).

Table 3-6: Status Interface Signals

Name	Avalon-ST Type	Direction	Description
status_error_value	conduit	Output	Error correction value for every valid data symbol.
status_num_error_symbol	conduit	Output	Number of error symbols in a codeword. This signal is valid when the IP core asserts out_endofpacket . Only available when you turn on **Error symbol count** .
status_num_error_bit	conduit	Output	Number of error bits in a codeword. This signal is valid when the IP core asserts the out_endofpacket . Only available when you turn on **Error bit count** and select **Full for Error bits count format** .
status_num_error_bit0	conduit	Output	Number of bit errors for the correction from bit 1 to bit 0. The latest is the correct bit. This signal is valid when the IP core asserts the out_endofpacket . Only available when you turn on **Error bit count** and select **Split for Error bits count format** .
status_num_error_bit0	conduit	Output	Number of bit errors for the correction from bit 0 to bit 1. The latest is the correct bit. This signal is valid when the IP core asserts the out_endofpacket . Only available when you turn on **Error bit count** and select **Split for Error bits count format** .

Document Revision History 4

2016.05.02

UG-01090 Subscribe Send Feedback

Reed-Solomon II IP Core User Guide revision history.

Date	Version	Changes
2016.05.02	16.0	Added two new CCSDS parameters.
2015.11.17	15.1	Removed latency information.
2015.10.01	15.1	• Added latency information • Added **Control signal** and **Optimization type** parameters for variable codeword lengths. • Changed decoder error correction and detection table.
2015.05.01	15.0	• Added `in_data` information
2014.12.15	14.1	• Added final support for MAX10 and Arria 10 devices • Removed Appendix A
August 2014	14.0 Arria 10 Edition	• Added support for Arria 10 devices. • Added Arria 10 generated files description. • Removed table with generated file descriptions.
June 2014	14.0	• Removed support for Cyclone III and Stratix III devices • Added support for MAX 10 FPGAs • Added instructions for using IP Catalog

Date	Version	Changes

November 2013	13.1	• Updated performance data
		• Added erasures-supporting decoder
		• Added status signals to parameters description
		• Added in_error signal description (for erasures-supporting decoder).
		• Added new wizard parameters:
		  • Number of bits per symbol
		  • Field polynomial
		  • Type of generator polynomial
		  • First root of the generator polynomial
		  • Root spacing in the generator polynomial
		  • Erasures-supporting decoder
		  • Decoder status signals
		•
		•
		• Removed support for the following devices:
		  • Arria GX
		  • Cyclone II
		  • HardCopy II,
		  • HardCopy III
		  • HardCopy IV
		  • Stratix
		  • Stratix II
		  • Stratix GX
		  • Stratix II GX
		• Added final support for the following devices
		  • Arria V
		  • Stratix V

May 2013	13.0	Added support for Cyclone IV E devices.
November 2012	12.1	Added support for Arria V GZ devices.

Date	Version	Changes
May 2011	2.0	• Updated About This MegaCore Function with new device family support. • Updated Functional Description with new status ports and timing diagrams.
December 2010	1.0	Initial release.