# Reed-Solomon Compiler MegaCore Function User Guide

November 1999

Printed on Recycled Paper.

I.S. EN ISO 9001

# About this User Guide

User Guide

This user guide provides comprehensive information about the Altera® Reed-Solomon Compiler MegaCore™ function.

For the most-up-to-date information about Altera products, go to the Altera world-wide web site at **http://www.altera.com**.

## How to Contact Altera

For additional information about Altera products, consult the sources shown in Table 1.

*Table 1. How to Contact Altera*

| Information Type | Access | USA & Canada | All Other Locations |
|---|---|---|---|
| Altera Literature Services | Telephone hotline | (888) 3-ALTERA *(1)* | (888) 3-ALTERA *(1)* |
| | Electronic mail | **lit_req@altera.com** *(1)* | **lit_req@altera.com** *(1)* |
| Non-technical customer service | Telephone hotline | (800) SOS-EPLD | (408) 544-7000 |
| | Fax | (408) 544-7606 | (408) 544-7606 |
| Technical support | Telephone hotline (6:00 a.m. to 6:00 p.m. Pacific Time) | (800) 800-EPLD | (408) 544-7000 *(1)* |
| | Fax | (408) 544-6401 | (408) 544-6401 *(1)* |
| | Electronic mail | **support@altera.com** | **support@altera.com** |
| | FTP site | **ftp.altera.com** | **ftp.altera.com** |
| General product information | Telephone | (408) 544-7104 | (408) 544-7104 *(1)* |
| | World-wide web site | **http://www.altera.com** | **http://www.altera.com** |

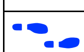
***Note:***

(1) You can also contact your local Altera sales office or sales representative.

# Typographic Conventions

The ***Reed-Solomon Compiler MegaCore Function User Guide*** uses the typographic conventions shown in Table 2.

*Table 2. Conventions*

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Command names, dialog box titles, checkbox options and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box. |
| **bold type** | External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\maxplus2** directory, **d:** drive, **chiptrip.gdf** file. |
| ***Bold italic type*** | Book titles are shown in bold italic type with initial capital letters. Example: ***1999 Data Book***. |
| *Italic Type with Initial Capital Letters* | Document titles are shown in italic type with initial capital letters. Example: *AN 75 (High-Speed Board Design).* |
| *Italic type* | Internal timing parameters and variables are shown in italic type. Examples: $t_{PIA}$, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <*file name*>, <*project name*>**.pof** file. |
| Initial Capital Letters | Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu. |
| "Subheading Title" | References to sections within a document and titles of MAX+PLUS II Help topics are shown in quotation marks. Example: "Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable." |
| `Courier type` | Reserved signal and port names are shown in uppercase Courier type. Examples: `DATA1`, `TDI`, `INPUT`.<br><br>User-defined signal and port names are shown in lowercase Courier type. Examples: `my_data`, `ram_input`.<br><br>Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\max2work\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in `Courier`. |
| 1., 2., 3., and a., b., c.,... | Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ | Bullets are used in a list of items when the sequence of the items is not important. |
| ↵ | The checkmark indicates a procedure that consists of one step only. |
| ↵ | The hand points to information that requires special attention. |
| ↵ | The angled arrow indicates you should press the Enter key. |
|  | The feet direct you to more information on a particular topic. |

# Contents

*Notes:*

# Introduction

## Contents

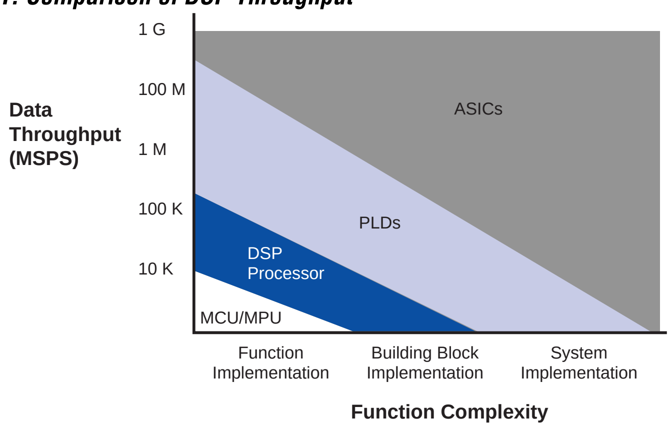
*Notes:*

## Altera MegaCore Functions

As programmable logic device (PLD) densities grow to over 1 million gates, design flows must be as efficient and productive as possible. Altera® provides ready-made, pre-tested, and optimized megafunctions that let you rapidly implement the functions you need, instead of building them from the ground up. Altera® MegaCore™ functions, which are reusable blocks of pre-designed intellectual property, improve your productivity by allowing you to concentrate on adding proprietary value to your design. When you use MegaCore functions, you can focus on your high-level design and spend more time and energy on improving and differentiating your product.

Traditionally, designers have been forced to make a tradeoff between the flexibility of digital signal processors and the performance of application-specific integrated circuit (ASIC) and application-specific standard product (ASSP) digital signal processing (DSP) solutions. The Altera FLEX® and APEX™ DSP solution eliminates the need for this tradeoff by providing exceptional performance combined with the flexibility of PLDs. See Figure 1.

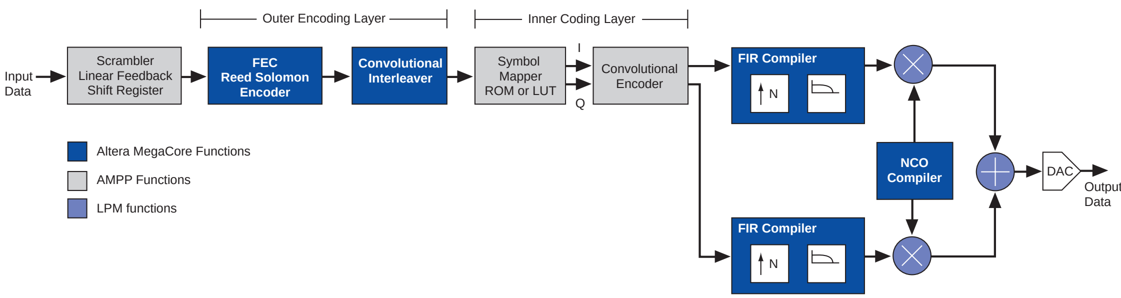
*Figure 1. Comparison of DSP Throughput*



Altera DSP solutions include MegaCore functions developed and supported by Altera and Altera Megafunction Partners Program (AMPP℠) functions. In addition, many commonly used functions, such as adders and multipliers, are available from the industry-standard library of parameterized modules (LPM). Altera devices easily implement DSP applications, while leaving ample room for your custom logic. The devices are supported by Altera's MAX+PLUS® II and Quartus™ development system, which allow you to perform a complete design cycle including design entry, synthesis, place-and-route, simulation, timing analysis, and device programming. Altera devices, software, and DSP MegaCore functions provide you with a complete design solution.

Figure 2 shows a complete system and highlights the functions that are available from Altera.

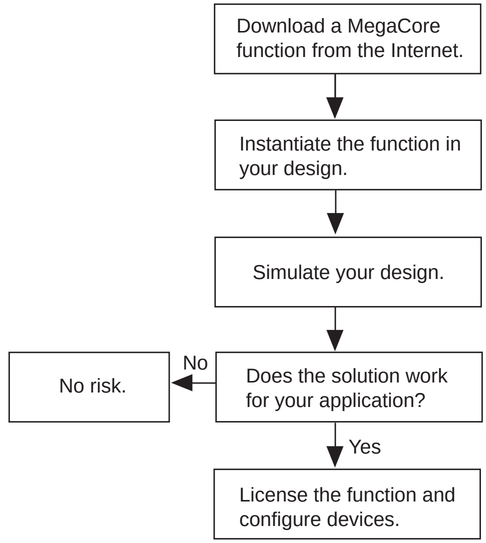
*Figure 2. Typical Modulator*




## OpenCore Feature

Altera's exclusive OpenCore™ feature allows you to evaluate MegaCore functions before deciding to license them. You can instantiate a MegaCore function in your design, compile and simulate the design, and then verify the MegaCore function's size and performance. This evaluation provides first-hand functional, timing, and other technical data that allows you to make an informed decision on whether to license the MegaCore function. Once you license a MegaCore function, you can use the MAX+PLUS II or Quartus software to generate programming files as well as EDIF, VHDL, or Verilog HDL output netlist files for simulation in third-party EDA tools. Figure 3 on page 5 shows a typical design flow using MegaCore functions and the OpenCore feature.

*Figure 3. OpenCore Design Flow*




## Altera Devices

The Reed-Solomon (RS) Compiler MegaCore function has been optimized and targeted for Altera FLEX 10K devices and APEX 20K devices; the RS encoder can also be implemented in FLEX 6000 devices.

The FLEX 10K embedded PLD family delivers the flexibility of traditional programmable logic with the efficiency and density of gate arrays with embedded memory. FLEX 10K devices feature embedded array blocks (EABs), which are 2 Kbits of RAM that can be configured as 256 × 8, 512 × 4, 1,024 × 2, or 2,048 × 1 blocks. The new 2.5-V FLEX 10KE devices support efficient implementation of dual-port RAM, and further enhance the performance of the FLEX 10K family.

APEX 20K devices offer complete system-level integration on a single device. The APEX MultiCore™ architecture delivers the ultimate in design flexibility and efficiency for high-performance System-on-a-Programmable Chip™ applications. With densities ranging from 100,000 to 1,000,000 gates, the APEX 20K architecture integrates look-up table (LUT) logic, product-term logic, and memory into a single architecture, eliminating the need for multiple devices, saving board space, and simplifying the implementation of complex designs.

In the APEX MultiCore architecture, embedded system blocks (ESBs) and logic array blocks (LABs) are combined into MegaLAB™ structures. Each APEX 20K ESB can be configured as product-term logic, enabling APEX 20K devices to achieve unmatched integration efficiency, as LUT logic or as memory. The ESB can be configured as dual-port RAM, with a wide range of RAM widths and depths, or ROM in APEX 20K devices, and as content-addressable memory (CAM), a memory technology that accelerates applications requiring fast searches, in APEX 20KE devices.

For more information on FLEX 10K and APEX 20K devices, refer to the following documents:

- *FLEX 10K Embedded Programmable Logic Family Data Sheet*
- *FLEX 10KE Embedded Programmable Logic Family Data Sheet*
- *APEX 20K Programmable Logic Device Family Data Sheet*

## Software Tools

Altera offers the fastest, most powerful, and most flexible programmable logic development software in the industry. The MAX+PLUS II and Quartus software offer a completely integrated development flow and an intuitive, graphical user interface, making them easy to learn and use.

The MAX+PLUS II software offers a seamless development flow, allowing you to enter, compile, and simulate your design and program devices using a single, integrated tool, regardless of the Altera device you choose. Altera’s fourth-generation Quartus software offers designers the ideal platform for processing multi-million gate designs, with state-of-the-art features that shorten design cycles, streamline development flows, and reduce verification times. The Quartus software shortens design cycles with the revolutionary nSTEP™ Compiler, which permits incremental recompilation, and multiple processor support that can operate locally or across networks and even platforms.

## MegaWizard Plug-Ins

MegaWizard™ Plug-Ins are parameterization tools that help you integrate megafunctions into your designs without requiring the use of third-party tools. You can use this feature in the MAX+PLUS II and Quartus software or as a stand-alone tool with third-party EDA design interfaces. MegaWizard Plug-Ins provide maximum flexibility, allowing you to customize megafunctions without changing your design's source code. You can integrate a parameterized megafunction in any hardware description language (HDL) or netlist file using any EDA tool.

The MAX+PLUS II MegaWizard Plug-In Manager allows you to launch the megafunction’s wizard so you can set the megafunction’s parameters to fit your design. A custom megafunction is then generated, which you can instantiate in your design file.

## EDA Interfaces

Altera has worked closely with major EDA vendors to develop the best interface any PLD software has to offer. As a standard feature, the MAX+PLUS II software interfaces with all major EDA design tools, including tools for ASIC designers. Once a design is captured and simulated using the tool of your choice, you can transfer your EDIF file directly into the MAX+PLUS II software. After synthesis and fitting, you can transfer your file back into your tool of choice for simulation. The MAX+PLUS II system outputs the full-timing VHDL, Verilog HDL, Standard Delay Format (SDF), and EDIF netlists that can be used for post-route device- and system-level simulation. Altera opened the Quartus interface to various EDA partners to enable them to provide unmatched levels of integration. NativeLink™ integration provides a truly seamless interface to major EDA software tools to support existing design flows, eliminating the need to learn new design tools.

To simplify the design flow between the MAX+PLUS II software and other EDA tools, Altera has developed the MAX+PLUS II Altera Commitment to Cooperative Engineering Solutions (ACCESS℠) Key Guidelines. These guidelines provide complete instructions on how to create, compile, and simulate your design with tools from leading EDA vendors. The MAX+PLUS II ACCESS Key Guidelines are part of Altera's ongoing efforts to give you state-of-the-art tools that fit into your design flow, and to enhance your productivity for even the highest-density devices. The MAX+PLUS II ACCESS Key Guidelines are available on the Altera web site (**http://www.altera.com**) and the MAX+PLUS II CD-ROM.

*Notes:*

# Specifications

## Contents

User Guide



2 Specifications

*Notes:*

User Guide



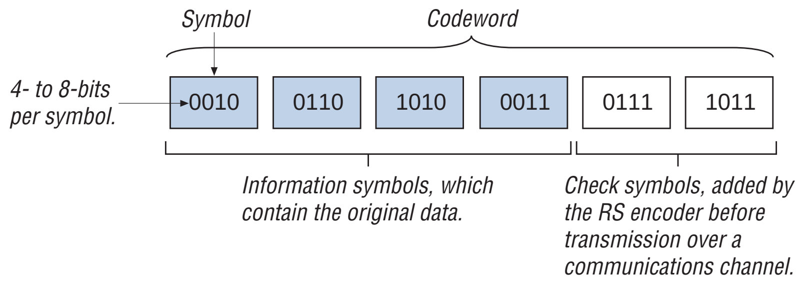
## Features

- High-performance encoder/decoder for error detection and correction
- Easy-to-use MegaWizard™ Plug-In
  - Generates parameterized encoders or decoders
  - Generates example test vectors
- Efficient VHDL simulation model
- Fully parameterized Reed-Solomon function including:
  - Number of bits per symbol
  - Number of symbols per codeword
  - Number of check symbols per codeword
  - Field polynomial
  - First root of generator polynomial
- Decoder implementation options:
  - Standard or erasure-supporting
  - Discrete or streaming (pipelined)
  - Fast or small
- Optimized for the FLEX® 10K and APEX™ 20K device architectures

## General Description

Reed-Solomon (RS) codes are widely used for error detection and correction. To use RS codes, a data stream is first broken into a series of codewords. Each codeword consists of several information symbols followed by several check symbols (also known as parity symbols). Symbols can contain an arbitrary number of bits. The Altera® RS Compiler MegaCore™ function supports four to eight bits per symbol. In an error correction system, the encoder adds parity symbols to the data stream prior to its transmission over a communications channel. Once the data is received, the decoder checks for and corrects any errors (see Figure 1).

**Figure 1. RS Codeword Example**



RS codes are described as ($N$,$K$) where $N$ is the total number of symbols per codeword, and $K$ is the number of information symbols. Errors are defined on a symbol basis (i.e., any number of bit errors within a symbol is considered as only one error). A RS decoder can correct one symbol error for every two check symbols in a codeword. However, if a codeword contains many errors, the decoder can only detect up to one symbol error for each check symbol.

RS codes are based on finite-field (i.e., Galois field) arithmetic. All arithmetic operations (i.e., addition, subtraction, multiplication, and division) on field elements give results that are an element of the field. The size of the Galois field is determined by the number of bits per symbol; specifically, the field has $2^m$ elements, where $m$ is the number of bits per symbol. A specific Galois field is defined by a polynomial, which is user-defined for the RS Compiler MegaCore function. The MegaWizard Plug-In only lets the user select valid field polynomials.

The maximum number of symbols in a codeword is limited by the size of the finite field to $2^m - 1$. For example, a code based on 8-bit symbols can have up to 255 symbols per codeword.

# Functional Description

Table 1 shows the RS compiler MegaCore function's parameters. The parameters are specified using the MegaWizard Plug-In, and define the specific RS code for the encoder or decoder.

*Table 1. RS Compiler MegaCore Function Parameters*

| Parameter | Range | Description |
|---|---|---|
| N | $R$ + 1 to $2^m - 1$ | Specifies the total number of symbols per codeword. |
| R | 4 to $N - 1$ maximum is 50 *(1)* | Specifies the number of check symbols per codeword. |
| m | 4 to 8 | Specifies the number of bits per symbol. |
| field | Any valid polynomial *(2)* | Specifies the polynomial defining the Galois field. |
| genstart | 0 to $(2^m - 1 - R)$ | Indicates the first root of the generator polynomial. |
| type_of_decoder | Standard or erasure-supporting | Specifies a standard or erasure-supporting decoder. Erasure-supporting substantially increases the logic resources used. |
| architecture | Discrete or streaming *(3)* | Specifies a discrete or streaming architecture. The discrete architecture processes one entire codeword before starting on the next. The streaming architecture creates a pipelined decoder with a depth of three codewords. |
| speed | Half or full *(3)* | Controls the amount of logic used to determine the location and number of errors. The full-speed setting creates a large, high-performance function. The half-speed setting creates a small, lower-performance function. |

***Notes:***
(1) An even and odd number of check symbols are allowed.
(2) The MegaWizard Plug-in only allows the selection of legal values.
(3) These parameters apply to the decoder only.

2 Specifications

## MegaWizard Plug-In

You can launch the MegaWizard Plug-In Manager from within the MAX+PLUS II or Quartus software, or you can run it from the command line. The wizard generates a custom megafunction that you can instantiate in your design. Table 2 highlights the wizard's main features.

*Table 2. RS Compiler MegaCore Function Wizard Options*

| Option | Description |
|---|---|
| Type of function | Specifies an RS encoder or decoder. |
| Implementation parameters (decoder only) | Specifies a standard or erasure-supporting decoder, a discrete or streaming decoder, and whether to use a half- or full-speed implementation. |
| Parameters | Specifies the number of bits per symbol, the number of symbols per codeword, the number of check symbols per codeword, the field polynomial, and the first root of the generator polynomial. |

When you finish going through the wizard, it generates a VHDL component declaration file (**.cmp**) and the following:

- Text Design File (**.tdf**), VHDL Design File (**.vhd**), or Verilog Design File (**.v**) depending on your selection, used to instantiate an instance of the RS encoder/decoder in your design
- MAX+PLUS II Vector File (**.vec**) used for simulation within the MAX+PLUS II environment
- Symbol File (**.sym**) used to instantiate the RS encoder/decoder into a schematic design
- Hexadecimal (Intel-Format) File (**.hex**) with the contents of the ROM blocks used in the function

The RS compiler MegaCore function has an interactive wizard-driven interface that allows you to create custom encoders or decoders easily. The wizard generates an instance of the function with your choice of parameters in AHDL, VHDL, or Verilog HDL, which you can integrate into your system design.

## Encoder Signals

The RS encoder is fully synchronous and operates on the rising edge of `sysclk`. The function's internal registers are cleared asynchronously by setting `reset` high. To begin the encoding process, a high pulse of at least one clock cycle must be applied on `start`. After `start` is de-asserted, when `enable` is asserted, data is entered at `rsin[]`and encoded at the rising edge of `sysclk`. The remaining information symbols appear in subsequent cycles as long as `enable` is asserted. The function outputs the first codeword symbol after the same clock edge that clocks in the first information symbol. The `check` symbols are output during the *R* clock cycles after the information symbols. The `start` signal must be asserted before each codeword is encoded.

The `enable` signal allows the encoder to perform either continuous or discrete (discontinuous) operation. If `enable` is de-asserted, the encoder stops processing data; if `enable` is asserted, the encoder restarts processing data.
Table 3 describes the signals used by the RS encoder.

***Table 3. Encoder Signals***

| Name | Type | Description |
|---|---|---|
| `sysclk` | Input | System clock. |
| `reset` | Input | Resets the encoder asynchronously. |
| `start` | Input | Sets the encoder to process a new codeword. |
| `enable` | Input | Enables data into the encoder and on the output data bus. |
| `rsin[]` | Input | *m*-bit wide input data bus. |
| `rsout[]` | Output | *m*-bit wide output data bus. |

Figures 2 and 3 show the waveforms for discrete and continuous encoding.

2 Specifications

*Figure 2. Discrete Encoding*

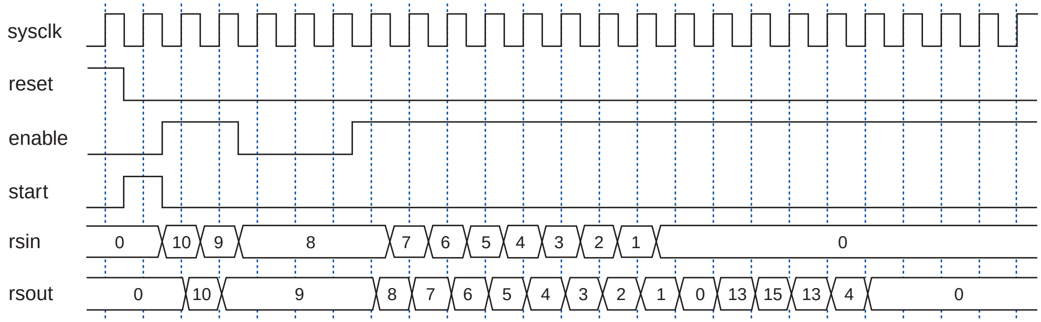
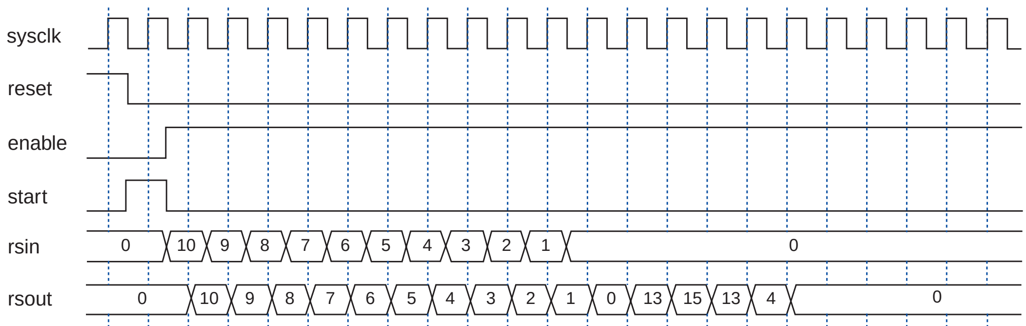



*Figure 3. Continuous Encoding*




## Decoder Signals

A discrete decoder processes one codeword at a time and must be reset between each codeword. The decoder receives the codeword's first symbol on the rising edge of `sysclk` after `reset` is de-asserted, depending on the value of `dsin`. The `rdyin` signal is asserted during reset and remains asserted until the decoder receives the last codeword symbol.

The `outvalid` signal is asserted when the decoder outputs valid data on `rsout[]`. When `dsout` is asserted, the discrete decoder outputs one symbol on each rising clock edge until all data is transferred. The `outvalid` signal is de-asserted when all data is transferred. If the discrete decoder detects more errors than it can correct, it asserts `decfail` and presents the uncorrected data on `rsout[]`.

The streaming decoder interface is similar to the discrete decoder, except the streaming decoder is only reset once. If `reset` is asserted while codewords are being decoded, the codewords are lost. The `rdyin` signal indicates when the streaming decoder can accept a new codeword; once `rdyin` goes high, the decoder expects a new symbol at each rising clock edge, controlled by `dsin`. At startup, `dsin` must remain de-asserted for at least one clock cycle after `reset` is de-asserted.

The streaming decoder has a pipeline depth of three codewords. Therefore, the decoder must receive three codewords before it places the first corrected codeword at `rsout[]`. The `outvalid` signal is not asserted until the first valid codeword is available at `rsout[]`. When `outvalid` is asserted, corrected symbols are presented at `rsout[]` on every clock cycle, controlled by `dsout`. If the streaming decoder detects more errors than it can correct, it asserts `decfail` and presents the uncorrected codeword on `rsout[]`. The `eras_ind` input indicates an erasure (when the erasure-supporting decoder is selected).

Table 4 describes the signals used by the RS decoder.

**Table 4. RS Decoder Signals**

| Name | Type | Description |
|---|---|---|
| `sysclk` | Input | System clock. |
| `reset` | Input | Decoder reset. The discrete decoder must be reset between each codeword. |
| `rsin[]` | Input | *m*-bit wide input data bus. |
| `eras_ind` | Input | When asserted the symbol in `rsin[]` is erased (erasure-supporting decoder only). |
| `dsin` | Input | Data input control. Data input only takes place in clock cycles when `dsin` is asserted. In the streaming decoder, `dsin` must remain de-asserted for at least one clock cycle after `reset` is de-asserted. |
| `dsout` | Input | Data output control. Data can be output when `dsout` is asserted. If `dsout` is not asserted, data is not output. There is a three-cycle latency between a change in `dsout` and the time when the output starts and stops. |
| `bypass` | Input | When asserted, the decoder outputs the uncorrected input data instead of the corrected data. All other operations are unaffected and the decoder's latency remains the same. |
| `rsout[]` | Output | *m*-bit wide output data bus. |
| `rdyin` | Output | Indicates the decoder is ready to accept a new codeword. |
| `outvalid` | Output | Asserted when the decoder outputs a codeword. |
| `decfail` | Output | Asserted at the beginning of a data output when the decoder has detected errors and cannot correct them. |
| `numerr[]` | Output | Number of symbol errors found. |

Figures 4 and 5 show decoder timing diagrams. Note that at least one clock cycle separation is required between `reset` deassertion and `dsin` assertion.

*Figure 4. Using dsin to Control Input of Data to Decoder*

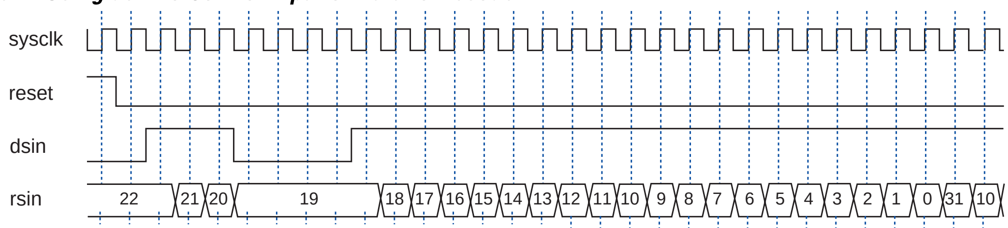
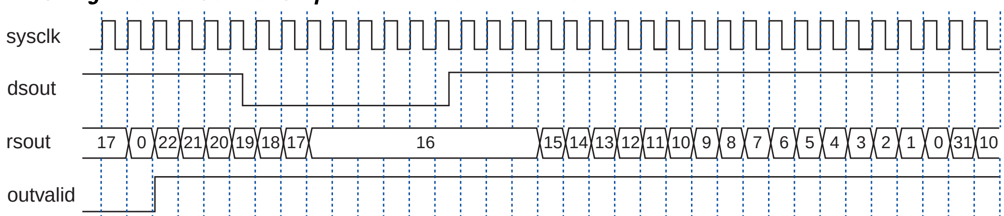



*Figure 5. Using dsout to Control Output of Data from the Decoder*




## Performance

You can calculate the decoder performance using the formulae in Table 5, which define the maximum number of clock cycles required to process each codeword.

*Table 5. Performance Calculations - Standard Decoder*

| Speed | Discrete (1) | Streaming |
|---|---|---|
| half | $3N + 11R + 4$ | max $[(N + R),(10R + 4)]$ |
| full | $3N + 8R + 5$ | max $[(N + R),(7R + 5)]$ |

*Note:*
(1) First symbol in to last symbol out.

*Table 6. Performance Calculations - Erasure-supporting Decoder*

| Speed | Discrete (1) | Streaming |
|---|---|---|
| half | $3N + 11R + 6$ | max $[(N + R),(10R + 6)]$ |
| full | $3N + 9R + 6$ | max $[(N + R),(8R + 6)]$ |

*Note:*
(1) First symbol in to last symbol out.

Tables 7 and 8 show the function's performance and area utilization for FLEX 10KE devices. The performance is derived from the formulae in Table 5 and maximum frequency at which the design can operate.

*Table 7. Performance & Area Utilization - Standard*

| Architecture | Speed | m | N | R | Utilization *(1)* | | Performance | |
|---|---|---|---|---|---|---|---|---|
| | | | | | LEs | EABs | Megabits/ Second | $f_{MAX}$ (MegaHertz) |
| streaming | full | 4 | 15 | 4 | 516 | 6 | 150 | 82.6 |
| streaming | full | 5 | 31 | 6 | 772 | 6 | 259 | 78.7 |
| streaming | half | 6 | 55 | 6 | 906 | 6 | 381 | 74 |
| streaming | half | 8 | 204 | 16 | 2431 | 6 | 455 | 61 |
| discrete | half | 8 | 204 | 16 | 2194 | 3 | 115 | 55 |
| streaming | half | 8 | 207 | 20 | 3439 | 6 | 388 | 53.1 |
| streaming | half | 8 | 225 | 45 | 6603 | 6 | 178 | 45 |

*Note:*
(1) LEs = logic elements, EABs = embedded array blocks.

*Table 8. Performance & Area Utilization - Erasure-supporting*

| Architecture | Speed | m | N | R | Utilization *(1)* | | Performance | |
|---|---|---|---|---|---|---|---|---|
| | | | | | LEs | EABs | Megabits/ Second | $F_{max}$ (MegaHertz) |
| streaming | half | 6 | 55 | 6 | 1562 | 6 | 381 | 76.3 |
| streaming | half | 8 | 204 | 16 | 4853 | 6 | 441 | 59.5 |
| discrete | half | 8 | 204 | 16 | 4190 | 3 | 118 | 57.8 |
| streaming | half | 8 | 207 | 20 | 6312 | 6 | 383 | 52.6 |
| streaming | half | 8 | 225 | 45 *(2)* | 13101 | 6 | 149 | 37.9 |

*Note:*
(1) LEs = logic elements, EABs = embedded array blocks.
(2) Implemented in an APEX 20K device.

Overall resource requirements vary widely depending on the parameter values used. The number of logic cells required to implement the function is linearly dependent on both the field size and the number of check symbols. The number of EABs used depends on whether the decoder is discrete (3 EABs) or streaming (6 EABs).

*Notes:*

# Getting Started

## Contents

User Guide

*Notes:*

Altera® digital signal processing (DSP) MegaCore™ functions provide solutions for integrating Reed-Solomon (RS) functions into your digital communications system. The functions are optimized for Altera FLEX® 10K and APEX™ 20K devices, greatly enhancing your productivity by allowing you to focus efforts on the custom logic in the system. This section describes how to obtain the RS compiler MegaCore function, explains how to install it on your PC, and walks you through the process of implementing the function in a design. You can test-drive MegaCore functions using Altera's OpenCore™ feature to simulate the functions within your custom logic. When you are ready to generate programming or configuration files, you should license the function by contacting your local Altera sales representative.

This design walkthrough involves the following steps:

1. Download and install the RS compiler MegaCore function.

2. Generate a custom RS function.

3. Implement your system using AHDL, VHDL, Verilog HDL, or schematic entry.

4. Compile your design and perform place-and-route.

5. Use the RS compiler MegaCore function wizard-generated VHDL or Verilog HDL simulation models to confirm the operation of your system.

6. License the RS compiler MegaCore function and configure the devices.

The instructions assume that:

- You are using a PC.
- You are familiar with the MAX+PLUS II software.
- MAX+PLUS II version 9.2 or higher is installed in the default location (**c:\maxplus2**).
- You are using the OpenCore feature to test-drive the RS compiler MegaCore function or you have licensed the function.

# Download & Install the RS Compiler MegaCore Function

Before you can start using Altera MegaCore functions, you must obtain the MegaCore files and install them on your PC. The following instructions describe this process.

## Obtaining MegaCore Functions

If you have Internet access, you can download the RS compiler MegaCore function from the Altera web site at **http://www.altera.com**. Follow the instructions below to obtain the RS compiler MegaCore function via the Internet. If you do not have Internet access, you can obtain the RS compiler MegaCore function from your local Altera representative.

1. Run your web browser (e.g., Netscape Navigator or Microsoft Internet Explorer).
2. Open the URL **http://www.altera.com**.
3. Click the Tools icon on the home page toolbar.
4. Click the MegaCore Functions link.
5. Click the link for the RS compiler MegaCore function.
6. Follow the on-line instructions to download the function and save it to your hard disk.

## Installing the MegaCore Files

For Windows 95/98 and Windows NT 4.0, follow the instructions below:

1. Click **Run** (Start menu).
2. Type <*path name*>\<*filename*>`.exe`, where <*path name*> is the location of the downloaded MegaCore function and <*filename*> is the filename of the function. Click **OK**.
3. The **MegaCore Installer** dialog box appears. Follow the on-line instructions to finish installation.
4. After you have finished installing the MegaCore files, you must specify the directory in which you installed them (e.g., <*path name*>**\reed_solomon\lib**) as a user library in the MAX+PLUS II software. Search for “User Libraries” in MAX+PLUS II Help for instructions on how to add these libraries.

# Generate a Custom RS Function

This section describes the design flow using the Altera RS compiler MegaCore function and the MAX+PLUS II development system. Altera provides a MegaWizard Plug-In with the RS compiler MegaCore function. The MegaWizard™ Plug-In Manager, which you can use within the MAX+PLUS II software or as a stand-alone application, lets you create or modify design files to meet the needs of your application. You can then instantiate the custom megafunction in your design file.

You can use Altera's OpenCore feature to compile and simulate the MegaCore functions, allowing you to evaluate the functions before deciding to license them. However, you must obtain a license from Altera before you can generate programming files or EDIF, VHDL, or Verilog HDL netlist files for simulation in third-party EDA tools.

☞ You should use the MAX+PLUS II software version 9.2 or higher when designing with the RS compiler MegaCore function. The function relies on library files that only exist in these versions of software.

To create a custom version of the RS compiler MegaCore function, follow these steps:

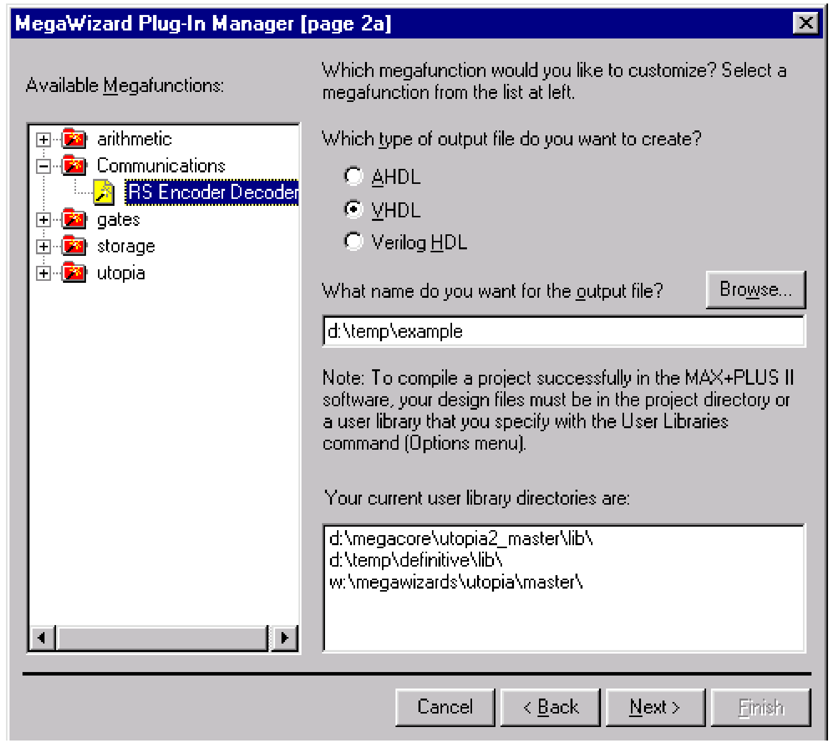
1. Start the MegaWizard Plug-In Manager by choosing the **MegaWizard Plug-In Manager** command (File menu) in any MAX+PLUS II application, or by starting the stand-alone version of the MegaWizard Plug-In Manager by typing the command `megawiz` ↵ at a command line. The **MegaWizard Plug-In Manager** dialog box is displayed.

☞ Refer to MAX+PLUS II Help for detailed instructions on how to use the MegaWizard Plug-In Manager.

2. Specify that you want to create a new custom megafunction and click **Next**.

3. Select **RS Encoder Decoder** in the **Communications** folder (see Figure 1).

*Figure 1. Selecting the Megafunction*




4. Choose the language for the output file(s)—either AHDL, VHDL, or Verilog HDL—and specify a filename. Click **Next**.

5. Select whether you wish to create a RS encoder or decoder. If you are creating a decoder, you must also specify the implementation parameters: whether the decoder is discrete or streaming, and whether to use a faster or smaller implementation (see Figures 2 and 3). Click **Next**.

*Figure 2. Selecting the Decoder*

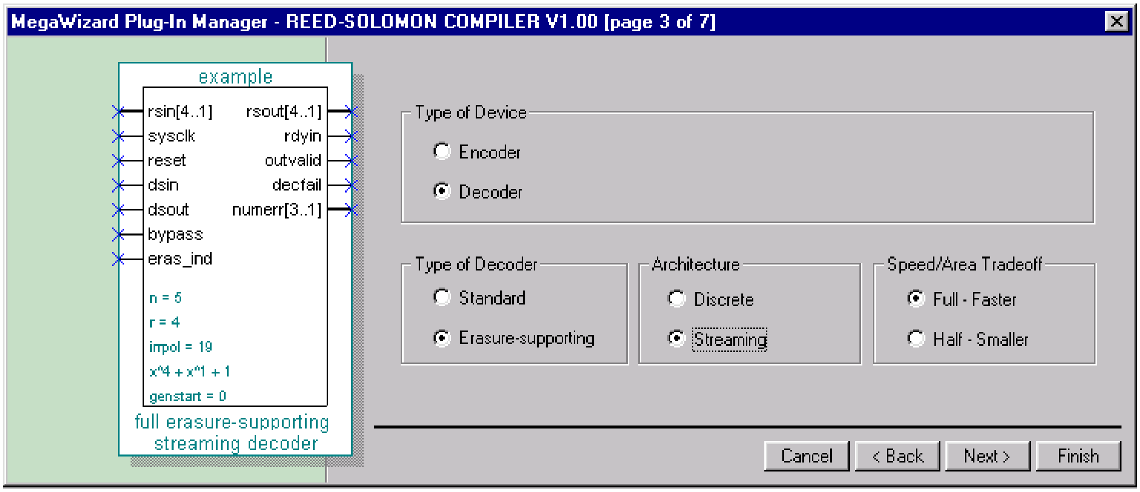
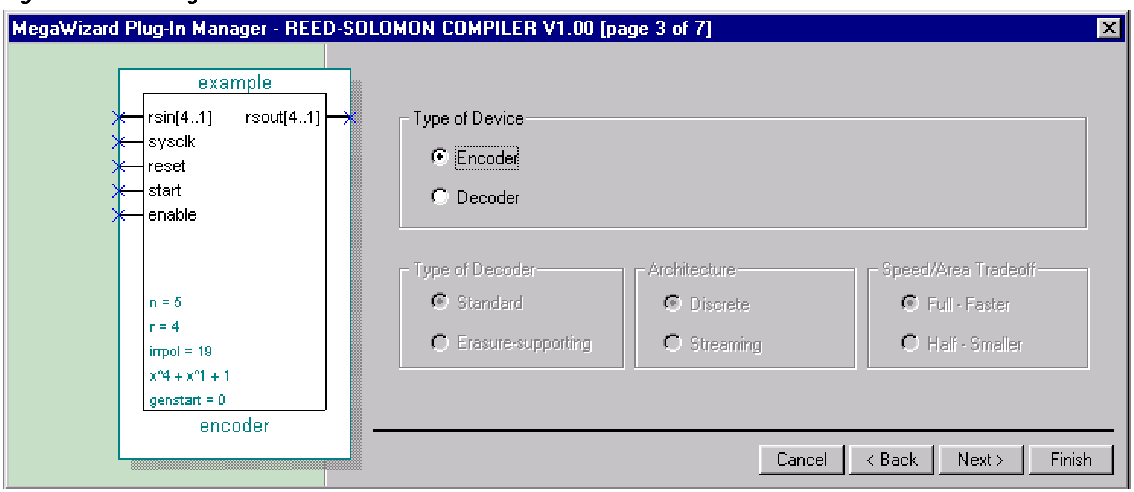
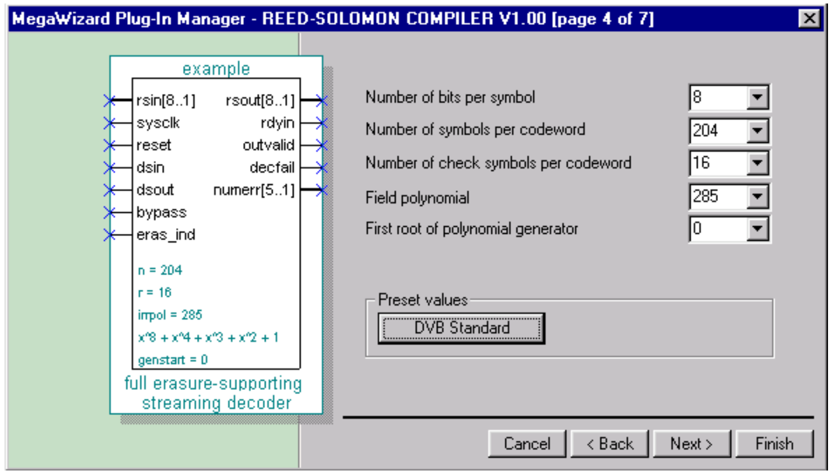



*Figure 3. Selecting the Encoder*



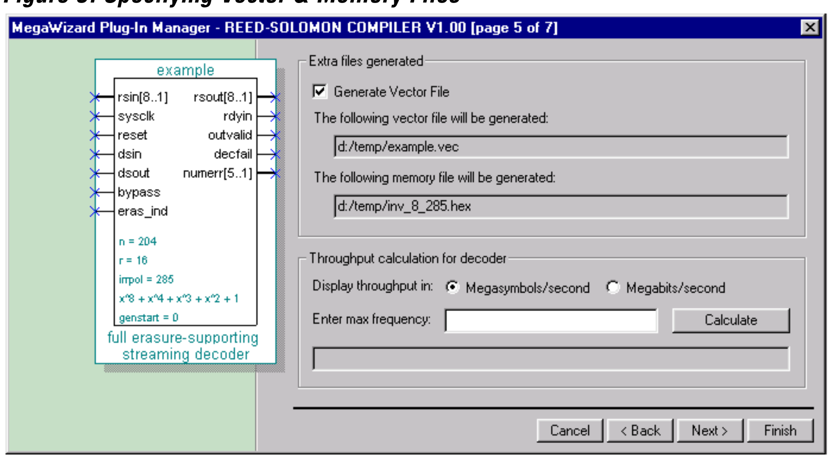

6. Choose the parameters that define the specific RS code you wish to implement (see Figure 4). You can either enter the parameters one by one or click **DVB Standard** to use digital video broadcast (DVB) standard values. See Table 1 on page 13 for a description of the parameters. The MegaWizard Plug-In only allows you to select legal combinations of parameters. Click **Next** when you are finished.

*Figure 4. Choosing the Parameters*



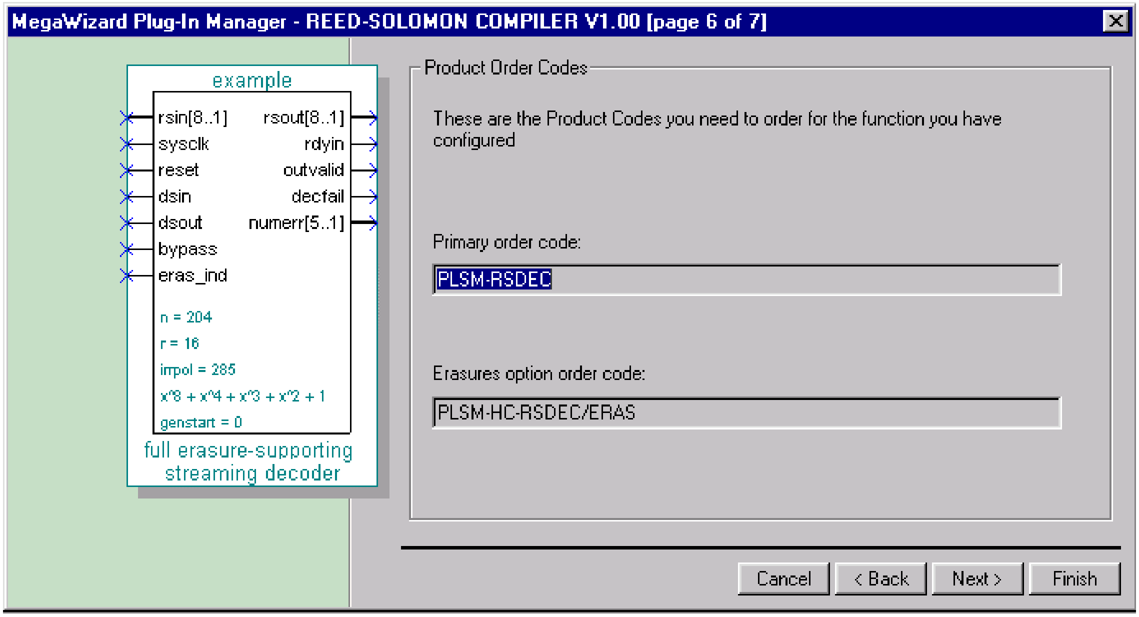

7. Choose whether you want to generate vector files. The MegaWizard Plug-In also lists any memory or vector files generated. For a throughput calculation, enter the frequency in MHz, select the desired units and click **Calculate** (see Figure 5). Click **Next** when you are finished.

*Figure 5. Specifying Vector & Memory Files*




8. The MegaWizard Plug-In lists the product order codes for your custom megafunction (see Figure 6). You will need these codes when you want to license the MegaCore function. Click **Next** when you are finished.

*Figure 6. Product Order Codes*




9. The final screen lists the design files that the wizard creates. In addition to these files, the wizard creates one hexadecimal file (**.hex**) that defines the contents of the ROM used by the MegaCore function, and a Vector File (**.vec**) that can be used to simulate the function in the MAX+PLUS II software. Click **Finish**.

Once you have created a custom megafunction, you can integrate it into your system design and compile.

## Simulate Using VHDL Models

Altera provides register transfer level (RTL) VHDL models that you can use to simulate the functionality of the RS function in your system. The VHDL models are supplied as pre-compiled libraries for the ModelSim simulator and support both encoding and decoding functions. You can integrate these RTL models into your system, speeding simulation. Additionally, you can synthesize the MegaCore function in the MAX+PLUS II software and then generate VHDL Output Files (**.vho**) or Verilog Output Files (**.vo**) for simulation in third-party simulators.

The following instructions describe how to set up your system and how to simulate the VDHL models using ModelSim.

## Setting Up Your System

The pre-compiled VHDL model is provided with the RS compiler MegaCore function, and is installed in the directory **sim_lib\vhdl\ModelSim\ReedS**. Follow the steps below to set up your system to use the model.

1. Run the ModelSim software and create a logical map called **ReedS** to the directory containing the compiled library by typing the following command in the ModelSim software:

   ```
   vmap ReedS <Drive:>\<RS MegaCore Path>
       \sim_lib\vhdl\ModelSim\ReedS
   ```

   You can also use the ModelSim graphical user interface to create the logical map. Refer to the ModelSim on-line help for details.

2. Altera provides sample testbenches and configuration files with the models in the **sim_lib\vhdl\ModelSim\testbench** directory. You should compile these files and save them into the **ReedS** library before simulating in the ModelSim software by performing the following steps:

   a. Choose **Compile** (File menu).

   b. In the **Compile HDL Source Files** dialog box, click **Default Options** (see Figure 7). The **Compiler Options** dialog box appears
   (see Figure 8).

*Figure 7. Compile HDL Source Files Dialog Box*

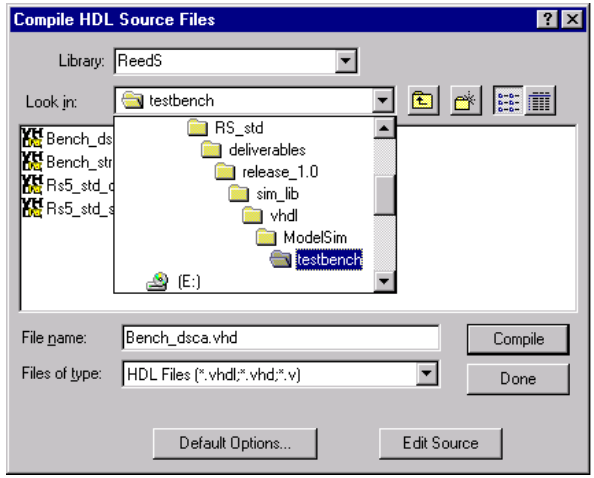
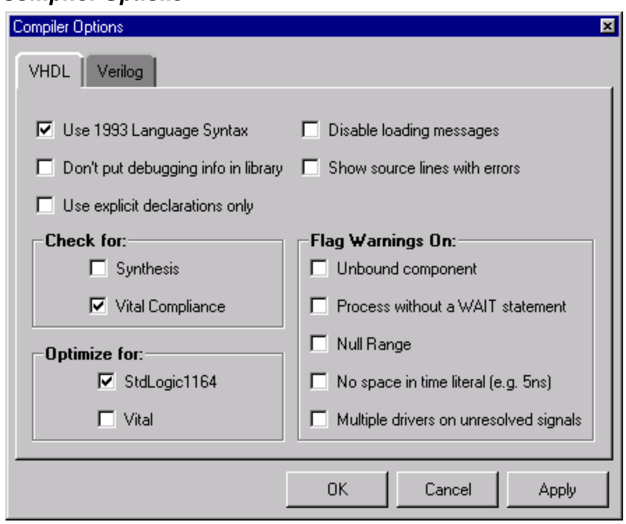



*Figure 8. Compiler Options*




c. In the **Compiler Options** dialog box, turn on the **Use 1993 Language Syntax** option in the **VHDL** tab. Click **OK**.

d. In the **Compile HDL Source Files** dialog box, select **ReedS** in the **Library** drop down list.

e. Select the files you wish to compile and click **Compile**.

f. Once compilation finishes, click **Done**.

### Using the VHDL Model

To use the model, you must first instantiate it in your system. Altera provides four generic testbenches and configurations to demonstrate how to use the models: **cfg_Rs_std_str_tb** for the standard streaming decoder, **cfg_Rs_std_dsc_tb** for the discrete decoder, **cfg_Rs_eras_str_tb** for the erasure-supporting streaming decoder, and **cfg_Rs_eras_dsc_tb** for the erasure-supporting discrete decoder. All files include the encoder and a testbench with a generic stimulus and a generic channel introducing some errors. These configurations must be loaded specifying the parameter values (for non-default values). Five of the parameters (`n`, `check`, `m`, `irrpol`, and `genstart`) can be obtained from page 4 of the MegaWizard Plug-In. Refer to Figure 1 on page 26 for an example of the wizard.

The `speed` parameter can be set to “full” or “half;” the `clock_period` parameter can be any valid time period (for example, 30 ns).
See Table 1 on page 13 for a detailed description of the parameters.

You can quickly load the configuration from the command line in the ModelSim software, for example:

```
vsim -Girrpol=37 -Ggenstart=1 -Gm=5 -Gspeed=\"full\"
    -Gn=31 -Gcheck=6 {-Gclock_period=30 ns}
    ReedS.cfg_rs_std_str_tb
```

You can also use the ModelSim graphical user interface to load the configuration. Refer to the ModelSim on-line help for details.

To instantiate the VHDL model in your system, you can instantiate the parameterized models by using the generic testbenches as templates. Alternatively, you can instantiate the VHDL entity created by the MegaWizard Plug-In, which acts as a wrapper to the parameterized model.

## Compile & Simulate in the MAX+PLUS II Software

The following steps explain how to compile and simulate your design in the MAX+PLUS II software.

☞ For the best results, you should turn on the **Manual Carry Chain** logic option, in the **Global Project Logic Synthesis** dialog box, when compiling your design in the MAX+PLUS II software. Also you should turn on **Register Packing**, to reduce the size by 10 to 15%. However, this may reduce speed (in some cases drastically), so if speed is a premium turn off **Register Packing**.

1. Open the MAX+PLUS II Compiler.

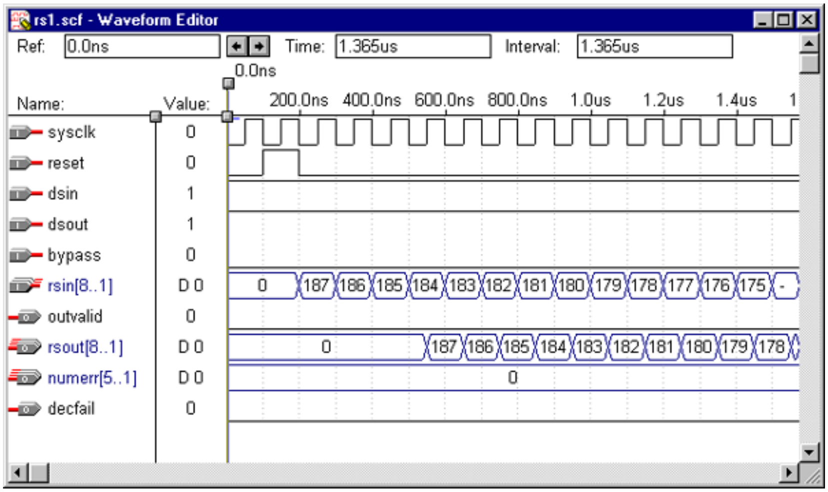
2. Click **Start** to compile your design.

3. Run the MAX+PLUS II Simulator. The Vector File created by the MegaWizard Plug-In Manager for your custom RS compiler MegaCore function is loaded automatically. Click **Start** to begin simulation.

4. When simulation has completed, click the **Open SCF** button to view the design's waveform.

Figure 9 shows an example waveform display simulated in the MAX+PLUS II Simulator.

***Figure 9. Example RS MegaCore Simulation***




After you have verified that your design is functionally correct, you are ready to perform system verification.

## Perform Synthesis Compilation & Post-Routing Simulation

As a standard feature, the Altera MAX+PLUS II software works seamlessly with tools from all EDA vendors, including Cadence, Exemplar Logic, Mentor Graphics, Synopsys, Synplicity, and Viewlogic. After you have licensed the MegaCore function, you can generate EDIF, VHDL, Verilog HDL, and Standard Delay Output Files from the MAX+PLUS II software and use them with your existing EDA tools to perform functional modeling and post-route simulation of your design.

The following sections describe the design flow to compile and simulate your custom MegaCore design with a third-party EDA tool. Refer to Figure 2 on page 4, which shows the design flow for interfacing your third-party EDA tool with the MAX+PLUS II software.

To synthesize your design in a third-party EDA tool and perform post-route simulation, perform the following steps:

1. Create your custom design instantiating a RS compiler MegaCore function.

2. Synthesize the design using your third-party EDA tool. Your EDA tool should treat the MegaCore instantiation as a black box by either setting attributes or ignoring the instantiation.

☞ For more information on setting compiler options in your third-party EDA tool, refer to the MAX+PLUS II ACCESS Key Guidelines.

3. After compilation, generate a hierarchical EDIF netlist file in your third-party EDA tool.

4. Open your EDIF file in the MAX+PLUS II software.

5. Set your EDIF file as the current project in the MAX+PLUS II software.

6. Choose **EDIF Netlist Reader Settings** (Interfaces menu).

7. In the **EDIF Netlist Reader Settings** dialog box, select the vendor for your EDIF netlist file in the **Vendor** drop-down list box and click **OK**.

8. Make logic option and/or place-and-route assignments for your custom logic using the commands in the Assign menu.

9. In the MAX+PLUS II Compiler, make sure **Functional SNF Extractor** (Processing menu) is turned off.

10. Turn on the **Verilog Netlist Writer** or **VHDL Netlist Writer** command (Interfaces menu), depending on the type of output file you want to use in your third-party simulator. Use the **1993 VHDL language** option.

11. Compile your design. The MAX+PLUS II Compiler synthesizes and performs place-and-route on your design, and generates output and programming files.

12. Import your MAX+PLUS II-generated output files (**.edo**, **.vho**, **.vo**, or **.sdo**) into your third-party EDA tool for post-route, device-level, and system-level simulation.

## Configuring a Device

After you have compiled and analyzed your design, you are ready to configure your targeted Altera device. If you are evaluating the MegaCore function with the OpenCore feature, you must license the function before you can generate configuration files.