The CRC Compiler is scheduled for product obsolescence and discontinued support as described in PDN1304. Therefore, Altera does not recommend use of this IP in new designs. For more information about Altera’s current IP offering, refer to Altera’s Intellectual Property website.
Chapter 1. About This Compiler
- Release Information .................................................. 1–1
- Device Family Support .............................................. 1–1
- Features ...................................................................... 1–2
- General Description ................................................... 1–3
- CRC MegaCore Function Verification .............................. 1–3
- Performance and Resource Utilization ........................... 1–3
- Installation and Licensing .......................................... 1–4
  - OpenCore Plus Evaluation ......................................... 1–5
  - OpenCore Plus Time-Out Behavior ............................ 1–5

Chapter 2. Getting Started
- Design Flow ............................................................... 2–1
- CRC Compiler Walkthrough ...................................... 2–1
  - Create a New Quartus II Project ................................. 2–2
  - Launch the MegaWizard Plug-In Manager .................. 2–3
  - Parameterize ............................................................ 2–3
  - Set Up Simulation .................................................... 2–4
  - Generate Files ......................................................... 2–5
- Simulate the Design ................................................... 2–6
- Compile the Design .................................................... 2–6
- Program a Device ....................................................... 2–6

Chapter 3. Functional Description
- Functional Overview .................................................. 3–1
- CRC Generator .......................................................... 3–1
- CRC Checker ............................................................. 3–3
- Multi-Channel Operation ............................................ 3–4
- Parameters ............................................................... 3–6
- Signals ........................................................................ 3–7
- Testbench ................................................................. 3–7
- Running the Testbench Example .................................. 3–8
  - Create a New Project for the Testbench .................. 3–9
  - Create the Generator and Checker Files .................. 3–9
  - Create a Simulation Model for the Checker .......... 3–10
  - Compile and Simulate the Testbench ..................... 3–10

Additional Information
- Revision History ......................................................... Info–i
- How to Contact Altera ................................................ Info–i
- Typographic Conventions .......................................... Info–ii
1. About This Compiler

Release Information

Table 1–1 provides information about this release of the Cyclic Redundancy Check (CRC) compiler.

Table 1–1. CRC Compiler Release Information

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>9.1</td>
</tr>
<tr>
<td>Release Date</td>
<td>November 2009</td>
</tr>
<tr>
<td>Ordering Code</td>
<td>IP-CRC</td>
</tr>
<tr>
<td>Product ID(s)</td>
<td>00BC</td>
</tr>
<tr>
<td>Vendor ID(s)</td>
<td>6AF7</td>
</tr>
</tbody>
</table>

For more information about this release, refer to the MegaCore IP Library Release Notes and Errata.

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore® function. The MegaCore IP Library Release Notes and Errata report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

Device Family Support

MegaCore® functions provide either full or preliminary support for target Altera device families, as described below:

- Full support means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs.
- Preliminary support means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the CRC Compiler function to each Altera device family.

Table 1–2. Device Family Support (Part 1 of 2)

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria® GX</td>
<td>Full</td>
</tr>
<tr>
<td>Arria II GX</td>
<td>Preliminary</td>
</tr>
<tr>
<td>Cyclone®</td>
<td>Full</td>
</tr>
<tr>
<td>Cyclone II</td>
<td>Full</td>
</tr>
<tr>
<td>Cyclone III</td>
<td>Full</td>
</tr>
<tr>
<td>HardCopy® II</td>
<td>Full</td>
</tr>
</tbody>
</table>
The following list summarizes the features of the CRC Compiler:

- Highly parameterized Cyclic Redundancy Check (CRC) generator and checker
- CRC-32, CRC-16-ANSI, and CRC-16-CCITT generator polynomials
- High-speed operation, over 250 MHz for many configurations
- Configurable input datapath width from 1 to 256 bits (power-of-two)
- Configurable CRC starting value
- Built-in support for the following:
  - Inverting output data
  - Reversing input and output data
  - Partial first word
  - Multi-channel operation
- Avalon® Streaming (Avalon-ST) interface without backpressure for message/codeword bits
- Support for all possible end-of-packet byte residues
- Verilog and VHDL demonstration testbenches
- Easy-to-use MegaWizard® interface
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore Plus evaluation
General Description

The CRC Compiler generates high-performance circuits to generate or check Cyclic Redundancy Check (CRC) checksums for packet-based communication.

The CRC generator uses an Avalon-ST interface to receive data and emits generated checksums on a dedicated output. The CRC checker similarly uses an Avalon-ST interface to receive a packet with a CRC checksum and uses a dedicated output to indicate if the checksum is correct. The CRC generator and checker MegaCore functions do not store any data, checksums, or status.

CRC MegaCore Function Verification

Before releasing the CRC Compiler, Altera runs comprehensive regression tests to verify the quality and correctness of the CRC Compiler.

Custom variations generated by the CRC Compiler exercise the CRC compiler’s various parameter options. The resulting simulation models are thoroughly simulated, and the results are verified against bit-accurate master simulation models.

Performance and Resource Utilization

Parameterization allows you to generate the most efficient implementation that meets your design functionality, size, and performance goals.

The section lists the performance and resource utilization for several sample implementations in different device families. The performance metrics were generated using the Quartus® II software version 8.0 and the TimeQuest timing analyzer, with the fastest speed grade selected for the device family. Neither the generator nor checker MegaCore function uses any memory.

Table 1–3 shows the typical expected performance and resource utilization for Cyclone II, Cyclone III, and Stratix GX devices.

<table>
<thead>
<tr>
<th>Device and Speed Grade</th>
<th>MegaCore Function</th>
<th>Parameter Settings (1)</th>
<th>Logic Elements</th>
<th>f_{max} MHz</th>
<th>Throughput Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone II -6</td>
<td>CRC generator</td>
<td>8-bit datapath</td>
<td>34</td>
<td>420.17</td>
<td>3.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 symbol per word</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cyclone III -6</td>
<td>CRC generator</td>
<td>Inputs and outputs not registered</td>
<td>34</td>
<td>450.05</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC-16-CCITT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimize for area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stratix GX -5</td>
<td>CRC checker</td>
<td>16-bit datapath</td>
<td>147</td>
<td>277</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 symbols per word</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC-16-ANSI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimize for speed</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note to Table 1–3:
(1) Parameters set to their default values are not mentioned.
Table 1–4 shows the typical expected performance for Stratix II, Stratix II GX, and Stratix III devices. The performance of the MegaCore function in Stratix IV devices is similar to Stratix III devices.

### Table 1–4. Performance and Resource Utilization Stratix II, Stratix II GX and Stratix III Devices

<table>
<thead>
<tr>
<th>Device and Speed Grade</th>
<th>MegaCore Function</th>
<th>Parameter Settings (1)</th>
<th>ALUTs</th>
<th>Logic Registers</th>
<th>f&lt;sub&gt;max&lt;/sub&gt; MHz</th>
<th>Throughput Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix II –3</td>
<td>CRC generator</td>
<td>32-bit datapath CRC-32 4 symbols per word Optimize for speed</td>
<td>510</td>
<td>373</td>
<td>274</td>
<td>8.7</td>
</tr>
<tr>
<td>Stratix II –3</td>
<td>CRC generator</td>
<td>32-bit datapath CRC-32 4 symbols per word Optimize for speed 8 channels</td>
<td>869</td>
<td>1177</td>
<td>226.96</td>
<td>7.3</td>
</tr>
<tr>
<td>Stratix II GX –3</td>
<td>CRC checker</td>
<td>64-bit datapath 8 symbols per word CRC-16-CCITT Optimize for speed</td>
<td>551</td>
<td>358</td>
<td>241</td>
<td>15.4</td>
</tr>
<tr>
<td>Stratix II GX –3</td>
<td>CRC checker</td>
<td>64-bit datapath 8 symbols per word CRC-16-CCITT Optimize for speed 8 channels</td>
<td>440</td>
<td>614</td>
<td>201.9</td>
<td>12.9</td>
</tr>
<tr>
<td>Stratix III –2</td>
<td>CRC generator</td>
<td>32-bit datapath CRC-32 4 symbols per word Optimize for speed</td>
<td>519</td>
<td>372</td>
<td>330</td>
<td>10.5</td>
</tr>
</tbody>
</table>

**Note to Table 1–4:**

(1) Parameters set to their default values are not mentioned.

### Installation and Licensing

The CRC Compiler is part of the MegaCore<sup>®</sup> IP Library, which is distributed with the Quartus<sup>®</sup> II software and downloadable from the Altera<sup>®</sup> website, [www.altera.com](http://www.altera.com).

For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows and Linux Workstations*.

**Figure 1–1** shows the directory structure after you install the CRC Compiler User Guide Compiler, where `<path>` is the installation directory. The default installation directory on Windows is `c:\altera\<version>`; on UNIX and Solaris it is `/opt/altera/<version>`.
OpenCore Plus Evaluation

With Altera’s free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafuction (Altera MegaCore function or AMPPSM megafuction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include megafuctions.
- Program a device and verify your design in hardware.

You only need to purchase a license for the megafuction when you are completely satisfied with its functionality and performance, and want to take your design to production.

After you purchase a license for CRC Compiler MegaCore function, you can request a license file from the Altera website at www.altera.com/licensing and install it on your computer. When you request a license file, Altera emails you a license.dat file. If you do not have Internet access, contact your local Altera representative.

For more information on OpenCore Plus hardware evaluation, refer to AN 320: OpenCore Plus Evaluation of Megafunctions.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following two operation modes:

- Untethered—the design runs for a limited time.
- Tethered—requires a connection between your board and the host computer. If tethered mode is supported by all megafuctions in a design, the device can operate for a longer time or indefinitely.
All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction’s time-out behavior may be masked by the time-out behavior of the other megafunctions.

For MegaCore functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

Your design stops working after the evaluation time expires; the checksum output signal in a generator and the crcbad output signal in a checker are forced low.
Design Flow

To evaluate the CRC compiler using the OpenCore Plus feature include these steps in your design flow:

1. Obtain and install the CRC compiler.
2. Create a custom variation of the CRC compiler.
3. Implement the rest of your design using the design entry method of your choice.
4. Use the IP functional simulation model to verify the operation of your design.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the Quartus II Handbook.

5. Use the Quartus II software to compile your design.

   You also can generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware.

6. Purchase a license for the CRC compiler.

   After you have purchased a license for the CRC compiler, follow these additional steps:

   1. Set up licensing.
   2. Generate a programming file for the Altera® device(s) on your board.
   3. Program the Altera device(s) with the completed design.

CRC Compiler Walkthrough

This walkthrough explains how to create a CRC compiler MegaCore function variation using the Quartus II MegaWizard™ Plug-In Manager. When you finish generating your custom variation, you can incorporate it into your overall project.

This walkthrough requires the following steps:

- Create a New Quartus II Project
- Launch the MegaWizard Plug-In Manager
- Parameterize
- Set Up Simulation
- Generate Files
Create a New Quartus II Project

You need to create a new Quartus II project with the New Project Wizard, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity.

To create a new project, follow these steps:

1. Choose Programs > Altera > Quartus II <version> (Windows Start menu) to run the Quartus II software. Alternatively, you can use the Quartus II Web Edition software.

2. Choose New Project Wizard (File menu).

3. Click Next in the New Project Wizard Introduction page (the introduction page does not display if you turned it off previously).

4. In the New Project Wizard: Directory, Name, Top-Level Entity page, enter the following information:
   a. Specify the working directory for your project. For example, this walkthrough uses the c:\altera\projects\crc_project directory.
   b. Specify the name of the project. This walkthrough uses example for the project name.

   The Quartus II software automatically specifies a top-level design entity that has the same name as the project. This walkthrough assumes the names are the same.

5. Click Next to close this page and display the New Project Wizard: Add Files page.

   When you specify a directory that does not already exist, a message asks you if the specified directory should be created. Click Yes to create the directory.

6. If you installed the MegaCore IP Library in a different directory from where you installed Quartus II, you must add the user libraries:
   a. Click User Libraries.
   b. Type <path>\ip\altera in the Library name box, where <path> is the directory in which you installed the CRC compiler.
   c. Click Add to add the path to the Quartus II project.
   d. Click OK to save the library path in the project.

7. Click Next to close this page and display the New Project Wizard: Family & Device Settings page.


9. The remaining pages in the New Project Wizard are optional. Click Finish to complete the Quartus II project setup.

You have finished creating your new Quartus II project.
Launch the MegaWizard Plug-In Manager

To launch the MegaWizard Plug-In Manager in the Quartus II software, follow these steps:


   Refer to Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

2. Ensure Create a new custom megafuction variation is selected and click Next.

3. Expand the folder under Installed Plug-Ins by clicking the + icon next to Communications. Expand the Additional Functions directory and select CRC Compiler v9.1.

4. Choose the device family that you want to use for this MegaCore function variation; for this example, select Stratix II GX.

5. Select the output file type for your design; the CRC compiler supports VHDL and Verilog HDL.

6. The MegaWizard Plug-In Manager shows the project path that you specified in the New Project Wizard. Append a variation name for the MegaCore function output files <project path>\<variation name>.

7. Click Next to display the Parameter Settings page for the CRC compiler.

   You can change the page displayed in the MegaWizard Plug-In Manager by clicking Next or Back at the bottom of the dialog box. You can also move directly to a named page by clicking Parameter Settings, EDA, or Summary.

Parameterize

To parameterize your CRC compiler MegaCore function, specify the following values on the Parameter Settings page:

- For Mode, ensure that CRC generator is selected.
- For Input Options, specify these values:
  - Datapath, specify 64 bits.
  - Symbols per word, specify 8.
- For Processing Options, specify these options:
  - Generator, CRC Code, specify CRC-16-CCITT.
  - Optimize for, select Speed.
- For the rest of the options, use the default values.
Figure 2–1 shows you the **Parameter Settings** page.

**Figure 2–1. Parameters Setting Page**

To generate a simulation model or a netlist, click **Next** (or the **EDA** tab) to display the **EDA** page. Turn on the options as required.

**Set Up Simulation**

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software. The model allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

You may only use these models for simulation and expressly not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Turn on **Generate Simulation Model**.

   Some third-party synthesis tools can use a netlist that contains only the structure of the MegaCore function, but not detailed logic, to optimize performance of the design that contains the MegaCore function. Turn on **Generate netlist** only if your synthesis tool supports this feature.

2. Click **Next** (or the **Summary** tab) to display the **Summary** page.

   You can use the check boxes on the **Summary** page to enable or disable the generation of specified files. A gray checkmark indicates a file that is automatically generated; other files are optional.
You can click Back to display the previous page or click Parameter Setting, EDA or Summary, if you want to change any of the MegaWizard options.

**Generate Files**

To generate the files, follow these steps:

1. Turn on the files you wish to generate.

   At this stage you can still click Back or the pages to display any of the other pages in the MegaWizard interface, if you want to change any of the parameters.

2. To generate the specified files and close the MegaWizard interface, click Finish. The Generation Status Window appears. After file generation is successful, click Exit to close the window.

   The generation phase may take several minutes to complete. The generation status window tells you the files that are generated and reports whether the generation phase was successful or not.

3. After you review the generation report, click Exit to close the MegaWizard Plug-In Manager and click Yes on the Quartus II IP Files message.

   The Quartus II IP File (.qip) is a file generated by the MegaWizard interface or SOPC Builder that contains information about a generated IP core. You are prompted to add this .qip file to the current Quartus II project at the time of file generation. In most cases, the .qip file contains all of the necessary assignments and information required to process the core or system in the Quartus II compiler. Generally, a single .qip file is generated for each MegaCore function and for each SOPC Builder system. However, some more complex SOPC Builder components generate a separate .qip file, so the system .qip file references the component .qip file.

**Table 2–1** describes the generated files and other files that may be in your project directory. The names and types of files specified in the summary vary based on whether you created your design with VHDL or Verilog HDL.

**Table 2–1. Generated Files (Sheet 1 of 2)**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;variation name&gt;.bsf</code></td>
<td>Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.</td>
</tr>
<tr>
<td><code>&lt;variation name&gt;.cmp</code></td>
<td>A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.</td>
</tr>
<tr>
<td><code>&lt;variation name&gt;.html</code></td>
<td>Report file which contains lists of the generated files and ports for the MegaCore function variation.</td>
</tr>
<tr>
<td><code>&lt;variation name&gt;.vo</code> or <code>.vho</code></td>
<td>VHDL or Verilog HDL IP functional simulation model.</td>
</tr>
</tbody>
</table>
You can now integrate your custom MegaCore function variation into your design, simulate, and compile your design.

### Simulate the Design

You can simulate your design using the MegaWizard-generated VHDL or Verilog HDL IP functional simulation models. These simulation models are generated in the language you selected in “Set Up Simulation” on page 2–4.

For an example of simulating a design, refer to the section, “Running the Testbench Example” on page 3–8.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the Quartus II Handbook.

### Compile the Design

You can use the Quartus II software to compile your design. Refer to Quartus II Help for instructions on compiling your design.

### Program a Device

After you compile your design, program your targeted Altera device and verify your design in hardware.
Functional Overview

The CRC Compiler MegaCore function generates high-performance circuits to generate or check Cyclic Redundancy Check checksums for packet-based communication. The CRC Compiler dynamically generates RTL based on requirements you supply, rather than parameterizing static RTL.

The generated CRC MegaCore function uses an Avalon-ST interface to receive data. Generated checksums for transmission or the status of received checksums are emitted on dedicated ports. The CRC MegaCore function does not provide an Avalon® Memory-Mapped (Avalon-MM) interface or any other software access interface. The generated CRC MegaCore function also does not store any data, checksums, or status.

CRC Generator

Figure 3–1 shows the block diagram of the CRC generator.

![CRC Generator Block Diagram](image)

**Figure 3–1.** CRC Generator Block Diagram

<table>
<thead>
<tr>
<th>CRC GENERATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
</tr>
<tr>
<td>reset_n</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>datavalid</td>
</tr>
<tr>
<td>startofpacket</td>
</tr>
<tr>
<td>endofpacket</td>
</tr>
<tr>
<td>empty</td>
</tr>
<tr>
<td>datachannel (1)</td>
</tr>
<tr>
<td>checksum</td>
</tr>
<tr>
<td>crcvalid</td>
</tr>
<tr>
<td>chaninitdone (1)</td>
</tr>
<tr>
<td>crcchannel (1)</td>
</tr>
</tbody>
</table>

**Note to Figure 3–1:**
(1) These signals are present only in a multi-channel generator.

The CRC Generator receives message bits from the data input interface on each clock cycle during which datavalid is asserted. The datavalid signal may be asserted or deasserted arbitrarily during or between packets. When datavalid is asserted, all data pins must contain valid data and the empty signal must be zero, with the exception of the final word of a packet. The empty signal, if used, must be set to zero during all cycles except the last word of a packet, that is when endofpacket is asserted. If datavalid is deasserted, data, startofpacket, endofpacket, and
empty are ignored. Following a fixed number of clock cycles after the final word, indicated by assertion of endofpacket, crcvalid is asserted and the computed CRC value is available on the checksum output. The value of checksum is undefined at other times. For the delay between assertion of endofpacket and crcvalid, refer to the Latency panel of the MegaWizard interface.

The CRC generator can start computing the checksum of a new packet while it is completing the calculation for the previous packet, that is before it asserts crcvalid. The data source may assert startofpacket on the clock cycle immediately following endofpacket, regardless of the generator’s latency.

Figure 3–2 and Figure 3–3 shows a typical application of the CRC Generator and operations respectively.

**Figure 3–2.** Typical CRC Generator Application

![CRC Generator Application Diagram](image)

**Figure 3–3.** CRC Generator Operation

![CRC Generator Operation Diagram](image)
CRC Checker

Figure 3–4 shows the block diagram for the CRC Checker.

Figure 3–4. CRC Checker Block Diagram

Note to Figure 3–4:
(1) These signals are present only in a multi-channel checker.

The CRC Compiler MegaCore function receives message and checksum (codeword) bits from the data input interface on each clock cycle during which datavalid is asserted. The datavalid signal can be asserted or deasserted arbitrarily during or between packets as shown in Figure 3–5 on page 3–3 and Figure 3–6 on page 3–4. When datavalid is asserted, all data pins must contain valid data and the empty signal must be zero, with the exception of the final word of a packet. The empty signal, if used, must be set to 0 during all cycles except the last word of a packet, that is when endofpacket is asserted. If datavalid is deasserted, data, startofpacket, endofpacket, and empty are ignored. Following a fixed number of clock cycles after the final word, indicated by assertion of endofpacket, crcvalid is asserted and the crcbad output is set to 1 if the calculated syndrome is incorrect. The value of crcbad is undefined at other times. For the delay between assertion of endofpacket and crcvalid, refer to the Latency panel of the MegaWizard interface.

The CRC checker can start computing the checksum of a new packet while it is completing the calculation for the previous packet, that is before it asserts crcvalid. The data source may assert startofpacket on the clock cycle immediately following endofpacket, regardless of the checker’s latency.

Figure 3–5. Typical CRC Checker Application
For multi-channel (Number of CRC Channels > 1) operations, the CRC Compiler inserts context memories which allow a generator or checker to process multiple packets concurrently. Both speed and area optimization variants support multi-channel operation.

On each datavalid cycle, user applications must set the datachannel input signal to the channel context in use. The MegaCore function sets the crcchannel output signal to the same value after a fixed number of clock latency (at the assertion of crcvalid), to indicate the channel the crcbad or checksum output signal applies to.

There is no channel switch latency or minimum number of cycles between context switches, and no restriction on channel ordering is imposed. The latency of a multi-channel MegaCore function is identical to that of the corresponding single-channel MegaCore function.

However, there is an fmax penalty in multi-channel operation. After each system reset (reset_n = 0), there will be a slight delay in terms of the number of clock cycles due to an internal initialization of the context to a known state. The chaninitdone output signal is asserted when the internal initialization is done. You must wait until this output signal is asserted before sending any data to the MegaCore function for processing.
Figure 3–8 shows the operation of a multi-channel generator.

Figure 3–8. Multi-Channel Generator Operation

Figure 3–9 and Figure 3–10 show the operation of a multi-channel checker when processing a good and incorrect codeword, respectively.

Figure 3–9. Multi-Channel Checker Operation with Good Codeword

Figure 3–10. Multi-Channel Checker Operation with Incorrect Codeword
Parameters

You specify the parameters for the CRC Compiler MegaCore functions using the MegaWizard interface. Table 3–1 describes each available parameter.

Table 3–1. MegaWizard Control Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC Generator</td>
<td>Creates a generator MegaCore function</td>
</tr>
<tr>
<td>CRC Checker</td>
<td>Creates a checker MegaCore function</td>
</tr>
<tr>
<td>Number of CRC Channels</td>
<td>Specifies the number of CRCs supported. All input and output signals are shared among the channels. Hence, only 1 channel can be processed at any one time.</td>
</tr>
<tr>
<td>Datapath</td>
<td>Selects the width of the data input</td>
</tr>
<tr>
<td>Symbols per word</td>
<td>Specifies the number of symbols in a datapath, and thus defines the width of each symbol in bits. A symbol represents the smallest unit of data in a word. For example, if Datapath is set to 32 bits and Symbols per word is set to 4, the width of each symbol is 8 bits.</td>
</tr>
<tr>
<td>Process low-numbered bit in each symbol first</td>
<td>Processes each symbol in reverse order. For example, given the Datapath is 12 bits and Symbols per word is 3, turning on this parameter causes the MegaCore function to process “abcd efgh ijk’l” as “dcba hgfe lkji”.</td>
</tr>
<tr>
<td>Process low-numbered symbol first</td>
<td>Processes the symbols in reverse order. For example, given the Datapath is 12 bits and Symbols per word is 3, turning on this parameter causes the MegaCore function to process “abcd efgh ijk’l” as “ijkl efgh abcd”. If both parameters, Process low-numbered bit in each symbol first and Process low-numbered symbol first are turned on, the MegaCore processes the entire data in reverse order.</td>
</tr>
<tr>
<td>Register all Avalon-ST inputs</td>
<td>Buffers data, datavalid, startofpacket, endofpacket, and empty signals with flip-flops</td>
</tr>
<tr>
<td>Optimize for: speed</td>
<td>Inserts logic to increase the maximum clock speed</td>
</tr>
<tr>
<td>Optimize for: area</td>
<td>Removes logic to reduce logic utilization and latency</td>
</tr>
<tr>
<td>CRC Code</td>
<td>Selects the generator polynomial by name</td>
</tr>
<tr>
<td>Starting value</td>
<td>Presets the CRC register to this value before each packet.</td>
</tr>
<tr>
<td>Number of leading bits to ignore in First word</td>
<td>Specifies the number of bits to ignore at the beginning of a packet.</td>
</tr>
<tr>
<td>Negate checksum</td>
<td>Sets checksum to the one’s complement of the result of the polynomial division. Not applicable to the CRC checker.</td>
</tr>
<tr>
<td>Bit-swap checksum</td>
<td>Applies only to CRC Generator. Selecting this parameter reverses the sequence of the entire output bits. The low-order bits of the remainder are placed in the high-order bits of the checksum. Select this parameter if data transmission starts from low-order bit.</td>
</tr>
<tr>
<td>Register all outputs</td>
<td>Buffers crcvalid and checksum, or crcbad with flip-flops.</td>
</tr>
</tbody>
</table>
Signals

Table 3–2 describes the I/O signals of a CRC MegaCore function.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>In</td>
<td>System clock.</td>
</tr>
<tr>
<td>reset_n</td>
<td>In</td>
<td>Asynchronous reset signal, which is active at 0.</td>
</tr>
<tr>
<td>data[n-1:0]</td>
<td>In</td>
<td>Message bits for generator; codeword bits for checker, where n is a power of 2, from 1 through 256, (1, 2, 4, 8,16, 32, 64,128, or 256).</td>
</tr>
<tr>
<td>datavalid</td>
<td>In</td>
<td>Asserted by the source to indicate new packet data.</td>
</tr>
<tr>
<td>startofpacket</td>
<td>In</td>
<td>Asserted by the source to mark the beginning of a packet. This signal is present only when the parameter Optimize for: speed is set to ON or Number of leading bits to ignore in First word is greater than 0.</td>
</tr>
<tr>
<td>endofpacket</td>
<td>In</td>
<td>Asserted by the source to mark the end of a packet.</td>
</tr>
<tr>
<td>empty[n-1:0]</td>
<td>In</td>
<td>The number of empty symbols during cycles that contain the end of a packet. This signal must be set to 0 during other cycles when datavalid is asserted. The value of n equals log₂(number of symbols per word). This signal is present only when the parameter Symbols per word is greater than 1.</td>
</tr>
<tr>
<td>checksum[n-1:0]</td>
<td>Out</td>
<td>Computed CRC output, where n equals the order of the generator polynomial. This signal is present only in CRC generator.</td>
</tr>
<tr>
<td>crcbad</td>
<td>Out</td>
<td>Asserted if the previous packet had an incorrect CRC. This signal is present only in CRC checker.</td>
</tr>
<tr>
<td>crcvalid</td>
<td>Out</td>
<td>Asserted when the core has completed calculation of the checksum or crcbad value.</td>
</tr>
</tbody>
</table>

Multi-Channel Signals

<table>
<thead>
<tr>
<th>datachannel[n-1:0]</th>
<th>In</th>
<th>The channel on which data is received by the generator or checker. The value of n equals log₂(number of CRC channels).</th>
</tr>
</thead>
<tbody>
<tr>
<td>chaninitdone</td>
<td>Out</td>
<td>Asserted when the MegaCore function has completed its internal initialization process and ready to receive data for processing. User applications must wait for chaninitdone to be asserted before sending data to the MegaCore function for processing. The MegaCore function reinitializes itself each time the reset_n signal is asserted.</td>
</tr>
<tr>
<td>crcchannel[n-1:0]</td>
<td>Out</td>
<td>Indicates the channel whose result is presented at the checksum (generator) or crcbad (checker) output signal. The value of n equals log₂(number of CRC channels).</td>
</tr>
</tbody>
</table>

Testbench

The testbench shows you a typical variant created by the CRC Compiler. The variant and the testbench are fixed and may differ from the variant you create.

The testbench is available in both Verilog and VHDL, plaintext except for the CRC IP Functional Simulation models. The testbench models are synthesizable RTL which allows you to compile and to run the testbench in hardware. The architecture is shown in Figure 3–11.
The packet generator module creates sample message words (packets without CRCs). The CRC generator creates a checksum, which the merge module appends to the end of the packet. The merge module delays the data to account for generator processing time. The result is a codeword. The channel module inserts errors in the codewords, as indicated by the errorInserted signal. The CRC checker indicates the presence of errors in the received codewords. The testbench shows detected errors using the errorDetected signal.

**Running the Testbench Example**

In this section you run the testbench example to illustrate how you do the following tasks:

- Create a testbench project using the Quartus II New Project Wizard
- Create simulation models for the generator and checker using the MegaWizard interface
- Set EDA Tool Settings to simulate your testbench project
- Run compilation and analysis

When you create a MegaCore function variation and turn on the Generate Simulation Model option on the EDA page, the Quartus II software generates the simulation model in the project directory. The simulation model is an Intellectual Property Functional Simulation (IPFS) model with a file extension of .vo or .vho, depending on the language you select. It also creates an example design in the testbench subdirectory. The testbench subdirectory contains the data generator, merge, and channel modules for the example testbench in the following files:

- A top-level simulation file, `tb.v` or `tb.vhd`.
- `crcdemo.v(hd)` for a single channel MegaCore function or `crcdemo_multichan.v(hd)` for a multichannel MegaCore function
- A generator variation file, `crcgen.v(hd)` for a single channel MegaCore function or `crcgen_multichan.v(hd)` for a multichannel MegaCore function.
- The corresponding checker variation file, `crcchk.v(hd)` for a single channel MegaCore or `crcchk_multichan.v(hd)` for a multichannel MegaCore.
The CRC demonstration testbench is supported by the following simulators:

**Verilog**: Mentor ModelSim SE and AE, Cadence NCSim, Synopsys VCS and VCS-MX

**VHDL**: Mentor ModelSim SE and AE, Cadence NCSim, and Synopsys VCS-MX.

For up-to-date information on the supported versions of these simulators, reference the Altera website: [www.altera.com](http://www.altera.com).

### Create a New Project for the Testbench

Following the steps below, create a Quartus II project in the testbench directory.

1. Choose **New Project Wizard** (File menu).
2. Click **Next** in the **New Project Wizard Introduction** page (the introduction page does not display if you turned it off previously).
3. In the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:
   a. Specify the working directory for your testbench project, which should be the **testbench** directory created in your previous project directory. For example, this walkthrough uses the directory `c:\altera\projects\crc_project\testbench`.
   b. Specify **crcdemo** for the name of the project.
      The Quartus II software automatically specifies a top-level design entity that has the same name as the project. This walkthrough assumes the names are the same.
   c. If the directory already exists, a dialog displays that asks what action to take. Click the **Delete the existing project and continue using the New Project Wizard** and then click **OK** to continue.
4. The **New Project Wizard: Add Files** page displays.
   a. Select the file **crcdemo.v** or **crcdemo.vhd**, depending on the language in which your MegaCore function was created, and click **Add** to add the file to your project.
   b. Click **Finish** to add the file to your project.

### Create the Generator and Checker Files

In this section, you create the generator and checker variation files.

1. Open the MegaWizard Plug-In Manager and click **Edit an existing custom megafuction variation** to edit the CRC generator wrapper file **crcgen.v** or **crcgen.vhd**
2. Click **Next** to display the page from which you select the generator wrapper file, **crcgen.v** or **crcgen.vhd**, in the testbench directory.
3. After you select the **crcgen.v** or **crcgen.vhd** the file, click **Next** to continue.

   ![You can review the parameters, but it is recommended that you don’t change the settings.](image-url)

---

© November 2009  Altera Corporation  CRC Compiler User Guide
4. Click the Simulation tab and ensure that Generate Simulation Model is checked. Click Finish to create the IPFS model for the CRC generator.

5. The generation window displays the files being generated and the generation status. After the files are generated successfully, click Exit.

Create a Simulation Model for the Checker

1. Open the MegaWizard Plug-In Manager again (see Step 1 in the previous section “Create the Generator and Checker Files” on page 3–9) to edit the CRC checker wrapper file crcchk.v or crcchk.vhd, in the testbench directory.

2. Click Next to display the page from which you select the CRC checker wrapper file, crcchk.v or crcchk.vhd.

3. After you select the crcchk.v or crcchk.vhd the file, click Next to continue.

   You can review the parameters, but it is recommended that you don’t change the settings.

4. Click the Simulation tab and ensure that Generate Simulation Model is checked.

5. Click Finish to create the IPFS model for the CRC checker.

Compile and Simulate the Testbench

In this section, you navigate the Quartus II software interface to compile and simulate the testbench.

1. Using the Quartus II software, open the Assignments/EDA Tool Settings... menu, then click the plus sign (+) to expand the EDA Tool Settings category and select Simulation to display the Quartus II Simulation page.

2. Set the Tool Name to the simulator of your choice. In this example, it is ModelSim-Altera.

3. Click Compile test bench: in the NativeLink settings, then click Test Benches...

4. In the Test Benches page, click New...

5. In the New Test Bench Settings page, set Test bench name:, Test bench entity:, and Instance: all to tb. Set Run for: to 100 us. In the Test bench files, set File name: to tb.v or tb.vhd and then click Add.

6. Click OK to apply these settings. The Test Benches page displays the settings.

7. You can review the testbench settings. If these setting are correct, click OK to confirm this is the testbench file and settings that you want to use.

8. Click OK to close the EDA Tools ->Simulation page.

9. Open the Tools->Options menu and select EDA Tool Options.

10. Double-click the simulator you chose in Step 1 and enter the path to the executable. Click OK to close the EDA Tool Options page.

11. In the Quartus II Processing menu, select Start->Start Analysis & Elaboration. After the testbench compiles successfully, a message displays telling you that the compilation was successful and informs you of the number of warnings that exist.
12. Click **OK** to close the compilation message.

13. In the Quartus II Tools menu, select **EDA Simulation Tool->Run EDA RTL Simulation**.

14. The Quartus II software starts the simulator you chose in interactive mode and simulates the design. Consult the vendor’s documentation for information on how to use the simulator. When you have finished using the simulator, quit the simulator and continue using Quartus II software.
Revision History

The following table shows the revision history for this user guide.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 2013</td>
<td>13.0</td>
<td>Added obsolescence notice. Refer to PDN1304.</td>
</tr>
<tr>
<td>November 2009</td>
<td>9.1</td>
<td>Maintenance release.</td>
</tr>
<tr>
<td>March 2009</td>
<td>9.0</td>
<td>■ Added support for Arria II GX device family.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed support for HardCopy Stratix device family.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Maintenance release.</td>
</tr>
<tr>
<td>November 2008</td>
<td>8.1</td>
<td>Maintenance release.</td>
</tr>
<tr>
<td>May 2008</td>
<td>8.0</td>
<td>■ Added support for Stratix IV device family.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Maintenance release; updated version numbers.</td>
</tr>
<tr>
<td>October 2007</td>
<td>7.2</td>
<td>■ Added information for multi-channel operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added block diagrams of the MegaCore functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added description of new signals and parameters.</td>
</tr>
<tr>
<td>May 2007</td>
<td>7.1</td>
<td>■ Added support for Arria GX device family.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Revised descriptions for parameters: Symbols per word, Process low-numbered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bits first and Bit-swap checksum.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added new sections: Bit Order Reversal in a Symbol and Partial Word.</td>
</tr>
<tr>
<td>December 2006</td>
<td>7.0</td>
<td>■ Added support for Cyclone III device family.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Maintenance release; updated version numbers.</td>
</tr>
</tbody>
</table>

How to Contact Altera

For the most up-to-date information about Altera® products, see the following table.

<table>
<thead>
<tr>
<th>Contact</th>
<th>Contact Method</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical support</td>
<td>Website</td>
<td><a href="http://www.altera.com/support">www.altera.com/support</a></td>
</tr>
<tr>
<td>Technical training</td>
<td>Website</td>
<td><a href="http://www.altera.com/training">www.altera.com/training</a></td>
</tr>
<tr>
<td></td>
<td>Email</td>
<td><a href="mailto:custrain@altera.com">custrain@altera.com</a></td>
</tr>
<tr>
<td>Altera literature services</td>
<td>Email</td>
<td><a href="mailto:literature@altera.com">literature@altera.com</a></td>
</tr>
<tr>
<td>Non-technical support (General) (Software Licensing)</td>
<td>Email</td>
<td><a href="mailto:nacom@altera.com">nacom@altera.com</a></td>
</tr>
<tr>
<td></td>
<td>Email</td>
<td><a href="mailto:authorization@altera.com">authorization@altera.com</a></td>
</tr>
</tbody>
</table>

Note:
(1) You can also contact your local Altera sales office or sales representative.
## Typographic Conventions

The following table shows the typographic conventions that this document uses.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Indicates command names, dialog box titles, dialog box options, and other GUI labels. For example, <strong>Save As</strong> dialog box.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>Indicates directory names, project names, disk drive names, file names, file name extensions, and software utility names. For example, <code>\qdesigns</code> directory, <code>d:</code> drive, and <code>chiptrip.gdf</code> file.</td>
</tr>
<tr>
<td><strong>Italic Type with Initial Capital Letters</strong></td>
<td>Indicates document titles. For example, <em>AN 519: Stratix IV Design Guidelines.</em></td>
</tr>
<tr>
<td><strong>Italic type</strong></td>
<td>Indicates variables. For example, <code>n + 1</code>. Variable names are enclosed in angle brackets (<code>&lt;&gt;</code>). For example, <code>&lt;file name&gt;</code> and <code>&lt;project name&gt;.pof</code> file.</td>
</tr>
<tr>
<td><strong>Initial Capital Letters</strong></td>
<td>Indicates keyboard keys and menu names. For example, Delete key and the Options menu.</td>
</tr>
<tr>
<td><strong>“Subheading Title”</strong></td>
<td>Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”</td>
</tr>
<tr>
<td><strong>Courier type</strong></td>
<td>Indicates signal, port, register, bit, block, and primitive names. For example, <code>data1</code>, <code>tdi</code>, and <code>input</code>. Active-low signals are denoted by suffix <code>n</code>. For example, <code>resetn</code>. Indicates command line commands and anything that must be typed exactly as it appears. For example, <code>c:\qdesigns\tutorial\chiptrip.gdf</code>. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword <code>SUBDESIGN</code>), and logic function names (for example, <code>TRI</code>).</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., and so on.</td>
<td>Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td>■ ■ ▪ ▣ ▤ ▥</td>
<td>Bullets indicate a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td><img src="image" alt="hand" /></td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td><img src="image" alt="caution" /></td>
<td>A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.</td>
</tr>
<tr>
<td><img src="image" alt="warning" /></td>
<td>A warning calls attention to a condition or possible situation that can cause you injury.</td>
</tr>
<tr>
<td><img src="image" alt="angled arrow" /></td>
<td>The angled arrow instructs you to press Enter.</td>
</tr>
<tr>
<td><img src="image" alt="feet" /></td>
<td>The feet direct you to more information about a particular topic.</td>
</tr>
</tbody>
</table>