Virtex-5 FPGA CRC Wizard v1.3 User Guide

UG189 (v1.4.1) March 24, 2008

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/30/06	1.1	Initial Xilinx release.
05/17/07	1.2	LogiCORE CRC Wizard v1.1 release.
10/10/07	1.3	LogiCORE CRC Wizard v1.2 release. Removed TX_ and RX_ prefix from TX CRC Module ports and RX_ prefix from RX CRC Module port names in text, figures, and tables throughout. Corrected other port names throughout. Updated Figure 3-1. Major updates to Chapter 5, “Example Design Overview.”
03/24/08	1.4	LogiCORE CRC Wizard v1.3 release. Revised installation instructions to comply with ISE 10.1 software procedure in Chapter 2, “Installation and Licensing.” Added “System Requirements,” page 15. Added ISE Simulator (ISim) instructions in “Using Testbench and Simulation Scripts,” page 26.
03/24/08	1.4.1	Minor non-technical edits.

.

Table of Contents

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Installation and Licensing

Chapter 3: Customizing CRC Wizard

Chapter 4: Project Directory Structure

Chapter 5: Example Design Overview

Schedule of Tables

About This Guide

The *Virtex-5 FPGA CRC Wizard v1.3 User Guide* provides information about the LogiCORE™ IP Cyclic Redundancy Check (CRC) Wizard. This guide describes the function and operation of the CRC Wizard for the Virtex™-5 LXT, SXT, and FXT platforms.

Contents

This manual contains the following chapters:

- Preface, "About this Guide" introduces the organization and purpose of this guide, a list of additional resources, and the conventions used in this document.
- Chapter 1, "Introduction" describes the core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Installation and Licensing" provides instructions for installing the CRC Wizard in the CORE Generator tool. It is not necessary to obtain a license to use the Wizard.
- Chapter 3, "Customizing CRC Wizard" describes how to customize a CRC Wizard core using the available parameters.
- Chapter 4, "Project Directory Structure" illustrates the layout of the file directory created by the CRC Wizard.
- Chapter 5, "Example Design Overview" describes the example design included with the CRC Wizard core, detailing the available ports and signals.

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/literature.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets `[ ]`	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** `[`*`option_name`*`]` *design_name*
Braces `{ }`	A list of items from which you must choose one or more	**`lowpwr =`**`{`**`on`**`|`**`off`**`}`
Vertical bar `|`	Separates items in a list of choices	**`lowpwr =`**`{`**`on`**`|`**`off`**`}`

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis `...`	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-5 Platform FPGA User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Chapter 1

Introduction

This chapter introduces the Virtex-5 FPGA CRC Wizard core and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

The Virtex-5 FPGA CRC Wizard is a CORE Generator™ tool designed to support both Verilog and VHDL design environments. In addition, the example design delivered with the core is provided in both Verilog and VHDL.

The Wizard generates customized cores that adapt the CRC block to a wide variety of applications.

The Wizard produces cores that instantiate properly configured CRC blocks for custom applications (see Figure 1-1).

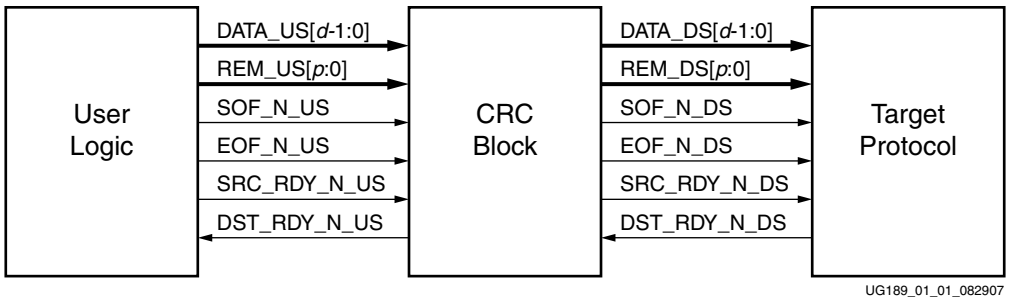

Figure 1-1: **Typical CRC Block Application**

About the Wizard

The Virtex-5 FPGA CRC Wizard is a CORE Generator tool, available at the Xilinx IP Center. For information about system requirements, installation, and licensing options, see Chapter 2, "Installation and Licensing."

Additional Wizard Resources

For detailed information and updates about the Virtex-5 FPGA CRC Wizard core, see the following documents located at Xilinx IP Center Page.

- DS589: *Virtex-5 FPGA CRC Wizard v1.3 Data Sheet*
- Virtex-5 FPGA CRC Wizard Release Notes

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the Virtex-5 FPGA CRC Wizard.

Xilinx provides technical support for use of this product as described in the *Virtex-5 FPGA CRC Wizard v1.3 User Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the Virtex-5 FPGA CRC Wizard and the accompanying documentation.

Virtex-5 FPGA CRC Wizard

For comments or suggestions about the Virtex-5 FPGA CRC Wizard, submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Product name
- Wizard version number
- List of parameter settings
- Explanation of your comments, including whether the case is requesting an *enhancement* (you believe something could be improved) or reporting a *defect* (you believe something isn't working correctly).

Document

For comments or suggestions about the Virtex-5 FPGA CRC Wizard, please submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Document title
- Document number
- Document revision number
- Page number(s) to which your comments refer
- Explanation of your comments, including whether the case is requesting an *enhancement* (you believe something could be improved) or reporting a *defect* (you believe something isn't documented correctly).

Chapter 2

Installation and Licensing

This chapter provides instructions for installing the Virtex-5 FPGA CRC Wizard in the CORE Generator tool. It is not necessary to obtain a license to use the Wizard.

System Requirements

Windows

- Windows XP® Professional 32-bit/64-bit
- Windows Vista® Business 32-bit/64-bit

Linux

- Red Hat® Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat® Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

Tools

- ISE™ v10.1
- Mentor Graphics® ModelSim®: v6.3c
- Cadence® IUS v6.1
- Synopsys® vcs_mxY-2006.06-SP1

Check the release notes for the required Service Pack; ISE Service Packs can be downloaded from www.xilinx.com/support/download.htm.

Before You Begin

Before installing the Wizard, you must have a MySupport account and the ISE 10.1 software installed on your system. If you have already completed these steps, go to "Verifying Your Installation," page 16, otherwise do the following:

1. Click **Login** at the top of the Xilinx home page then follow the onscreen instructions to create a MySupport account.
2. Install the ISE 10.1 software. For the software installation instructions, refer to the ISE 10.1 Design Suite Release Notes and Installation Guide available in ISE Documentation at http://www.xilinx.com/support.

Installing the Wizard

The Virtex-5 FPGA CRC Wizard is included with the ISE 10.1 software. See "Before You Begin" for installation details.

Verifying Your Installation

Use the following procedure to verify that you have successfully installed the Virtex-5 FPGA CRC Wizard in the CORE Generator tool.

1. Start CORE Generator.
2. After creating a new LXT/SXT/FXT project or opening an existing one, the IP core functional categories appear at the left side of the window, as shown in Figure 2-1.

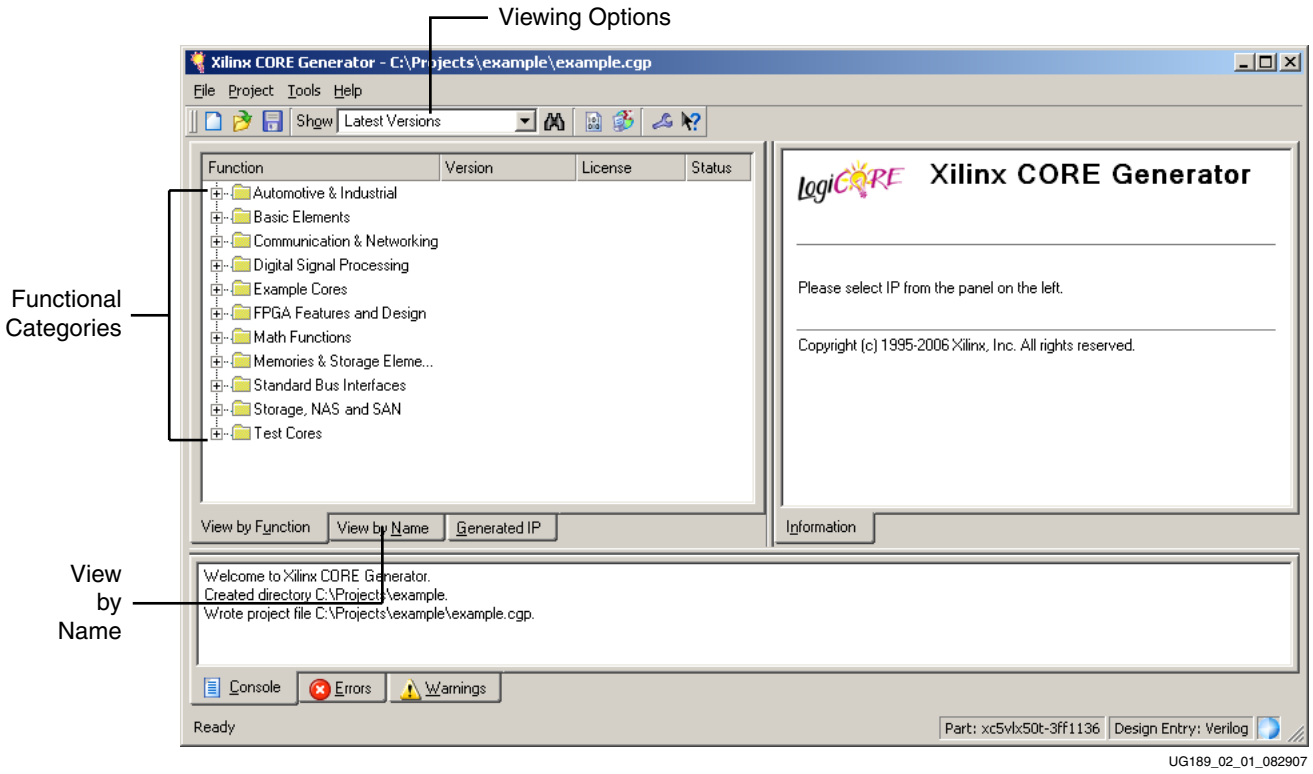

Figure 2-1: **CORE Generator Window**

3. Click to expand or collapse the view of individual functional categories, or click the **View by Name** tab at the bottom of the list to see an alphabetical list of all cores in all categories.
4. To view specific versions of the cores, choose an option from the **Show** drop-down list at the top of the window:
   a. **Latest Versions.** Display the latest versions of all cores.
   b. **All Versions.** Display all versions of cores, including new cores and new versions of cores.
   c. **All Versions including Obsolete.** Display all cores, including those scheduled to become obsolete.

5. Determine if the installation was successful by verifying that CRC Wizard appears at the following location in the Functional Categories list:
   `/Communication & Networking/Error Correction`

   For additional assistance installing the IP Update, visit the Xilinx Support website at www.xilinx.com/support.

XILINX®

Chapter 3

Customizing CRC Wizard

Introduction

This chapter details the customization of parameters available to the user and how these parameters are specified within the IP Customizer interface.

Using the IP Customizer

The CRC Wizard Customizer is presented when the user selects the Virtex-5 FPGA CRC Wizard in the CORE Generator tool. For help starting and using the tool, see *CORE Generator Help*, available in ISE documentation. Each numbered item in Figure 3-1, page 20 corresponds to a section that describes the purpose of the feature.

IP Customizer Page

Figure 3-1 shows the customizer page.

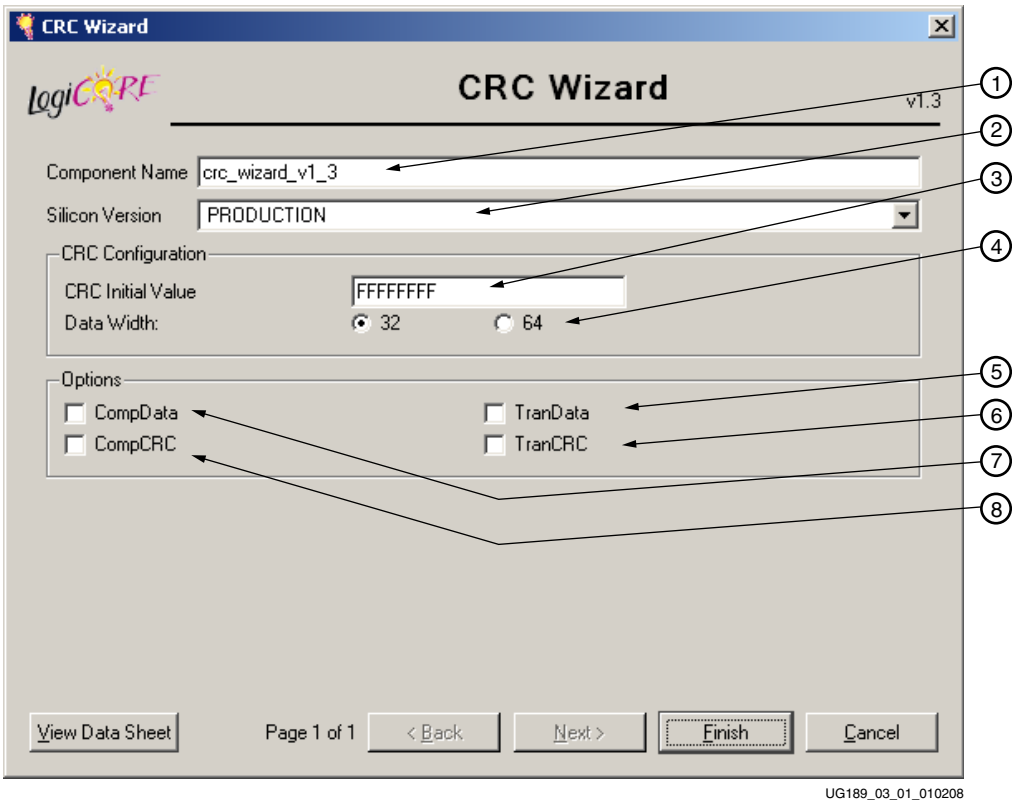

Figure 3-1: **CRC Wizard IP Customizer Page**

1. Component Name

Enter the top-level name for the core in this text box. Illegal names are highlighted in red until they are corrected. All files for the generated core are placed in a subdirectory using this name. The top-level module for the core also uses this name.

Default: `crc_wizard_v1_3`

2. Silicon Version

Select the respective silicon version for the target device.

- ES - Engineering sample
- PRODUCTION - Production silicon

Default: PRODUCTION

3. CRC Initial Value

The initial value loaded to the internal CRC registers for CRC calculation.

The value is:

- User programmable for production silicon
- Fixed to FFFFFFFF for LX30T and LX50T engineering samples

Default: `FFFFFFFF`

4. Data Width

Select the input data width from the user interface to CRC interface.

Default: 32

5. TranData

Transpose Input Data Bytes: If this option is set, each RX/TX_DATA byte is transposed before it is passed through the CRC calculator.

Default: Not checked

6. TranCRC

Transpose CRC Bytes: If this option is set, each byte of the CRC generated from the CRC generator is transposed.

Default: Not checked

7. CompData

Complement Input Data: Some protocols specify that each data byte should be complemented before it is passed through the CRC calculator. Selecting this allows the user to implement this feature.

Default: Not checked

8. CompCRC

Complement CRC Bytes: This feature is useful for protocols, such as Ethernet and Fibre Channel which require a final complementing of each of the CRC bytes.

Default: Not checked

XILINX®

Chapter 4

Project Directory Structure

The customized CRC core is delivered as a set of HDL source modules in the selected language with supporting script and documentation files. These files are arranged in a predetermined directory structure under the project directory name provided to the CORE Generator system when the project is created, as shown below.

```
<top-level name>
|___crc_wizard_readme.txt
|
|___ug189.pdf (user guide)
|
|___src
|   |
|   |___(top-level source file)
|   |___(remaining CRC module source files)
|
|___ucf
|   |
|   |___crc_sample.ucf (crc sample design constraints)
|
|___examples
|   |
|   |___(crc_sample source file)
|   |___(remaining source files for submodules used in example design)
|
|___scripts
|   |
|   |___(simulation script for modelsim and ISim)
|
|___testbench
    |
    |___(testbench files for simulating crc_sample design)
```

Chapter 5

Example Design Overview

Each customized CRC core includes a sample design that uses the core in a simple data transfer system. In the sample design, a frame generator is connected to the TX user interface, and a frame checker is connected to the RX user interface (Figure 5-1).

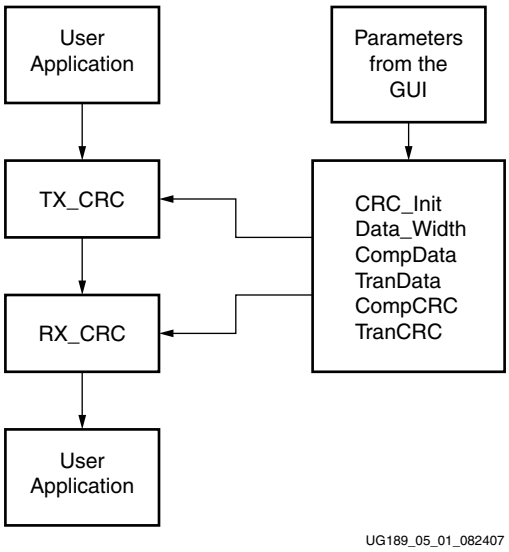

Figure 5-1: **Example Design Block Diagram**

The FRAME GENERATOR block produces a constant stream of data for cores with a framing interface. The data from the FRAME GENERATOR block passes through the TX_CRC block and loops back to the RX_CRC block. The output from the RX_CRC block is passed to the FRAME CHECKER block, which verifies the incoming data and reports an error if the data received does not match the expected data.

When using the example design on a board, be sure to edit the `crc_sample.ucf` file in the *ucf* subdirectory to supply the correct pins and clock constraints. Also ensure that the location of the CRC hard block is specified if a particular position/tile is desired.

Example Design Ports

Table 5-1: **Example Design Ports**

Port	Direction	Description
RESET	Input	Reset signal for the sample design.
ERROR_COUNT[0:7]	Output	Count of the number of data words received by the frame checker that did not match the expected value.
CRC_PASS_FAIL_N	Output	Indicates the CRC Calculation passed. This signal is to be considered only when CRC_VALID is asserted.
CRC_VALID	Output	Indicates the CRC is valid.
CLK	Input	Reference clock for the CRC core.

Using Testbench and Simulation Scripts

The testbench environment consisting of a TX_CRC module looped back to RX_CRC module is included in the `testbench` subdirectory.

The simulation scripts are included in the **`scripts`** folder.

- For simulating with Modelsim, run the following command after setting up the Modelsim environment:

```
xilperl simulate_mti.pl
```

Additionally, the IP also provides simulation script for ISE Simulator (ISim).

- For simulating with ISim, run the following command after setting up the Xilinx environment:

```
xilperl simulate_isim.pl
```

The CRC Primitive

Each CRC block computes a 32-bit CRC using the CRC-32 polynomial specified for PCI Express, Gigabit Ethernet, and other common protocols. The CRC-32 polynomial is:

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

CRC-32 Polynomial *Equation 5-1*

There are two primitives for instantiating CRC hard blocks. The 32-bit CRC primitive, CRC32, can process 8, 16, 24, or 32-bit input data and generates a 32-bit CRC. The 64-bit primitive, CRC64, can process 8, 16, 24, 32, 40, 56, or 64-bit input data and also generate a 32-bit CRC. Using the CRC64 primitive consumes both CRC hard blocks paired with a given transceiver tile.

Using CRC Blocks

Figure 5-2 and Figure 5-3 show a CRC block calculating the CRC for input data. In this figure, the CRC64 primitive is shown. This operation is performed when the CRC is being generated or checked.

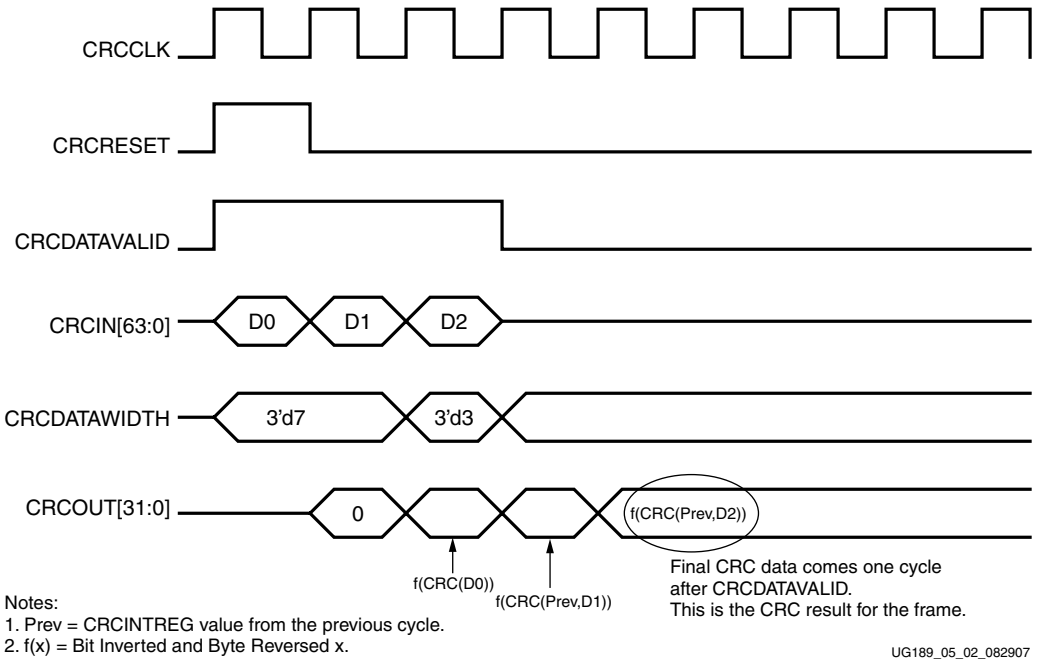

Figure 5-2: **Normal CRC Operation (1 of 2)**

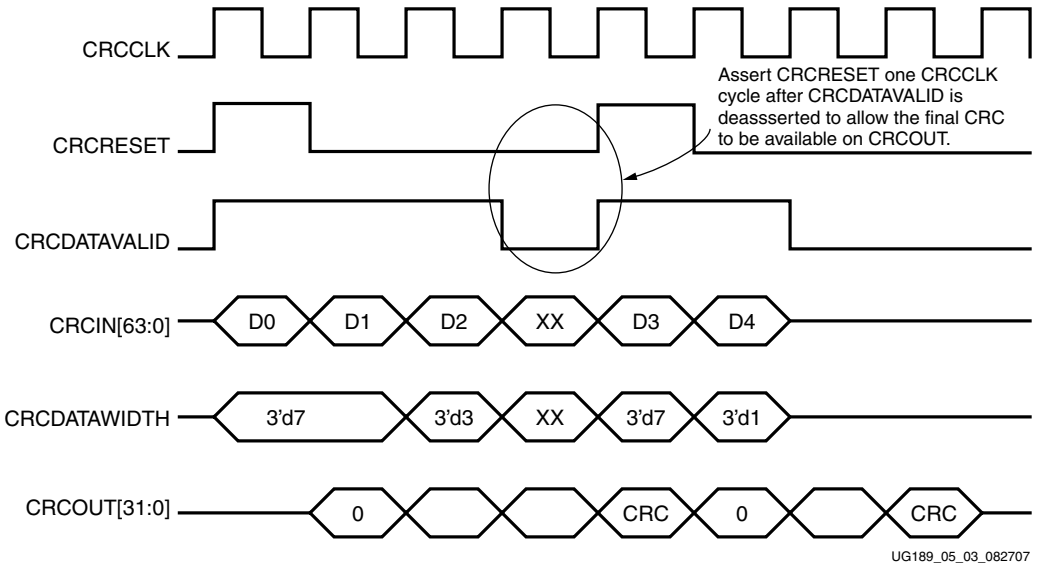

Figure 5-3: **Normal CRC Operation (2 of 2)**

At the start of each frame, **CRCRESET** must be applied to set the initial CRC value to **CRC_INIT**. CRC calculations are cumulative, so this step is required to start the CRC calculation at a known value. **CRC_INIT** is a 32-bit value used for the initial state of the CRC internal register. Its default value is `0xFFFFFFFF`. The **CRC_INIT** value required for a given protocol is specified as part of that protocol's CRC algorithm. Table 5-2 shows the **CRC_INIT** values for some common protocols that use the CRC-32 polynomial.

Table 5-2: **CRC_INIT Values for some Common Protocols**

Protocol	CRC_INIT
Ethernet	`32'hFFFF FFFF`
PCI Express	`32'hFFFF FFFF`
Infiniband	`32'hFFFF FFFF`
Fibre Channel	`32'hFFFF FFFF`
Serial ATA	`32'h5232 5032`

Use of the CRC block is done by applying the first date byte(s) and a reset on the same clock cycle for the fastest throughput. If the fastest throughput is not required, the CRC module can be reset on any prior cycle as long as the CRCDATAVALID input is "0". If CRC32 is being used, the first or most significant byte will be applied to CRCIN[31:24]. The 2nd byte is applied to CRCIN[23:16]. Similarly, the 3rd byte is applied to CRCIN[15:8] and the 4th byte to CRCIN[7:0]. If CRC64 is being used, the same sequence applies. The first or most significant byte will be applied to CRCIN[63:56]. The 2nd byte is applied to CRCIN[55:48]. This continues to where the 8th byte is applied to CRCIN[7:0].

When applying the first byte(s) and a reset, inputs CRCDATAVALID and CRCDATAWIDTH must also be activated. Whenever valid input data is being applied to a CRC module, input CRCDATAVALID must be 1. It is possible to halt computation for a number of clock cycles by setting CRCDATAVALID to 0. CRCDATAWIDTH determines how many input bytes are valid. If only 1 byte is valid, CRCDATAVALID is set to 0. If 2 bytes are valid, CRCDATAVALID is set to 1. This sequence continues until either 3 for CRC32 or 7 for CRC64.

The CRC results will appear on the output CRCOUT[31:0] 1 clock cycle after the last valid data byte has been applied (Figure 5-2, page 27). No new data can be applied to the CRC block until at least 1 clock cycle after the last valid byte has been applied from a CRC computation (Figure 5-3, page 27). The valid CRC will only last until CRCRESET is applied. **CRCOUT** provides the bit-inverted, byte reversed version of the CRC output at the same time. Figure 5-4 shows an example of the Byte rotation and Bit inversion operations.

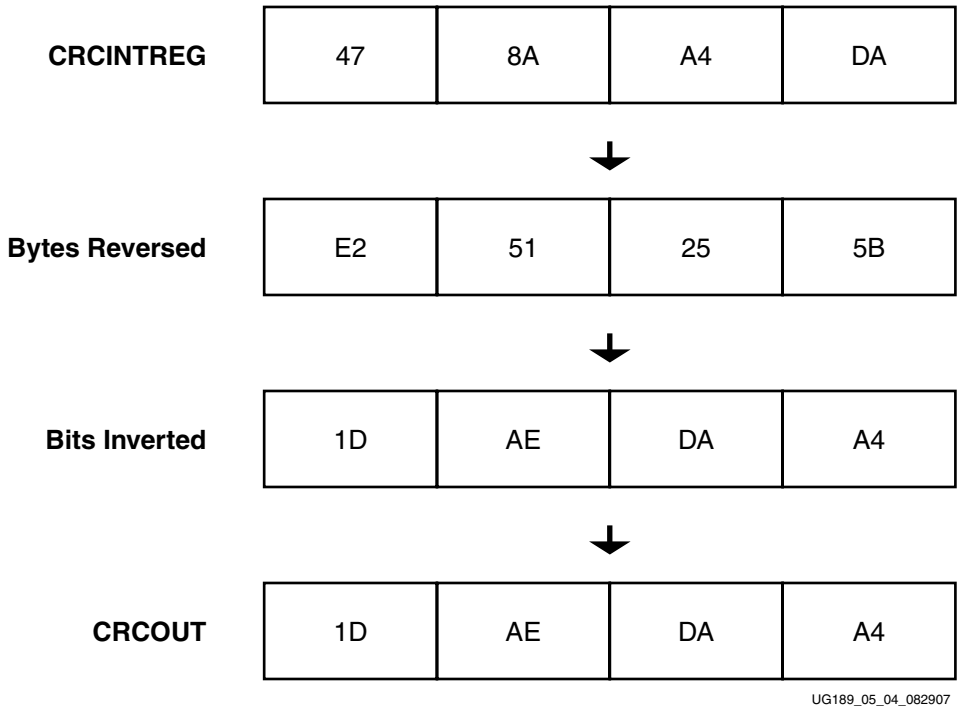

Figure 5-4: **Byte Rotation and Bit Inversion**

Example Module Usage

Figure 5-5 shows an example of the CRC modules used with a framing protocol module.

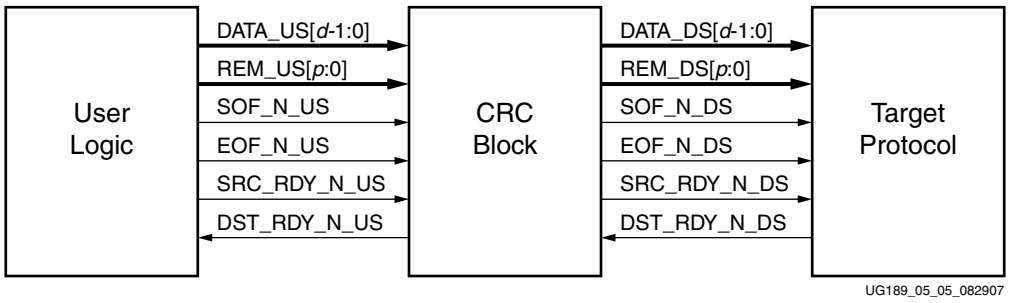

Figure 5-5: **Example Module Usage**

The port names for the CRC Transmit and Receive modules have been divided into two parts: *upstream* and *downstream*. The upstream interface provides the data to the CRC modules, while the downstream interface receives data from the CRC modules. The interface signals are suffixed with **_US** for upstream and **_DS** for downstream.

The configurable CRC solution comprises two parts:

- "TX_CRC Module," page 30 generates and inserts CRC on the transmit path
- "RX_CRC Module," page 33 verifies the CRC on the receive path.

TX_CRC Module

The TX_CRC module has two LocalLink interfaces shown in Figure 5-6. The left side is the upstream interface, while the right side is the downstream interface. This module is both a LocalLink source and destination, which allows it to be inserted into the transmit path of the system. The module generates a CRC value from data input on the upstream interface. The module then inserts the CRC byte(s) at the end of the frame on the downstream interface.

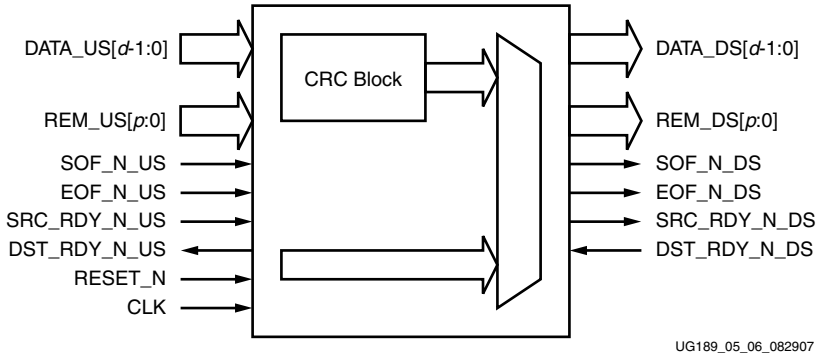

Figure 5-6: **TX_CRC Module**

DST_RDY_N_US can be used to pause the transfer of data from the upstream module. When the CRC word(s) are being inserted into the downstream interface, **DST_RDY_N_US** is de-asserted, indicating that the CRC module is not ready to accept new data. After the CRC word(s) have been inserted, **DST_RDY_N_US** is asserted.

Transmitting Data

LocalLink is a synchronous interface. The LocalLink-based CRC core samples the data on the interface only on the positive edge of **CLK**, and only on the cycles when both **DST_RDY_N** and **SRC_RDY_N** are asserted (low). When LocalLink signals are sampled, they are only considered valid if **SRC_RDY_N** is asserted. The user application can de-assert **SRC_RDY_N** on any clock cycle; this will cause the LocalLink based CRC core to ignore the LocalLink input for that cycle. If this occurs in the middle of a frame, idle symbols are sent through the core, which eventually result in an idle cycles during the frame when it is received at the RX user interface.

TX_CRC Module Interface

The interface signals to the TX_CRC module are listed in Table 5-3.

Table 5-3: **TX_CRC Module Interface Signals**

Port	Direction	Active Level	Description
RESET	Input	High	Reset: The TX_CRC module is reset. The reset is synchronous to CLK.
CLK	Input	N/A	Clock Input: All signals are synchronous to this clock.
Upstream LocalLink Interface			
DATA_US[*d*-1:0] (*d*=[width in bits])	Input	N/A	Data Bus: Packet data is transmitted across this bus. Bus width of 32 or 64 bits is selectable.
REM_US[*p*:0] ($p=\log_2(d/8)-1$)	Input	N/A	Remainder: Indicates the byte offset of the last valid byte on the data bus for the last packet transfer. Remainder value is valid concurrent with end-of-frame assertions. The remainder is binary encoded. If the data width (*d*) is 32-bits, then the remainder bus value (*p*) is 1 yielding a two-bit bus (REM_US[1:0]). A remainder value of 0 indicates bytes [31:24] are valid, while a value of 3 indicates all bytes are valid.
SOF_N_US	Input	Low	Start of Frame: Indicates the beginning of a frame transfer on the data bus. This signal is coincident with the first data word. This signal also indicates the start of calculation.
EOF_N_US	Input	Low	End of Frame: Indicates the end of a frame transfer.
SRC_RDY_N_US	Input	Low	Source Ready: Indicates data that is provided by the source interface on the data bus is valid during the current cycle.
DST_RDY_N_US	Output	Low	Destination Ready: Indicates that the destination interface is ready to accept data presented to it on the data bus in the current cycle.
Downstream LocalLink Interface			
DATA_DS[*d*-1:0] (*d*=[width in bits])	Output	N/A	Data Bus: Packet data is transmitted across this bus. The CRC that is calculated over the entire frame is appended to the end of the frame.
REM_DS[*p*:0]	Output	N/A	Remainder: Indicates the byte offset of the last valid byte on the data bus for the last packet transfer. Remainder value is valid concurrent with end-of-frame assertions. The remainder is binary encoded.
SOF_N_DS	Output	Low	Start of Frame: Indicates the start of frame transfer.
EOF_N_DS	Output	Low	End of Frame: Indicates the end of a frame transfer including the CRC.
SRC_RDY_N_DS	Output	Low	Source Ready: Indicates data that is provided by the source interface on the data bus is valid during the current cycle.
DST_RDY_N_DS	Input	Low	Destination Ready: Indicates that the destination interface is ready to accept data presented to it on the data bus in the current cycle. The DST_RDY_N_DS signal is passed through to the upstream interface without any latency.

Data Transfer through the TX_CRC Module

Figure 5-7 shows the functional waveforms of the TX_CRC module. The calculation of the CRC starts on the assertion of the **SOF_N_US** signal on the upstream interface. The calculation is paused whenever **SRC_RDY_N_US** or **DST_RDY_N_US** is de-asserted and resumes when both are asserted.The IP handles variable REM feature. It appends CRC calculated at the end of frame and updates the REM value accordingly. For REM > 3, the CRC value is distributed over two clock cycles and sent over the downstream end.

There is a fixed latency between the upstream and downstream interface. For engineering sample version of device, the latency will increase between the upstream and downstream interface. The following waveform depicts the timing for a production device.The **EOF_N_US** signal terminates the calculation of the CRC. In Figure 5-7, the data width shown is 64 bits, while the CRC polynomial used is CRC32. The CRC packet is appended to the end of the packet. While the CRC bytes are being appended, the **DST_RDY_N_US** signal is de-asserted indicating that the CRC block is not ready to accept new data on the upstream interface. The waveform for TX_CRC considers a 64-bit framing interface with REM = 5. After CRC insertion at the end of data, REM value is updated for the downstream end.

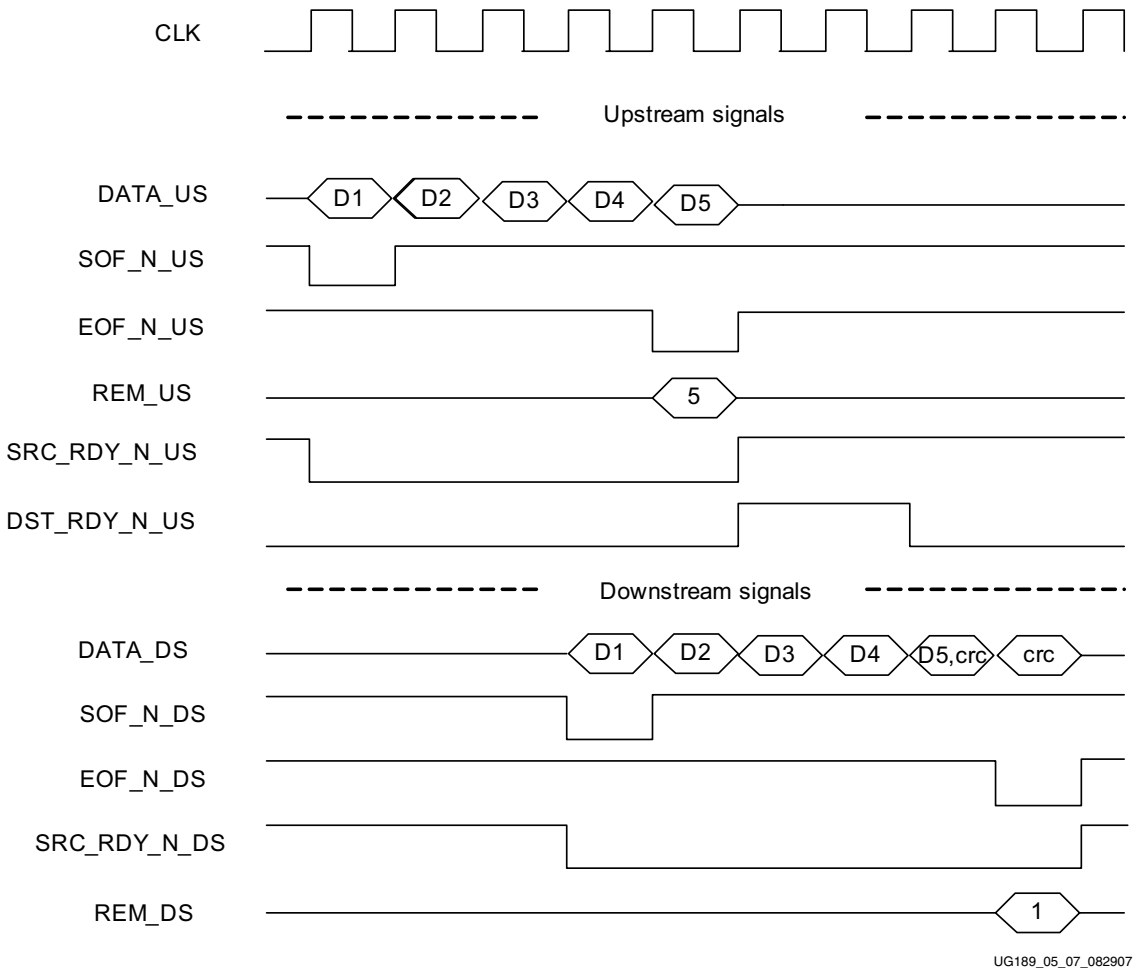

Figure 5-7: **TX_CRC Data Transfer**

RX_CRC Module

The RX_CRC module calculates the CRC on incoming data from the upstream LocalLink interface. It also passes the data through to its downstream LocalLink interface. The downstream interface includes two extra signals, **CRC_PASS_FAIL_N** and **CRC_VALID**. The **CRC_PASS_FAIL_N** signal indicates whether or not the frame contains a valid CRC.

This signal is valid only when **CRC_VALID** is asserted high. The interface of the RX_CRC module is shown in Figure 5-8.

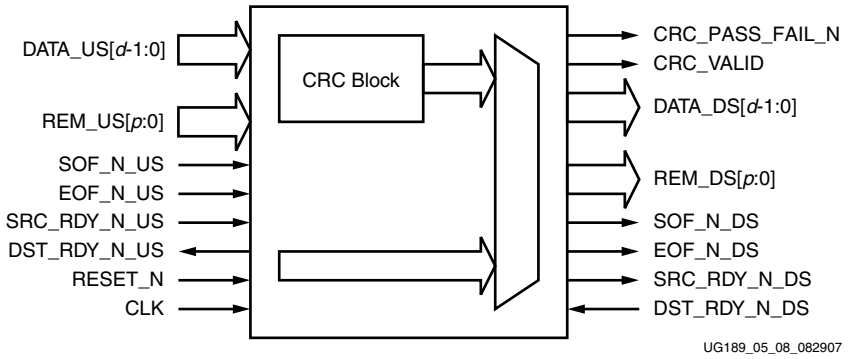

Figure 5-8: **RX_CRC Module**

RX_CRC Module Interface

The interface signals to the RX_CRC module are listed in the Table 5-4 below.

Table 5-4: **RX_CRC Module Interface Signals**

Port	Direction	Active Level	Description
RESET	Input	High	Reset: The TX_CRC module is reset. The reset is synchronous to CLK.
CLK	Input	N/A	Clock Input: All signals are synchronous to this clock.
Upstream LocalLink Interface			
DATA_US[*d*-1:0] (*d*=[width in bits])	Input	N/A	Data Bus: Packet data is transmitted across this bus. Bus width of 32 or 64 bits is selectable.
REM_US[*p*:0] ($p=\log_2(d/8)$-1)	Input	N/A	Remainder: Indicates the byte offset of the last valid byte on the data bus for the last packet transfer. Remainder value is valid concurrent with end-of-frame assertions. The remainder is binary encoded. If the data width (*d*) is 32-bits, then the remainder bus value (*p*) is 1 yielding a two-bit bus (REM_US[1:0]). A remainder value of 0 indicates bytes [31:24] are valid, while a value of 3 indicates all bytes are valid. Current release assumes all data bytes to be valid.
SOF_N_US	Input	Low	Start of Frame: Indicates the beginning of a frame transfer on the data bus. This signal is coincident with the first data word. This signal also indicates the start of calculation of the receive CRC.
EOF_N_US	Input	Low	End of Frame: Indicates the end of a frame transfer on the data.

Table 5-4: **RX_CRC Module Interface Signals *(Cont'd)***

Port	Direction	Active Level	Description
SRC_RDY_N_US	Input	Low	Source Ready: Indicates data that is provided by the source interface on the data bus is valid during the current cycle.
DST_RDY_N_US	Output	Low	Destination Ready: Indicates that the destination interface is ready to accept data presented to it on the data bus in the current cycle.
Downstream LocalLink Interface			
DATA_DS[*d*-1:0] (*d*=[width in bits])	Output	N/A	Data Bus: Packet data is transmitted across this bus.
REM_DS[*p*:0]	Output	N/A	Remainder: Indicates the byte offset of the last valid byte on the data bus for the last packet transfer. Remainder value is valid concurrent with end-of-frame assertions. The remainder is binary encoded.
SOF_N_DS	Output	Low	Start of Frame: Indicates the start of frame transfer.
EOF_N_DS	Output	Low	End of Frame: Indicates the end of a frame transfer including the CRC.
SRC_RDY_N_DS	Output	Low	Source Ready: Indicates data that is provided by the source interface on the data bus is valid during the current cycle.
DST_RDY_N_DS	Input	Low	Destination Ready: Indicates that the destination interface is ready to accept data presented to it on the data bus in the current cycle. The DST_RDY_N_DS signal is passed through to the upstream interface without any latency.

Data Transfer through RX_CRC Module

Figure 5-9 shows the data transfer through the RX_CRC module. The calculation of the CRC starts on the assertion of the **SOF_N_US** signal on the upstream interface. The CRC calculated for data at the receive end is compared with the CRC received from the transmit end. Latency is fixed between the upstream and downstream interface. In Figure 5-9, the CRC polynomial is CRC32. The user should note that the CRC is stripped by realigning the **EOF_N_DS** signal and REM_DS is adjusted to reflect the correct value. Figure 5-9 shows a packet that has passed the CRC verification. This is indicated by the **CRC_PASS_FAIL_N** signal, which is valid at the assertion of **CRC_VALID**.

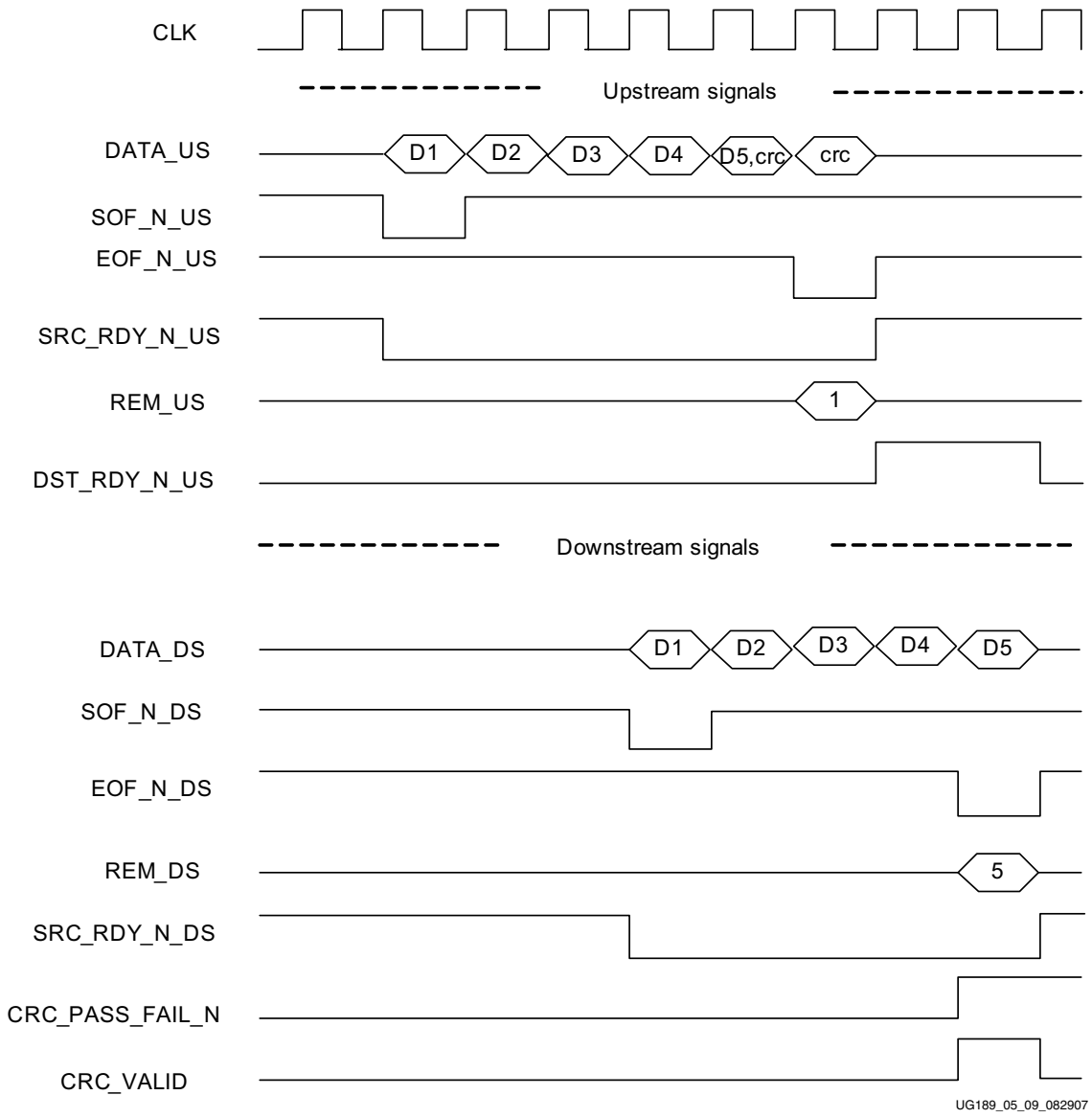

Figure 5-9: **RX_CRC Data Transfer**