SYNOPSYS®

DesignWare Cores SuperSpeed USB 3.0 Controller

Programming Guide

DWC_usb3 – Product Codes

Version 3.30b
May 2018

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com

Contents

Revision History

Date	Version	Description
May 2018	3.30b	No updates to this document
February 2018	3.30a	This is the first version of this document. All of the chapters in this guide have been moved from the DWC SuperSpeed USB 3.0 Controller Databook. Removed support for the following features: ■ OTG ■ ADP ■ SSIC Chapter 1, “Register Descriptions” ■ Added: - `DWC_usb3_block_link` and `DWC_usb3_block_debug` register blocks - `GUCTL3` and `GUSB2RHBCTL` registers - `DisUSB2RefClkGtng`, `DisRefClkGtng`, and `OVRD_FSLS_DISC_TIME` register fields ■ Updated `ULSTCHNGREQ`, `NOLOWPWRDUR`, `USBLNKST`, and `SOFFN` register field descriptions Chapter 2, “Additional Register Information” ■ Updated “Controller CSR Memory Map” on page 282 Chapter 4, “Programming DWC_usb3 in Device Mode” ■ Updated “Initialization after U3 Exit” on page 339 ■ Updated Isoc as Isoc-First in “Example of Setting Up TRBs” on page 308 (Figure 3-12 on page 311 and Figure 3-13 on page 312)

Preface

This programming guide describes the programming sequences and requirements for the Synopsys DesignWare® Cores SuperSpeed USB 3.0 controller in device and host mode. It also describes the programming flow for various features.

For more details on USB 3.0 protocol, refer to *USB 3.0 Protocol Overview* training module of the Synopsys *USB University* series.

Product Codes

Table 1 lists the products and product codes for the DWC_usb3 controller.

Table 1 Product Codes for DWC SuperSpeed USB 3.0 Controller

Component Name or Add-On	Product Code
Base Products	
DWC USB 3.0 Device-AHB[a]	6793-0
DWC USB 3.0 Host-AHB[b]	6897-0
DWC USB 3.0 Dual Role Device-AHB[c]	7567-0
DWC USB 3.0 Hub	6921-0
DWC AP USB 3.0 Device	C234-0
DWC AP USB 3.0 Host	C233-0
DWC AP USB 3.0 Dual Role Device	C232-0
Add-Ons	
DWC USB 3.0 Device-Isoch Add-On	7571-0
DWC USB 3.0 Host-Isoch Add-On	7593-0
DWC USB 3.0 DRD-Isoch Add-On	7568-0
DWC USB 3.0 Hibernation Add-On	9228-0
DWC USB 3.0 HSIC Add-On	7387-0

a. DWC USB 3.0 Device-AHB includes DWC USB 3.0 Device-AXI Add-On and DWC USB 3.0 Device-Isoch Add-On.
b. DWC USB 3.0 Host-AHB includes DWC USB 3.0 Host-AXI Add-On and DWC USB 3.0 Host-Isoch Add-On.
c. DWC USB 3.0 Dual Role Device-AHB includes DWC USB 3.0 DRD-AXI Add-On and DWC USB 3.0 DRD-Isoch Add-On.

Databook Organization

The chapters and appendices of this databook are organized as follows:

- Chapter 1, "Register Descriptions", provides the memory map and descriptions for the DWC_usb3 controller's control and status registers (CSRs).
- Chapter 2, "Additional Register Information", provides additional details on the registers including register resets and considerations for register write and read testing.
- Chapter 3, "Device Data Structures", describes the descriptor structure, command structure and event buffer structure used by the DWC_usb3 controller in device mode.
- Chapter 4, "Programming DWC_usb3 in Device Mode", gives instructions for programming the DWC_usb3 controller in device mode.
- Chapter 5, "Programming DWC_usb3 in Host Mode", gives instructions for programming the DWC_usb3 controller in host mode.
- Chapter 6, "Programming DWC_usb3 for Hibernation", gives instructions for programming the DWC_usb3 controller for Hibernation.
- Chapter A, "Internal Parameter Descriptions", describes the internal parameters that might be indirectly referenced in expressions in the Registers chapter.

Related Product Documentation

USB 3.0 Controller

Before installing the USB 3.0 Controller, you can download the latest document set, including the Datasheet, Installation Guide, QuickStart, Databook, and Release Notes, at:

Device: http://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_device

Host: http://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_host

Dual-Role Device (DRD): http://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_drd

Hub: http://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_hub

(SolvNet ID required)

USB 3.0 Controller Linux Driver Software

You can download the Linux Driver Software at:

https://www.kernel.org/

Directory: /drivers/usb/dwc3

Synopsys supports these USB drivers.

Verification IP (VIP)

To run simulations in coreConsultant, you must download and install the Synopsys Verification IP for USB 3.0 and AMBA from the SolvNet Download Center, at:

https://solvnet.synopsys.com/DownloadCenter/dc/product.jsp

For more information, refer to the *DesignWare Cores SuperSpeed USB 3.0 Installation Guide*.

Synopsys Tools

- *coreConsultant User Guide*, Synopsys, Inc. (included with the coreConsultant tool)
  https://www.synopsys.com/dw/doc.php/doc/coretools/latest/coreconsultant_user.pdf

Web Resources

- DesignWare IP product information: http://www.designware.com
- Your custom DesignWare IP page: http://www.mydesignware.com
- Documentation through SolvNet: http://solvnet.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Reference Documentation

The following standards are applicable to the USB 3.0 Controller product:

USB Protocol (from USB Implementers Forum)

- *Universal Serial Bus 3.0 Specificat*ion, Revision 1.0, November 12, 2008
- *Universal Serial Bus Mass Storage Class UAS Protocol Specification,* Revision 0.6, June 1, 2008
- *Universal Serial Bus Specification*, Revision 2.0, USB-IF, April 27, 2000
- *Errata for "USB Revision 2.0 April 27 2000"* as of May 28, 2002, USB-IF
- *High-Speed Inter-Chip USB Electrical Specification, Version 1.0,* USB-IF, *September 23, 2007*
- *Inter-Chip Supplement to the USB Specification, Version 1.0,* USB-IF, *March 13, 2006*
  http://www.usb.org/developers/docs
- *Battery Charging Specification*, Revision 1.2, Nov 2, 2010

USB Protocol (other)

- *Universal Serial Bus PC Legacy Compatibility Specification*, 0.9 Draft Revision, May 30, 1996

  http://wwwindev.synopsys.com/~dr/products/usb30/uploads/Main.ExternalSpecs/usb_le9.pdf

PHYs and Transceivers

- *PHY Interface for the PCI Express™, SATA, and USB 3.0 Architectures*, Version 4.00, Intel Corp.

  http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/phy-interface-pci-express-sata-usb30-architectures.pdf

- *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification*, Revision 1.05, Intel Corp., March 29, 2001

  http://www.intel.com/technology/usb/download/2_0_Xcvr_macrocell_1_05.pdf

- *UTMI+ Specification*, Revision 1.0, ULPI Working Group, February 25, 2004

  http://www.ulpi.org/ (May require registration)

- *UTMI+ Low Pin Interface (ULPI) Specification*, Revision 1.1, ULPI Working Group, October 20, 2004

  http://www.ulpi.org/ (May require registration)

- *PDIUSBP11A Universal Serial Bus Transceiver Product Specification*, Philips Semiconductor, June 12, 2001

  http://www.semiconductors.philips.com/acrobat_download/datasheets/PDIUSBP11A_3.pdf

- *Mini USB Analog Carkit Interface Specification*, CEA-936, Revision 2, December 2002

  http://global.ihs.com/ (Available for purchase)

SoC Buses

- *AMBA Specification*, Revision 2.0, ARM Ltd, ARM IHI 0001, 1999 (May require registration)

  http://www.arm.com/products/solutions/AMBA_Spec.html

- *AMBA AXI(3) Protocol Specification*, v2.0, ARM Ltd, ARM IHI 0022C, 2003-2010 (May require registration)

  http://www.arm.com/products/solutions/axi_spec.html

Host Controllers

- *eXtensible Host Controller Interface for Universal Serial Bus (xHCI)*, Revision 1.1 with errata to 12/20/13, Intel Corp., December 20, 2013 (Request from USB-IF)

  http://www.intel.com/technology/usb/xhcispec.htm

Design Methodology

- *Reuse Methodology Manual: For System-On-A-Chip Designs*, Third Edition, Kluwer Academic Publishers, 2002

  http://www.springer.com/engineering/circuits+%26+systems/book/978-0-387-74098-0

Customer Support

To obtain support for your product, choose one of the following:

- First, prepare the following debug information, if applicable:
  - For environment setup problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, use the following menu entry:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This menu entry gathers all the Synopsys product data needed to begin debugging an issue and writes it to the file <*core tool startup directory*>/debug.tar.gz.
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveforms file (such as VPD or VCD)
    - Identify the hierarchy path to the DesignWare instance
    - Identify the timestamp of any signals or locations in the waveforms that are not understood
- Then, contact the Support Center, with a description of your question and supply the above information, using one of the following methods:
  - *For fastest response*, use the SolvNet website. If you fill in your information as explained below, your issue is automatically routed to a support engineer who is experienced with your product. The **Sub Product** entry is critical for correct routing.

    Go to http://solvnet.synopsys.com/support/open_case.action. Provide the requested information, including:
    - Product: DesignWare Cores
    - Sub Product: USB 3.0 Device, USB 3.0 Host, USB 3.0 Hub, USB 3.0 DRD
    - Version: 3.30b
    - Problem Type
    - Priority
    - Title: Provide a short summary of the issue or list the error message you have encountered
    - Description: For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

    After creating the case, attach any debug files you created in the previous step.
  - Or, send an e-mail message to support_center@synopsys.com (your email is queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
    - Include the Product name, Sub Product name, Product version, and Tool Version number in your e-mail (as identified above) so it can be routed correctly.
    - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
    - Attach any debug files you created in the previous step.
  - Or, telephone your local support center:
    - North America:

      Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
    - All other countries: https://www.synopsys.com/support/global-support-centers.html

1 Register Descriptions

This chapter details all possible registers in the controller. They are arranged hierarchically into maps and blocks (banks). For configurable IP titles, your actual configuration might not contain all of these registers.

Attention: For configurable IP titles, do not use this document to determine the exact attributes of your register map. It is for reference purposes only.

When you configure the controller in coreConsultant, you must access the register attributes for your actual configuration at `workspace/report/ComponentRegisters.html` or `workspace/report/ComponentRegisters.xml` after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the registers that are not in your actual configuration. This does not apply to non-configurable IP titles. In addition, all parameter expressions are evaluated to actual values. Therefore, the Offset and Memory Access values might change d pending on your actual configuration.

Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>**) that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Exists Expressions

These expressions indicate the combination of configuration parameters required for a register, field, or block to exist in the memory map. The expression is only valid in the local context and does not indicate the conditions for existence of the parent. For example, the expression for a bit field in a register assumes that the register exists and does not include the conditions for existence of the register.

Offset

The term *Offset* is synonymous with *Address.*

Memory Access Attributes

The Memory Access attribute is defined as `<ReadBehavior>/<WriteBehavior>` which are defined in the following table.

Table 1-1 Possible Read and Write Behaviors

Read (or Write) Behavior	Description
RC	A read clears this register field.
RS	A read sets this register field.
RM	A read modifies the contents of this register field.
Wo	You can only write to this register once field.
W1C	A write of 1 clears this register field.
W1S	A write of 1 sets this register field.
W1T	A write of 1 toggles this register field.
W0C	A write of 0 clears this register field.
W0S	A write of 0 sets this register field.
W0T	A write of 0 toggles this register field.
WC	Any write clears this register field.
WS	Any write sets this register field.
WM	Any write toggles this register field.
no Read Behavior attribute	You cannot read this register. It is Write-Only.
no Write Behavior attribute	You cannot write to this register. It is Read-Only.

Table 1-2 Memory Access Examples

Memory Access	Description
R	Read-only register field.
W	Write-only register field.
R/W	Read/write register field.
R/W1C	You can read this register field. Writing 1 clears it.
RC/W1C	Reading this register field clears it. Writing 1 clears it.
R/Wo	You can read this register field. You can only write to it once.

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 1-3 Optional Attributes

Attribute	Description
Volatile	As defined by the IP-XACT specification. If true, indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this field can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. For example, when the controller updates the register field contents.
Testable	As defined by the IP-XACT specification. Possible values are unconstrained, untestable, readOnly, writeAsRead, restore. Untestable means that this field is untestable by a simple automated register test. For example, the read-write access of the register is controlled by a pin or another register. readOnly means that you should not write to this register; only read from it. This might apply for a register that modifies the contents of another register.
Reset Mask	As defined by the IP-XACT specification. Indicates that this register field has an unknown reset value. For example, the reset value is set by another register or an input pin; or the register is implemented using RAM.
* Varies	Indicates that the memory access (or reset) attribute (read, write behavior) is not fixed. For example, the read-write access of the register is controlled by a pin or another register. Or when the access depends on some configuration parameter; in this case the post-configuration report in coreConsultant gives the actual access value.

get_field_val Tcl Proc:

For the **Value After Reset** for some of the register fields, there are references to the get_field_val Tcl procedure.

- Syntax:

  mpformat "0x%x" [get_field_val "DWC_USB3_EXAMPLE_PARAMETER" x y]
- Description:

  The get_field_val is an internal Tcl procedure that is used to return the value of DWC_USB3_EXAMPLE_PARAMETER[(x+y-1):x].
- For the reset value specific to your configuration, see coreConsultant ComponentRegisters report.

Register definitions for each component memory map.

Table 1-4 Registers for the DWC_usb3_map Memory Map

Register	Offset	Description
USB 3.0 eXtensible Host Controller Capability Register Block DWC_usb3_block_eXtensible_Host_Cntrl_Cap_Regs		
“CAPLENGTH” on page 27	0x0	Capability Registers Length Host Controller Operational Registers = Base address + CAPLENGTH where...

Register	Offset	Description
“HCSPARAMS1” on page 28	0x4	Structural Parameters 1 Register For register definitions, refer to the xHCI specification.
“HCSPARAMS2” on page 30	0x8	Structural Parameters 2 Register For register definitions, refer to the xHCI specification.
“HCSPARAMS3” on page 32	0xc	Structural Parameters 3 Register For register definitions, refer to the xHCI specification.
“HCCPARAMS1” on page 33	0x10	Capability Parameters 1 Register For register definitions, refer to the xHCI specification.
“DBOFF” on page 37	0x14	Doorbell Offset Register For register definitions, refer to the xHCI specification.
“RTSOFF” on page 38	0x18	Runtime Register Space Offset Register
“HCCPARAMS2” on page 39	0x1c	Host Controller Capability Parameters 2 For register definitions, refer to the xHCI...
USB 3.0 Global Register Block DWC_usb3_block_gbl		
“GSBUSCFG0” on page 42	0xc100	Global SoC Bus Configuration Register 0 This register configures system bus DMA options for the...
“GSBUSCFG1” on page 47	0xc104	Global SoC Bus Configuration Register 1 xHCI Register Power-On Value: If you are using a standard...
“GTXTHRCFG” on page 49	0xc108	Global Tx Threshold Control Register For more information on - Using this register, refer to "Packet...
“GRXTHRCFG” on page 53	0xc10c	Global Rx Threshold Control Register In a normal case, a Tx burst starts as soon as one packet...
“GCTL” on page 58	0xc110	Global Core Control Register Refer to <workspace>/src/DWC_usb3_params.v for details on `DWC_USB3_GCTL_INIT. Note:...
“GPMSTS” on page 69	0xc114	Global Power Management Status Register This debug register gives information on which event caused...
“GSTS” on page 71	0xc118	Global Status Register
“GUCTL1” on page 74	0xc11c	Global User Control Register 1
“GSNPSID” on page 87	0xc120	Global Synopsys ID Register This is a read-only register that contains the release number of the...
“GGPIO” on page 88	0xc124	Global General Purpose Input/Output Register The application can use this register for general...
“GUID” on page 89	0xc128	Global User ID Register This is a read/write register containing the User ID. The power-on value...

Register	Offset	Description
“GUCTL” on page 90	0xc12c	Global User Control Register: This register provides a few options for the software to control...
“GBUSERRADDR” on page 96	0xc130	Global Soc Bus Error Address Register When the AHB or AXI Master Bus returns an "Error" response,...
“GBUSERRADDRLO” on page 97	0xc130	Gobal SoC Bus Error Address Register - Low This is an alternate register for the GBUSERRADDR...
“GBUSERRADDRHI” on page 98	0xc134	Gobal SoC Bus Error Address Register - High This is an alternate register for the GBUSERRADDR...
“GPRTBIMAP” on page 99	0xc138	Global SS Port to Bus Instance Mapping Register This register specifies the SuperSpeed USB instance...
“GPRTBIMAPLO” on page 103	0xc138	Global SS Port to Bus Instance Mapping Register - Low This is an alternate register for the GPRTBIMAP...
“GPRTBIMAPHI” on page 105	0xc13c	Global SS Port to Bus Instance Mapping Register - High This is an alternate register for the GPRTBIMAP...
“GHWPARAMS0” on page 107	0xc140	Global Hardware Parameters Register 0 This register contains the hardware configuration options...
“GHWPARAMS1” on page 109	0xc144	Global Hardware Parameters Register 1 This register contains the hardware configuration options...
“GHWPARAMS2” on page 112	0xc148	Global Hardware Parameters Register 2 This register contains the hardware configuration options...
“GHWPARAMS3” on page 113	0xc14c	Global Hardware Parameters Register 3 This register contains the hardware configuration options...
“GHWPARAMS4” on page 116	0xc150	Global Hardware Parameters Register 4 This register contains the hardware configuration options...
“GHWPARAMS5” on page 120	0xc154	Global Hardware Parameters Register 5 This register contains the hardware configuration options...
“GHWPARAMS6” on page 122	0xc158	Global Hardware Parameters Register 6 This register contains the hardware configuration options...
“GHWPARAMS7” on page 125	0xc15c	Global Hardware Parameters Register 7 This register contains the hardware configuration options...
“GDBGFIFOSPACE” on page 126	0xc160	Global Debug Queue/FIFO Space Available Register Bit Bash test should not be done on this debug...
“GDBGLTSSM” on page 128	0xc164	Global Debug LTSSM Register In multi-port host configuration, the port-number is defined by Port-Select[3:0]...
“GDBGLNMCC” on page 133	0xc168	Global Debug LNMCC Register Bit Bash test should not be done on this debug register.

Register	Offset	Description
“GDBGBMU” on page 134	0xc16c	Global Debug BMU Register Bit Bash test should not be done on this debug register.
“GDBGLSPMUX_DEV” on page 137	0xc170	Global Debug LSP MUX Register - Device This register is for internal use only. If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT...
“GDBGLSPMUX_HST” on page 135	0xc170	Global Debug LSP MUX Register - Host This register is for internal use only. If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT...
“GDBGLSP” on page 139	0xc174	Global Debug LSP Register This register is for internal debug purposes only. This register is...
“GDBGEPINFO0” on page 140	0xc178	Global Debug Endpoint Information Register 0 This register is for internal use only. If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT...
“GDBGEPINFO1” on page 141	0xc17c	Global Debug Endpoint Information Register 1 This register is for internal use only. If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT...
“GPRTBIMAP_HSLO” on page 142	0xc180	Global High-Speed Port to Bus Instance Mapping Register - Low This is an alternate register for...
“GPRTBIMAP_HS” on page 144	0xc180	Global High-Speed Port to Bus Instance Mapping Register This register specifies the High Speed...
“GPRTBIMAP_HSHI” on page 148	0xc184	Global High-Speed Port to Bus Instance Mapping Register - High This is an alternate register for...
“GPRTBIMAP_FS” on page 150	0xc188	Global Full-Speed Port to Bus Instance Mapping Register This register specifies the Full Speed...
“GPRTBIMAP_FSLO” on page 154	0xc188	Global Full-Speed Port to Bus Instance Mapping Register - Low This is an alternate register for...
“GPRTBIMAP_FSHI” on page 156	0xc18c	Global Full-Speed Port to Bus Instance Mapping Register - High This is an alternate register for...
“GERRINJCTL_1” on page 158	0xc194	Global Error Injection 1 Control Register This register is reserved for future use.
“GERRINJCTL_2” on page 159	0xc198	Global Error Injection 2 Control Register This register is reserved for future use.
“GUCTL2” on page 160	0xc19c	Global User Control Register 2: This register provides a few options for the software to control...
“GUSB2PHYCFG(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)” on page 163	0xc200 + (*i* * 0x4)	Global USB2 PHY Configuration Register The application must program this register before starting...

Register	Offset	Description
"GUSB2I2CCTL(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)" on page 174	0xc240 + (*i* * 0x4)	Reserved Register
"GUSB2PHYACC_UTMI(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)" on page 175	0xc280 + (*i* * 0x4)	Global USB 2.0 UTMI PHY Vendor Control Register The application uses this register to access PHY...
"GUSB2PHYACC_ULPI(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)" on page 178	0xc280 + (*i* * 0x4)	Global USB 2.0 ULPI PHY Vendor Control Register The application uses this register to access the...
"GUSB3PIPECTL(#n) (for n = 0; n <= DWC_USB3_NUM_U3_ROOT_PORTS-1)" on page 181	0xc2c0 + (*i* * 0x4)	Global USB 3.0 PIPE Control Register The application uses this register to configure the USB3 PHY...
"GTXFIFOSIZ(#n) (for n = 0; n <= 31)" on page 189	0xc300 + (n*0x4)	Global Transmit FIFO Size Register This register specifies the RAM start address and depth (both...
"GRXFIFOSIZ(#n) (for n = 0; n <= 31)" on page 191	0xc380 + (n*0x4)	Global Receive FIFO Size Register This register specifies the RAM start address and depth (both...
"GEVNTADR(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)" on page 193	0xc400 + (*i* * 0x10)	Global Event Buffer Address Register This register holds the Event Buffer DMA Address pointer....
"GEVNTADRLO(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)" on page 194	0xc400 + (*i* * 0x10)	Global Event Buffer Address (Low) Register This is an alternate register for the GEVNTADRn...
"GEVNTADRHI(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)" on page 195	0xc404 + (*i* * 0x10)	Global Event Buffer Address (High) Register This is an alternate register for the GEVNTADRn...
"GEVNTSIZ(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)" on page 196	0xc408 + (*i* * 0x10)	Global Event Buffer Size Register This register holds the Event Buffer Size and the Event Interrupt...
"GEVNTCOUNT(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)" on page 197	0xc40c + (*i* * 0x10)	Global Event Buffer Count Register This register holds the number of valid bytes in the Event Buffer....
"GHWPARAMS8" on page 199	0xc600	Global Hardware Parameters Register 8 This register contains the hardware configuration options...
"GUCTL3" on page 200	0xc60c	Global User Control Register 3 This register provides a few options for the software to control...
"GTXFIFOPRIDEV" on page 201	0xc610	Global Device TX FIFO DMA Priority Register This register specifies the relative DMA priority level...
"GTXFIFOPRIHST" on page 203	0xc618	Global Host TX FIFO DMA Priority Register This register specifies the relative DMA priority level...

Register	Offset	Description
"GRXFIFOPRIHST" on page 205	0xc61c	Global Host RX FIFO DMA Priority Register This register specifies the relative DMA priority level...
"GFIFOPRIDBC" on page 207	0xc620	Global Host Debug Capability DMA Priority Register This register specifies the relative priority...
"GDMAHLRATIO" on page 208	0xc624	Global Host FIFO DMA High-Low Priority Ratio Register This register specifies the relative priority...
"GFLADJ" on page 210	0xc630	Global Frame Length Adjustment Register This register provides options for the software to control...
"GUSB2RHBCTL(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)" on page 215	0xc640 + (*i* * 0x4)	Global USB 2.0 Root Hub Control Register The application must program this register before starting...
USB 3.0 Device Register Block DWC_usb3_block_dev		
"DCFG" on page 217	0xc700	Device Configuration Register. This register configures the controller in Device mode after power-on...
"DCTL" on page 221	0xc704	Device Control Register Note: When Hibernation is not enabled using GCTL.GblHibernationEn field, ...
"DEVTEN" on page 231	0xc708	Device Event Enable Register This register controls the generation of device-specific events (see...
"DSTS" on page 235	0xc70c	Device Status Register This register indicates the status of the device controller with respect...
"DGCMDPAR" on page 241	0xc710	Device Generic Command Parameter Register This register indicates the device command parameter....
"DGCMD" on page 242	0xc714	Device Generic Command Register This register enables software to program the controller using...
"DALEPENA" on page 244	0xc720	Device Active USB Endpoint Enable Register. This register indicates whether a USB endpoint is...
"Rsvd[0:31]" on page 245	0xc724 + (*i* * 0x4)	Reserved
"DEPCMDPAR2[0:7]" on page 246	0xc800 + (*i* * 0x10)	Device Physical Endpoint-n Command Parameter 2 Register (DEPCMDPAR2n) This register indicates the...
"DEPCMDPAR1[0:7]" on page 247	0xc804 + (*i* * 0x10)	Device Physical Endpoint-n Command Parameter 1 Register (DEPCMDPAR1n)
"DEPCMDPAR0[0:7]" on page 248	0xc808 + (*i* * 0x10)	Device Physical Endpoint-n Command Parameter 0 Register (DEPCMDPAR0n)
"DEPCMD[0:7]" on page 249	0xc80c + (*i* * 0x10)	Device Physical Endpoint-n Command Register This register enables software to issue physical endpoint-specific...

Register	Offset	Description
"DEV_IMOD[0]" on page 253	0xca00 + (*i* * 0x4)	Device Interrupt Moderation Register (DEV_IMOD) This register controls the Interrupt Moderation...
USB 3.0 BC Register Block DWC_usb3_block_bc		
"BCFG" on page 256	0xcc30	BC Configuration Register
"BCEVT" on page 257	0xcc38	BC Event Register Writing 1 to the info bit in this register clears the register bit and the associated...
"BCEVTEN" on page 259	0xcc3c	BC Event Enable Register
USB 3.0 ECC Register Block DWC_usb3_block_ecc		
"BRSERRCNT" on page 270	0xd850	Block RAM Single Bit Error Count. 8 bit count for each of the 3 RAMs.
"BRMERRCNT" on page 271	0xd854	Block RAM Multiple Bit Error Count. 8 bit count for each of the 3 RAMs.
"BRMECCERR" on page 272	0xd858	Block RAM ECC Error Vector Register
"BRERRCTL" on page 274	0xd85c	Block RAM ECC Error Control Register
"BRAM0ADDRERR" on page 275	0xd860	Block RAM Address of RAM0 having uncorrectable error
"BRAM1ADDRERR" on page 276	0xd864	Block RAM Address of RAM1 having uncorrectable error
"BRAM2ADDRERR" on page 277	0xd868	Block RAM Address of RAM2 having uncorrectable error
USB 3.0 Link Register Block DWC_usb3_block_link		
"LU1LFPSRXTIM[0]" on page 261	0xd000 + (*i* * 0x80)	U1/U2 LFPS Rx Timer Register - Link Layer Register for U1/U2 LFPS RX Timers. - This register...
"LINK_SETTINGS[0]" on page 263	0xd020 + (*i* * 0x80)	Link Setting Register - Link Layer User Control Register for enabling Link/PHY-specific options....
"LLUCTL[0]" on page 265	0xd024 + (*i* * 0x80)	Link User Control Register - Link Layer User Control Register for enabling Link/PHY-specific options....
"LPTMDPDELAY[0]" on page 268	0xd028 + (*i* * 0x80)	Link Datapath Delay Register - Link Layer Timer Register for P3CPM/P4 Residency. - This register...
USB 3.0 Debug Register Block DWC_usb3_block_debug		
"BU3RHBDBG(#n) (for n = 0; n <= DWC_USB3_NUM_U3_ROOT_PORTS-1)" on page 279	0xd800 + (*i* * 0x4)	U3 Root Hub Debug Register

1.1 DWC_usb3_block_eXtensible_Host_Cntrl_Cap_Regs Registers

1.1.1 CAPLENGTH

- **Description:** Capability Registers Length
  Host Controller Operational Registers = Base address + CAPLENGTH
  where CAPLENGTH is `DWC_USB3_HOST_CAP_REG_LEN whose default value is 20h.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

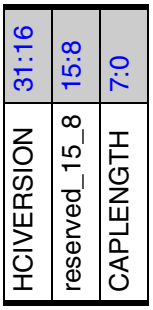


Table 1-5 Fields for Register: CAPLENGTH

Bits	Name	Memory Access	Description
31:16	HCIVERSION	R	HC Interface Version Number (HCIVERSION) **Value After Reset:** DWC_USB3_HC_HCIVERSION **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:8	reserved_15_8	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead
7:0	CAPLENGTH	R	Capability Registers Length (CAPLENGTH) **Value After Reset:** DWC_USB3_HC_CAPLENGTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.1.2 HCSPARAMS1

- **Description:** Structural Parameters 1 Register
  For register definitions, refer to the xHCI specification.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** Always

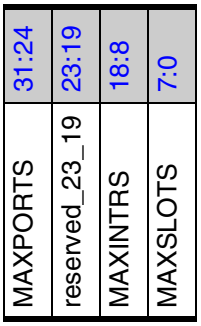


Table 1-6 Fields for Register: HCSPARAMS1

Bits	Name	Memory Access	Description
31:24	MAXPORTS	R	Number of Ports (MaxPorts) ■ Number of ports implemented is defined by the parameter (`DWC_USB3_HOST_NUM_U2_ROOT_PORTS + `DWC_USB3_HOST_NUM_U3_ROOT_PORTS) ■ Number of ports enabled is controlled by the controller input signals host_num_u2_port[3:0]+host_num_u3_port[3:0] **Note:** In USB 2.0-only mode, the host_num_u3_port signal is zero. **Value After Reset:** expr [DWC_USB3_HOST_NUM_U2_ROOT_PORTS_PIN] + [DWC_USB3_HOST_NUM_U3_ROOT_PORTS_PIN] **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead
23:19	reserved_23_19	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1f **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
18:8	MAXINTRS	R	Number of Interrupters (MaxIntrs) Defined by the configurable parameter `DWC_USB3_HOST_NUM_INTERRUPTER_SUPT **Value After Reset:** DWC_USB3_HC_MAXINTRS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:0	MAXSLOTS	R	Number of device slots (MaxSlots) Defined by configurable parameter `DWC_USB3_NUM_DEVICE_SUPT **Value After Reset:** DWC_USB3_HC_MAXSLOTS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.1.3 HCSPARAMS2

- **Description:** Structural Parameters 2 Register
  For register definitions, refer to the xHCI specification.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

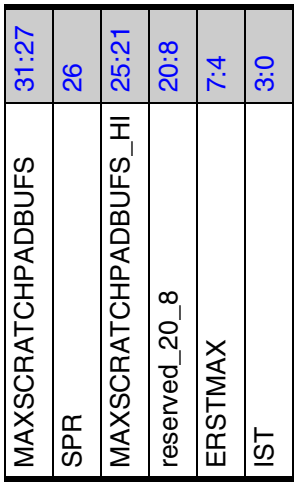


Table 1-7 Fields for Register: HCSPARAMS2

Bits	Name	Memory Access	Description
31:27	MAXSCRATCHPADBUFS	R	Max Scratchpad Bufs Lo The value is calculated based on chosen configuration parameter values. Possible values are 1-4. **Value After Reset:** HCACHE_NUM_SP_BUFS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
26	SPR	R	Scratchpad Restore (SPR) **Value After Reset:** DWC_USB3_HC_SPR **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
25:21	MAXSCRATCHPADBUFS_HI	R	Max Scratchpad Bufs HI The controller automatically updates this field. **Value After Reset:** expr [HCACHE_NUM_SP_BUFS] / 32 **Exists:** DWC_USB3_HC_HCIVERSION_10 == 1 ? 1 : 0 **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
20:8	reserved_20_8	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1fff **Write Constraint:** writeAsRead
7:4	ERSTMAX	R	Event Ring Segment Table Max (ERST Max) **Value After Reset:** DWC_USB3_HC_ERSTMAX **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	IST	R	Isochronous Scheduling Threshold (IST) **Value After Reset:** DWC_USB3_HC_IST **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.1.4 HCSPARAMS3

- **Description:** Structural Parameters 3 Register
  For register definitions, refer to the xHCI specification.
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** Always

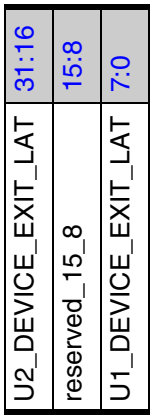
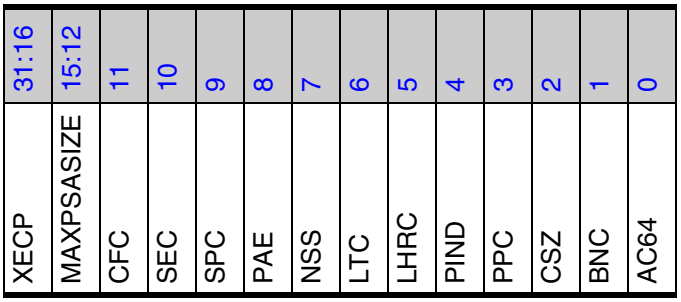


Table 1-8 Fields for Register: HCSPARAMS3

Bits	Name	Memory Access	Description
31:16	U2_DEVICE_EXIT_LAT	R	U2 Device Exit Latency **Value After Reset:** DWC_USB3_HC_U2_DEVICE_EXIT_LATENCY **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:8	reserved_15_8	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead
7:0	U1_DEVICE_EXIT_LAT	R	U1 Device Exit Latency **Value After Reset:** DWC_USB3_HC_U1_DEVICE_EXIT_LATENCY **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.1.5 HCCPARAMS1

- **Description:** Capability Parameters 1 Register
  For register definitions, refer to the xHCI specification.
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** Always



31:16	15:12	11	10	9	8	7	6	5	4	3	2	1	0
XECP	MAXPSASIZE	CFC	SEC	SPC	PAE	NSS	LTC	LHRC	PIND	PPC	CSZ	BNC	AC64

Table 1-9 Fields for Register: HCCPARAMS1

Bits	Name	Memory Access	Description
31:16	XECP	R	xHCI Extended Capabilities Pointer (xECP) Based on configuration, controller automatically updates it. Refer to <workspace>/src/DWC_usb3_params.v for details on DWC_USB3_HC_XECP. **Value After Reset:** DWC_USB3_HC_XECP **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	MAXPSASIZE	R	Maximum Primary Stream Array Size (MaxPSASize) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0*. **Value After Reset:** DWC_USB3_HC_MAXPSASIZE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11	CFC	R	Contiguous Frame ID Capability (CFC) **Value After Reset:** DWC_USB3_HC_CFC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
10	SEC	R	Stopped EDLTA Capability (SEC) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0*. **Value After Reset:** DWC_USB3_HC_SEC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
9	SPC	R	Short Packet Capability (SPC) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0*. **Value After Reset:** DWC_USB3_HC_SPC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
8	PAE	R	Parse All Event Data (PAE) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0*. **Value After Reset:** DWC_USB3_HC_PAE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7	NSS	R	No Secondary SID Support (NSS) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0*. **Value After Reset:** DWC_USB3_HC_NSS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
6	LTC	R	Latency Tolerance Messaging Capability (LTC) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0*. **Value After Reset:** DWC_USB3_HC_LTC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
5	LHRC	R	Light HC Reset Capability For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_LHRC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
4	PIND	R	Port Indicators (PIND) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_PIND **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3	PPC	R	Port Power Control For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_PPC **Exists:** Always **Testable:** untestable **Volatile:** true **Write Constraint:** writeAsRead
2	CSZ	R	Context Size (CSZ) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_CSZ **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
1	BNC	R	BW Negotiation Capability (BNC) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_BNC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
0	AC64	R	64-bit Addressing Capability (AC64) For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_AC64 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.1.6 DBOFF

- **Description:** Doorbell Offset Register
  For register definitions, refer to the xHCI specification.
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** Always

Table 1-10 Fields for Register: DBOFF

Bits	Name	Memory Access	Description
31:2	DOORBELL_ARRAY_OFFSET	R	Doorbell Array Offset - RO Based on configuration, the controller automatically updates it. For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** expr [DWC_USB3_HC_DBOFF] / 4 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
1:0	reserved_1_0	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead

1.1.7 RTSOFF

- **Description:** Runtime Register Space Offset Register
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** Always

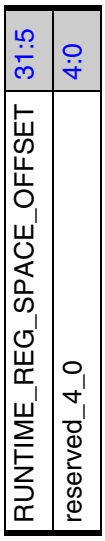


Table 1-11 Fields for Register: RTSOFF

Bits	Name	Memory Access	Description
31:5	RUNTIME_REG_SPACE_OFFSET	R	Runtime Register Space Offset Based on configuration, the controller automatically updates it. For a description of this standard USB register field, see the *eXtensible Host Controller Interface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** expr [DWC_USB3_HC_RTSOFF] / 32 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
4:0	reserved_4_0	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1f **Write Constraint:** writeAsRead

1.1.8 HCCPARAMS2

- **Description:** Host Controller Capability Parameters 2
  For register definitions, refer to the xHCI specification.
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** Always

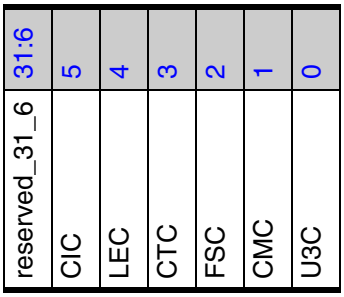


Table 1-12 Fields for Register: HCCPARAMS2

Bits	Name	Memory Access	Description
31:6	reserved_31_6	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
5	CIC	R	Configuration Information Capability (CIC) For a description of this standard USB register field, see the *eXtensible Host Controller I nterface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_CIC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
4	LEC	R	Large ESIT Payload Capability (LEC) For a description of this standard USB register field, see the *eXtensible Host Controller I nterface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_LEC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
3	CTC	R	Compliance Transition Capability (CTC) For a description of this standard USB register field, see the *eXtensible Host Controller I nterface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_CTC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
2	FSC	R	Force Save Context Capability (FSC) For a description of this standard USB register field, see the *eXtensible Host Controller I nterface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_FSC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
1	CMC	R	Configure Endpoint Command Max Exit Latency Too Large Capability (CMC) For a description of this standard USB register field, see the *eXtensible Host Controller I nterface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_CMC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
0	U3C	R	U3 Entry Capability (U3C) For a description of this standard USB register field, see the *eXtensible Host Controller I nterface for Universal Serial Bus (USB) Specification 3.0.* **Value After Reset:** DWC_USB3_HC_U3C **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2 DWC_usb3_block_gbl Registers

1.2.1 GSBUSCFG0

- **Description:** Global SoC Bus Configuration Register 0
  This register configures system bus DMA options for the master bus, which may be configured as AHB, AXI, or Native. Options include burst length and cache type (bufferable/posted, cacheable/snoop, and so on). The application can program this register upon power-on, or a change in mode of operation after the DMA engine is halted.
  xHCI Register Power-On Value:
  If you are using a standard xHCI host driver, make sure to set the register's power-on value during coreConsultant configuration (DWC_USB3_GSBUSCFG0_INIT parameter) because the standard xHCI driver does not access this register.
  For more details on this register, refer to the following sections:
  - "Usage of Global SoC Bus Configuration Register 0 (GSBUSCFG0)" section in the Programming Guide
  - "System Bus Interface" section in the Databook
- **Size:** 32 bits
- **Offset:** 0xc100
- **Exists:** Always

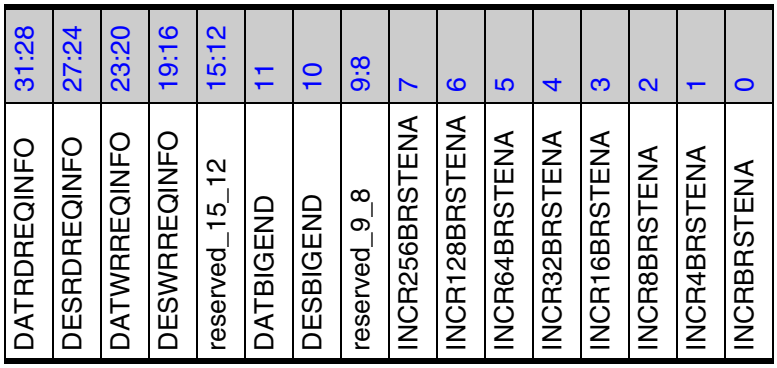


Table 1-13 Fields for Register: GSBUSCFG0

Bits	Name	Memory Access	Description
31:28	DATRDREQINFO	R/W	DATRDREQINFO AHB-prot/AXI-cache/OCP-ReqInfo for Data Read (DatRdReqInfo) Input to BUS-GM. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 28 4] **Exists:** Always

Bits	Name	Memory Access	Description
27:24	DESRDREQINFO	R/W	DESRDREQINFO AHB-prot/AXI-cache/OCP-ReqInfo for Descriptor Read (DesRdReqInfo). Input to BUS-GM. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 24 4] **Exists:** Always
23:20	DATWRREQINFO	R/W	DATWRREQINFO AHB-prot/AXI-cache/OCP-ReqInfo for Data Write (DatWrReqInfo). Input to BUS-GM. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 20 4] **Exists:** Always
19:16	DESWRREQINFO	R/W	DESWRREQINFO AHB-prot/AXI-cache/OCP-ReqInfo for Descriptor Write (DesWrReqInfo) Input to BUS-GM. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 16 4] **Exists:** Always
15:12	reserved_15_12	R	Reserved for future use **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 12 4] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
11	DATBIGEND	R/W	Data Access is Big Endian This bit controls the endian mode for data accesses. ■ Little-endian (default); ■ Big-endian; In big-endian mode, DMA access (both read and write) for packet data a Byte Invariant Big-Endian mode (see "Little-Endian and Big-Endian" section in the User Guide). **Note:** Since AXI requires byte invariant endianness, setting DescBigend and DatBigEnd to one causes an address invariant transform to be applied, which is not appropriate. See section 9.3 and 9.4 of the *AMBA AXI Specification*. Hence for an AXI master (DWC_USB3_MBUS_TYPE=1), this bit must be set to zero. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 11 1] **Exists:** Always

Bits	Name	Memory Access	Description
10	DESBIGEND	R/W	Descriptor Access is Big Endian This bit controls the endian mode for descriptor accesses. ■ Little-endian (default) ■ Big-endian In big-endian mode, DMA access (both read and write) for descriptors uses a Byte Invariant Big-Endian mode (see "Little-Endian and Big-Endian" section in the User Guide. Data is considered as 'embedded data' in the descriptors in the following cases: ■ Device mode: The buffer pointer of a Setup TRB points to the Setup TRB itself. ■ Host mode: The Immediate Data (IDT) bit in a Transfer TRB is set to 1. In device mode, if the system uses different endian modes for descriptor and data, software must not use 'embedded' data. In host mode, if the system uses different endian modes for data and descriptors, the controller treats 'embedded data' as descriptor (not as data) in terms of endian mode handling. If this is not the expectation of the system, the software must manipulate the 'embedded data' accordingly. Note: Since AXI requires byte invariant endianness, setting DescBigend and DatBigEnd to one causes an address invariant transform to be applied, which is not appropriate. See section 9.3 and 9.4 of the AMBA AXI Specification. Hence for an AXI master (DWC_USB3_MBUS_TYPE=1), this bit must be set to zero. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 10 1] **Exists:** Always
9:8	reserved_9_8	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 8 2] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead
7	INCR256BRSTENA	R/W	INCR256 Burst Type Enable Input to BUS-GM. For the AHB/AXI configuration, if software set this bit to 1, the AHB/AXI master uses INCR to do the 256-beat burst. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 7 1] **Exists:** Always

Bits	Name	Memory Access	Description
6	INCR128BRSTENA	R/W	INCR128 Burst Type Enable Input to BUS-GM; For the AHB/AXI configuration, if software set this bit to 1, the AHB/AXI master uses INCR to do the 128-beat burst. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 6 1] **Exists:** Always
5	INCR64BRSTENA	R/W	INCR64 Burst Type Enable ■ Input to BUS-GM; For the AHB/AXI configuration, if software set this bit to 1, the AHB/AXI master uses INCR to do the 64-beat burst. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 5 1] **Exists:** Always
4	INCR32BRSTENA	R/W	INCR32 Burst Type Enable Input to BUS-GM; For the AHB/AXI configuration, if software set this bit to 1, the AHB/AXI master uses INCR to do the 32-beat burst. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 4 1] **Exists:** Always
3	INCR16BRSTENA	R/W	INCR16 Burst Type Enable Input to BUS-GM. For the AHB/AXI configuration, if software set this bit to '1', the AHB/AXI master uses INCR to do the 16-beat burst. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 3 1] **Exists:** Always
2	INCR8BRSTENA	R/W	INCR8 Burst Type Enable Input to BUS-GM; For the AHB/AXI configuration, if software set this bit to "1", the AHB/AXI master uses INCR to do the 8-beat burst. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 2 1] **Exists:** Always

Bits	Name	Memory Access	Description
1	INCR4BRSTENA	R/W	INCR4 Burst Type Enable Input to BUS-GM; For the AXI configuration, when this bit is enabled the controller is allowed to do bursts of beat length 1, 2, and 4. It is highly recommended that this bit is enabled to prevent descriptor reads and writes from being broken up into separate transfers. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 1 1] **Exists:** Always
0	INCRBRSTENA	R/W	Undefined Length INCR Burst Type Enable (INCRBrstEna) Input to BUS-GM; This bit determines the set of burst lengths the master interface uses. It works in conjunction with the GSBUSCFG0[7:1] enables (INCR256/128/64/32/16/8/4). *0: INCRX burst mode* HBURST (for AHB configurations) and ARLEN/AWLEN (for AXI configurations) do not use INCR except in case of non-aligned burst transfers. In the case of address-aligned transfers, they use only the following burst lengths: ■ 1 ■ 2, 4 (if GSBUSCFG0.INCR4BrstEna = 1) ■ 8 (if GSBUSCFG0.INCR8BrstEna = 1) ■ 16 (if GSBUSCFG0.INCR16BrstEna = 1) ■ 32 (if GSBUSCFG0.INCR32BrstEna = 1) ■ 64 (if GSBUSCFG0.INCR64BrstEna = 1) ■ 128 (if GSBUSCFG0.INCR128BrstEna = 1) ■ 256 (if GSBUSCFG0.INCR256BrstEna = 1) **Note:** ■ In case of non-address-aligned transfers, INCR may get generated at the beginning and end of the transfers to align the address boundaries, even though INCR is disabled. ■ In AHB mode, if INCRX burst mode is enabled, but none of the supported INCRx bursts bits are enabled, then the controller will perform (undefined length) INCR bursts. *1: INCR (undefined length) burst mode* ■ AHB configurations: HBURST uses SINGLE or INCR of any length with handling 1KB boundary breakup. ■ AXI configurations: ARLEN/AWLEN uses any length less than or equal to the largest-enabled burst length of INCR32/64/128/256. For cache line-aligned applications, this bit is typically set to 0 to ensure that the master interface uses only power-of-2 burst lengths (as enabled via GSBUSCFG0[7:0]). **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG0_INIT" 0 1] **Exists:** Always

1.2.2 GSBUSCFG1

- **Description:** Global SoC Bus Configuration Register 1
  xHCI Register Power-On Value:
  If you are using a standard xHCI host driver, make sure to set the register's power-on value during coreConsultant configuration (DWC_USB3_GSBUSCFG1_INIT parameter) because the standard xHCI driver does not access this register.
  For more details on this register, refer to "System Bus Interface" section in the Databook.
- **Size:** 32 bits
- **Offset:** 0xc104
- **Exists:** Always

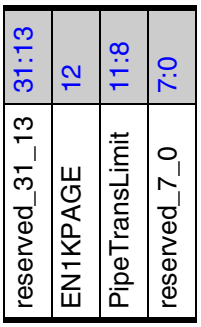


Table 1-14 Fields for Register: GSBUSCFG1

Bits	Name	Memory Access	Description
31:13	reserved_31_13	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG1_INIT" 13 19] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7ffff **Write Constraint:** writeAsRead
12	EN1KPAGE	R/W	1k Page Boundary Enable By default (this bit is disabled) the AXI breaks transfers at the 4k page boundary. When this bit is enabled, the AXI master (DMA data) breaks transfers at the 1k page boundary. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG1_INIT" 12 1] **Exists:** Always

Bits	Name	Memory Access	Description
11:8	PipeTransLimit	R/W	AXI Pipelined Transfers Burst Request Limit The field controls the number of outstanding pipelined transfer requests the AXI master pushes to the AXI slave. When the AXI master reaches this limit, it does not make any more requests on the AXI ARADDR and AWADDR buses until the associated data phases complete. This field is encoded as follows: ■ 'h0: 1 request ■ 'h1: 2 requests ■ 'h2: 3 requests ■ 'h3: 4 requests ■ ... ■ 'hF: 16 requests **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG1_INIT" 8 4] **Exists:** Always
7:0	reserved_7_0	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GSBUSCFG1_INIT" 0 8] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead

1.2.3 GTXTHRCFG

- **Description:** Global Tx Threshold Control Register
  For more information on
  - Using this register, refer to "Packet Threshold and Burst Features for High Latency Systems" section in the Databook.
  - Selecting values for the fields of this register, see the "TX/RX Data FIFO Sizes and TX/RX Threshold Control Register Settings" section in the User Guide.

  Note:
  - GTXTHRCFG register is not applicable for Debug Target.
  - GTXTHRCFG register is not applicable in USB 2.0-only mode.
- **Size:** 32 bits
- **Offset:** 0xc108
- **Exists:** Always

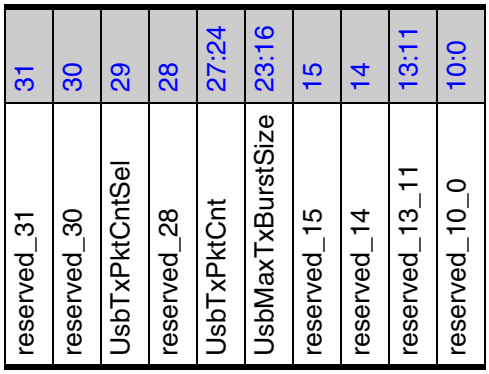


Table 1-15 Fields for Register: GTXTHRCFG

Bits	Name	Memory Access	Description
31	reserved_31	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 31 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
30	reserved_30	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 30 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
29	UsbTxPktCntSel	R/W	USB Transmit Packet Count Enable This field enables/disables the USB transmission multi-packet thresholding: ■ 0: USB transmission multi-packet thresholding is disabled; the controller can start transmission on the USB after the entire (one full) packet has been fetched into the corresponding TXFIFO. ■ 1: USB transmission multi-packet thresholding is enabled. The controller can only start transmission on the USB after USB Transmit Packet Count amount of packets for the USB transaction (burst) are already in the corresponding TXFIFO. This mode is valid in both host and device modes. It is only used for SuperSpeed operation. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 29 1] **Exists:** Always **Testable:** unconstrained
28	reserved_28	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 28 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
27:24	UsbTxPktCnt	R/W	USB Transmit Packet Count This field specifies the number of packets that must be in the TXFIFO before the controller can start transmission for the corresponding USB transaction (burst). This field is only valid when the USB Transmit Packet Count Enable field is set to one. Valid values are from 1 to 15. **Note:** ■ In device mode, if device controller does not have the TRBs for the number of packets or if it cannot fetch the TRBs because of high latency or switching between other endpoints, then it does not wait for the threshold number of packets. The threshold number of packets will be honored only when the TRBs are available in the controller for the number of packets before it starts the data fetch. ■ This field must be less than or equal to the USB Maximum TX Burst Size field. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 24 4] **Exists:** Always **Testable:** unconstrained

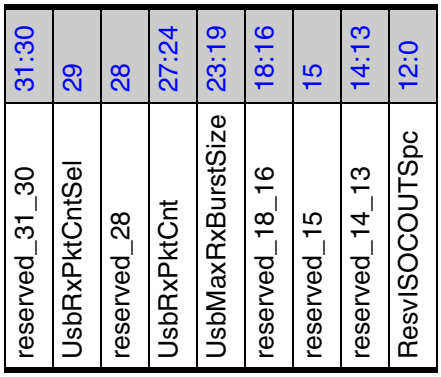
Bits	Name	Memory Access	Description
23:16	UsbMaxTxBurstSize	R/W	USB Maximum TX Burst Size When UsbTxPktCntSel is one, this field specifies the Maximum Bulk OUT burst the controller can do. When the system bus is slower than the USB, TX FIFO can underrun during a long burst. User can program a smaller value to this field to limit the TX burst size that the controller can do. Host mode: It only applies to SS Bulk, Isochronous, and Interrupt OUT endpoints. Device mode: This value is not used in device mode, but users need to program a value when using the TX threshold feature to make sure that the value programmed in UsbTxPktCnt is less than this value. Valid values are from 1 to 16. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 16 8] **Exists:** Always **Testable:** unconstrained
15	reserved_15	R	Reserved_15 **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 15 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
14	reserved_14	R	Reserved1(Rsvd/Rs) Register field must write only 0 by the application. The read value must be treated as X (unknown). **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 14 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
13:11	reserved_13_11	R	Reserved (Rsvd/Rs) The register field must write only 0 by the application. The read value must be treated as X (unknown). **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 11 3] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
10:0	reserved_10_0	R	Reserved for future use **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GTXTHRCFG_INIT" 0 11] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7ff **Write Constraint:** writeAsRead

1.2.4 GRXTHRCFG

- **Description:** Global Rx Threshold Control Register
  In a normal case, a Tx burst starts as soon as one packet is prefetched; an Rx burst starts as soon as 1-packet space is available. This works well as long as the system bus is faster than the USB 3.0 bus (a 1024-bytes packet takes ~2.2 microseconds on the USB bus in SS mode).
  If the system bus latency is larger than 2.2 microseconds to access a 1024-byte packet, then starting a burst on 1-packet condition leads to an early abort of the burst causing unnecessary performance reduction.
  To avoid underrun and overrun during the burst, in a high-latency bus system (like USB), threshold and burst size control is provided through GTXTHRCFG and GRXTHRCFG registers. Bit [29] of the GTXTHRCFG and GRXTHRCFG registers enables this feature.
  For more information on
  - Using this register, refer to "Packet Threshold and Burst Features for High Latency Systems" section in the Databook.
  - Selecting values for the fields of this register, see the "TX/RX Data FIFO Sizes and TX/RX Threshold Control Register Settings" section in the User Guide.

  Note:
  - GRXTHRCFG register is not applicable for Debug Target.
  - There is an issue when ACK TP with NumP=0 followed by ACK TP with NumP=1 without ERDY TP sent by the device controller during a burst bulk OUT transfer. This may cause third-party USB 3.0 host controllers to keep waiting for the ERDY TP.
    The USB 3.0 specification states that "When an endpoint is not in a flow control condition, it shall not send an ERDY TP unless the endpoint is a Bulk endpoint that supports streams." In this case, after the device sent the ACK TP (nump=1), the endpoint was not in the flow control, so it did not send an ERDY.
    The device would have sent ERDY if the next OUT packet was not received. When the next OUT packet was received, at that time there was enough buffer space to accept it, so the device accepted the packet by informing host that it is not no longer in the flow control. The Host should wait for the responses for all the OUT packets to return and then decide if the endpoint is still in flow control or not.
    The USB 3.1 specification supersedes all the USB 3.0 specification. The errata states that "If the host continues, or resumes, transactions to an endpoint, the endpoint shall re-evaluate its flow control state and respond appropriately." However, there are no ECNs on the USB 3.0 for this issue.
    To work around this issue, the Global Rx Threshold mode must be disable by setting GRXTHRCFG.UsbRxPktCntSel=0. Instead, software can program the DCFG.NUMP mode (where fixed NUMP is transmitted always) instead of the RX threshold based nump mode to prevent the device from sending ACK TP with NumP=0. The NUMP in the ACK TP is the minimum value of (DCFG.NUMP, bMaxBurstSize) for each endpoint.
- **Size:** 32 bits
- **Offset:** 0xc10c
- **Exists:** Always



31:30	29	28	27:24	23:19	18:16	15	14:13	12:0
reserved_31_30	UsbRxPktCntSel	reserved_28	UsbRxPktCnt	UsbMaxRxBurstSize	reserved_18_16	reserved_15	reserved_14_13	ResvISOCOUTSpc

Table 1-16 Fields for Register: GRXTHRCFG

Bits	Name	Memory Access	Description
31:30	reserved_31_30	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 30 2] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead
29	UsbRxPktCntSel	R/W	USB Receive Packet Count Enable This field enables/disables the USB reception multi-packet thresholding: ■ 0: The controller can only start reception on the USB when the RX FIFO has space for at least one packet. ■ 1: The controller can only start reception on the USB when the RX FIFO has space for at least UsbRxPktCnt amount of packets. This mode is valid in both host and device mode. It is only used for SuperSpeed. In device mode, ■ Setting this bit to 1 also enables the functionality of reporting NUMP in the ACK TP based on the RX FIFO space instead of reporting a fixed NUMP derived from DCFG.NUMP for non-control endpoints. ■ If you are using external buffer control (EBC) feature, disable this mode by setting UsbRxPktCntSel to 0. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 29 1] **Exists:** Always

Bits	Name	Memory Access	Description
28	reserved_28	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 28 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
27:24	UsbRxPktCnt	R/W	USB Receive Packet Count In host mode, this field specifies the space (in terms of the number of packets) that must be available in the RX FIFO before the controller can start the corresponding USB RX transaction (burst). In device mode, this field specifies the space (in terms of the number of packets) that must be available in the RX FIFO before the controller can send ERDY for a flow-controlled endpoint. This field is valid only when the USB Receive Packet Count Enable field is set to 1. The valid values for this field are from 1 to 15. **Note:** This field must be less than or equal to the USB Maximum Receive Burst Size field. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 24 4] **Exists:** Always
23:19	UsbMaxRxBurstSize	R/W	USB Maximum Receive Burst Size In host mode, this field specifies the Maximum Bulk IN burst the DWC_usb3 controller can perform. When the system bus is slower than the USB, RX FIFO can overrun during a long burst. You can program a smaller value to this field to limit the RX burst size that the controller can perform. It only applies to SS Bulk, Isochronous, and Interrupt IN endpoints in the host mode. In device mode, this field specifies the NUMP value that is sent in ERDY for an OUT endpoint. The programmed value should not exceed the RXFIFO size. This field is valid only when UsbRxPktCntSel is one. The valid values for this field are from 1 to 16. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 19 5] **Exists:** Always

Bits	Name	Memory Access	Description
18:16	reserved_18_16	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 16 3] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7 **Write Constraint:** writeAsRead
15	reserved_15	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 15 1] **Exists:** Always **Reset Mask:** 0x1
14:13	reserved_14_13	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 13 2] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
12:0	ResvISOCOUTSpc	R/W	Space reserved in Rx FIFO for ISOC OUT In host mode, this field is not applicable and must be programmed to 0. In device mode, this value represents the amount of space to be reserved for ISOC OUT packets. The value to be programmed should be chosen so as to ensure that non ISOC packets are not completely dropped. If no space needs to be reserved for ISOC OUT packets, program this field to 0. This field is valid only in device mode. The maximum configurable depth of RX FIFO is 8192. Therefore, this field is 13 bits wide. The value of space reserved is in terms of DWC_USB3_MDWIDTH. For SS, the space reservation is always rounded off to the nearest packet boundary. Therefore, it is always recommended to program a value corresponding to MPS or its multiples. For HS/FS, the space reservation is the actual value. **Note:** For SS, reserve space for ISOC when the Rx FIFO space can accommodate two MPS or more. Otherwise, this may result in degraded performance for non-ISOC packets. If the space is entirely allocated for ISOC, the non-ISOC packets will be completely dropped. To help you decide during the time of configuring the controller, refer to the "Device-Mode Receive Path" section in the Databook. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GRXTHRCFG_INIT" 0 12] **Exists:** Always

1.2.5 GCTL

- **Description:** Global Core Control Register
  Refer to <workspace>/src/DWC_usb3_params.v for details on `DWC_USB3_GCTL_INIT.
  Note:
  When Hibernation is not enabled, you can write any value to GblHibernationEn. It always returns 0 when read.
- **Size:** 32 bits
- **Offset:** 0xc110
- **Exists:** Always

31:19	18	17	16	15:14	13:12	11	10	9	8	7:6	5:4	3	2	1	0
PWRDNSCALE	MASTERFILTBYPASS	BYPSSETADDR	U2RSTECN	FRMSCLDWN	PRTCAPDIR	CORESOFTRESET	SOFITPSYNC	U1U2TimerScale	DEBUGATTACH	RAMCLKSEL	SCALEDOWN	DISSCRAMBLE	U2EXIT_LFPS	GblHibernationEn	DSBLCLKGTNG

Table 1-17 Fields for Register: GCTL

Bits	Name	Memory Access	Description
31:19	PWRDNSCALE	R/W	Power Down Scale (PwrDnScale) The USB3 suspend_clk input replaces pipe3_rx_pclk as a clock source to a small part of the USB3 controller that operates when the SS PHY is in its lowest power (P3) state, and therefore does not provide a clock. The Power Down Scale field specifies how many suspend_clk periods fit into a 16 kHz clock period. When performing the division, round up the remainder. For example, when using an 8-bit/16-bit/32-bit PHY and 25-MHz Suspend clock, Power Down Scale = 25000 kHz/16 kHz = 13'd1563 (rounder up) **Note:** ■ Minimum Suspend clock frequency is 32 kHz ■ Maximum Suspend clock frequency is 125 MHz The LTSSM uses Suspend clock for 12-ms and 100-ms timers during suspend mode. According to the USB 3.0 specification, the accuracy on these timers is 0% to +50%. ■ 12 ms + 0~+50% accuracy = 18 ms (Range is 12 ms - 18 ms) ■ 100 ms + 0~+50% accuracy = 150 ms (Range is 100 ms - 150 ms). The suspend clock accuracy requirement is: ■ (12,000/62.5) * (GCTL[31:19]) * actual suspend_clk_period must be between 12,000 and 18,000 ■ (100,0000/62.5) * (GCTL[31:19]) * actual suspend_clk_period must be between 100,000 and 150,000 For example, if your suspend_clk frequency varies from 7.5 MHz to 10.5MHz, then the value needs to programmed is: Power Down Scale = 10500/16 = 657 (rounded up; and fastest frequency used). **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 19 13] **Exists:** Always
18	MASTERFILTBYPASS	R/W	Master Filter Bypass When this bit is set to 1'b1, irrespective of the parameter `DWC_USB3_EN_BUS_FILTERS chosen, all the filters in the DWC_usb3_filter module are bypassed. The double synchronizers to mac_clk preceding the filters are also bypassed. For enabling the filters, this bit must be 1'b0. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 18 1] **Exists:** Always

Bits	Name	Memory Access	Description
17	BYPSSETADDR	R/W	Bypass SetAddress in Device Mode. When BYPSSETADDR bit is set, the device controller uses the value in the DCFG[DevAddr] bits directly for comparing the device address in the tokens. For simulation, you can use this feature to avoid sending an actual SET ADDRESS control transfer on the USB, and make the device controller respond to a new address. When the xHCI Debug capability is enabled and this bit is set, the Debug Target immediately enters the configured state without requiring the Debug Host to send a SetAddress or SetConfig request. **Note:** You can set this bit for simulation purposes only. In the actual hardware, this bit must be set to 1'b0. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 17 1] **Exists:** Always
16	U2RSTECN	R/W	U2RSTECN If the SuperSpeed connection fails during POLL or LMP exchange, the device connects at non-SS mode. If this bit is set, then device attempts three more times to connect at SS, even if it previously failed to operate in SS mode. For each attempt, the device checks receiver termination eight times. From 2.60a release, this bit controls whether to check for Rx.Detect eight times or one time for every attempt. Device controller on USB 2.0 reset checks for receiver termination eight times per attempt if this bit is set to zero, or only once per attempt if the bit is set to one. **Note:** This bit is applicable only in device mode. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 16 1] **Exists:** Always

Bits	Name	Memory Access	Description
15:14	FRMSCLDWN	R/W	FRMSCLDWN This field scales down device view of a SOF/USOF/ITP duration. For SS/HS mode: ■ Value of 2'h3 implements interval to be 15.625 us ■ Value of 2'h2 implements interval to be 31.25 us ■ Value of 2'h1 implements interval to be 62.5 us ■ Value of 2'h0 implements interval to be 125us For FS mode, the scale-down value is multiplied by 8. When xHCI Debug Capability is enabled, this field also scales down the MaxPacketSize of the IN and OUT bulk endpoint to allow more traffic during simulation. It can only be changed from a non-zero value during simulation. ■ 2'h0: 1024 bytes ■ 2'h1: 512 bytes ■ 2'h2: 256 bytes ■ 2'h3: 128 bytes **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 14 2] **Exists:** Always

Bits	Name	Memory Access	Description
13:12	PRTCAPDIR	R/W	PRTCAPDIR: Port Capability Direction (PrtCapDir) ■ 2'b01: for Host configurations ■ 2'b10: for Device configurations **Note:** For static Host-only/Device-only applications, use DRD Host or DRD Device mode. The combination of GCTL.PrtCapDir=2'b11 with SRP and HNP/RSP disabled is not recommended for these applications. The sequence for switching modes in DRD configuration is as follows: *Switching from Device to Host:* 1. Reset the controller using GCTL[11] (CoreSoftReset). 2. Set GCTL[13:12] (PrtCapDir) to 2'b01 (Host mode). 3. Reset the host using USBCMD.HCRESET. 4. Follow the steps in "Initializing Host Registers" section of the Programming Guide. *Switching from Host to Device:* 1. Reset the controller using GCTL[11] (CoreSoftReset). 2. Set GCTL[13:12] (PrtCapDir) to 2'b10 (Device mode). 3. Reset the device by setting DCTL[30] (CSftRst). 4. Follow the steps in "Register Initialization" section of the Programming Guide. Programming this field with random data causes the controller to keep toggling between the host mode and the device mode . Bit Bash register testing is not recommended. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 12 2] **Exists:** Always **Testable:** writeAsRead

Bits	Name	Memory Access	Description
11	CORESOFTRESET	R/W	Core Soft Reset (CoreSoftReset) ■ 1'b0 - No soft reset ■ 1'b1 - Soft reset to controller Clears the interrupts and all the CSRs except the following registers: ■ GCTL ■ GUCTL ■ GSTS ■ GSNPSID ■ GGPIO ■ GUID ■ GUSB2PHYCFGn registers ■ GUSB3PIPECTLn registers ■ DCFG ■ DCTL ■ DEVTEN ■ DSTS When you reset PHYs (using GUBS3PHYCFG or GUSB3PIPECTL registers), you must keep the controller in reset state until PHY clocks are stable. This controls the bus, ram, and mac domain resets. Refer to the "Reset Generation" section in the Databook. **Note:** This bit is for debug purposes only. Use USBCMD.HCRESET in xHCI Mode and DCTL.SoftReset in device mode for soft reset. Programming this field with random data will reset the internal logic of the host controller. Due to this side effect Bit Bash register testing is not recommended. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 11 1] **Exists:** Always **Testable:** writeAsRead

Bits	Name	Memory Access	Description
10	SOFITPSYNC	R/W	SOFITPSYNC If this bit is set to '0' operating in host mode, the controller keeps the UTMI/ULPI PHY on the first port in a non-suspended state whenever there is a SuperSpeed port that is not in Rx.Detect, SS.Disable and U3. If this bit is set to '1' operating in host mode, the controller keeps the UTMI/ULPI PHY on the first port in a non-suspended state whenever the other non-SuperSpeed ports are not in a suspended state. This feature is useful because it saves power by suspending UTMI/ULPI when SuperSpeed only is active, and it helps resolve when the PHY does not transmit a host resume unless it is placed in suspend state. This bit must be programmed as a part of initialization at power-on reset, and must not be dynamically changed afterwards. **Note:** ■ USB2PHYCFGn[6].PhySusp eventually decides to put the UTMI/ULPI PHY in to suspend state. In addition, when this bit is set to '1', the controller generates ITP from the ref_clk based counter. Otherwise, ITP and SOF are generated from utmi/ulpi_clk[0] based counter. To program the reference clock period inside the controller, refer to GUCTL[31:22].REFCLKPER. ■ This feature is valid in Host and DRD configurations and used only in Host mode operation. ■ If you never use this feature or the GFLADJ.GFLADJ_REFCLK_LPM_SEL, the minimum frequency for the ref_clk can be as low as 32KHz. You can connect the suspend_clk (as low as 32 KHz) to the ref_clk. ■ If you plan to enable hardware-based LPM or software-based LPM (PORTPMSC. HLE=1), then you cannot use this feature. Turn off this feature by setting this bit to '0' and use the GFLADJ.GFLADJ_REFCLK_LPM_SEL feature. ■ If you set this bit to '1', the GUSB2PHYCFG.U2_FRECLK_EXISTS bit and the DWC_USB3_FRECLK_USB2_EXIST parameter must be set to '0'. Program this bit to 0 if the controller is intended to be operated in USB 3.0 mode. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 10 1] **Exists:** Always
9	U1U2TimerScale	R/W	Disable U1/U2 timer Scaledown (U1U2TimerScale). If set to '1' along with GCTL[5:4] (ScaleDown) = 2'bX1, disables the scale down of U1/U2 inactive timer values. This is for simulation mode only. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 9 1] **Exists:** Always

Bits	Name	Memory Access	Description
8	DEBUGATTACH	R/W	Debug Attach When this bit is set, ■ SS Link proceeds directly to the Polling link state (after RUN/STOP in the DCTL register is asserted) without checking remote termination; ■ Link LFPS polling timeout is infinite; ■ Polling timeout during TS1 is infinite (in case link is waiting for TXEQ to finish). **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 8 1] **Exists:** Always
7:6	RAMCLKSEL	R/W	RAM Clock Select (RAMClkSel) ■ 2'b00: bus clock ■ 2'b01: pipe clock (Only used in device mode) ■ 2'b10: In device mode , pipe/2 clock.In Host mode, controller switches ram_clk between pipe/2 clock, mac2_clk and bus_clk based on the status of the U2/U3 ports ■ 2'b11: In device mode, selects mac2_clk as ram_clk (when 8-bit UTMI or ULPI used. Not supported in 16-bit UTMI mode) In Host mode, controller switches ram_clk between pipe_clk, mac2_clk and bus_clk based on the status of the U2/U3 ports. In device mode, upon a USB reset and USB disconnect, the hardware clears these bits to 2'b00. For more information on how to select the RAM clock, see the "Clock Generation and Clock Tree Synthesis (CTS) Requirements" section in the Databook. **Note:** ■ In device mode, if you set RAMClkSel to 2'b11 (mac2_clk), the controller internally switches the ram_clk to bus_clk when the link state changes to Suspend (L2 or L3), and switches the ram_clk back to mac2_clk when the link state changes to resume or U2.

Bits	Name	Memory Access	Description
7:6...(cont.)	RAMCLKSEL.	R/W	■ In host mode, if a value of 2/3 is chosen, then controller switches ram_clk between bus_clk, mac2_clk and pipe_clk, pipe_clk/2, based on the state of the U2/U3 ports. For example, if only the U2 port is active and the U3 ports are suspended, then the ram_clk is switched to mac2_clk. When only the U3 ports are active and the U2 ports are suspended, the controller internally switches the ram_clk to pipe3 clock and when all U2 and U3 ports are suspended, it switches the ram_clk to bus_clk. This allows decoupling the ram_clk from the bus_clk, and depending on the bandwidth requirement allows the bus_clk to be run at a lower frequency than the ram_clk requirements. The bus_clk frequency still cannot be less than 60MHz in host mode, and this is not verified. A value of 2 can be chosen only if the pipe data width is 8 or 16 bits. In this case the when the ram_clk is switched to pipe_clk, it uses pipe_clk/2 instead of pipe_clk. If a value of 3 is chosen for RAMClkSel, then when ram_clk is switched to pipe_clk, then pipe_clk is used without any divider. ■ In device mode, when RAMClkSel != 2'b00, the bus_clk_early frequency can be a minimum of 1 MHz. This is tested in simulation and also in hardware with Linux, Microsoft Windows 8, and MCCI Windows7 host drivers. Only control and non periodic transfers are supported when bus_clk is 1 MHz. For periodic applications, the bus_clk_early minimum frequency is higher depending on your application and SoC bus. Even though 1 MHz has been tested with standard host drivers, Synopsys recommends 5 MHz minimum for ASIC designs to provide a margin or at least have a backup option to increase the bus_clk frequency to 5 MHz if needed. Programming this field with random data will cause side effect. Bit Bash register testing is not recommended. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 6 2] **Exists:** Always **Testable:** writeAsRead

Bits	Name	Memory Access	Description
5:4	SCALEDOWN	R/W	Scale-Down Mode (ScaleDown) When Scale-Down mode is enabled for simulation, the controller uses scaled-down timing values, resulting in faster simulations. When Scale-Down mode is disabled, actual timing values are used. This is required for hardware operation. *HS/FS/LS Modes* ■ 2'b00: Disables all scale-downs. Actual timing values are used. ■ 2'b01: Enables scale-down of all timing values except Device mode suspend and resume. These include Speed enumeration, HNP/SRP, and Host mode suspend and resume ■ 2'b10: Enables scale-down of Device mode suspend and resume timing values only. ■ 2'b11: Enables bit 0 and bit 1 scale-down timing values. *SS Mode* ■ 2'b00: Disables all scale-downs. Actual timing values are used. ■ 2'b01: Enables scaled down SS timing and repeat values including: (1) Number of TxEq training sequences reduce to 8; (2) LFPS polling burst time reduce to 256 nS; (3) LFPS warm reset receive reduce to 30 uS. Refer to the rtl_vip_scaledown_mapping.xls file under <workspace>/sim/SoC_sim directory for the complete list. ■ 2'b10: No TxEq training sequences are sent. Overrides Bit 4. ■ 2'b11: Enables bit 0 and bit 1 scale-down timing values. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 4 2] **Exists:** Always
3	DISSCRAMBLE	R/W	Disable Scrambling (DisScramble) Transmit request to Link Partner on next transition to Recovery or Polling. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 3 1] **Exists:** Always

Bits	Name	Memory Access	Description
2	U2EXIT_LFPS	R/W	U2EXIT_LFPS If this bit is, ■ 0: the link treats 248ns LFPS as a valid U2 exit. ■ 1: the link waits for 8us of LFPS before it detects a valid U2 exit. This bit is added to improve interoperability with a third-party host/device controller. This host/device controller in U2 state while performing receiver detection generates an LFPS glitch of about 4ms duration. This causes the host/device to exit from U2 state because the LFPS filter value is 248ns. With the new functionality enabled, the host/device can stay in U2 while ignoring this glitch from the host/device controller. This bit is applicable for both host and device controller. This bit is added to improve interoperability with a third party host controller. This host controller in U2 state while performing receiver detection generates an LFPS glitch of about 4ms duration. This causes the device to exit from U2 state because the LFPS filter value is 248ns. With the new functionality enabled, the device can stay in U2 while ignoring this glitch from the host controller. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 2 1] **Exists:** Always
1	GblHibernationEn	* Varies	GblHibernationEn This bit enables hibernation at the global level. If hibernation is not enabled through this bit, the PMU immediately accepts the D0->D3 and D3->D0 power state change requests, but does not save or restore any controller state. In addition, the PMUs never drive the PHY interfaces and let the controller continue to drive the PHY interfaces. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 1 1] **Exists:** Always **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_EN_PWROPT==2 ? "read-write" : "read-only"}
0	DSBLCLKGTNG	R/W	Disable Clock Gating (DsblClkGtng) This bit is set to 1 and the controller is in Low Power mode, internal clock gating is disabled. You can set this bit to 1'b1 after Power On Reset. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GCTL_INIT" 0 1] **Exists:** Always

1.2.6 GPMSTS

- **Description:** Global Power Management Status Register
  This debug register gives information on which event caused the hibernation exit. It provides internal status and state machine information, and is for Synopsys use only for debugging purposes.
  This register is not applicable in USB 2.0-only mode.
- **Size:** 32 bits
- **Offset:** 0xc114
- **Exists:** Always

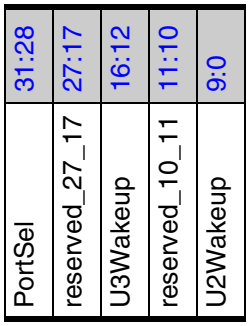


Table 1-18 Fields for Register: GPMSTS

Bits	Name	Memory Access	Description
31:28	PortSel	W	Global Power Management Status Register, PortSel This field selects the port number. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable
27:17	reserved_27_17	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7ff **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
16:12	U3Wakeup	R	U3Wakeup This field gives the following USB 3.0 port wakeup conditions: ■ Bit [12]: Overcurrent Detected ■ Bit [13]: Resume Detected ■ Bit [14]: Connect Detected ■ Bit [15]: Disconnect Detected ■ Bit [16]: Last Connection State **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:10	reserved_10_11	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead
9:0	U2Wakeup	R	U2Wakeup This field indicates the following USB 2.0 port wakeup conditions: ■ Bit [0]: Overcurrent Detected ■ Bit [1]: Resume Detected ■ Bit [2]: Connect Detected ■ Bit [3]: Disconnect Detected ■ Bit [4]: Last Connection State ■ Bit [5]: ID Change Detected ■ Bit [6]: SRP Request Detected ■ Bit [7]: ULPI Interrupt Detected ■ Bit [8]: USB Reset Detected ■ Bit [9]: Resume Detected Changed **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.7 GSTS

- **Description:** Global Status Register
- **Size:** 32 bits
- **Offset:** 0xc118
- **Exists:** Always

31:20	19:12	11	10	9	8	7	6	5	4	3:2	1:0
CBELT	reserved_19_12	SSIC_IP	OTG_IP	BC_IP	ADP_IP	Host_IP	Device_IP	CSRTimeout	BUSERRADDRVLD	reserved_3_2	CURMOD

Table 1-19 Fields for Register: GSTS

Bits	Name	Memory Access	Description
31:20	CBELT	R	Current BELT Value In Host mode, this field indicates the minimum value of all received device BELT values and the BELT value that is set by the Set Latency Tolerance Value command. **Value After Reset:** {DWC_USB3_MODE == 0 ? 0x0 : 0x7e8} **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
19:12	reserved_19_12	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead
11	SSIC_IP	R	This field is not used. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
10	OTG_IP	R	This field is not used. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
9	BC_IP	R	Battery Charger Interrupt Pending This field indicates that there is a pending interrupt pertaining to BC in BCEVT register. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
8	ADP_IP	R	This field is not used. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7	Host_IP	R	Host Interrupt Pending: This field indicates that there is a pending interrupt pertaining to xHC in the Host event queue. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
6	Device_IP	R	Device Interrupt Pending This field indicates that there is a pending interrupt pertaining to peripheral (device) operation in the Device event queue. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
5	CSRTimeout	R/W1C	CSR Timeout When this bit is 1'b1, it indicates that the software performed a write or read to a controller register that could not be completed within `DWC_USB3_CSR_ACCESS_TIMEOUT bus clock cycles (default: h1FFFF). **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x1 **Volatile:** true

Bits	Name	Memory Access	Description
4	BUSERRADDRVLD	R/W1C	Bus Error Address Valid (BusErrAddrVld) Indicates that the GBUSERRADDR register is valid and reports the first bus address that encounters a bus error. **Note:** Only supported in AHB and AXI configurations. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x1 **Volatile:** true
3:2	reserved_3_2	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead
1:0	CURMOD	R	Current Mode of Operation (CurMod) Indicates the current mode of operation: ■ 2'b00: Device mode ■ 2'b01: Host mode **Value After Reset:** {DWC_USB3_MODE==2 ? 0 : DWC_USB3_MODE} **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.8 GUCTL1

- **Description:** Global User Control Register 1
- **Size:** 32 bits
- **Offset:** 0xc11c
- **Exists:** Always

31	30	29	28	27	26	25	24	23:21	20	19	18	17	16	15	14:13	12	11	10	9	8	7:4	3	2	1	0
DEV_DECOUPLE_L1L2_EVT	DS_RXDET_MAX_TOUT_CTRL	FILTER_SE0_FSLS_EOP	TX_IPGAP_LINECHECK_DIS	DEV_TRB_OUT_SPR_IND	DEV_FORCE_20_CLK_FOR_30_CLK	P3_IN_U2	DEV_L1_EXIT_BY_HW	IP_GAP_ADD_ON	DEV_LSP_TAIL_LOCK_DIS	NAK_PER_ENH_FS	NAK_PER_ENH_HS	PARKMODE_DISABLE_SS	PARKMODE_DISABLE_HS	PARKMODE_DISABLE_FSLS	reserved_14_13	DisUSB2RefClkGtng	DisRefClkGtng	RESUME_OPMODE_HS_HOST	DEV_HS_NYET_BULK_SPR	L1_SUSP_THRLD_EN_FOR_HOST	L1_SUSP_THRLD_FOR_HOST	HC_ERRATA_ENABLE	HC_PARCHK_DISABLE	OVRLD_L1_SUSP_COM	LOA_FILTER_EN

Table 1-20 Fields for Register: GUCTL1

Bits	Name	Memory Access	Description
31	DEV_DECOUPLE_L1L2_EVT	R/W	DEV_DECOUPLE_L1L2_EVT ■ 0: Default behavior, no change in device events L1/L2U3 events are not decoupled (old behavior of v2.90a and before) ■ 1: Feature enabled, L1 and L2 events are separated when operating in 2.0 mode. Separate event enable bits for L1 suspend and wake events. This bit is applicable for device mode only. If this feature is enabled, L1 suspend and wake events have individual controls to enable/mask them. Enable this feature if you want to get L1 (LPM) events separtely and not combined with L2 events when operating in 2.0 speeds. **Exists:** Always

Bits	Name	Memory Access	Description
30	DS_RXDET_MAX_TOUT_CTRL	R/W	DS_RXDET_MAX_TOUT_CTRL This bit is used to control the tRxDetectTimeoutDFP timer for the SuperSpeed link. ■ 0: Default behavior; 12ms is used as tRxDetectTime-outDFP. ■ 1: 120ms is used as the tRxDetectTimeoutDFP. This bit is used only in host mode. For more details, refer to ECN020 for USB 3.0 Specification. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 30 1] **Exists:** Always
29	FILTER_SE0_FSLS_EOP	R/W	FILTER_SE0_FSLS_EOP ■ 0: Default behavior, no change in Linestate check for SE0 detection in FS/LS ■ 1: Feature enabled, FS/LS SE0 is filtered for 2 clocks for detecting EOP This bit is applicable for FS/LS operation. If this feature is enabled, then SE0 on the linestate is validated for 2 consecutive utmi/ulpi clock edges for EOP detection. This feature is applicable only in FS in device mode and FS/LS mode of operation in host mode. *Device mode:* FS - If GUCTL1.FILTER_SE0_FSLS_EOP is set, then for device LPM handshake, the controller will ignore single SE0 glitch on the linestate during transmit. Only 2 or more SE0 is considered as a valid EOP on FS. *Host mode:* FS/LS - If GUCTL1.FILTER_SE0_FSLS_EOP is set, then the controller will ignore single SE0 glitch on the linestate during transmit. Only 2 or more SE0 is considered as a valid EOP on FS/LS port. Enable this feature if the LineState has SE0 glitches during transmission. This bit is quasi-static, that is, it must not be changed during device operation. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 29 1] **Exists:** Always

Bits	Name	Memory Access	Description
28	TX_IPGAP_LINECHECK_DIS	R/W	TX_IPGAP_LINECHECK_DIS ■ 0: Default behavior, no change in Linestate check ■ 1: Feature enabled, 2.0 MAC disables Linestate check during HS transmit This bit is applicable for HS operation of u2mac. If this feature is enabled, then the 2.0 mac operating in HS ignores the UTMI/ULPI Linestate during the transmit of a token (during token-to-token and token-to-data IPGAP). When enabled, the controller implements a fixed 40-bit TxEndDelay after the packet is given on UTMI and ignores the Linestate during this time. This feature is applicable only in HS mode of operation. *Device mode:* If GUCTL1.TX_IPGAP_LINECHECK_DIS is set, then for device LPM handshake, the controller will ignore the linestate after TX and wait for fixed clocks (40 bit times equivalent) after transmitting ACK on utmi. *Host mode:* If GUCTL1.TX_IPGAP_LINECHECK_DIS is set, then the ipgap between (tkn to tkn/data) is added by 40 bit times of TXENDDELAY, and linestate is ignored during this 40 bit times delay. Enable this bit if the LineState will not reflect the expected line state (J) during transmission. This bit is quasi-static, that is, it must not be changed during device operation. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 28 1] **Exists:** Always
27	DEV_TRB_OUT_SPR_IND	R/W	DEV_TRB_OUT_SPR_IND ■ 0: Default behavior, no change in TRB status dword ■ 1: Feature enabled, OUT TRB status indicates Short Packet This bit is applicable for device mode only (and ignored in host mode). If the device application (software/hardware) wants to know if a short packet was received for an OUT in the TRB status itself, then this feature can be enabled, so that a bit is set in the TRB writeback in the buf_size dword. Bit[26] - SPR of the {trbstatus, RSVD, SPR, PCM1, bufsize} dword will be set during an OUT transfer TRB write back if this is the last TRB used for that transfer descriptor. This bit is quasi-static, that is, it must not be changed during device operation. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 27 1] **Exists:** Always

Bits	Name	Memory Access	Description
26	DEV_FORCE_20_CLK_FOR_30_CLK	R/W	DEV_FORCE_20_CLK_FOR_30_CLK ■ 0: Default behavior, Uses 3.0 clock when operating in 2.0 mode ■ 1: Feature enabled This bit is applicable (and to be set) for device mode (DCFG.Speed != SS) only. In the 3.0 device controller, if the controller is programmed to operate in 2.0 only (that is, Device Speed is programmed to 2.0 speeds in DCFG[Speed]), then setting this bit makes the internal 2.0 (utmi/ulpi) clock to be routed as the 3.0 (pipe) clock. Enabling this feature allows the pipe3 clock to be not-running when forcibly operating in 2.0 device mode. **Note:** ■ When using this feature, all pipe3 inputs must be in inactive mode. In particular, the pipe3 clocks must not be running and the pipe3_phystatus_async must be tied to 0. This bit should not be set if the controller is programmed to operate in SuperSpeed mode (even when it falls back to 2.0). ■ This bit is quasi-static, that is, it must not be changed during operation. ■ If the parameter "DWC_USB3_REMOVE_PIPE_-CLK_MUX_FOR_20_MODE" is enabled, then muxing 2.0 clock/signals to pipe_clk/signals have to be done outside the controller for this feature to work. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 26 1] **Exists:** Always
25	P3_IN_U2	R/W	P3_IN_U2 ■ 0: Default behavior, When SuperSpeed link is in U2, PowerState P2 is attempted on the PIPE Interface. ■ 1: When SuperSpeed link is in U2, PowerState P3 is attempted if GUSB3PIPECTL[17] is set. Setting this bit enables P3 Power State when the SuperSpeed link is in U2. Another Power Saving option. Check with your PHY vendor before enabling this option.When setting this bit to 1 to enable P3 in P2, GUSB3PIPECTL[27] should be set to 0 to make sure that the U2 exit is attempted in P0. This bit should be set only when GCTL.SOFITPSYNC=1 or GFLADJ.GFLADJ_REFCLK_LPM_SEL=1. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 25 1] **Exists:** Always

Bits	Name	Memory Access	Description
24	DEV_L1_EXIT_BY_HW	R/W	DEV_L1_EXIT_BY_HW ■ 0: Default behavior, disables device L1 hardware exit logic ■ 1: feature enabled This bit is applicable for device mode (2.0) only. This field enables device controller sending remote wakeup for L1 if the device becomes ready for sending/accepting data when in L1 state. If the host expects the device to send remote wakeup signaling to resume after going into L1 in flow controlled state, then this bit can be set to send the remote wake signal automatically when the device controller becomes ready. This hardware remote wake feature is applicable only to bulk and interrupt transfers, and not for Isoch/Control ■ When control transfers are in progress, the LPM will be rejected (NYET response). Only after control transfers are completed (either with ACK/STALL), LPM will be accepted ■ For Isoch transfers, the host needs to do the wake-up and start the transfer. Device controller will not do remote-wakeup when Isoch endpoints get ready. The device SW needs to keep the GUSB2PHYCFG[EnblSlpM] reset in order to keep the PHY clock to be running for keeping track of SOF intervals. ■ When L1 hibernation is enabled, the controller will not do automatic exit for hibernation requests thru L1. This bit is quasi-static, that is, it must not be changed during device operation. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 24 1] **Exists:** Always
23:21	IP_GAP_ADD_ON	R/W	This register field is used to add on to the default inter packet gap setting in the USB 2.0 MAC. This should be programmed to a non zero value only in case where you need to increase the default inter packet delay calculations in the USB 2.0 MAC module DWC_usb3_u2mac.v **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 21 3] **Exists:** Always

Bits	Name	Memory Access	Description
20	DEV_LSP_TAIL_LOCK_DIS	R/W	DEV_LSP_TAIL_LOCK_DIS ■ 0: Default behavior, enables device lsp lock logic for tail TRB update ■ 1: Fix disabled This is a bug fix for STAR 9000716195 that affects the CSP mode for OUT endpoints in device mode. The issue is that tail TRB index is not synchronized with the cache Scratchpad bytecount update. If the fast-forward request comes in-between the bytecount update on a newly fetched TRB and the tail-index write update in TPF, the RDP works on an incorrect tail index and misses the byte count decrement for the newly fetched TRB in the fast-forwarding process. This fix needs to be present all the times. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 20 1] **Exists:** Always
19	NAK_PER_ENH_FS	R/W	NAK_PER_ENH_FS ■ 1: Enables performance enhancement for FS async endpoints in the presence of NAKs ■ 0: Enhancement not applied If a periodic endpoint is present , and if a bulk endpoint which is also active is being NAKed by the device, then this could result in a decrease in performance of other Full Speed bulk endpoint which is ACked by the device. Setting this bit to 1, will enable the host controller to schedule more transactions to the async endpoints (bulk/ control) and hence will improve the performance of the bulk endpoint. This control bit should be enabled only if the existing performance with the default setting is not sufficient for your FullSpeed application. Setting this bit will only control, and is only required for Full Speed transfers. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 19 1] **Exists:** Always

Bits	Name	Memory Access	Description
18	NAK_PER_ENH_HS	R/W	NAK_PER_ENH_HS ■ 1: Enables performance enhancement for HS async endpoints in the presence of NAKs ■ 0: Enhancement not applied If a periodic endpoint is present, and if a bulk endpoint which is also active is being NAKed by the device, then this could result in decrease in performance of other High Speed bulk endpoint which is ACked by the device. Setting this bit to 1, will enable the host controller to schedule more transactions to the async endpoints (bulk/ control) and hence will improve the performance of the bulk endpoint. This control bit should be enabled only if the existing performance with the default setting is not sufficient for your HighSpeed application. Setting this bit will only control, and is only required for High Speed transfers. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 18 1] **Exists:** Always
17	PARKMODE_DISABLE_SS	R/W	PARKMODE_DISABLE_SS This bit is used only in host mode, and is for debug purpose only. When this bit is set to '1' all SS bus instances in park mode are disabled. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 17 1] **Exists:** Always

Bits	Name	Memory Access	Description
16	PARKMODE_DISABLE_HS	R/W	PARKMODE_DISABLE_HS This bit is used only in host mode. When this bit is set to '1' all HS bus instances park mode are disabled. To improve performance in park mode, the xHCI scheduler queues in three requests of 4 packets each for High Speed asynchronous endpoints in a micro-frame. But if a device is slow and if it NAKs more than 3 times, then it is rescheduled only in the next micro-frame. This could decrease the performance of a slow device even further. In a few high speed devices (such as Sandisk Cruzer Blade 4GB VID:1921, PID:21863 and Flex Drive VID:3744, PID:8552) when an IN request is sent within 900ns of the ACK of the previous packet, these devices send a NAK. When connected to these devices, if required, the software can disable the park mode if you see performance drop in your system. When park mode is disabled, pipelining of multiple packet is disabled and instead one packet at a time is requested by the scheduler. This allows up to 12 NAKs in a micro-frame and improves performance of these slow devices. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 16 1] **Exists:** Always
15	PARKMODE_DISABLE_FSLS	R/W	PARKMODE_DISABLE_FSLS This bit is used only in host mode, and is for debug purpose only. When this bit is set to '1' all FS/LS bus instances in park mode disabled. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 15 1] **Exists:** Always
14:13	reserved_14_13	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 13 2] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
12	DisUSB2RefClkGtng	* Varies	Disable ref_clk gating for 2.0 PHY (DisUSB2RefClkGtng) If ref_clk gating is disabled, then the ref_clk input cannot be turned off to the USB 2.0 PHY and controller. This is independent of the GCTL[DisClkGtng] setting. ■ 1'b0: ref_clk gating enabled for USB 2.0 PHY ■ 1'b1: ref_clk gating disabled for USB 2.0 PHY **Value After Reset:** =mpformat "0x%x" [{DWC_USB3_REF_CLK_OFF == 0 ? 1 : [get_field_val "DWC_USB3_GUCTL1" 12 1]}] **Exists:** Always **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_REF_CLK_OFF==1 ? "read-write" : "read-only"}
11	DisRefClkGtng	* Varies	Disable ref_clk gating (DisRefClkGtng) If the ref_clk gating is disabled then input ref_clk cannot be turned off to SSPHY and controller. This is independent of GCTL[DisClkGtng] setting. ■ 1'b0: ref_clk gating Enabled for SSPHY ■ 1'b1: ref_clk gating Disabled for SSPHY **Value After Reset:** =mpformat "0x%x" [{DWC_USB3_REF_CLK_OFF == 0 ? 1 : [get_field_val "DWC_USB3_GUCTL1" 11 1]}] **Exists:** Always **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_REF_CLK_OFF==1 ? "read-write" : "read-only"}
10	RESUME_OPMODE_HS_HOST	R/W	RESUME_OPMODE_HS_HOST This bit is used only in host mode, and is for USB 2.0 opmode behavior in HS Resume. ■ When this bit is set to '1', the UTMI/ULPI opmode will be changed to "normal" along with HS terminations after EOR. This option is to support certain legacy UTMI/ULPI PHYs. ■ When this bit is set to '0', the UTMI/ULPI opmode will be changed to "normal" 2us after HS terminations change after EOR. This is the default behavior. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 10 1] **Exists:** Always

Bits	Name	Memory Access	Description
9	DEV_HS_NYET_BULK_SPR	R/W	DEV_HS_NYET_BULK_SPR ■ 0: Default behavior, no change in device response ■ 1: Feature enabled, HS bulk OUT short packet gets NYET response This bit is applicable for device mode only (and ignored in host mode) to be used in 2.0 operation. If this bit is set, the device controller sends NYET response instead of ACK response for a successfully received bulk OUT short packet. If NYET is sent after receiving short packet, then the host would PING before sending the next OUT; this improves the performance as well as clears up the buffer/cache on the host side. Internal to the device controller, short packet (SPR=1) processing takes some time, and during this time, the USB is flow controlled. With NYET response instead of ACK on short packet, the host does not send another OUT-DATA without pinging in HS mode. This bit is quasi-static, that is, it must not be changed during device operation. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 9 1] **Exists:** Always
8	L1_SUSP_THRLD_EN_FOR_HOST	R/W	L1_SUSP_THRLD_EN_FOR_HOST This bit is used only in host mode. The host controller asserts the utmi_l1_suspend_n and utmi_sleep_n output signals (see "LPM Interface Signals" table in the Databook) as follows: The controller asserts the utmi_l1_suspend_n signal to put the PHY into deep low-power mode in L1 when both of the following are true: ■ The HIRD/BESL value used is greater than or equal to the value in L1_SUSP_THRLD_FOR_HOST field. ■ The L1_SUSP_THRLD_EN_FOR_HOST bit is set to 1'b1. The controller asserts utmi_sleep_n on L1 when one of the following is true: ■ The HIRD/BESL value used is less than the value in L1_SUSP_THRLD_FOR_HOST field. ■ The L1_SUSP_THRLD_EN_FOR_HOST bit is set to 1'b0. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 8 1] **Exists:** Always

Bits	Name	Memory Access	Description
7:4	L1_SUSP_THRLD_FOR_HOST	R/W	L1_SUSP_THRLD_FOR_HOST This field is effective only when the L1_SUSP_THRLD_EN_FOR_HOST bit is set to 1. For more details, refer to the description of the L1_SUSP_THRLD_EN_FOR_HOST bit. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 4 4] **Exists:** Always
3	HC_ERRATA_ENABLE	R/W	Host ELD Enable (HELDEn) When this bit is set to 1, it enables the Exit Latency Delta (ELD) support defined in the xHCI 1.0 Errata. This bit is used only in the host mode. This bit has to be set to 1 in Host mode. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 3 1] **Exists:** Always
2	HC_PARCHK_DISABLE	R/W	Host Parameter Check Disable (HParChkDisable) When this bit is set to '0' (by default), the xHC checks that the input slot/EP context fields comply to the xHCI Specification. Upon detection of a parameter error during command execution, the xHC generates an event TRB with completion code indicating 'PARAMETER ERROR'. When the bit is set to '1', the xHC does not perform parameter checks and does not generate 'PARAMETER ERROR' completion code. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 2 1] **Exists:** Always

Bits	Name	Memory Access	Description
1	OVRLD_L1_SUSP_COM	R/W	OVRLD_L1_SUSP_COM If this bit is set, the utmi_l1_suspend_com_n is overloaded with the utmi_sleep_n signal. This bit is usually set if the PHY stops the port clock during L1 sleep condition. **Note:** The recommended connection for the SUSPENDM/SLEEPM signals to the PHY with respect to this bit is as follows. **For non-zero ports:** Connect: ■ utmi_sleep_n[n] to SLEEPM[n] ■ (utmi_suspend_n[n] & utmi_l1_suspend_n[n]) to SUSPENDM[n] ■ USB2 PHYCLK[n] to utmi_clk[n] GUCTL1.OVRLD_L1_SUSP_COM impacts only Port0. **For Port0:** *For Synopsys PHY,* GUSB2PHYCFGn.U2_FREECLK_EXISTS=1; With this connection, the PHY keeps PLL active so that FREECLK is always available irrespective of suspend/sleep. ■ Connect USB2 PHY COMMONONN to 0. ■ Connect utmi_sleep_n[0] to SLEEPM[0]. ■ Connect (utmi_suspend_n[0] & utmi_l1_suspend_n[0]) to SUSPENDM[0]. ■ Connect USB2 PHY FREECLK to utmi_clk[0]. ■ Leave utmi_suspend_com_n, utmi_l1_suspend_com_n unconnected. ■ GUCTL1.OVRLD_L1_SUSP_COM can be set to any value. *For Third Party PHY,* GUSB2PHYCFGn.U2_FREECLK_EXISTS=0; With this connection the PHY can shut off all the clocks when the required conditions are met (like, GUSB2PHYCFGn[8,6], GUCTL1[1], GFLADJ[23], GCTL[10], Suspend condition, HW LPM enable etc). ■ Connect -utmi_suspend_com_n to SUSPENDM[0] (or equivalent). ■ Connect -utmi_l1_suspend_com_n to SLEEPM[0] (or equivalent). ■ Connect PHYCLK0 (first port clock) to utmi_clk[0]. ■ Leave utmi_suspend_n[0], utmi_l1_suspend_ n[0], utmi_sleep_n[0] unconnected. ■ Set GUCTL1.OVRLD_L1_SUSP_COM to 1'b1. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 1 1] **Exists:** Always

Bits	Name	Memory Access	Description
0	LOA_FILTER_EN	R/W	LOA_FILTER_EN If this bit is set, the USB 2.0 port babble is checked at least three consecutive times before the port is disabled. This prevents false triggering of the babble condition when using low quality cables. **Note:** This bit is valid only in host mode. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL1" 0 1] **Exists:** Always

1.2.9 GSNPSID

- **Description:** Global Synopsys ID Register
  This is a read-only register that contains the release number of the controller.
- **Size:** 32 bits
- **Offset:** 0xc120
- **Exists:** Always

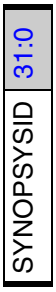


Table 1-21 Fields for Register: GSNPSID

Bits	Name	Memory Access	Description
31:0	SYNOPSYSID	R	Synopsys ID ■ SynopsysID[31:16] indicates Core Identification Number. 0x5533 is ASCII for U3 (DWC_usb3). ■ SynopsysID[15:0] indicates the release number. Current Release is 3.30a. Software uses this register to configure release-specific features in the driver. **Value After Reset:** =format "0x5533%s" [mem_getCKVersion] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.10 GGPIO

- **Description:** Global General Purpose Input/Output Register
  The application can use this register for general purpose input and output ports or for debugging.
- **Size:** 32 bits
- **Offset:** 0xc124
- **Exists:** Always

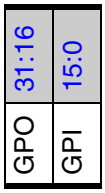


Table 1-22 Fields for Register: GGPIO

Bits	Name	Memory Access	Description
31:16	GPO	R/W	General Purpose Output The value of this field is driven out on the gp_out[15:0] output port. **Value After Reset:** 0x0 **Exists:** Always
15:0	GPI	R	General Purpose Input The read value of this field reflects the gp_in[15:0] input signal value. **Note:** Register bit-bash test should not check for reset value of this field since its not predictable; depends on the gp_in port. **Value After Reset:** gp_in [15:0] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.11 GUID

- **Description:** Global User ID Register
  This is a read/write register containing the User ID. The power-on value for this register is specified as the User Identification Register. Power-on value during coreConsultant configuration (parameter DWC_USB3_USERID). This register can be used in the following ways:
  - To store the version or revision of your system;
  - To store hardware configurations that are outside the controller;
  - As a scratch register.
- **Size:** 32 bits
- **Offset:** 0xc128
- **Exists:** Always

Table 1-23 Fields for Register: GUID

Bits	Name	Memory Access	Description
31:0	USERID	R/W	USERID Application-programmable ID field. **Value After Reset:** DWC_USB3_USERID **Exists:** Always

1.2.12 GUCTL

- **Description:** Global User Control Register:
  This register provides a few options for the software to control the controller behavior in the Host mode. Most of the options are used to improve host inter-operability with different devices.
- **Size:** 32 bits
- **Offset:** 0xc12c
- **Exists:** Always

31:22	21	20:18	17	16	15	14	13	12	11	10:9	8:0
REFCLKPER	NoExtrDl	reserved_20_18	SprsCtrlTransEn	ResBwHSEPS	reserved_15	USBHstInAutoRetryEn	EnOverlapChk	ExtCapSupptEN	InsrtExtrFSBODI	DTCT	DTFT

Table 1-24 Fields for Register: GUCTL

Bits	Name	Memory Access	Description
31:22	REFCLKPER	R/W	REFCLKPER This field indicates in terms of nano seconds the period of ref_clk. The default value of this register is set to 'h8 (8ns/125 MHz). This field needs to be updated during power-on initialization, if GCTL.SOFITPSYNC or GFLADJ.GFLADJ_REFCLK_LPM_SEL is set to '1'. The programmable maximum value is 62ns, and the minimum value is 8ns. You must use a reference clock with a period that is an integer multiple, so that ITP can meet the jitter margin of 32ns. The allowable ref_clk frequencies whose period is not integer multiples are 16/17/19.2/24/39.7MHz. This field must not be set to '0' at any time. If you never plan to use this feature, then set this field to 'h8, the default value. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 22 10] **Exists:** Always

Bits	Name	Memory Access	Description
21	NoExtrDl	R/W	No Extra Delay Between SOF and the First Packet(NoExtrDl) Some HS devices misbehave when the host sends a packet immediately after a SOF. However, adding an extra delay between a SOF and the first packet can reduce the USB data rate and performance. This bit is used to control whether the host must wait for 2 microseconds before it sends the first packet after a SOF, or not. User can set this bit to one to improve the performance if those problematic devices are not a concern in the user's host environment. ■ 1'b0: Host waits for 2 microseconds after a SOF before it sends the first USB packet. ■ 1'b1: Host doesn't wait after a SOF before it sends the first USB packet. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 21 1] **Exists:** Always
20:18	reserved_20_18	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 18 3] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7 **Write Constraint:** writeAsRead
17	SprsCtrlTransEn	R/W	Sparse Control Transaction Enable Some devices are slow in responding to Control transfers. Scheduling multiple transactions in one microframe/frame can cause these devices to misbehave. If this bit is set to 1'b1, the host controller schedules transactions for a Control transfer in different microframes/frames. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 17 1] **Exists:** Always

Bits	Name	Memory Access	Description
16	ResBwHSEPS	R/W	Reserving 85% Bandwidth for HS Periodic EPs (ResBwHSEPS) By default, HC reserves 80% of the bandwidth for periodic EPs. If this bit is set, the bandwidth is relaxed to 85% to accommodate two high speed, high bandwidth ISOC EPs. USB 2.0 required 80% bandwidth allocated for ISOC traffic. If two High-bandwidth ISOC devices (HD Webcams) are connected, and if each requires 1024-bytes X 3 packets per Micro-Frame, then the bandwidth required is around 82%. If this bit is set, then it is possible to connect two Webcams of 1024bytes X 3 paylod per Micro-Frame each. Otherwise, you may have to reduce the resolution of the Webcams. This bit is valid in Host and DRD configuration and is used in host mode operation only. Ignore this bit in device mode. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 16 1] **Exists:** Always
15	reserved_15	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 15 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1
14	USBHstInAutoRetryEn	R/W	Host IN Auto Retry (USBHstInAutoRetryEn) When set, this field enables the Auto Retry feature. For IN transfers (non-isochronous) that encounter data packets with CRC errors or internal overrun scenarios, the auto retry feature causes the Host controller to reply to the device with a non-terminating retry ACK (that is, an ACK transaction packet with Retry = 1 and NumP != 0). If the Auto Retry feature is disabled (default), the controller will respond with a terminating retry ACK (that is, an ACK transaction packet with Retry = 1 and NumP = 0). ■ 1'b0: Auto Retry Disabled ■ 1'b1: Auto Retry Enabled **Note:** When enabling Auto Retry feature, if the system latency is large enough to cause the internal PSQ full (PSQ can be full as the result of messages not being processed because of pending fetches before flushing the TxQ due to NRDY/ERDY conditions), then the host controller can generate a transaction error. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 14 1] **Exists:** Always

Bits	Name	Memory Access	Description
13	EnOverlapChk	R/W	Enable Check for LFPS Overlap During Remote Ux Exit: If this bit is set to, ■ 1'b1: The SuperSpeed link when exiting U1/U2/U3 waits for either the remote link LFPS or TS1/TS2 training symbols before it confirms that the LFPS handshake is complete. This is done to handle the case where the LFPS glitch causes the link to start exiting from the low power state. Looking for the LFPS overlap makes sure that the link partner also sees the LFPS. ■ 1'b0: When the link exists U1/U2/U3 because of a remote exit, it does not look for an LFPS overlap. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 13 1] **Exists:** Always
12	ExtCapSupptEN	R/W	External Extended Capability Support Enable (ExtCapSuptEN) When set, this field enables extended capabilities to be implemented outside the controller. When the ExtCapSupEN is set and the Debug Capability is enabled, the Next Capability pointer in "Debug Capability" returns 16. A read to the first DWORD of the last internal extended capability (the "xHCI Supported Protocol Capability for USB 3.0" when the Debug Capability is not enabled) returns a value of 4 in the Next Capability Pointer field. This indicates to software that there is another capability four DWORDs after this capability (for example, at address N+16 where N is the address of this DWORD). If enabled, an external address decoder that snoops the xHC slave interface must be implemented. If it sees an access to N+16 or greater, the slave access is re-routed to a piece of hardware which returns the external capability pointer register of the new capability and also handles reads/writes to this new capability and the side effects. If disabled, a read to the first DWORD of the last internal extended capability returns 0 in the 'Next Capability Pointer field. This indicates there are no more capabilities. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 12 1] **Exists:** Always

Bits	Name	Memory Access	Description
11	InsrtExtrFSBODI	R/W	Insert Extra Delay Between FS Bulk OUT Transactions (InsrtExtrFSBODI). Some FS devices are slow to receive Bulk OUT data and can get stuck when there are consecutive Bulk OUT transactions with short inter-transaction delays. This bit is used to control whether the host inserts extra delay between consecutive Bulk OUT transactions to a FS Endpoint. ■ 1'b0: Host doesn't insert extra delay between consecutive Bulk OUT transactions to a FS Endpoint. ■ 1'b1: Host inserts about 12us extra delay between consecutive Bulk OUT transactions to a FS Endpoint to work around the device issue. **Note:** Setting this bit to one will reduce the Bulk OUT transfer performance for most of the FS devices. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 11 1] **Exists:** Always
10:9	DTCT	R/W	Device Timeout Coarse Tuning (DTCT) This field is a Host mode parameter which determines how long the host waits for a response from device before considering a timeout. The controller first checks the DTCT value. If it is 0, then the timeout value is defined by the DTFT. If it is non-zero, then it uses the following timeout values: ■ 2'b00: 0 usec -> use DTFT value instead ■ 2'b01: 500 usec ■ 2'b10: 1.5 msec ■ 2'b11: 6.5 msec **Note:** When the system latency is larger than the programmed DTCT/DTFT value, if the host controller is not able to accept certain transactions on the bus (because of system bus delays), the controller may not release header credits which in turn can cause the host to report a transaction error. Therefore, program this value to be larger than your system delay. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 9 2] **Exists:** Always

Bits	Name	Memory Access	Description
8:0	DTFT	R/W	Device Timeout Fine Tuning (DTFT) This field is a Host mode parameter which determines how long the host waits for a response from device before considering a timeout. For the DTFT field to take effect, DTCT must be set to 2'b00. The DTFT value is the number of 125 MHz clocks * 256 to count before considering a device timeout. The minimum value of DTFT is 2. For example, if the mac3_clk is 125 MHz clk (8 ns period), this is calculated as follows: (DTFT value) * 256 * (8 ns) Quick Reference: ■ if DTFT = 0x2, 2*256*8 = 4usec timeout ■ if DTFT = 0x5, 5*256*8 = 10usec timeout ■ if DTFT = 0xA, 10*256*8 = 20usec timeout ■ if DTFT = 0x10, 16*256*8 = 32usec timeout ■ if DTFT = 0x19, 25*256*8 = 51usec timeout ■ if DTFT = 0x31, 49*256*8 = 100usec timeout ■ if DTFT = 0x62, 98*256*8 = 200usec timeout **Note:** When the system latency is larger than the programmed DTCT/DTFT value, if the host controller is not able to accept certain transactions on the bus (because of system bus delays), the controller may not release header credits which in turn can cause the host to report a transaction error. Therefore, program this value to be larger than your system delay. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL" 0 9] **Exists:** Always

1.2.13 GBUSERRADDR

- **Description:** Global Soc Bus Error Address Register
  When the AHB or AXI Master Bus returns an "Error" response, the "SoC Bus Error" is generated. In the Host mode, the host_system_err port indicates this condition. In addition, it is also indicated in the USBSTS.HSE field. In the Device mode, the GSTS.BusErrAddrVld field is the only indication of the SoC Bus Error.
  Note for AXI configuration:
  Due to the nature of AXI, it is possible that multiple AXI transactions are active at a time. The DWC_usb3 controller does not keep track of the start address of all outstanding transactions. Instead, it keeps track of the start address of the DMA transfer associated with all active transactions. It is this address that is reported in the GBUSERRADDR when a bus error occurs.
  For example, if the DWC_usb3 controller initiates a DMA transfer to write 1k of packet data starting at buffer address 0xABCD0000, and this DMA is broken up into multiple 256B bursts on the AXI, then if a bus error occurs on any of these associated AXI transfers, the GBUSERRADDR reflects the DMA start address of 0xABCD0000 regardless of which AXI transaction received the error.
- **Size:** 64 bits
- **Offset:** 0xc130
- **Exists:** DWC_USB3_SDWIDTH != 32 && DWC_USB3_32BIT_REGS == 0

Table 1-25 Fields for Register: GBUSERRADDR

Bits	Name	Memory Access	Description
63:0	BUSERRADDR	R	Bus Address This 64-bit register contains the address of the first bus that encountered a SoC bus error. It is valid when the GSTS.BusErrAddrVld field is 1. It can only be cleared by resetting the controller. **Note:** Only supported in AHB and AXI configurations. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.14 GBUSERRADDRLO

- **Description:** Gobal SoC Bus Error Address Register - Low
  This is an alternate register for the GBUSERRADDR register.
- **Size:** 32 bits
- **Offset:** 0xc130
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

Table 1-26 Fields for Register: GBUSERRADDRLO

Bits	Name	Memory Access	Description
31:0	BUSERRADDR	R	Bus Address - Low (BusAddrLo) This register contains the lower 32 bits of the first bus address that encountered a SoC bus error. It is valid when the GSTS.BusErrAddrVld field is 1. It can only be cleared by resetting the controller. **Note:** Only supported in AHB and AXI configurations. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.15 GBUSERRADDRHI

- **Description:** Gobal SoC Bus Error Address Register - High
  This is an alternate register for the GBUSERRADDR register.
- **Size:** 32 bits
- **Offset:** 0xc134
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

Table 1-27 Fields for Register: GBUSERRADDRHI

Bits	Name	Memory Access	Description
31:0	BUSERRADDR	R	Bus Address - High (BusAddrHi) This register contains the higher 32 bits of the first bus address that encountered a SoC bus error. It is valid when the GSTS.BusErrAddrVld field is 1. It can only be cleared by resetting the controller. **Note:** Only supported in AHB and AXI configurations. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.16 GPRTBIMAP

- **Description:** Global SS Port to Bus Instance Mapping Register
  This register specifies the SuperSpeed USB instance number to which each USB 3.0 port is connected. By default, USB 3.0 ports are evenly distributed among all SS USB instances.
  The software can program this register to specify how USB 3.0 ports are connected to SS USB instances.
  - Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  The reset value for each field in the GPRTBIMAP register is calculated as shown below:
  for (i=0; i<`DWC_USB3_HOST_NUM_U3_ROOT_PORTS; i=i+1)
  GPRTBIMAP[4*i+3:4*i] = i%`DWC_USB3_NUM_SS_USB_INSTANCES;
  Note:
  - GPRTBIMAP register is not applicable for USB 2.0-only mode.
- **Size:** 64 bits
- **Offset:** 0xc138
- **Exists:** DWC_USB3_SDWIDTH != 32 && DWC_USB3_32BIT_REGS == 0

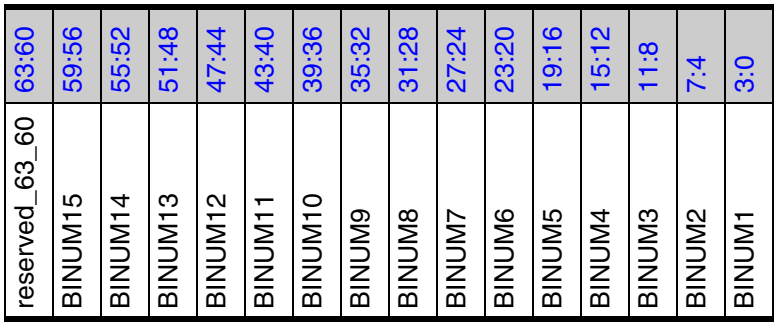


Table 1-28 Fields for Register: GPRTBIMAP

Bits	Name	Memory Access	Description
63:60	reserved_63_60	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 60 4] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
59:56	BINUM15	R/W	BINUM15: SS USB Instance Number for Port 15. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 56 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
55:52	BINUM14	R/W	BINUM14: SS USB Instance Number for Port 14. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 52 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
51:48	BINUM13	R/W	BINUM13: SS USB Instance Number for Port 13. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 48 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
47:44	BINUM12	R/W	BINUM12: SS USB Instance Number for Port 12. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 44 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
43:40	BINUM11	R/W	BINUM11: SS USB Instance Number for Port 11. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 40 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
39:36	BINUM10	R/W	BINUM10: SS USB Instance Number for Port 10. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 36 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
35:32	BINUM9	R/W	BINUM9: SS USB Instance Number for Port 9. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 32 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
31:28	BINUM8	R/W	BINUM8: SS USB Instance Number for Port 8. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27:24	BINUM7	R/W	BINUM7: SS USB Instance Number for Port 7. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:20	BINUM6	R/W	BINUM6: SS USB Instance Number for Port 6. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
19:16	BINUM5	R/W	BINUM5: SS USB Instance Number for Port 5. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM4	R/W	BINUM4: SS USB Instance Number for Port 4. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM3	R/W	BINUM3: SS USB Instance Number for Port 3. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_64_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
7:4	BINUM2	R/W	BINUM2: SS USB Instance Number for Port 2. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_64_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM1	R/W	BINUM1: SS USB Instance Number for Port 1. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_64_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.17 GPRTBIMAPLO

- **Description:** Global SS Port to Bus Instance Mapping Register - Low
  This is an alternate register for the GPRTBIMAP register.
  Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  For a configuration with number of USB 3.0 ports same as number of SS Bus Instances, do not remap during debug session. If you remap for some reason, then the debug host must be connected to a port which has a dedicated SS Bus Instance.
  For example, if DWC_USB3_NUM_U3_ROOT_PORTS =3 and DWC_USB3_NUM_SS_USB_INSTANCES=3, and software maps the first SS port to the first SS BI and the second/third port to the second BI, then the debug host can be connected to the first port only.
  Note: For reset values, refer to the corresponding values in the GPRTBIMAP register.
- **Size:** 32 bits
- **Offset:** 0xc138
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

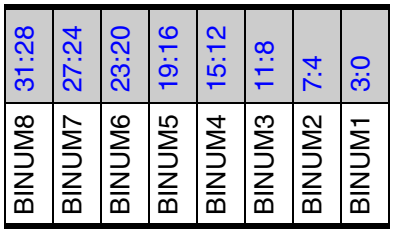


Table 1-29 Fields for Register: GPRTBIMAPLO

Bits	Name	Memory Access	Description
31:28	BINUM8	R/W	BINUM8: SS USB Instance Number for Port 8. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_lo_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27:24	BINUM7	R/W	BINUM7: SS USB Instance Number for Port 7. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_lo_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
23:20	BINUM6	R/W	BINUM6: SS USB Instance Number for Port 6. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_lo_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
19:16	BINUM5	R/W	BINUM5: SS USB Instance Number for Port 5. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_lo_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM4	R/W	BINUM4: SS USB Instance Number for Port 4. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_lo_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM3	R/W	BINUM3: SS USB Instance Number for Port 3. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_lo_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:4	BINUM2	R/W	BINUM2: SS USB Instance Number for Port 2. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_lo_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM1	R/W	BINUM1: SS USB Instance Number for Port 1. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_lo_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.18 GPRTBIMAPHI

- **Description:** Global SS Port to Bus Instance Mapping Register - High
  This is an alternate register for the GPRTBIMAP register.
  For a configuration with number of USB 3.0 ports same as number of SS Bus Instances, do not remap during debug session. If you remap for some reason, then the debug host must be connected to a port which has a dedicated SS Bus Instance.
  For example, if DWC_USB3_NUM_U3_ROOT_PORTS =3 and DWC_USB3_NUM_SS_USB_INSTANCES=3, and software maps the first SS port to the first SS BI and the second/third port to the second BI, then the debug host can be connected to the first port only.
  Note: For reset values, refer to the corresponding values in the GPRTBIMAP register.
- **Size:** 32 bits
- **Offset:** 0xc13c
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

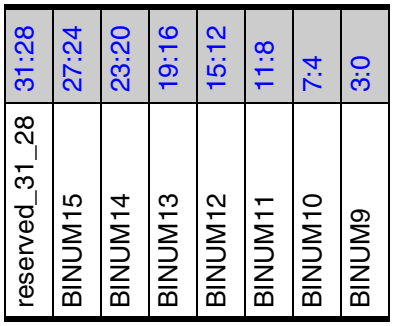


Table 1-30 Fields for Register: GPRTBIMAPHI

Bits	Name	Memory Access	Description
31:28	reserved_31_28	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
27:24	BINUM15	R/W	BINUM15: SS USB Instance Number for Port 15. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
23:20	BINUM14	R/W	BINUM14: SS USB Instance Number for Port 14. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
19:16	BINUM13	R/W	BINUM13: SS USB Instance Number for Port 13. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM12	R/W	BINUM12: SS USB Instance Number for Port 12. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM11	R/W	BINUM11: SS USB Instance Number for Port 11. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:4	BINUM10	R/W	BINUM10: SS USB Instance Number for Port 10. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM9	R/W	BINUM9: SS USB Instance Number for Port 9. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_hi_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.19 GHWPARAMS0

- **Description:** Global Hardware Parameters Register 0
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc140
- **Exists:** Always

31:24	23:16	15:8	7:6	5:3	2:0
ghwparams0_31_24	ghwparams0_23_16	ghwparams0_15_8	ghwparams0_7_6	ghwparams0_5_3	ghwparams0_2_0

Table 1-31 Fields for Register: GHWPARAMS0

Bits	Name	Memory Access	Description
31:24	ghwparams0_31_24	R	`DWC_USB3_AWIDTH **Value After Reset:** DWC_USB3_AWIDTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:16	ghwparams0_23_16	R	`DWC_USB3_SDWIDTH **Value After Reset:** DWC_USB3_SDWIDTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:8	ghwparams0_15_8	R	`DWC_USB3_MDWIDTH **Value After Reset:** DWC_USB3_MDWIDTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
7:6	ghwparams0_7_6	R	`DWC_USB3_SBUS_TYPE **Value After Reset:** DWC_USB3_SBUS_TYPE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
5:3	ghwparams0_5_3	R	`DWC_USB3_MBUS_TYPE **Value After Reset:** DWC_USB3_MBUS_TYPE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
2:0	ghwparams0_2_0	R	`DWC_USB3_MODE **Value After Reset:** DWC_USB3_MODE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.20 GHWPARAMS1

- **Description:** Global Hardware Parameters Register 1
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc144
- **Exists:** Always

31	30	29	28	27	26	25:24	23	22:21	20:15	14:12	11:9	8:6	5:3	2:0
ghwparams1_31	ghwparams1_30	ghwparams1_29	ghwparams1_28	ghwparams1_27	ghwparams1_26	ghwparams1_25_24	ghwparams1_23	ghwparams1_22_21	ghwparams1_20_15	ghwparams1_14_12	ghwparams1_11_9	ghwparams1_8_6	ghwparams1_5_3	ghwparams1_2_0

Table 1-32 Fields for Register: GHWPARAMS1

Bits	Name	Memory Access	Description
31	ghwparams1_31	R	`DWC_USB3_EN_DBC **Value After Reset:** DWC_USB3_EN_DBC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
30	ghwparams1_30	R	`DWC_USB3_RM_OPT_FEATURES **Value After Reset:** DWC_USB3_RM_OPT_FEATURES **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
29	ghwparams1_29	R	Reserved1 **Value After Reset:** DWC_USB3_SYNC_RST **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
28	ghwparams1_28	R	`DWC_USB3_RAM_BUS_CLKS_SYNC **Value After Reset:** DWC_USB3_RAM_BUS_CLKS_SYNC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27	ghwparams1_27	R	`DWC_USB3_MAC_RAM_CLKS_SYNC **Value After Reset:** DWC_USB3_MAC_RAM_CLKS_SYNC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
26	ghwparams1_26	R	`DWC_USB3_MAC_PHY_CLKS_SYNC **Value After Reset:** DWC_USB3_MAC_PHY_CLKS_SYNC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
25:24	ghwparams1_25_24	R	`DWC_USB3_EN_PWROPT **Value After Reset:** DWC_USB3_EN_PWROPT **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23	ghwparams1_23	R	`DWC_USB3_SPRAM_TYP **Value After Reset:** DWC_USB3_SPRAM_TYP **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
22:21	ghwparams1_22_21	R	`DWC_USB3_NUM_RAMS **Value After Reset:** DWC_USB3_NUM_RAMS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
20:15	ghwparams1_20_15	R	`DWC_USB3_DEVICE_NUM_INT For details on `DWC_USB3_DEVICE_NUM_INT, refer to <workspace>/src/DWC_usb3_params.v file. **Value After Reset:** DWC_USB3_DEVICE_NUM_INT **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
14:12	ghwparams1_14_12	R	`DWC_USB3_ASPACEWIDTH **Value After Reset:** DWC_USB3_ASPACEWIDTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:9	ghwparams1_11_9	R	`DWC_USB3_REQINFOWIDTH **Value After Reset:** DWC_USB3_REQINFOWIDTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
8:6	ghwparams1_8_6	R	`DWC_USB3_DATAINFOWIDTH **Value After Reset:** DWC_USB3_DATAINFOWIDTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
5:3	ghwparams1_5_3	R	`DWC_USB3_BURSTWIDTH-1 **Value After Reset:** expr [DWC_USB3_BURSTWIDTH] - 1 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
2:0	ghwparams1_2_0	R	`DWC_USB3_IDWIDTH-1 **Value After Reset:** expr [DWC_USB3_IDWIDTH] - 1 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.21 GHWPARAMS2

- **Description:** Global Hardware Parameters Register 2
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc148
- **Exists:** Always

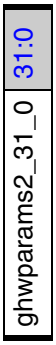


Table 1-33 Fields for Register: GHWPARAMS2

Bits	Name	Memory Access	Description
31:0	ghwparams2_31_0	R	`DWC_USB3_USERID **Value After Reset:** DWC_USB3_USERID **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.22 GHWPARAMS3

- **Description:** Global Hardware Parameters Register 3
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc14c
- **Exists:** Always

31	30:23	22:18	17:12	11	10	9:8	7:6	5:4	3:2	1:0
ghwparams3_31	ghwparams3_30_23	ghwparams3_22_18	ghwparams3_17_12	ghwparams3_11	ghwparams3_10	ghwparams3_9_8	ghwparams3_7_6	ghwparams3_5_4	ghwparams3_3_2	ghwparams3_1_0

Table 1-34 Fields for Register: GHWPARAMS3

Bits	Name	Memory Access	Description
31	ghwparams3_31	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
30:23	ghwparams3_30_23	R	`DWC_USB3_CACHE_TOTAL_XFER_RESOURCES **Value After Reset:** DWC_USB3_CACHE_TOTAL_XFER_RESOURCES **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
22:18	ghwparams3_22_18	R	`DWC_USB3_NUM_IN_EPS **Value After Reset:** DWC_USB3_NUM_IN_EPS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
17:12	ghwparams3_17_12	R	`DWC_USB3_NUM_EPS **Value After Reset:** DWC_USB3_NUM_EPS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11	ghwparams3_11	R	`DWC_USB3_ULPI_CARKIT **Value After Reset:** DWC_USB3_ULPI_CARKIT **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
10	ghwparams3_10	R	`DWC_USB3_VENDOR_CTL_INTERFACE **Value After Reset:** {DWC_USB3_VENDOR_CTL_INTERFACE == 1 ? 1 : 0} **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
9:8	ghwparams3_9_8	R	Reserved **Value After Reset:** DWC_USB3_I2C_INTERFACE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:6	ghwparams3_7_6	R	`DWC_USB3_HSPHY_DWIDTH **Value After Reset:** DWC_USB3_HSPHY_DWIDTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
5:4	ghwparams3_5_4	R	`DWC_USB3_FSPHY_INTERFACE **Value After Reset:** DWC_USB3_FSPHY_INTERFACE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
3:2	ghwparams3_3_2	R	`DWC_USB3_HSPHY_INTERFACE **Value After Reset:** DWC_USB3_HSPHY_INTERFACE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
1:0	ghwparams3_1_0	R	`DWC_USB3_SSPHY_INTERFACE **Value After Reset:** DWC_USB3_SSPHY_INTERFACE **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.23 GHWPARAMS4

- **Description:** Global Hardware Parameters Register 4
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc150
- **Exists:** Always

31:28	27:24	23	22	21	20:17	16:13	12	11	10:9	8:7	6	5:0
ghwparams4_31_28	ghwparams4_27_24	ghwparams4_23	ghwparams4_22	ghwparams4_21	ghwparams4_20_17	ghwparams4_16_13	ghwparams4_12	ghwparams4_11	ghwparams4_10_9	ghwparams4_8_7	ghwparams4_6	ghwparams4_5_0

Table 1-35 Fields for Register: GHWPARAMS4

Bits	Name	Memory Access	Description
31:28	ghwparams4_31_28	R	`DWC_USB3_BMU_LSP_DEPTH **Value After Reset:** DWC_USB3_BMU_LSP_DEPTH **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
27:24	ghwparams4_27_24	R	`DWC_USB3_BMU_PTL_DEPTH-1 **Value After Reset:** expr [DWC_USB3_BMU_PTL_DEPTH] - 1 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
23	ghwparams4_23	R	`DWC_USB3_EN_ISOC_SUPT **Value After Reset:** DWC_USB3_EN_ISOC_SUPT **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
22	ghwparams4_22	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
21	ghwparams4_21	R	`DWC_USB3_EXT_BUFF_CONTROL **Value After Reset:** DWC_USB3_EXT_BUFF_CONTROL **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
20:17	ghwparams4_20_17	R	`DWC_USB3_NUM_SS_USB_INSTANCES **Value After Reset:** DWC_USB3_NUM_SS_USB_INSTANCES **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
16:13	ghwparams4_16_13	R	`DWC_USB3_HIBER_SCRATCHBUFS Number of external scratchpad buffers the controller requires to save its internal state in the device mode. Each buffer is assumed to be 4KB. The scratchpad buffer array must have this many buffer pointers. **Value After Reset:** DWC_USB3_HIBER_SCRATCHBUFS **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
12	ghwparams4_12	R	Reserved **Value After Reset:** {DWC_USB3_NUM_SSIC_PORTS == 0 ? 0 : 1} **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
11	ghwparams4_11	R	Reserved **Value After Reset:** {DWC_USB3_NUM_SSIC_PORTS == 0 ? 0 : DWC_USB3_SSIC_NON_SNPS_MPHY} **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
10:9	ghwparams4_10_9	R	Reserved **Value After Reset:** {DWC_USB3_NUM_SSIC_PORTS == 0 ? 0 : DWC_USB3_SSIC_GEAR} **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead
8:7	ghwparams4_8_7	R	Reserved **Value After Reset:** {DWC_USB3_NUM_SSIC_PORTS == 0 ? 0 : DWC_USB3_SSIC_NUM_LANE==4 ? 0 : DWC_USB3_SSIC_NUM_LANE} **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead
6	ghwparams4_6	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
5:0	ghwparams4_5_0	R	`DWC_USB3_CACHE_TRBS_PER_TRANSFER **Value After Reset:** DWC_USB3_CACHE_TRBS_PER_TRANSFER **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3f **Write Constraint:** writeAsRead

1.2.24 GHWPARAMS5

- **Description:** Global Hardware Parameters Register 5
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc154
- **Exists:** Always

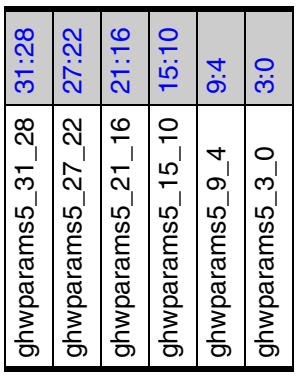


Table 1-36 Fields for Register: GHWPARAMS5

Bits	Name	Memory Access	Description
31:28	ghwparams5_31_28	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27:22	ghwparams5_27_22	R	`DWC_USB3_DFQ_FIFO_DEPTH **Value After Reset:** DWC_USB3_DFQ_FIFO_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
21:16	ghwparams5_21_16	R	`DWC_USB3_DWQ_FIFO_DEPTH **Value After Reset:** DWC_USB3_DWQ_FIFO_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
15:10	ghwparams5_15_10	R	`DWC_USB3_TXQ_FIFO_DEPTH **Value After Reset:** DWC_USB3_TXQ_FIFO_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
9:4	ghwparams5_9_4	R	`DWC_USB3_RXQ_FIFO_DEPTH **Value After Reset:** DWC_USB3_RXQ_FIFO_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	ghwparams5_3_0	R	`DWC_USB3_BMU_BUSGM_DEPTH **Value After Reset:** DWC_USB3_BMU_BUSGM_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.25 GHWPARAMS6

- **Description:** Global Hardware Parameters Register 6
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc158
- **Exists:** Always

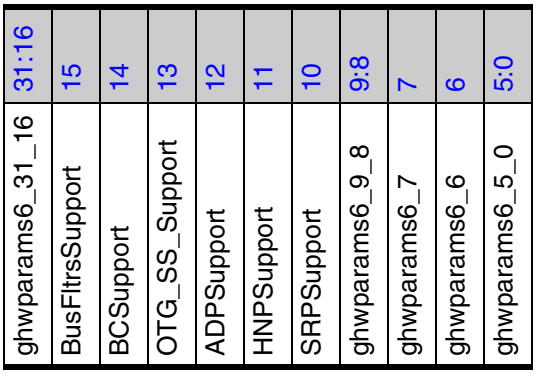


Table 1-37 Fields for Register: GHWPARAMS6

Bits	Name	Memory Access	Description
31:16	ghwparams6_31_16	R	`DWC_USB3_RAM0_DEPTH **Value After Reset:** DWC_USB3_RAM0_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15	BusFltrsSupport	R	`DWC_USB3_EN_BUS_FILTERS **Value After Reset:** DWC_USB3_EN_BUS_FILTERS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
14	BCSupport	R	`DWC_USB3_EN_BC **Value After Reset:** DWC_USB3_EN_BC **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
13	OTG_SS_Support	R	Reserved **Value After Reset:** DWC_USB3_EN_OTG_SS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
12	ADPSupport	R	`DWC_USB3_EN_ADP **Value After Reset:** DWC_USB3_EN_ADP **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11	HNPSupport	R	Reserved **Value After Reset:** {DWC_USB3_MODE==2 ? [DWC_USB3_EN_OTG!=0] : 0} **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
10	SRPSupport	R	Reserved **Value After Reset:** {DWC_USB3_EN_OTG!=0 ? 1 : 0} **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
9:8	ghwparams6_9_8	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7	ghwparams6_7	R	`DWC_USB3_EN_FPGA **Value After Reset:** DWC_USB3_EN_FPGA **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
6	ghwparams6_6	R	`DWC_USB3_EN_DBG_PORTS **Value After Reset:** DWC_USB3_EN_DBG_PORTS **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
5:0	ghwparams6_5_0	R	`DWC_USB3_PSQ_FIFO_DEPTH **Value After Reset:** DWC_USB3_PSQ_FIFO_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.26 GHWPARAMS7

- **Description:** Global Hardware Parameters Register 7
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc15c
- **Exists:** Always

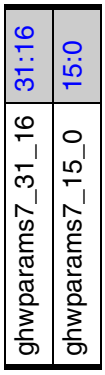


Table 1-38 Fields for Register: GHWPARAMS7

Bits	Name	Memory Access	Description
31:16	ghwparams7_31_16	R	`DWC_USB3_RAM2_DEPTH **Value After Reset:** DWC_USB3_RAM2_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:0	ghwparams7_15_0	R	`DWC_USB3_RAM1_DEPTH **Value After Reset:** DWC_USB3_RAM1_DEPTH **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.27 GDBGFIFOSPACE

- **Description:** Global Debug Queue/FIFO Space Available Register
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc160
- **Exists:** Always

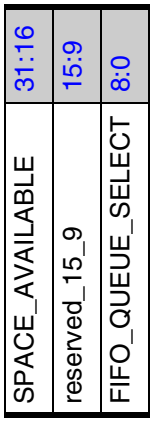


Table 1-39 Fields for Register: GDBGFIFOSPACE

Bits	Name	Memory Access	Description
31:16	SPACE_AVAILABLE	R	SPACE_AVAILABLE **Value After Reset:** =mpformat "0x%x" [{DWC_USB3_MODE == 1 ? DWC_USB3_HOST_TXFIFO_DEPTH_PER_FSLS_INSTANCE : DWC_USB3_DEV_TXF0_DEPTH}] **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffff
15:9	reserved_15_9	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x7f

Bits	Name	Memory Access	Description
8:0	FIFO_QUEUE_SELECT	R/W	FIFO/Queue Select (or) Port-Select ■ FIFO/Queue Select[8:5] indicates the FIFO/Queue Type ■ FIFO/Queue Select[4:0] indicates the FIFO/Queue Number For example, 9'b0_0010_0001 refers to RxFIFO_1 and 9'b0_0101_1110 refers to TxReqQ_30. ■ 9'b0_0001_1111 to 9'b0_0000_0000: TxFIFO_31 to TxFIFO_0 ■ 9'b0_0011_1111 to 9'b0_0010_0000: RxFIFO_31 to RxFIFO_0 ■ 9'b0_0101_1111 to 9'b0_0100_0000: TxReqQ_31 to TxReqQ_0 ■ 9'b0_0111_1111 to 9'b0_0110_0000: RxReqQ_31 to RxReqQ_0 ■ 9'b0_1001_1111 to 9'b0_1000_0000: RxInfoQ_31 to RxInfoQ_0 ■ 9'b0_1010_0000: DescFetchQ_0 (for backwards compatibility) ■ 9'b0_1010_0001: EventQ_0 (for backwards compatibility) ■ 9'b0_1010_0010: ProtocolStatusQ_0 ■ 9'b0_1101_1111 to 9'b0_1110_0000: DescFetchQ_31 to DescFetchQ_0 ■ 9'b0_1111_1111 to 9'b0_1110_0000: WriteBack/EventQ_31 to WriteBack/EventQ_0 ■ 9'b1_0000_0111 to 9'b1_0000_0000: AuxEventQ_7 to AuxEventQ_0 (if EN_SEPARATE_DESC_QUEUES=1) Port-Select[3:0] selects the port-number when accessing GDBGLTSSM register. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1ff

1.2.28 GDBGLTSSM

- **Description:** Global Debug LTSSM Register
  In multi-port host configuration, the port-number is defined by Port-Select[3:0] field in the GDBGFIFOSPACE register.
  Note:
  - GDBGLTSSM register is not applicable for USB 2.0-only mode.
  - Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc164
- **Exists:** Always

31	30	29	28	27	26	25:22	21:18	17	16	15	14	13:11	10:9	8	7:6	5:3	2	1	0
reserved_31_31	RxElecidle	X3_XS_SWAPPING	X3_DS_HOST_SHUTDOWN	PRTDIRECTION	LTDBTIMEOUT	LTDBLINKSTATE	LTDBSUBSTATE	ELASTICBUFFERMODE	TXELECLDLE	RXPOLARITY	TxDetRxLoopback	LTDBPhyCmdState	POWERDOWN	RXEQTRAIN	TXDEEMPHASIS	LTDBClkState	TXSWING	RXTERMINATION	TXONESZEROS

Table 1-40 Fields for Register: GDBGLTSSM

Bits	Name	Memory Access	Description
31	reserved_31_31	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
30	RxElecidle	R	RxElecidle For description of RxElecIdle, see table 5-4, "Status Interface Signals" of the *PIPE3 Specification.* **Value After Reset:** 0x1 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x0

Bits	Name	Memory Access	Description
29	X3_XS_SWAPPING	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
28	X3_DS_HOST_SHUTDOWN	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
27	PRTDIRECTION	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
26	LTDBTIMEOUT	R	LTDB Timeout (LTDBTimeout) **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
25:22	LTDBLINKSTATE	R	LTDB Link State (LTDBLinkState) **Value After Reset:** 0x4 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xb
21:18	LTDBSUBSTATE	R	LTDB Sub-State (LTDBSubState) **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xf
17	ELASTICBUFFERMODE	R	Elastic Buffer Mode (ElasticBufferMode) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUSB3PIPECTL_INIT" 0 1] **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1

Bits	Name	Memory Access	Description
16	TXELECLDLE	R	Tx Elec Idle (TxElecIdle) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x1 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x0
15	RXPOLARITY	R	Rx Polarity (RxPolarity) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
14	TxDetRxLoopback	R	Tx Detect Rx/Loopback (TxDetRxLoopback) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
13:11	LTDBPhyCmdState	R	LTSSM PHY command State (LTDBPhyCmdState) ■ 000: PHY_IDLE (PHY command state is in IDLE. No PHY request pending) ■ 001: PHY_DET (Request to start Receiver detection) ■ 010: PHY_DET_3 (Wait for Phy_Status (Receiver detection)) ■ 011: PHY_PWR_DLY (Delay Pipe3_PowerDown P0 -> P1/P2/P3 request) ■ 100: PHY_PWR_A (Delay for internal logic) ■ 101: PHY_PWR_B (Wait for Phy_Status(Power state change request)) **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x7
10:9	POWERDOWN	R	POWERDOWN (PowerDown) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x2 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1

Bits	Name	Memory Access	Description
8	RXEQTRAIN	R	RxEq Train For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
7:6	TXDEEMPHASIS	R	TXDEEMPHASIS (TxDeemphasis) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x1 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x2
5:3	LTDBClkState	R	LTSSM Clock State (LTDBClkState) In multi-port host configuration, the port number is defined by Port-Select[3:0] field in the GDBGFIFOSPACE register. Note: GDBGLTSSM register is not applicable for USB 2.0-only mode. ■ 000: CLK_NORM (PHY is in non-P3 state and PCLK is running) ■ 001: CLK_TO_P3 (P3 entry request to PHY); ■ 010: CLK_WAIT1 (Wait for Phy_Status (P3 request)); ■ 011: CLK_P3 (PHY is in P3 and PCLK is not running); ■ 100: CLK_TO_P0 (P3 exit request to PHY); ■ 101: CLK_WAIT2 (Wait for Phy_Status (P3 exit request)) **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x7
2	TXSWING	R	Tx Swing (TxSwing) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1

Bits	Name	Memory Access	Description
1	RXTERMINATION	R	Rx Termination (RxTermination) For details on `DWC_USB3_PIPE_RXTERM_RESET_VAL, refer to <workspace>/src/DWC_usb3_params.v **Value After Reset:** =mpformat 0x%x [get_configuration_parameter DWC_USB3_PIPE_RXTERM_RESET_VAL] **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
0	TXONESZEROS	R	Tx Ones/Zeros (TxOnesZeros) For field definition, refer to Table 5-3 of the *PIPE3 specification*. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1

1.2.29 GDBGLNMCC

- **Description:** Global Debug LNMCC Register
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc168
- **Exists:** Always

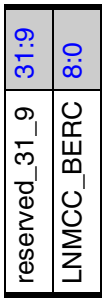


Table 1-41 Fields for Register: GDBGLNMCC

Bits	Name	Memory Access	Description
31:9	reserved_31_9	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x7fffff
8:0	LNMCC_BERC	R	This field indicates the bit error rate information for the port selected in the GDBGFIFOSPACE.PortSelect field. This field is for debug purposes only. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1ff **Write Constraint:** writeAsRead

1.2.30 GDBGBMU

- **Description:** Global Debug BMU Register
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc16c
- **Exists:** Always

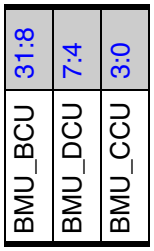


Table 1-42 Fields for Register: GDBGBMU

Bits	Name	Memory Access	Description
31:8	BMU_BCU	R	BMU_BCU Debug information **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffffff
7:4	BMU_DCU	R	BMU_DCU Debug information **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf
3:0	BMU_CCU	R	BMU_CCU Debug information **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xf

1.2.31 GDBGLSPMUX_HST

- **Description:** Global Debug LSP MUX Register - Host
  This register is for internal use only.
  If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT is enabled during controller configuration, then the default values readout is X (Undefined).
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc170
- **Exists:** Always

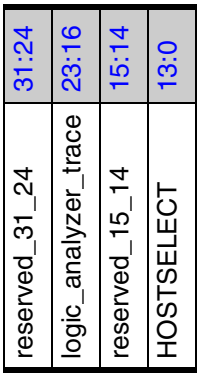


Table 1-43 Fields for Register: GDBGLSPMUX_HST

Bits	Name	Memory Access	Description
31:24	reserved_31_24	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xff
23:16	logic_analyzer_trace	R/W	logic_analyzer_trace Port MUX Select Currently only bits[21:16] are used. For details on how the mux controls the debug traces, refer to the "assign logic_analyzer_trace =" code section in the DWC_usb3.v file. A value of 6'h3F drives 0s on the logic_analyzer_trace signal. If you plan to OR (instead using a mux) this signal with other trace signals in your system to generate a common trace signal, you can use this feature. **Value After Reset:** 0x3f **Exists:** Always **Testable:** untestable **Reset Mask:** 0xc0

Bits	Name	Memory Access	Description
15:14	reserved_15_14	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x3
13:0	HOSTSELECT	R/W	Device LSP Select Selects the LSP debug information presented in the GDBGLSP register in host mode. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x3fff

1.2.32 GDBGLSPMUX_DEV

- **Description:** Global Debug LSP MUX Register - Device
  This register is for internal use only.
  If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT is enabled during controller configuration, then the default values readout is X (Undefined).
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc170
- **Exists:** Always

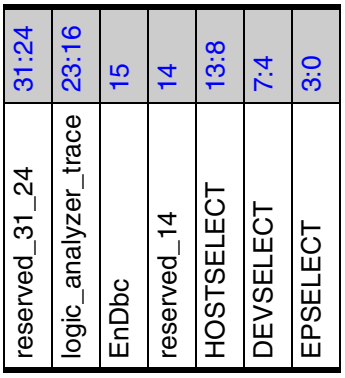


Table 1-44 Fields for Register: GDBGLSPMUX_DEV

Bits	Name	Memory Access	Description
31:24	reserved_31_24	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xff
23:16	logic_analyzer_trace	R/W	Logic Analyzer Trace Port MUX Select Currently only bits[21:16] are used. For details on how the mux controls the debug traces, refer to "assign logic_analyzer_trace =" code section in the DWC_usb3.v file. A value of 6'h3F drives "0"s on the logic_analyzer_trace signal. If you plan to OR (instead using a mux) this signal with other trace signals in your system to generate a common trace signal, you can use this feature. **Value After Reset:** 0x3f **Exists:** Always **Testable:** untestable **Reset Mask:** 0xc0

Bits	Name	Memory Access	Description
15	EnDbc	R/W	Enable debugging of Debug capablity LSP in Host mode. Use HostSelect to select DbC LSP debug information presented in the GDBGLSP register. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
14	reserved_14	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1
13:8	HOSTSELECT	R/W	Device LSP Select **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x3f
7:4	DEVSELECT	R/W	Device LSP Select Selects the LSP debug information presented in the GDBGLSP register in device mode. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xf
3:0	EPSELECT	R/W	Device Endpoint Select Selects the Endpoint debug information presented in the GDBGEPINFO registers in device mode. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xf

1.2.33 GDBGLSP

- **Description:** Global Debug LSP Register
  This register is for internal debug purposes only.
  This register is for internal use only.
  If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT is enabled during controller configuration, then the default values readout is X (Undefined).
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc174
- **Exists:** Always

Table 1-45 Fields for Register: GDBGLSP

Bits	Name	Memory Access	Description
31:0	LSPDEBUG	R	LSP Debug Information **Value After Reset:** =mpformat "0x%x" [{DWC_USB3_MODE == 1 ? DWC_USB3_GDBGLSP_RESET_CC : 0x0}] **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffffffff

1.2.34 GDBGEPINFO0

- **Description:** Global Debug Endpoint Information Register 0
  This register is for internal use only.
  If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT is enabled during controller configuration, then the default values readout is X (Undefined).
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc178
- **Exists:** Always

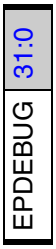


Table 1-46 Fields for Register: GDBGEPINFO0

Bits	Name	Memory Access	Description
31:0	EPDEBUG	R	Endpoint Debug Information, bits[31:0] **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffffffff

1.2.35 GDBGEPINFO1

- **Description:** Global Debug Endpoint Information Register 1
  This register is for internal use only.
  If DWC_USB3_PRESERVE_LOGIC_ANALYZER_SELECT is enabled during controller configuration, then the default values readout is X (Undefined).
  Bit Bash test should not be done on this debug register.
- **Size:** 32 bits
- **Offset:** 0xc17c
- **Exists:** Always

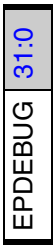
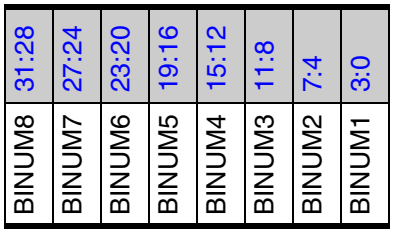


Table 1-47 Fields for Register: GDBGEPINFO1

Bits	Name	Memory Access	Description
31:0	EPDEBUG	R	Endpoint Debug Information, bits[63:32] **Value After Reset:** =mpformat "0x%x" [{DWC_USB3_MODE == 1 ? 0x0 : 0x800000}] **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffffffff

1.2.36 GPRTBIMAP_HSLO

- **Description:** Global High-Speed Port to Bus Instance Mapping Register - Low
  This is an alternate register for the GPRTBIMAP_HS register.
  - Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  Note: For reset values, refer to the corresponding values in the GPRTBIMAP_HS register.
- **Size:** 32 bits
- **Offset:** 0xc180
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1



31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
BINUM8	BINUM7	BINUM6	BINUM5	BINUM4	BINUM3	BINUM2	BINUM1

Table 1-48 Fields for Register: GPRTBIMAP_HSLO

Bits	Name	Memory Access	Description
31:28	BINUM8	R/W	BINUM8: HS USB Instance Number for Port 8. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27:24	BINUM7	R/W	BINUM7: HS USB Instance Number for Port 7. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:20	BINUM6	R/W	BINUM6 USB Instance Number for Port 6. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
19:16	BINUM5	R/W	BINUM5: HS USB Instance Number for Port 5. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM4	R/W	BINUM4: HS USB Instance Number for Port 4. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM3	R/W	BINUM3: HS USB Instance Number for Port 3. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:4	BINUM2	R/W	BINUM2: HS USB Instance Number for Port 2. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM1	R/W	BINUM1: HS USB Instance Number for Port 1. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_lo_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.37 GPRTBIMAP_HS

- **Description:** Global High-Speed Port to Bus Instance Mapping Register
  This register specifies the High Speed USB instance number to which each USB 3.0 port is connected. By default, USB 3.0 ports are evenly distributed among all HS USB instances. The software can program this register to specify how USB 3.0 ports are connected to HS USB instances.
  - Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  The *Reset Value* for each field in the GPRTBIMAP_HS register is calculated as shown below: for (i=0; i<`DWC_USB3_HOST_NUM_U2_ROOT_PORTS; i=i+1) GPRTBIMAP_HS[4*i+3:4*i] = i%`DWC_USB3_NUM_HS_USB_INSTANCES;
- **Size:** 64 bits
- **Offset:** 0xc180
- **Exists:** DWC_USB3_SDWIDTH != 32 && DWC_USB3_32BIT_REGS == 0

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
reserved_63_60	BINUM15	BINUM14	BINUM13	BINUM12	BINUM11	BINUM10	BINUM9	BINUM8	BINUM7	BINUM6	BINUM5	BINUM4	BINUM3	BINUM2	BINUM1

Table 1-49 Fields for Register: GPRTBIMAP_HS

Bits	Name	Memory Access	Description
63:60	reserved_63_60	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 60 4] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
59:56	BINUM15	R/W	BINUM15: HS USB Instance Number for Port 15. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 56 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
55:52	BINUM14	R/W	BINUM14: HS USB Instance Number for Port 14. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_64_val]" 52 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
51:48	BINUM13	R/W	BINUM13: HS USB Instance Number for Port 13. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_64_val]" 48 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
47:44	BINUM12	R/W	BINUM12: HS USB Instance Number for Port 12. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_64_val]" 44 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
43:40	BINUM11	R/W	BINUM11: HS USB Instance Number for Port 11. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_64_val]" 40 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
39:36	BINUM10	R/W	BINUM10: HS USB Instance Number for Port 10. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_64_val]" 36 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
35:32	BINUM9	R/W	BINUM9: HS USB Instance Number for Port 9. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_64_val]" 32 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
31:28	BINUM8	R/W	BINUM8: HS USB Instance Number for Port 8. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27:24	BINUM7	R/W	BINUM7: HS USB Instance Number for Port 7. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:20	BINUM6	R/W	BINUM6: HS USB Instance Number for Port 6. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
19:16	BINUM5	R/W	BINUM5: HS USB Instance Number for Port 5. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM4	R/W	BINUM4: HS USB Instance Number for Port 4. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM3	R/W	BINUM3: HS USB Instance Number for Port 3. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
7:4	BINUM2	R/W	BINUM2: HS USB Instance Number for Port 2. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM1	R/W	BINUM1: HS USB Instance Number for Port 1. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_64_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.38 GPRTBIMAP_HSHI

- **Description:** Global High-Speed Port to Bus Instance Mapping Register - High
  This is an alternate register for the GPRTBIMAP_HS register.
  - Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  Note: For reset values, refer to the corresponding values in the GPRTBIMAP register.
- **Size:** 32 bits
- **Offset:** 0xc184
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

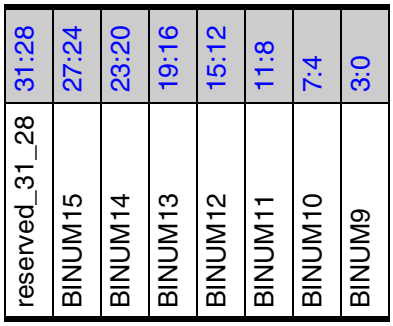


Table 1-50 Fields for Register: GPRTBIMAP_HSHI

Bits	Name	Memory Access	Description
31:28	reserved_31_28	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_hi_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
27:24	BINUM15	R/W	BINUM15: HS USB Instance Number for Port 15. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_hi_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:20	BINUM14	R/W	BINUM14: HS USB Instance Number for Port 14. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_HS_hi_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
19:16	BINUM13	R/W	BINUM13: HS USB Instance Number for Port 13. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_hi_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM12	R/W	BINUM12: HS USB Instance Number for Port 12. SApplication-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_hi_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM11	R/W	BINUM11: HS USB Instance Number for 11. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_hi_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:4	BINUM10	R/W	BINUM10: HS USB Instance Number for Port 10. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_hi_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM9	R/W	BINUM9: HS USB Instance Number for Port 9. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_HS_hi_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.39 GPRTBIMAP_FS

- **Description:** Global Full-Speed Port to Bus Instance Mapping Register
  This register specifies the Full Speed USB instance number to which each USB 3.0 port is connected. By default, USB 3.0 ports are evenly distributed among all FS USB instances.
  Software can program this register to specify how USB 3.0 ports are connected to FS USB instances.
  - Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  The *Reset Value* for each field in the GPRTBIMAP_FS register is calculated as shown below: for (i=0; i<`DWC_USB3_HOST_NUM_U2_ROOT_PORTS; i=i+1) GPRTBIMAP_FS[4*i+3:4*i] = i%`DWC_USB3_NUM_FSLS_USB_INSTANCES;
- **Size:** 64 bits
- **Offset:** 0xc188
- **Exists:** DWC_USB3_SDWIDTH != 32 && DWC_USB3_32BIT_REGS == 0

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
reserved_63_60	BINUM15	BINUM14	BINUM13	BINUM12	BINUM11	BINUM10	BINUM9	BINUM8	BINUM7	BINUM6	BINUM5	BINUM4	BINUM3	BINUM2	BINUM1

Table 1-51 Fields for Register: GPRTBIMAP_FS

Bits	Name	Memory Access	Description
63:60	reserved_63_60	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_64_val]" 60 4] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
59:56	BINUM15	R/W	BINUM15: FS USB Instance Number for Port 15. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_64_val]" 56 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
55:52	BINUM14	R/W	BINUM14: FS USB Instance Number for Port 14. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 52 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
51:48	BINUM13	R/W	BINUM13: FS USB Instance Number for Port 13. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 48 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
47:44	BINUM12	R/W	BINUM12: FS USB Instance Number for Port 12. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 44 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
43:40	BINUM11	R/W	BINUM11: FS USB Instance Number for Port 11. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 40 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
39:36	BINUM10	R/W	BINUM10: FS USB Instance Number for Port 10. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 36 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
35:32	BINUM9	R/W	BINUM9: FS USB Instance Number for Port 9. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 32 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
31:28	BINUM8	R/W	BINUM8: FS USB Instance Number for Port 8. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27:24	BINUM7	R/W	BINUM7: FS USB Instance Number for Port 7. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:20	BINUM6	R/W	BINUM6: FS USB Instance Number for Port 6. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
19:16	BINUM5	R/W	BINUM5: FS USB Instance Number for Port 5. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM4	R/W	BINUM4: FS USB Instance Number for Port 4. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM3	R/W	BINUM3: FS USB Instance Number for Port 3. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
7:4	BINUM2	R/W	BINUM2: FS USB Instance Number for Port 2. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM1	R/W	BINUM1: FS USB Instance Number for Port 1. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_64_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.40 GPRTBIMAP_FSLO

- **Description:** Global Full-Speed Port to Bus Instance Mapping Register - Low
  This is an alternate register for the GPRTBIMAP_FS register.
  - Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  Note: For reset values, refer to the corresponding values in the GPRTBIMAP_FS register.
- **Size:** 32 bits
- **Offset:** 0xc188
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
BINUM8	BINUM7	BINUM6	BINUM5	BINUM4	BINUM3	BINUM2	BINUM1

Table 1-52 Fields for Register: GPRTBIMAP_FSLO

Bits	Name	Memory Access	Description
31:28	BINUM8	R/W	BINUM8: FS USB Instance Number for Port 8. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_lo_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
27:24	BINUM7	R/W	BINUM7: FS USB Instance Number for Port 7. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_lo_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:20	BINUM6	R/W	BINUM6: FS USB Instance Number for Port 6. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_lo_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
19:16	BINUM5	R/W	BINUM5: FS USB Instance Number for Port 5. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_lo_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM4	R/W	BINUM4: FS USB Instance Number for Port 4. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_lo_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM3	R/W	BINUM3: FS USB Instance Number for Port 3. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_lo_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:4	BINUM2	R/W	BINUM2: FS USB Instance Number for Port 2. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_lo_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM1	R/W	BINUM1: FS USB Instance Number for Port 1. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val= [get_GPRTBIMAP_FS_lo_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.41 GPRTBIMAP_FSHI

- **Description:** Global Full-Speed Port to Bus Instance Mapping Register - High
  This is an alternate register for the GPRTBIMAP_FS register.
  - Register fields are read-write with respect to number of port instantiated. writeAsRead constraint is added to limit side effects for unused fields.
  Note: For reset values, refer to the corresponding values in the GPRTBIMAP_FS register.
- **Size:** 32 bits
- **Offset:** 0xc18c
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

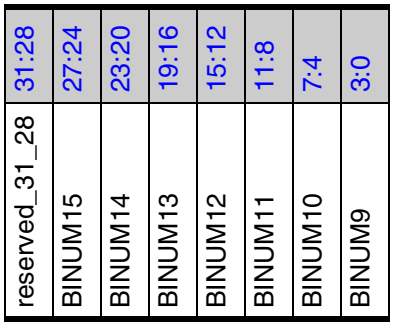


Table 1-53 Fields for Register: GPRTBIMAP_FSHI

Bits	Name	Memory Access	Description
31:28	reserved_31_28	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 28 4] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xf **Write Constraint:** writeAsRead
27:24	BINUM15	R/W	BINUM15: FS USB Instance Number for Port 15. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 24 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
23:20	BINUM14	R/W	BINUM14: FS USB Instance Number for Port 14. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 20 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
19:16	BINUM13	R/W	BINUM13: FS USB Instance Number for Port 13. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 16 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:12	BINUM12	R/W	BINUM12: FS USB Instance Number for Port 12. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 12 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11:8	BINUM11	R/W	BINUM11: FS USB Instance Number for Port 11. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 8 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:4	BINUM10	R/W	BINUM10: FS USB Instance Number for Port 10. Application-programmable ID field **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 4 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
3:0	BINUM9	R/W	BINUM9: FS USB Instance Number for Port 9. Application-programmable ID field. **Value After Reset:** =mpformat "0x%x" [get_field_val_b "val=[get_GPRTBIMAP_FS_hi_val]" 0 4] **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.42 GERRINJCTL_1

- **Description:** Global Error Injection 1 Control Register
  This register is reserved for future use.
- **Size:** 32 bits
- **Offset:** 0xc194
- **Exists:** Always

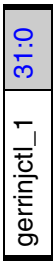


Table 1-54 Fields for Register: GERRINJCTL_1

Bits	Name	Memory Access	Description
31:0	gerrinjctl_1	R/W	Field for Error Injection Control 1 This field is reserved for future use. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable

1.2.43 GERRINJCTL_2

- **Description:** Global Error Injection 2 Control Register
  This register is reserved for future use.
- **Size:** 32 bits
- **Offset:** 0xc198
- **Exists:** Always

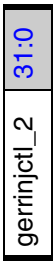


Table 1-55 Fields for Register: GERRINJCTL_2

Bits	Name	Memory Access	Description
31:0	gerrinjctl_2	R/W	Field for Error Injection control 2 This field is reserved for future use. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable

1.2.44 GUCTL2

- **Description:** Global User Control Register 2:
  This register provides a few options for the software to control the controller behavior in the Host and device mode. Most of the options are used to improve inter-operability with different hosts and devices.
- **Size:** 32 bits
- **Offset:** 0xc19c
- **Exists:** Always

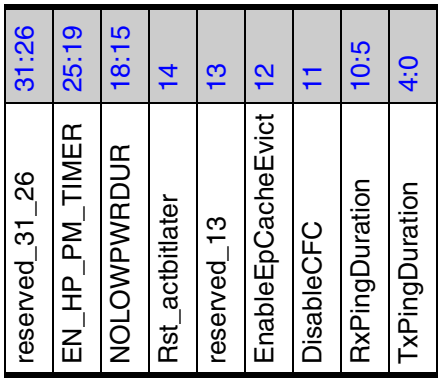


Table 1-56 Fields for Register: GUCTL2

Bits	Name	Memory Access	Description
31:26	reserved_31_26	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead
25:19	EN_HP_PM_TIMER	R/W	This register field is used to set new HP and PM timers. ■ To enable PM timer, set GUCTL2[19] bit as 1. ■ To enable HP timer, set GUCTL2[20] bit as 1. Default value of HP timer is 4us when HP PM timer is not enabled; when new HP timer is enabled default value is 12us. Use GUCTL2[25:21] to specify HP timer value in microseconds. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 19 7] **Exists:** Always

Bits	Name	Memory Access	Description
18:15	NOLOWPWRDUR	R/W	No Low Power Duration (NOLOWPWRDUR) This bit is applicable for device mode only and is ignored in host mode. After starting a transfer on a SS ISOC endpoint, the application must program these bits to prevent the device to lose frame synchronization over a period of time. Based on this count-down counter, the device will wake itself from U1/U2 low power states. After entering to U0 state and receiving two ITPs (which will sync-up the host and the device), U1/U2 low power entry is allowed. Each count represents the duration in terms of milliseconds. For example, a value of 3 represents 3ms. **Note:** ■ To disable this feature, set this field to 4'b0. ■ These bits are applicable only in device mode and ignored in host mode. ■ Some xHCI hosts do not send ITPs when performing ISOC transfers when the link enters U1/U2 low power states. This causes the device to lose frame synchronization over a period of time resulting in ISOC packets being dropped. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 15 4] **Exists:** Always
14	Rst_actbitlater	R/W	Enable clearing of the command active bit for the ENDXFER command after the command execution is completed. This bit is valid in device mode only. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 14 1] **Exists:** Always
13	reserved_13	R	Reserved for future use **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 13 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
12	EnableEpCacheEvict	R/W	Enable Evicting Endpoint cache after Flow Control for bulk endpoints. In 3.00a release, a performance enhancement was done to keep the non-stream capable bulk IN endpoint in cache after flow control. Setting this bit will disable this enhancement. This should be set only for debug purpose. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 12 1] **Exists:** Always

Bits	Name	Memory Access	Description
11	DisableCFC	R/W	Disable xHCI Errata Feature Contiguous Frame ID Capability This field controls the xHCI Errata feature Contiguous FrameID capability. When set, the xHCI HCCPARAMS1 bit 11 will be set to 0 indicating that CFC is not supported. Disable this feature only if your application cannot tolerate Misssed Service Error events for Isoc transfers, and your system latencies are large to cause Missed Service errors even if the software is following the Isochronous Thresholding rules. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 11 1] **Exists:** Always
10:5	RxPingDuration	R/W	Recieve Ping Maximum Duration This field is relevant to Host mode and controls the maximum duration of received LFPS to be treated as a Ping LFPS. The Max duration of the Ping LFPS is controlled by programming this value and is in terms of 8 ns granularity. Eg: A value of 32 indicates 256 ns. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 5 6] **Exists:** Always
4:0	TxPingDuration	R/W	Transmit Ping Maximum Duration This field is relevant to Device mode and controls the maximum duration for which the controller should instruct the PHY to transmit a Ping LFPS. The duration of the Ping LFPS is controlled by programming this value and is in terms of 8 ns granularity. Eg: A value of 13 indicates 104 ns. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL2" 0 4] **Exists:** Always

1.2.45 GUSB2PHYCFG(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

- **Description:** Global USB2 PHY Configuration Register
  The application must program this register before starting any transactions on either the SoC bus or the USB.
  In Device-only configurations, only one register is needed.
  In Host mode, per-port registers are implemented.
- **Size:** 32 bits
- **Offset:** 0xc200 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_NUM_U2_ROOT_PORTS]-1
- **Exists:** Always

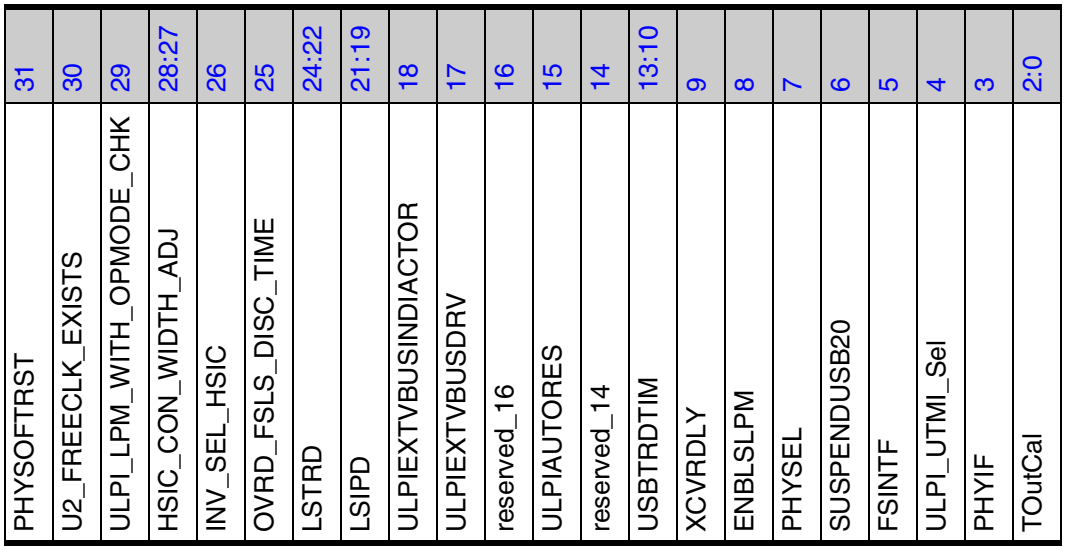


Table 1-57 Fields for Register: GUSB2PHYCFG(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

Bits	Name	Memory Access	Description
31	PHYSOFTRST	R/W	UTMI PHY Soft Reset (PHYSoftRst) Causes the usb2phy_reset signal to be asserted to reset a UTMI PHY. Not applicable to ULPI because ULPI PHYs are reset via their FunctionControl.Reset register, and the controller automatically writes to this register when the controller is reset (vcc_reset_n, USBCMD.HCRST, DCTL.SoftReset, or GCTL.SoftReset) **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 31 1] **Exists:** Always

Bits	Name	Memory Access	Description
30	U2_FREECLK_EXISTS	R/W	U2_FREECLK_EXISTS Specifies whether your USB 2.0 PHY provides a free-running PHY clock, which is active when the clock control input is active. If your USB 2.0 PHY provides a free-running PHY clock, it must be connected to the utmi_clk[0] input. The remaining utmi_clk[n] must be connected to the respective port clocks. The controller uses the Port-0 clock for generating the internal mac2 clock. ■ 1'b0: USB 2.0 free clock does not exist ■ 1'b1: USB 2.0 free clock exists **Note:** When the controller is configured as device-only (DWC_USB3_MODE = 0), do not set this bit to 1. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 30 1] **Exists:** Always
29	ULPI_LPM_WITH_OPMODE_CHK	R/W	ULPI_LPM_WITH_OPMODE_CHK Support the LPM over ULPI without NOPID token to the ULPI PHY. If this bit is set, the ULPI PHY is expected to qualify the EXT PID with OPMODE=2'b00 for LPM and not treat it as a NOPID. Check with your PHY vendor about your PHY behavior. This bit is valid only when the DWC_USB3_HSPHY_INTERFACE parameter is 2 or 3. ■ 1'b0: A NOPID is sent before sending an EXTPID for LPM; ■ 1'b1: An EXTPID is sent without previously sending a NOPID; **Note:** This bit is valid only in host mode. This bit should be '0' for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 29 1] **Exists:** Always
28:27	HSIC_CON_WIDTH_ADJ	* Varies	HSIC_CON_WIDTH_ADJ This bit is used in the HSIC device mode of operation. By default, the connect duration for the HSIC device controller is thrice the strobe period. You can change this duration to 4, 5, or 6 times the strobe period by setting the value of this field to 1, 2, or 3. This value is added to the default connect duration. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 27 2] **Exists:** Always **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_ENABLE_HSIC==0 ? "read-only" : "read-write"}

Bits	Name	Memory Access	Description
26	INV_SEL_HSIC	* Varies	INV_SEL_HSIC The application driver uses this bit to control the HSIC enable/disable function. When set to '1', this bit overrides and functionally inverts the "if_select_hsic" input signal. If {INV_SEL_HSIC, if_select_hsic} is: ■ 00: HSIC Capability is disabled. ■ 01: HSIC Capability is enabled. ■ 10: HSIC Capability is enabled. ■ 11: HSIC Capability is disabled. If the controller operates as non-HSIC-capable, it can only connect to non-HSIC-capable PHYs. If it operates as HSIC-capable, it can connect to HSIC-capable PHYs. This bit is reserved if the DWC_USB3_ENABLE_HSIC parameter is set to '0'. When selecting the HSIC feature, set the host side to HSIC mode first, then set the device mode side. If the device side is set to HSIC mode first and if the host does not see a connection in HSIC mode, then you must de-select the device HSIC mode and select it again using the if_select_hsic setting or register bit GUSB2PHYCFGn[26] to ensure that the device can connect to the host. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 26 1] **Exists:** Always **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_ENABLE_HSIC==0 ? "read-only" : "read-write"}
25	OVRD_FSLS_DISC_TIME	R/W	Overriding the FS/LS disconnect time to 32us. ■ If this value is 0, the FS/LS disconnect time is set to 2.5us as per the USB specification. ■ If this value is non-0, the disconnect detection time is set to 32us. Normally, this value is set to 0. However, if the USB 2.0 PHYs introduce noise on the UTMI linestate and cause SE0 glitches longer than 2.5us, then a false disconnect condition may get triggered. To avoid interoperability issues with these PHYs, this bit can be set to 1. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 25 1] **Exists:** Always

Bits	Name	Memory Access	Description
24:22	LSTRD	R/W	LS Turnaround Time (LSTRDTIM) This field indicates the value of the Rx-to-Tx packet gap for LS devices. The encoding is as follows: ■ 0: 2 bit times ■ 1: 2.5 bit times ■ 2: 3 bit times ■ 3: 3.5 bit times ■ 4: 4 bit times ■ 5: 4.5 bit times ■ 6: 5 bit times ■ 7: 5.5 bit times **Note:** ■ This field is applicable only in Host mode. ■ For normal operation (to work with most LS devices), set the default value of this field to 3'h0 (2 bit times). ■ The programmable LS device inter-packet gap and turn-around delays are provided to support some legacy LS devices that might require different delays than the default/fixed ones. For instance, the Open LS mouse requires 3 bit times of inter-packet gap to work correctly. ■ Include your PHY delays when programming the LSIPD/LSTRDTIM values. For example, if your PHY's TxEndDelay in LS mode is 30 UTMI/ULPI CLKs, then subtract this delay (~1 LS bit time) from the device's delay requirement. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 22 3] **Exists:** Always

Bits	Name	Memory Access	Description
21:19	LSIPD	R/W	LS Inter-Packet Time (LSIPD) This field indicates the value of Tx-to-Tx packet gap for LS devices. The encoding is as follows: ■ 0: 2 bit times ■ 1: 2.5 bit times ■ 2: 3 bit times ■ 3: 3.5 bit times ■ 4: 4 bit times ■ 5: 4.5 bit times ■ 6: 5 bit times ■ 7: 5.5 bit times **Note:** ■ This field is applicable only in Host mode. ■ For normal operation (to work with most LS devices), set the default value of this field to 3'h2 (3 bit times). ■ The programmable LS device inter-packet gap and turn-around delays are provided to support some legacy LS devices that might require different delays than the default/fixed ones. For instance, the AOpen LS mouse requires 3 bit times of inter-packet gap to work correctly. ■ Include your PHY delays when programming the LSIPD/LSTRDTIM values. For example, if your PHY's TxEndDelay in LS mode is 30 UTMI/ULPI CLKs, then subtract this delay (~1 LS bit time) from the device's delay requirement. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 19 3] **Exists:** Always
18	ULPIEXTVBUSINDIACTOR	R/W	ULPI External VBUS Indicator (ULPIExtVbusIndicator) Indicates the ULPI PHY VBUS over-current indicator. ■ 1'b0: PHY uses an internal VBUS valid comparator. ■ 1'b1: PHY uses an external VBUS valid comparator. Valid only when RTL parameter DWC_USB3_HSPHY_INTERFACE = 2 or 3 **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 18 1] **Exists:** Always

Bits	Name	Memory Access	Description
17	ULPIEXTVBUSDRV	R/W	ULPI External VBUS Drive (ULPIExtVbusDrv) Selects supply source to drive 5V on VBUS, in the ULPI PHY. ■ 1'b0: PHY drives VBUS with internal charge pump (default). ■ 1'b1: PHY drives VBUS with an external supply. (Only when RTL parameter DWC_USB3_HSPHY_INTERFACE = 2 or 3) **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 17 1] **Exists:** Always
16	reserved_16	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 16 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
15	ULPIAUTORES	R/W	ULPI Auto Resume (ULPIAutoRes) Sets the AutoResume bit in Interface Control register on the ULPI PHY. ■ 1'b0: PHY does not use the AutoResume feature. ■ 1'b1: PHY uses the AutoResume feature. Valid only when RTL parameter DWC_USB3_HSPHY_INTERFACE = 2 or 3 **Value After Reset:** =mpformat "0x%x" [{DWC_USB3_MODE == 0 ? 0 : [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 15 1]}] **Exists:** Always
14	reserved_14	R	Reserved **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 14 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
13:10	USBTRDTIM	R/W	USB 2.0 Turnaround Time (USBTrdTim) Sets the turnaround time in PHY clocks. Specifies the response time for a MAC request to the Packet FIFO Controller (PFC) to fetch data from the DFIFO (SPRAM). The following are the required values for the minimum SoC bus frequency of 60 MHz. USB turnaround time is a critical certification criteria when using long cables and five hub levels. The required values for this field: ■ 4'h5: When the MAC interface is 16-bit UTMI+. ■ 4'h9: When the MAC interface is 8-bit UTMI+/ULPI. If SoC bus clock is less than 60 MHz, and USB turnaround time is not critical, this field can be set to a larger value. **Note:** This field is valid only in device mode. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 10 4] **Exists:** Always
9	XCVRDLY	R/W	Transceiver Delay: Enables a delay between the assertion of the UTMI/ULPI Transceiver Select signal (for HS) and the assertion of the TxValid signal during a HS Chirp. When this bit is set to 1, a delay (of approximately 2.5 us) is introduced from the time when the Transceiver Select is set to 2'b00 (HS) to the time the TxValid is driven to 0 for sending the chirp-K. This delay is required for some UTMI/ULPI PHYs. **Note:** ■ If you enable the hibernation feature when the device controller comes out of power-off, you must re-initialize this bit with the appropriate value because the controller does not save and restore this bit value during hibernation. ■ This bit is valid only in device mode. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 9 1] **Exists:** Always

Bits	Name	Memory Access	Description
8	ENBLSLPM	R/W	Enable utmi_sleep_n and utmi_l1_suspend_n (EnblSlpM) The application uses this bit to control utmi_sleep_n and utmi_l1_suspend_n assertion to the PHY in the L1 state. ■ 1'b0: utmi_sleep_n and utmi_l1_suspend_n assertion from the controller is not transferred to the external PHY. ■ 1'b1: utmi_sleep_n and utmi_l1_suspend_n assertion from the controller is transferred to the external PHY. **Note:** This bit must be set high for Port0 if Synopsys PHY is used. **Note:** In Device mode - Before issuing any device endpoint command when operating in 2.0 speeds, disable this bit and enable it after the command completes. Without disabling this bit, if a command is issued when the device is in L1 state and if mac2_clk (utmi_clk/ulpi_clk) is gated off, the command will not get completed. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 8 1] **Exists:** Always
7	PHYSEL	W	USB 2.0 High-Speed PHY or USB 1.1 Full-Speed Serial Transceiver Select The application uses this bit to select a high-speed PHY or a full-speed transceiver. ■ 1'b0: USB 2.0 high-speed UTMI+ or ULPI PHY. This bit is always 0, with Write Only access. ■ 1'b1: USB 1.1 full-speed serial transceiver. This bit is always 1, with Write Only access. If both interface types are selected in coreConsultant (that is, parameters' values are not zero), the application uses this bit to select the active interface is active, with Read-Write bit access. **Note:** USB 1.1 full-serial transceiver is not supported. This bit always reads as 1'b0. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 7 1] **Exists:** Always

Bits	Name	Memory Access	Description
6	SUSPENDUSB20	R/W	Suspend USB2.0 HS/FS/LS PHY (SusPHY) When set, USB2.0 PHY enters Suspend mode if Suspend conditions are valid. For DRD configurations, it is recommended that this bit is set to 0 during coreConsultant configuration. If it is set to 1, then the application must clear this bit after power-on reset. Application needs to set it to 1 after the controller initialization completes. For all other configurations, this bit can be set to 1 during controller configuration. **Note:** ■ In host mode, on reset, this bit is set to 1. Software can override this bit after reset. ■ In device mode, before issuing any device endpoint command when operating in 2.0 speeds, disable this bit and enable it after the command completes. If you issue a command without disabling this bit when the device is in L2 state and if mac2_clk (utmi_clk/ulpi_clk) is gated off, the command will not get completed. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 6 1] **Exists:** Always
5	FSINTF	* Varies	Full-Speed Serial Interface Select (FSIntf) The application uses this bit to select a unidirectional or bidirectional USB 1.1 full-speed serial transceiver interface. ■ 1'b0: 6-pin unidirectional full-speed serial interface. This bit is set to 0 with Read Only access. ■ 1'b1: 3-pin bidirectional full-speed serial interface. This bit is set to 0 with Read Only access. **Note:** USB 1.1 full-speed serial interface is not supported. This bit always reads as 1'b0. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 5 1] **Exists:** Always **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_FSPHY_INTERFACE==0 ? "read-only" : "read-write"}

Bits	Name	Memory Access	Description
4	ULPI_UTMI_Sel	* Varies	ULPI or UTMI+ Select (ULPI_UTMI_Sel) The application uses this bit to select a UTMI+ or ULPI Interface. ■ 1'b0: UTMI+ Interface ■ 1'b1: ULPI Interface This bit is writable only if UTMI+ and ULPI is specified for High-Speed PHY Interface(s) in coreConsultant configuration (DWC_USB3_HSPHY_INTERFACE = 3). Otherwise, this bit is read-only and the value depends on the interface selected through DWC_USB3_HSPHY_INTERFACE. **Value After Reset:** DWC_USB3_CC_HS_PHYSEL **Exists:** Always **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_HSPHY_INTERFACE==3 ? "read-write" : "read-only"}
3	PHYIF	R/W	PHY Interface (PHYIf) If UTMI+ is selected, the application uses this bit to configure the controller to support a UTMI+ PHY with an 8- or 16-bit interface. ■ 1'b0: 8 bits ■ 1'b1: 16 bits ULPI Mode: 1'b0 **Note:** ■ All the enabled 2.0 ports must have the same clock frequency as Port0 clock frequency (utmi_clk[0]). ■ The UTMI 8-bit and 16-bit modes cannot be used together for different ports at the same time (that is, all the ports must be in 8-bit mode, or all of them must be in 16-bit mode, at a time). ■ If any of the USB 2.0 ports is selected as ULPI port for operation, then all the USB 2.0 ports must be operating at 60 MHz. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 3 1] **Exists:** Always

Bits	Name	Memory Access	Description
2:0	TOutCal	R/W	HS/FS Timeout Calibration (TOutCal) The number of PHY clocks, as indicated by the application in this field, is multiplied by a bit-time factor; this factor is added to the high-speed/full-speed interpacket timeout duration in the controller to account for additional delays introduced by the PHY. This may be required, since the delay introduced by the PHY in generating the linestate condition may vary among PHYs. The USB standard timeout value for high-speed operation is 736 to 816 (inclusive) bit times. The USB standard timeout value for full-speed operation is 16 to 18 (inclusive) bit times. The application must program this field based on the speed of connection. The number of bit times added per PHY clock are: High-speed operation: ■ One 30-MHz PHY clock = 16 bit times ■ One 60-MHz PHY clock = 8 bit times Full-speed operation: ■ One 30-MHz PHY clock = 0.4 bit times ■ One 60-MHz PHY clock = 0.2 bit times ■ One 48-MHz PHY clock = 0.25 bit times **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB2PHYCFG_REG_DEF_CC" 0 3] **Exists:** Always

1.2.46 GUSB2I2CCTL(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

- **Description:** Reserved Register
- **Size:** 32 bits
- **Offset:** 0xc240 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_NUM_U2_ROOT_PORTS]-1
- **Exists:** Always

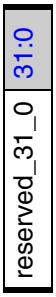


Table 1-58 Fields for Register: GUSB2I2CCTL(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

Bits	Name	Memory Access	Description
31:0	reserved_31_0	R	Reserved for future use **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffffffff **Write Constraint:** writeAsRead

1.2.47 GUSB2PHYACC_UTMI(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

- **Description:** Global USB 2.0 UTMI PHY Vendor Control Register
  The application uses this register to access PHY registers. It is implemented only if both UTMI PHY Interface and PHY Vendor Control Interface are enabled during coreConsultant configuration (parameter DWC_USB3_HSPHY_INTERFACE = 1 or 3, and DWC_USB3_VENDOR_CTL_INTERFACE = 1). In UTMI mode of operation, the functionality of this register can be used only if UTMI Vendor Control Interface is enabled.
  For an UTMI+ PHY, the controller uses the UTMI+ Vendor Control interface for PHY register access. There is no read/write differentiation for UTMI+. It is just a vendor control access. The application sets the Vendor Control register for PHY register access and times the PHY register access. The application polls the VStatus Done bit in this register for the completion of the PHY register access.
  In Device-only configurations, only one register is needed. In Host mode, per-port registers are implemented.
- **Size:** 32 bits
- **Offset:** 0xc280 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_NUM_U2_ROOT_PORTS]-1
- **Exists:** Always

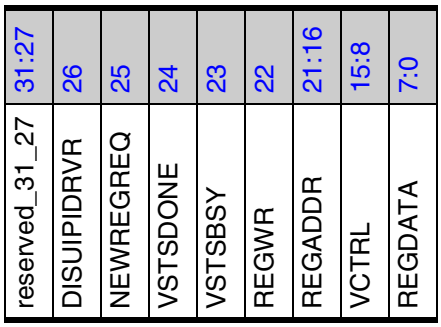


Table 1-59 Fields for Register: GUSB2PHYACC_UTMI(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

Bits	Name	Memory Access	Description
31:27	reserved_31_27	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1f **Write Constraint:** writeAsRead
26	DISUIPIDRVR	R	DISUIPIDRVR **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
25	NEWREGREQ	* Varies	NEWREGREQ **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_VENDOR_CTL_INTERFACE == 1 ? "read-write" : "read-only"}
24	VSTSDONE	R	VSTSDONE: **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Volatile:** true
23	VSTSBSY	* Varies	VSTSBSY **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_VENDOR_CTL_INTERFACE == 1 ? "read-write" : "read-only"}
22	REGWR	* Varies	REGWR **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_VENDOR_CTL_INTERFACE == 1 ? "read-write" : "read-only"}
21:16	REGADDR	* Varies	REGADDR **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_VENDOR_CTL_INTERFACE == 1 ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
15:8	VCTRL	* Varies	VCTRL This field contains the 4-bit register address, and the vendor-defined 4-bit parallel output bus. Bits [11:8] of this field are also placed on bits [3:0] of the utmi_vcontrol output signal. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_VENDOR_CTL_INTERFACE == 1 ? "read-write" : "read-only"}
7:0	REGDATA	* Varies	REGDATA **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {DWC_USB3_VENDOR_CTL_INTERFACE == 1 ? "read-write" : "read-only"}

1.2.48 GUSB2PHYACC_ULPI(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

- **Description:** Global USB 2.0 ULPI PHY Vendor Control Register
  The application uses this register to access the PHY registers. This register is always implemented when the ULPI PHY Interface is enabled during coreConsultant configuration (parameter DWC_USB3_HSPHY_INTERFACE = 2 or 3).
  For an ULPI PHY, the controller uses the ULPI interface for PHY register access.
  The application sets the Vendor Control register for PHY register access and times the PHY register access. The application polls the VStatus Done bit in this register for the completion of the PHY register access.
  In Device-only configurations, only one register is needed. In Host mode, per-port registers are implemented.
- **Size:** 32 bits
- **Offset:** 0xc280 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_NUM_U2_ROOT_PORTS]-1
- **Exists:** Always

31:27	26	25	24	23	22	21:16	15:8	7:0
reserved_31_27	DISUIPIDRVR	NEWREGREQ	VSTSDONE	VSTSBSY	REGWR	REGADDR	EXTREGADDR	REGDATA

Table 1-60 Fields for Register: GUSB2PHYACC_ULPI(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

Bits	Name	Memory Access	Description
31:27	reserved_31_27	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1f **Write Constraint:** writeAsRead
26	DISUIPIDRVR	* Varies	DISUIPIDRVR **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Volatile:** true **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_ULPI_CARKIT==1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
25	NEWREGREQ	* Varies	New Register Request The application sets this bit for a new vendor control access. Setting this bit to 1 asserts the utmi_vcontrolload_n (1'b0) on the UTMI interface. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Volatile:** true **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_VENDOR_CTL_INTERFACE==1 || DWC_USB3_HSPHY_INTERFACE==2 || DWC_USB3_HSPHY_INTERFACE==3) ? "read-write" : "read-only"}
24	VSTSDONE	R	VSTSDONE **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Volatile:** true
23	VSTSBSY	* Varies	VSTSBSY **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_VENDOR_CTL_INTERFACE==1 || DWC_USB3_HSPHY_INTERFACE==2 || DWC_USB3_HSPHY_INTERFACE==3) ? "read-write" : "read-only"}
22	REGWR	* Varies	Register Write The application sets this bit for register writes and clears it for register reads. **Note:** This bit is applicable for ULPI register read/write access only. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_VENDOR_CTL_INTERFACE==1 || DWC_USB3_HSPHY_INTERFACE==2 || DWC_USB3_HSPHY_INTERFACE==3) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
21:16	REGADDR	* Varies	Register Address The 6-bit PHY register address for immediate PHY Register Set access. Set to 6'h2F for Extended PHY Register Set access. Note: These bits are applicable for ULPI only. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_VENDOR_CTL_INTERFACE==1 || DWC_USB3_HSPHY_INTERFACE==2 || DWC_USB3_HSPHY_INTERFACE==3) ? "read-write" : "read-only"}
15:8	EXTREGADDR	* Varies	EXTREGADDR **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_VENDOR_CTL_INTERFACE==1 || DWC_USB3_HSPHY_INTERFACE==2 || DWC_USB3_HSPHY_INTERFACE==3) ? "read-write" : "read-only"}
7:0	REGDATA	* Varies	REGDATA **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_VENDOR_CTL_INTERFACE==1 || DWC_USB3_HSPHY_INTERFACE==2 || DWC_USB3_HSPHY_INTERFACE==3) ? "read-write" : "read-only"}

1.2.49 GUSB3PIPECTL(#n) (for n = 0; n <= DWC_USB3_NUM_U3_ROOT_PORTS-1)

- **Description:** Global USB 3.0 PIPE Control Register
  The application uses this register to configure the USB3 PHY and PIPE interface.
  Device-only configuration requires only one register. In Host mode, registers are implemented for each port.
  For more details on GUSB3PIPECTL(#n) bits, refer to section "GUSB3PIPECTLn Register" in the User Guide.
  Note:
  - GUSB3PIPECTLn registers are not applicable for USB 2.0-only mode.
- **Size:** 32 bits
- **Offset:** 0xc2c0 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_NUM_U3_ROOT_PORTS]-1
- **Exists:** Always

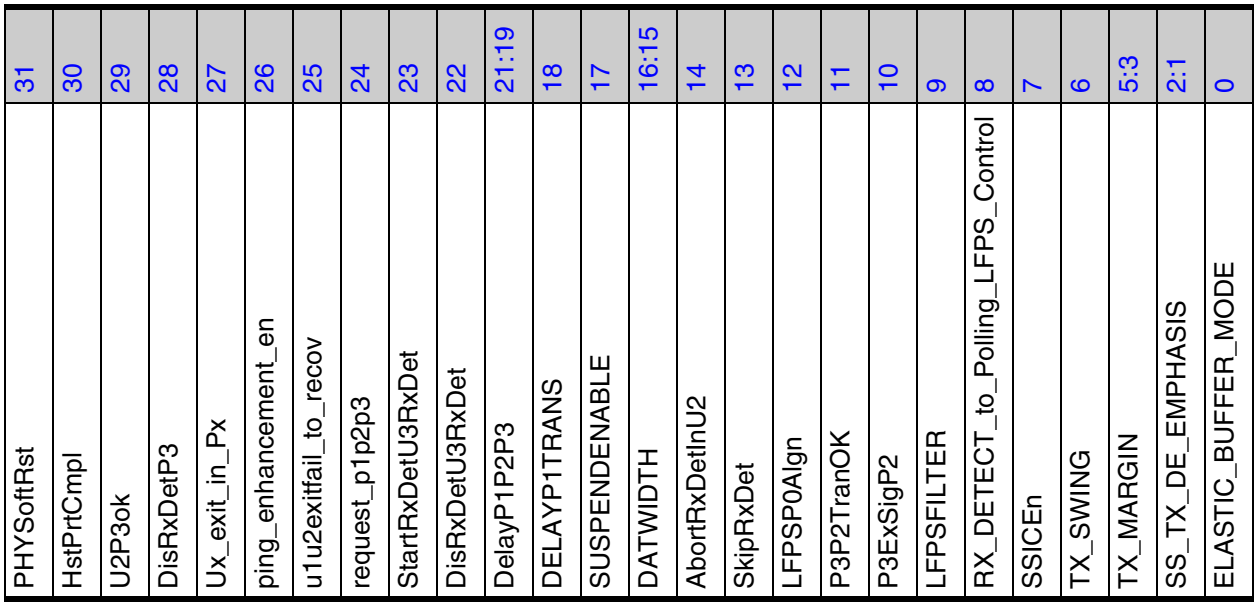


Table 1-61 Fields for Register: GUSB3PIPECTL(#n) (for n = 0; n <= DWC_USB3_NUM_U3_ROOT_PORTS-1)

Bits	Name	Memory Access	Description
31	PHYSoftRst	R/W	USB3 PHY Soft Reset After setting this bit to '1', the software needs to clear this bit. For more information, refer to Figure "Software Resets and PHY Clock Sequencing and Requirements" in the Databook. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 31 1] **Exists:** Always **Testable:** writeAsRead

Bits	Name	Memory Access	Description
30	HstPrtCmpl	R/W	HstPrtCmpl This feature tests the PIPE PHY compliance patterns without having to have a test fixture on the USB 3.0 cable. This bit enables placing the SS port link into a compliance state. By default, this bit must be set to 1'b0. In compliance lab testing, the SS port link enters compliance after failing the first polling sequence after power on. Set this bit to 0, when you run compliance tests. The sequence for using this functionality is as follows: ■ 1. Disconnect any plugged in devices. ■ 2. Perform USBCMD.HCRST or power-on-chip reset. ■ 3. Set PORTSC.PLS=0xA. ■ 4. Set PORTSC.PP=0. ■ 5. Set GUSB3PIPECTL. HstPrtCmpl=1. This places the link into compliance state. To advance the compliance pattern, follow this sequence (toggle the set GUSB3PIPECTL. HstPrtCmpl): ■ 1. Set GUSB3PIPECTL.HstPrtCmpl=0. ■ 2. Set GUSB3PIPECTL.HstPrtCmpl=1. This advances the link to the next compliance pattern. To exit from the compliance state perform USBCMD.HCRST or power-on-chip reset. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 30 1] **Exists:** Always
29	U2P3ok	R/W	P3 OK for SSInactive (SSIP3ok) ■ 0: During link state SS.Inactive, put PHY in P2 (Default) ■ 1: During link state SS.Inactive, put PHY in P3. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 29 1] **Exists:** Always
28	DisRxDetP3	R/W	Disabled receiver detection in P3 (DisRxDetP3) ■ 0: If PHY is in P3 and controller needs to perform receiver detection, The controller performs receiver detection in P3. (Default) ■ 1: If PHY is in P3 and controller needs to perform receiver detection, The controller changes the PHY power state to P2 and then performs receiver detection. After receiver detection, the cores changes PHY power state to P3. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 28 1] **Exists:** Always

Bits	Name	Memory Access	Description
27	Ux_exit_in_Px	R/W	Ux Exit in Px (Ux_exit_in_Px) ■ 0: The controller does U1/U2/U3 exit in PHY power state P0 (default behavior). ■ 1: The controller does U1/U2/U3 exit in PHY power state P1/P2/P3 respectively. **Note:** This bit is used by third-party SS PHY. It must be set to '0' for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 27 1] **Exists:** Always
26	ping_enhancement_en	R/W	Ping Enhancement Enable (ping_enhancement_en) When set, the Downstream port U1 ping receive timeout becomes 500 ms instead of 300 ms. Minimum Ping.LFPS receive duration is 8 ns (one mac3_clk). This field is valid for the downstream port only. Note: This bit is used by third-party SS PHY. It must be set to '0' for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 26 1] **Exists:** Always
25	u1u2exitfail_to_recov	R/W	U1U2exitfail to Recovery (u1u2exitfail_to_recov) When set, and U1/U2 LFPS handshake fails, the LTSSM transitions from U1/U2 to Recovery instead of SS Inactive. If Recovery fails, then the LTSSM can enter SS.Inactive. This is an enhancement only. It prevents interoperability issue if the remote link does not do proper handshake. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 25 1] **Exists:** Always
24	request_p1p2p3	R/W	Always Request P1/P2/P3 for U1/U2/U3 (request_p1p2p3) When set, the controller always requests PHY power change from P0 to P1/P2/P3 during U0 to U1/U2/U3 transition. If this bit is 0, and immediate Ux exit (remotely initiated, or locally initiated) happens, the controller does not request P1/P2/P3 power state change. **Note:** This bit must be set to '1' for Synopsys PHY. For third-party SS PHY, check with your PHY vendor. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 24 1] **Exists:** Always

Bits	Name	Memory Access	Description
23	StartRxDetU3RxDet	W	Start Receiver Detection in U3/Rx.Detect (StartRxdetU3RxDet) If DWC_USB3_GUSB3PIPECTL_INIT[22] is set, and the link is in either U3 or Rx.Detect state, the controller starts receiver detection on the rising edge of this bit. This can only be used for Downstream ports. This bit must be set to '0' for Upstream ports. This feature must not be enabled for normal operation. If have to use this feature, contact Synopsys. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 23 1] **Exists:** Always **Testable:** writeAsRead
22	DisRxDetU3RxDet	R/W	Disable Receiver Detection in U3/Rx.Det When set, the controller does not handle receiver detection in either U3 or Rx.Detect states. DWC_USB3_GUSB3PIPECTL_INIT[23] must be used to start receiver detection manually. This bit can only be used for the downstream port. This bit must be set to "0" for Upstream ports. This feature must not be enabled for normal operation. If you have to use this feature, contact Synopsys. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 22 1] **Exists:** Always
21:19	DelayP1P2P3	R/W	Delay P1P2P3 Delay P0 to P1/P2/P3 request when entering U1/U2/U3 until (DWC_USB3_GUSB3PIPECTL_INIT[21:19]*8) 8B10B error occurs, or Pipe3_RxValid drops to 0. DWC_USB3_GUSB3PIPECTL_INIT[18] must be 1 to enable this functionality. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 19 3] **Exists:** Always

Bits	Name	Memory Access	Description
18	DELAYP1TRANS	R/W	Delay PHY power change from P0 to P1/P2/P3 when link state changing from U0 to U1/U2/U3 respectively. ■ 1'b1: When entering U1/U2/U3, delay the transition to P1/P2/P3 until the pipe3 signals, Pipe3_RxElecIdle is 1 and pipe3_RxValid is 0 ■ 1'b0: When entering U1/U2/U3, transition to P1/P2/P3 without checking for Pipe3_RxElecIdle and pipe3_RxValid. **Note:** ■ If you are using Synopsys PHY, contact Synopsys Customer Support for recommendation on setting this bit because it is node dependent. ■ If you are using a third-party SS PHY, check with your PHY vendor for recommendation on setting this bit. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 18 1] **Exists:** Always
17	SUSPENDENABLE	R/W	Suspend USB3.0 SS PHY (Suspend_en) When set, and if Suspend conditions are valid, the USB 3.0 PHY enters Suspend mode. For DRD configurations, it is recommended that this bit is set to '0' during coreConsultant configuration. If it is set to '1', then the application must clear this bit after power-on reset. Application needs to set it to '1' after the controller initialization is completed. For all other configurations, this bit can be set to '1' during controller configuration. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 17 1] **Exists:** Always
16:15	DATWIDTH	R	PIPE Data Width (DatWidth) ■ 2'b00: 32 bits ■ 2'b01: 16 bits One clock after reset, these bits receive the value seen on the pipe3_DataBusWidth. The simulation testbench uses the coreConsultant parameter to configure the VIP. These bits in the coreConsultant parameter must match your PHY data width and the pipe3_DataBusWidth port. **Note:** 8-bit data width is not supported. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 15 2] **Exists:** Always **Testable:** untestable **Volatile:** true **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
14	AbortRxDetInU2	R/W	Abort Rx Detect in U2 (AbortRxDetInU2) When set and the link state is U2, the controller will abort receiver detection if it receives U2 exit LFPS from the remote link partner. This bit is for the downstream port only. **Note:** This bit is used by third-party SS PHY. It must be set to '0' for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 14 1] **Exists:** Always
13	SkipRxDet	R/W	Skip Rx Detect: When set, the controller skips Rx Detection if pipe3_RxElecIdle is low. Skip is defined as waiting for the appropriate timeout, then repeating the operation. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 13 1] **Exists:** Always
12	LFPSP0Algn	R/W	LFPS P0 Align: When set, ■ The controller deasserts LFPS transmission on the clock edge that it requests Phy power state 0 when exiting U1, U2, or U3 low power states. Otherwise, LFPS transmission is asserted one clock earlier. ■ The controller requests symbol transmission two pipe3_rx-_pclks periods after the PHY asserts PhyStatus as a result of the PHY switching from P1 or P2 state to P0 state. Currently, this bit is only used in USB 3.0 HUB with Synopsys PHY. For other USB 3.0 Host, Device, and DRD cores, this bit is not required. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 12 1] **Exists:** Always
11	P3P2TranOK	R/W	P3 P2 Transitions OK (P3P2TranOK) When set, the controller transitions directly from Phy power state P2 to P3 or from state P3 to P2. When not set, P0 is always entered as an intermediate state during transitions between P2 and P3, as defined in the *PIPE3 Specification*. According to the *PIPE3 Specification*, any direct transition between P3 and P2 is illegal. **Note:** This bit is used by third-party SS PHY. It must be set to '0' for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 11 1] **Exists:** Always

Bits	Name	Memory Access	Description
10	P3ExSigP2	R/W	P3 Exit Signal in P2 (P3ExSigP2) When this bit is set, the controller always changes the PHY power state to P2, before attempting a U3 exit handshake. This bit is used only for some non-Synopsys PHYs that cannot do LFPS in P3. **Note:** This bit is used by third-party SS PHY. It must be set to '0' for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 10 1] **Exists:** Always
9	LFPSFILTER	R/W	LFPS Filter (LFPSFilt) When set, filter LFPS reception with pipe3_RxValid in PHY power state P0, that is, ignore LFPS reception from the PHY unless both pipe3_Rxelecidle and pipe3_RxValid are deasserted. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 9 1] **Exists:** Always
8	RX_DETECT_to_Polling_LFPS_Control	R/W	RX_DETECT to Polling.LFPS Control ■ 1'b0 (Default): Enables a 400us delay to start Polling LFPS after RX_DETECT. This allows VCM offset to settle to a proper level. ■ 1'b1: Disables the 400us delay to start Polling LFPS after RX_DETECT. During controller certification with third party PHY it is observed that the PHY is not able to meet the Tx AC common mode voltage active (VTX-CM-ACPP_ACTIVE <100mv) if the link starts polling within 80us from the time rx.detect is performed. To meet this VTX-CM-ACPP_ACTIVE specification, the polling must be delayed further. If the PHY does not have issue then they can set this bit to 1 which allows polling to start within 80us. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 8 1] **Exists:** Always
7	SSICEn	R/W	This field is not used. **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 7 1] **Exists:** Always
6	TX_SWING	R/W	Tx Swing (TxSwing) Refer to the *PIPE3 specification.* **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 6 1] **Exists:** Always

Bits	Name	Memory Access	Description
5:3	TX_MARGIN	R/W	Tx Margin[2:0] (TxMargin) Refer to Table 5-3 of the *PIPE3 Specification.* **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 3 3] **Exists:** Always
2:1	SS_TX_DE_EMPHASIS	R/W	Tx Deemphasis (TxDeemphasis) The value driven to the PHY is controlled by the LTSSM during USB3 Compliance mode. (Refer to Table 5-3 of the *PIPE3 specification*.) **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 1 2] **Exists:** Always
0	ELASTIC_BUFFER_MODE	R/W	Elastic Buffer Mode (ElasticBufferMode) (Refer to Table 5-3 of the *PIPE3 specification*.) **Value After Reset:** =mpformat "0x%x" [get_field_val "GUSB3PIPECTL_REG_DEF_CC" 0 1] **Exists:** Always

1.2.50 GTXFIFOSIZ(#n) (for n = 0; n <= 31)

- **Description:** Global Transmit FIFO Size Register
  This register specifies the RAM start address and depth (both in MDWIDTH-bit words) for each implemented TxFIFO. The number of TxFIFOs depends on the configuration parameters including the number of Device IN Endpoints, number of Host Bus Instances, and presence of Debug Capability. The register default values for each mode are assigned in coreConsultant based on the maximum packet size, number of packets to be buffered, speed of host bus instance, bus latency, and mode of operation (host, device, or, DBC). Upon reset and mode transitions, hardware automatically programs these registers to the default values. Consequently, there is typically no need for the software to modify the pre-defined default values.
  For the debug capability mode, the currently mapped EP0 IN and EP1 IN TxFIFO numbers can be read from the GFIFOPRIDBC register.
  For more details on the usage of the GTXFIFOSIZn and GRXFIFOSIZn registers for different modes of operation, refer to "Memory Requirements" chapter in the Databook.
- **Size:** 32 bits
- **Offset:** 0xc300 + (n*0x4)
- **Exists:** DWC_USB3_NUM_TXFIFO>0

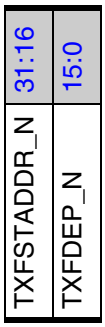


Table 1-62 Fields for Register: GTXFIFOSIZ(#n) (for n = 0; n <= 31)

Bits	Name	Memory Access	Description
31:16	TXFSTADDR_N	R/W	Transmit FIFOn RAM Start Address This field contains the memory start address for TxFIFOn in MDWIDTH-bit words. **Value After Reset:** =mpformat "0x%x" [get_field_val "GTXFIFOSIZ_REG_0" 16 16] **Exists:** Always

Bits	Name	Memory Access	Description
15:0	TXFDEP_N	R/W	TxFIFO Depth This field contains the depth of TxFIFOn in MDWIDTH-bit words. ■ Minimum value: 32 ■ Maximum value: 32,768 For more information, see "Integrating the Controller" chapter in the User Guide. **Value After Reset:** =mpformat "0x%x" [get_field_val "GTXFIFOSIZ_REG_0" 0 16] **Exists:** Always

1.2.51 GRXFIFOSIZ(#n) (for n = 0; n <= 31)

- **Description:** Global Receive FIFO Size Register
  This register specifies the RAM start address and depth (both in MDWIDTH-bit words) for each implemented RxFIFO. The number of RxFIFOs depends on the configuration parameters including the number of Host Bus Instances and presence of Debug Capability; device mode requires only one RxFIFO.
  The register default values for each mode are assigned in coreConsultant based on the maximum packet size, number of packets to be buffered, speed of the host bus instance, bus latency, and mode of operation (host, device, or DBC). Upon reset and mode transitions, hardware automatically programs these registers to the default values. Consequently, there is typically no need for the software to modify the pre-defined default values.
  For the debug capability mode, the currently mapped RxFIFO number can be read from the GFIFOPRIDBC register.
  For more details on the usage of the GTXFIFOSIZn and GRXFIFOSIZn registers for different modes of operation, refer to "Memory Requirements" chapter in the Databook.
- **Size:** 32 bits
- **Offset:** 0xc380 + (n*0x4)
- **Exists:** DWC_USB3_NUM_RXFIFO>0

Table 1-63 Fields for Register: GRXFIFOSIZ(#n) (for n = 0; n <= 31)

Bits	Name	Memory Access	Description
31:16	RXFSTADDR_N	R/W	RxFIFOn RAM Start Address (RxFStAddr_n) This field contains the memory start address for RxFIFO*n* in MDWIDTH-bit words. **Value After Reset:** =mpformat "0x%x" [get_field_val "GRXFIFOSIZ_REG_0" 16 16] **Exists:** Always

Bits	Name	Memory Access	Description
15:0	RXFDEP_N	R/W	RxFIFO Depth (RxFDep_n) This field contains the depth of RxFIFO*n* in MDWIDTH-bit words. ■ Minimum value: 32 ■ Maximum value: 16,384 **Value After Reset:** =mpformat "0x%x" [get_field_val "GRXFIFOSIZ_REG_0" 0 16] **Exists:** Always

1.2.52 GEVNTADR(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

- **Description:** Global Event Buffer Address Register
  This register holds the Event Buffer DMA Address pointer. Software must initialize this address once during power-on initialization. Software must not change the value of this register after it is initialized. Software must only use the GEVNTCOUNTn register for event processing. The lower n bits of the address must be GEVNTSIZn.EVNTSiz-aligned.
- **Size:** 64 bits
- **Offset:** 0xc400 + (*i* * 0x10), where *i* = 0..=[DWC_USB3_DEVICE_NUM_INT]-1
- **Exists:** DWC_USB3_SDWIDTH != 32 && DWC_USB3_32BIT_REGS == 0

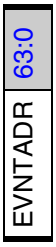


Table 1-64 Fields for Register: GEVNTADR(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

Bits	Name	Memory Access	Description
63:0	EVNTADR	R/W	EVNTADR **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0xffffffffffffffff **Volatile:** true

1.2.53 GEVNTADRLO(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

- **Description:** Global Event Buffer Address (Low) Register
  This is an alternate register for the GEVNTADRn register.
- **Size:** 32 bits
- **Offset:** 0xc400 + (*i* * 0x10), where *i* = 0..=[DWC_USB3_DEVICE_NUM_INT]-1
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

Table 1-65 Fields for Register: GEVNTADRLO(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

Bits	Name	Memory Access	Description
31:0	EVNTADRLO	R/W	Event Buffer Address (EvntAdrLo) Holds the lower 32 bits of start address of the external memory for the Event Buffer. During operation, hardware does not update this address. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0xffffffff **Volatile:** true

1.2.54 GEVNTADRHI(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

- **Description:** Global Event Buffer Address (High) Register
  This is an alternate register for the GEVNTADRn register.
- **Size:** 32 bits
- **Offset:** 0xc404 + (*i* * 0x10), where *i* = 0..=[DWC_USB3_DEVICE_NUM_INT]-1
- **Exists:** DWC_USB3_SDWIDTH == 32 || DWC_USB3_32BIT_REGS == 1

Table 1-66 Fields for Register: GEVNTADRHI(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

Bits	Name	Memory Access	Description
31:0	EVNTADRHI	R/W	Event Buffer Address (EvntAdrHi) Holds the higher 32 bits of start address of the external memory for the Event Buffer. During operation, hardware does not update this address. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0xffffffff **Volatile:** true

1.2.55 GEVNTSIZ(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

- **Description:** Global Event Buffer Size Register
  This register holds the Event Buffer Size and the Event Interrupt Mask bit. During power-on initialization, software must initialize the size with the number of bytes allocated for the Event Buffer. The Event Interrupt Mask will mask the interrupt, but events are still queued. After configuration, software must preserve the Event Buffer Size value when changing the Event Interrupt Mask.
- **Size:** 32 bits
- **Offset:** 0xc408 + (*i* * 0x10), where *i* = 0..=[DWC_USB3_DEVICE_NUM_INT]-1
- **Exists:** Always

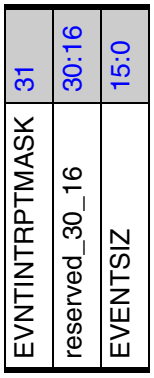


Table 1-67 Fields for Register: GEVNTSIZ(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

Bits	Name	Memory Access	Description
31	EVNTINTRPTMASK	R/W	Event Interrupt Mask (EvntIntMask). When set to '1', this prevents the interrupt from being generated. However, even when the mask is set, the events are queued. **Value After Reset:** 0x0 **Exists:** Always
30:16	reserved_30_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7fff **Write Constraint:** writeAsRead
15:0	EVENTSIZ	R/W	Event Buffer Size in bytes (EVNTSiz) Holds the size of the Event Buffer in bytes; must be a multiple of four. This is programmed by software once during initialization. The minimum size of the event buffer is 32 bytes. **Value After Reset:** 0x0 **Exists:** Always

1.2.56 GEVNTCOUNT(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

- **Description:** Global Event Buffer Count Register
  This register holds the number of valid bytes in the Event Buffer. During initialization, software must initialize the count by writing 0 to the Event Count field. Each time the hardware writes a new event to the Event Buffer, it increments this count. Most events are four bytes, but some events may span over multiple four byte entries. Whenever the count is greater than zero and if enabled, conditions for interrupt moderation are satisfied, the hardware raises the corresponding interrupt line (depending on the EvntIntMask bit in the GEVNTSIZn register). On an interrupt, software processes one or more events out of the Event Buffer. Afterwards, software must write the Event Count field with the number of bytes it processed. If Interrupt Moderation is enabled, then software needs to clear EVNT_HANDLER_BUSY bit.
  Clock crossing delays may result in the continuous assertion of the interrupt after software acknowledges the last event. Therefore, when the interrupt line is asserted, software must read the GEVNTCOUNT register and only process events if the GEVNTCOUNT is greater than 0.
- **Size:** 32 bits
- **Offset:** 0xc40c + (*i* * 0x10), where *i* = 0..=[DWC_USB3_DEVICE_NUM_INT]-1
- **Exists:** Always

Table 1-68 Fields for Register: GEVNTCOUNT(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

Bits	Name	Memory Access	Description
31	EVNT_HANDLER_BUSY	R/W	Event Handler Busy Device software event handler busy indication. The controller sets this bit when the interrupt line is asserted due to pending events. Software clears this bit (with 1'b1) when it has finished processing the events (along with updating the EVNTCOUNT in this register). The controller does not raise the interrupt line for a new event unless this bit is cleared. **Note:** When Interrupt moderation is disabled (that is, DEVICE_IMODI = 0), this bit is ignored. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x1 **Volatile:** true
30:16	reserved_30_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7fff **Write Constraint:** writeAsRead
15:0	EVNTCOUNT	R/W	Event Count (EVNTCount) When read, returns the number of valid events in the Event Buffer (in bytes). When written, hardware decrements the count by the value written. When Interrupt moderation is enabled (that is, DEVICE_IMODI!= 0), the interrupt line gets de-asserted when the first write happens on this register to decrement the count. When Interrupt moderation is disabled (that is, DEVICE_IMODI = 0), the Interrupt line continues to get asserted until the event count becomes zero (no-moderation behavior). **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0xffff **Volatile:** true

1.2.57 GHWPARAMS8

- **Description:** Global Hardware Parameters Register 8
  This register contains the hardware configuration options that you can select in the coreConsultant GUI.
  For a description of each parameter, refer to the "Parameter Descriptions" chapter in the Databook. This information is also available in coreConsultant by right-clicking the parameter label and selecting "What's This" or by clicking the Help tab.
  Note:
  Some of the global hardware parameters are not currently modifiable in coreConsultant. These settings are in the <workspace>/src/DWC_usb3_params.v file; you must not change them.
- **Size:** 32 bits
- **Offset:** 0xc600
- **Exists:** Always

Table 1-69 Fields for Register: GHWPARAMS8

Bits	Name	Memory Access	Description
31:0	ghwparams8_32_0	R	`DWC_USB3_DCACHE_DEPTH_INFO **Value After Reset:** DWC_USB3_DCACHE_DEPTH_INFO **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.2.58 GUCTL3

- **Description:** Global User Control Register 3
  This register provides a few options for the software to control the controller behavior in the Host mode. Most of the options are used to improve host inter-operability with different devices.
- **Size:** 32 bits
- **Offset:** 0xc60c
- **Exists:** Always

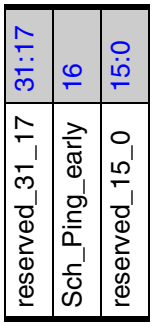


Table 1-70 Fields for Register: GUCTL3

Bits	Name	Memory Access	Description
31:17	reserved_31_17	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead
16	Sch_Ping_early	R/W	Enable SuperSpeed Ping Transaction Packet scheduling early in the microframe. This bit is valid in Host mode only. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GUCTL3" 16 1] **Exists:** Always
15:0	reserved_15_0	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead

1.2.59 GTXFIFOPRIDEV

- **Description:** Global Device TX FIFO DMA Priority Register
  This register specifies the relative DMA priority level among the Device TXFIFOs (one per IN endpoint). Each register bit[n] controls the priority (1: high, 0: low) of each TXFIFO[n]. When multiple TXFIFOs compete for DMA service at a given time (that is, multiple TXQs contain TX DMA requests and their corresponding TXFIFOs have space available), the TX DMA arbiter grants access on a packet-basis in the following manner:
  - 1. High-priority TXFIFOs are granted access using round-robin arbitration
  - 2. Low-priority TXFIFOs are granted access using round-robin arbitration only after the high-priority TXFIFOs have no further processing to do (that is, either the TXQs are empty or the corresponding TXFIFOs are full).

  For scatter-gather packets, the arbiter grants successive DMA requests to the same FIFO until the entire packet is completed.
  When configuring periodic IN endpoints, software must set register bit[n]=1, where n is the TXFIFO assignment. This ensures that the DMA for isochronous or interrupt IN endpoints are prioritized over bulk or control IN endpoints.
  This register is present only when the controller is configured to operate in the device mode (includes DRD mode). The register size corresponds to the number of Device IN endpoints.
  Note
  - Since the device mode uses only one RXFIFO, there is no Device RXFIFO DMA Priority Register.
- **Size:** 32 bits
- **Offset:** 0xc610
- **Exists:** Always

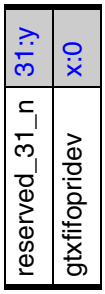


Table 1-71 Fields for Register: GTXFIFOPRIDEV

Bits	Name	Memory Access	Description
31:y	reserved_31_n	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffff **Write Constraint:** writeAsRead **Range Variable[y]:** DWC_USB3_NUM_IN_EPS

Bits	Name	Memory Access	Description
x:0	gtxfifopridev	R/W	Device TxFIFO priority **Value After Reset:** DWC_USB3_GTXFIFOPRIDEV_INIT **Exists:** Always **Range Variable[x]:** DWC_USB3_NUM_IN_EPS - 1

1.2.60 GTXFIFOPRIHST

- **Description:** Global Host TX FIFO DMA Priority Register
  This register specifies the relative DMA priority level among the Host TXFIFOs (one per USB bus instance) within the associated speed group (SS or HS/FSLS). Each register bit[n] controls the priority (1: high, 0: low) of TXFIFO[n] within a speed group. When multiple TXFIFOs compete for DMA service at a given time (i.e., multiple TXQs contain TX DMA requests and their corresponding TXFIFOs have space available), the TX DMA arbiter grants access on a packet-basis in the following manner:
  - 1. Among the FIFOs in the same speed group (SS or HS/FSLS):
    a. High-priority TXFIFOs are granted access using round-robin arbitration
    b. Low-priority TXFIFOs are granted access using round-robin arbitration only after the high-priority TXFIFOs have no further processing to do (that is, either the TXQs are empty or the corresponding TXFIFOs are full).
  - 2. The TX DMA arbiter prioritizes the SS speed group or HS/FSLS speed group according to the ratio programmed in the GDMAHLRATIO register.

  For scatter-gather packets, the arbiter grants successive DMA requests to the same FIFO until the entire packet is completed.
  This register is present only when the controller is configured to operate in the host mode (includes DRD mode). The register size corresponds to the number of configured USB bus instances; for example, in the default configuration, there are 3 USB bus instances (1 SS, 1 HS, and 1 FSLS).
- **Size:** 32 bits
- **Offset:** 0xc618
- **Exists:** Always

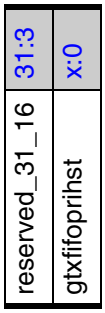


Table 1-72 Fields for Register: GTXFIFOPRIHST

Bits	Name	Memory Access	Description
31:3	reserved_31_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffff **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
x:0	gtxfifoprihst	R/W	Host TxFIFO priority **Value After Reset:** DWC_USB3_GTXFIFOPRIHST_INIT **Exists:** Always **Range Variable[x]:** DWC_USB3_NUM_TXFIFO_H - 1

1.2.61 GRXFIFOPRIHST

- **Description:** Global Host RX FIFO DMA Priority Register
  This register specifies the relative DMA priority level among the Host RXFIFOs (one per USB bus instance) within the associated speed group (SS or HS/FSLS). Each register bit[n] controls the priority (1: high, 0: low) of RXFIFO[n] within a speed group. When multiple RXFIFOs compete for DMA service at a given time (i.e., multiple RXQs contain RX DMA requests and their corresponding RXFIFOs have data available), the RX DMA arbiter grants access on a packet-basis in the following manner:
  - 1. Among the FIFOs in the same speed group (SS or HS/FSLS):
    a. High-priority RXFIFOs are granted access using round-robin arbitration
    b. Low-priority RXFIFOs are granted access using round-robin arbitration only after high-priority RXFIFOs have no further processing to do (that is, either the RXQs are empty or the corresponding RXFIFOs do not have the required data).
  - 2. The RX DMA arbiter prioritizes the SS speed group or HS/FSLS speed group according to the ratio programmed in the GDMAHLRATIO register.

  For scatter-gather packets, the arbiter grants successive DMA requests to the same FIFO until the entire packet is completed.
  This register is present only when the controller is configured to operate in the host mode (includes DRD mode). The register size corresponds to the number of configured USB bus instances; for example, in the default configuration, there are 3 USB bus instances (1 SS, 1 HS, and 1 FSLS).
- **Size:** 32 bits
- **Offset:** 0xc61c
- **Exists:** Always

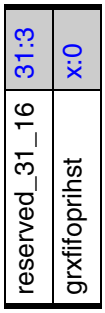


Table 1-73 Fields for Register: GRXFIFOPRIHST

Bits	Name	Memory Access	Description
31:3	reserved_31_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffff **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
x:0	grxfifoprihst	R/W	Host RxFIFO priority **Value After Reset:** DWC_USB3_GRXFIFOPRIHST_INIT **Exists:** Always **Range Variable[x]:** DWC_USB3_NUM_RXFIFO_H - 1

1.2.62 GFIFOPRIDBC

- **Description:** Global Host Debug Capability DMA Priority Register
  This register specifies the relative priority of the RXFIFOs and TXFIFOs associated with the DbC mode. It overrides the priority assigned in the corresponding indexes of the Host RXFIFO and TXFIFO DMA priority registers, when the DbC mode is enabled.
  Priority settings are specified in relation to the low-priority SS speed group:
  - 1. Normal priority indicates that the DbC FIFOs are considered identical to the Host SS low-priority FIFOs.
  - 2. Low priority indicates that the DbC FIFOs are considered to have lower priority than all Host SS FIFOs.
  - 3. High priority indicates that the DbC FIFOs are considered higher priority than the Host SS low-priority FIFOs but lower priority than the Host SS high-priority FIFOs.

  This register is present only when the controller is configured to operate in Host Debug Capability (DbC) mode.
- **Size:** 32 bits
- **Offset:** 0xc620
- **Exists:** DWC_USB3_EN_DBC==1

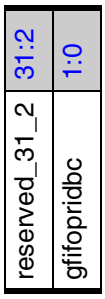


Table 1-74 Fields for Register: GFIFOPRIDBC

Bits	Name	Memory Access	Description
31:2	reserved_31_2	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3fffffff **Write Constraint:** writeAsRead
1:0	gfifopridbc	R/W	Host DbC DMA priority **Value After Reset:** DWC_USB3_GFIFOPRIDBC_INIT **Exists:** Always

1.2.63 GDMAHLRATIO

- **Description:** Global Host FIFO DMA High-Low Priority Ratio Register
  This register specifies the relative priority of the SS FIFOs with respect to the HS/FSLS FIFOs. The DMA arbiter prioritizes the HS/FSLS round-robin arbiter group every DMA High-Low Priority Ratio grants as indicated in the register separately for TX and RX.
  To illustrate, consider that all FIFOs are requesting access simultaneously, and the ratio is 4. SS gets priority for 4 packets, HS/FSLS gets priority for 1 packet, SS gets priority for 4 packets, HS/FSLS gets priority for 1 packet, and so on.
  If FIFOs from both speed groups are not requesting access simultaneously then,
  - if SS got grants 4 out of the last 4 times, then HS/FSLS get the priority on any future request.
  - if HS/FSLS got the grant last time, SS gets the priority on the next request.
  - if there is a valid request on either SS or HS/FSLS, a grant is always awarded; there is no idle.

  This register is present if the controller is configured to operate in host mode (includes DRD).
- **Size:** 32 bits
- **Offset:** 0xc624
- **Exists:** Always

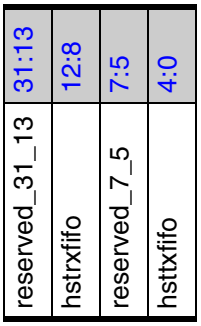


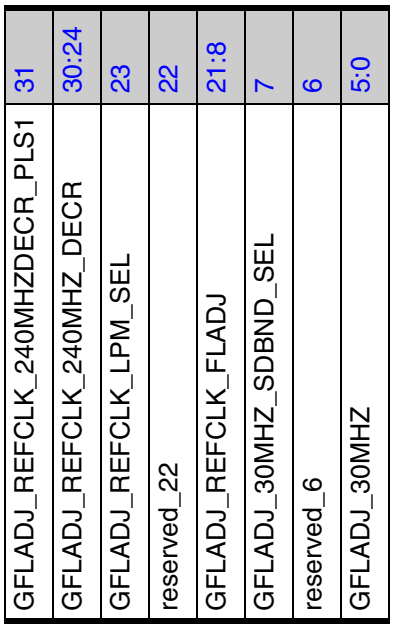
Table 1-75 Fields for Register: GDMAHLRATIO

Bits	Name	Memory Access	Description
31:13	reserved_31_13	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7ffff **Write Constraint:** writeAsRead
12:8	hstrxfifo	R/W	Host RXFIFO DMA High-Low Priority **Value After Reset:** DWC_USB3_GDMAHLRATIO_RX_INIT **Exists:** Always

Bits	Name	Memory Access	Description
7:5	reserved_7_5	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7 **Write Constraint:** writeAsRead
4:0	hsttxfifo	R/W	Host TXFIFO DMA High-Low Priority **Value After Reset:** DWC_USB3_GDMAHLRATIO_TX_INIT **Exists:** Always

1.2.64 GFLADJ

- **Description:** Global Frame Length Adjustment Register
  This register provides options for the software to control the controller behavior with respect to SOF (Start of Frame) and ITP (Isochronous Timestamp Packet) timers and frame timer functionality. It provides an option to override the fladj_30mhz_reg sideband signal. In addition, it enables running SOF or ITP frame timer counters completely from the ref_clk. This facilitates hardware LPM in host mode with the SOF or ITP counters being run from the ref_clk signal.
- **Size:** 32 bits
- **Offset:** 0xc630
- **Exists:** Always



31	30:24	23	22	21:8	7	6	5:0
GFLADJ_REFCLK_240MHZDECR_PLS1	GFLADJ_REFCLK_240MHZ_DECR	GFLADJ_REFCLK_LPM_SEL	reserved_22	GFLADJ_REFCLK_FLADJ	GFLADJ_30MHZ_SDBND_SEL	reserved_6	GFLADJ_30MHZ

Table 1-76 Fields for Register: GFLADJ

Bits	Name	Memory Access	Description

31	GFLADJ_REFCLK_240MHZDECR_PLS1	R/W	GFLADJ_REFCLK_240MHZDECR_PLS1
			This field indicates that the decrement value that the controller applies for each ref_clk must be GFLADJ_REFCLK_240MHZ_DECR and GFLADJ_REFCLK_240MHZ_DECR +1 alternatively on each ref_clk.
			Set this bit to a '1' only if GFLADJ_REFCLK_LPM_SEL is set to '1' and the fractional component of 240/ref_frequency is greater than or equal to 0.5.
			Examples:
			If the ref_clk is 19.2 MHz then
			■ GUCTL.REF_CLK_PERIOD = 52
			■ GFLADJ.GFLADJ_REFCLK_240MHZ_DECR = (240/19.2) = 12.5
			■ GFLADJ.GFLADJ_REFCLK_240MHZDECR_PLS1 = 1
			If the ref_clk is 24 MHz then
			■ GUCTL.REF_CLK_PERIOD = 41
			■ GFLADJ.GFLADJ_REFCLK_240MHZ_DECR = (240/24) = 10
			■ GFLADJ.GFLADJ_REFCLK_240MHZDECR_PLS1 = 0
			Value After Reset: =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 31 1]
			Exists: Always

Bits	Name	Memory Access	Description

30:24	GFLADJ_REFCLK_240MHZ_DECR	R/W	This field indicates the decrement value that the controller applies for each ref_clk in order to derive a frame timer in terms of a 240-MHz clock.
			This field must be programmed to a non-zero value only if GFLADJ_REFCLK_LPM_SEL is set to '1'.
			The value is derived as follows:
			GFLADJ_REFCLK_240MHZ_DECR = 240/ref_clk_frequency
			Examples: If the ref_clk is 24 MHz then
			■ GUCTL.REF_CLK_PERIOD = 41
			■ GFLADJ.GFLADJ_REFCLK_240MHZ_DECR = 240/24 = 10
			If the ref_clk is 48 MHz then
			■ GUCTL.REF_CLK_PERIOD = 20
			■ GFLADJ.GFLADJ_REFCLK_240MHZ_DECR = 240/48 = 5
			If the ref_clk is 17 MHz then
			■ GUCTL.REF_CLK_PERIOD = 58
			■ GFLADJ.GFLADJ_REFCLK_240MHZ_DECR = 240/17 = 14
			Value After Reset: =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 24 7]
			Exists: Always

23	GFLADJ_REFCLK_LPM_SEL	R/W	This bit enables the functionality of running SOF/ITP counters on the ref_clk. This bit must not be set to '1' if GCTL.SOFITPSYNC bit is set to '1'. Similarly, if GFLADJ_REFCLK_LPM_SEL set to '1', GCTL.SOFITPSYNC must not be set to '1'. In device mode, setting this bit to '1' enables SOF tracking using ref_clk. When GFLADJ_REFCLK_LPM_SEL is set to '1' the overloading of the suspend control of the USB 2.0 first port PHY (UTMI/ULPI) with USB 3.0 port states is removed. For example, for Synopsys PHY, the COMMONONN signal can be tied to '1'. Note that the ref_clk frequencies supported in this mode are 16/17/19.2/20/24/39.7/40 MHz. The utmi_clk[0] signal of the controller must be connected to the FREECLK of the PHY. **Note:** If you set this bit to '1', the GUSB2PHYCFG.U2_FREECLK_EXISTS bit must be set to '0'. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 23 1] **Exists:** Always

Bits	Name	Memory Access	Description
22	reserved_22	R	Reserved for future use **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 22 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
21:8	GFLADJ_REFCLK_FLADJ	R/W	This field indicates the frame length adjustment to be applied when SOF/ITP counter is running on the ref_clk. This register value is used to adjust the ITP interval when GCTL[SOFITPSYNC] is set to '1'; SOF and ITP interval when GLADJ.GFLADJ_REFCLK_LPM_SEL is set to '1'. This field must be programmed to a non-zero value only if GFLADJ_REFCLK_LPM_SEL is set to '1' or GCTL.SOFITPSYNC is set to '1'. The value is derived as follows: FLADJ_REF_CLK_FLADJ=((125000/ref_clk_period_integer)-(125000/ref_clk_period)) * ref_clk_period where ■ the ref_clk_period_integer is the integer value of the ref_clk period got by truncating the decimal (fractional) value that is programmed in the GUCTL.REF_CLK_PERIOD field. ■ the ref_clk_period is the ref_clk period including the fractional value. Examples: If the ref_clk is 24 MHz then ■ GUCTL.REF_CLK_PERIOD = 41 ■ GFLADJ.GLADJ_REFCLK_FLADJ = ((125000/41)-(125000/41.6666))*41.6666 = 2032 (ignoring the fractional value) If the ref_clk is 48 MHz then ■ GUCTL.REF_CLK_PERIOD = 20 ■ GFLADJ.GLADJ_REFCLK_FLADJ = ((125000/20)-(125000/20.8333))*20.8333 = 5208 (ignoring the fractional value) **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 8 14] **Exists:** Always

Bits	Name	Memory Access	Description
7	GFLADJ_30MHZ_SDBND_SEL	R/W	GFLADJ_30MHZ_SDBND_SEL This field selects whether to use the input signal fladj_30mhz_reg or the GFLADJ.GFLADJ_30MHZ to adjust the frame length for the SOF/ITP. When this bit is set to, ■ 1, the controller uses the register field GFLADJ.GFLADJ_30MHZ value ■ 0, the controller uses the input signal fladj_30mhz_reg value **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 7 1] **Exists:** Always
6	reserved_6	R	Reserved for future use **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 6 1] **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
5:0	GFLADJ_30MHZ	R/W	GFLADJ_30MHZ This field indicates the value that is used for frame length adjustment instead of considering from the sideband input signal fladj_30mhz_reg. This enables post-silicon frame length adjustment in case the input signal fladj_30mhz_reg is connected to a wrong value or is not valid. For details on how to set this value, refer to section 5.2.4, "Frame Length Adjustment Register (FLADJ)," of the *xHCI Specification.* **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_GFLADJ" 0 6] **Exists:** Always

1.2.65 GUSB2RHBCTL(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

- **Description:** Global USB 2.0 Root Hub Control Register
  The application must program this register before starting any transactions on the USB if a non-default value is desired.
  In Device-only configurations, only one register is needed.
  In Host mode, per-port registers are implemented.
- **Size:** 32 bits
- **Offset:** 0xc640 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_NUM_U2_ROOT_PORTS]-1
- **Exists:** Always

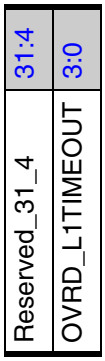


Table 1-77 Fields for Register: GUSB2RHBCTL(#n) (for n = 0; n <= DWC_USB3_NUM_U2_ROOT_PORTS-1)

Bits	Name	Memory Access	Description
31:4	Reserved_31_4	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x0
3:0	OVRD_L1TIMEOUT	R/W	Overriding the driver programmed L1TIMEOUT value. If this value is 0, the L1 Timeout value is taken from the xHCI PORTHLPMC register. If this value is non-0, then this will override the L1 Timeout value programmed in the xHCI PORTHLPMC register. In that case the actual L1 Timeout would be 2 ^ <OVRD_L1TIMEOUT-1> * 8us. (1=8us, 2=16us, 3=32us etc) **Value After Reset:** 0x0 **Exists:** Always

1.3 DWC_usb3_block_dev Registers

1.3.1 DCFG

- **Description:** Device Configuration Register.
  This register configures the controller in Device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming.
- **Size:** 32 bits
- **Offset:** 0xc700
- **Exists:** Always

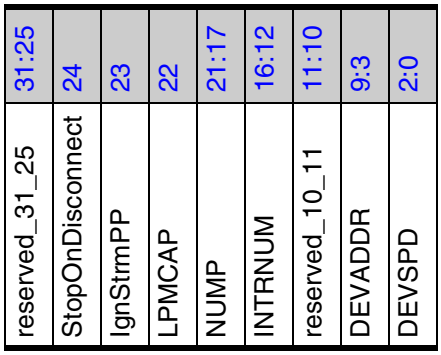


Table 1-78 Fields for Register: DCFG

Bits	Name	Memory Access	Description
31:25	reserved_31_25	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3f **Write Constraint:** writeAsRead
24	StopOnDisconnect	* Varies	Reserved **Value After Reset:** 0x0 **Exists:** DWC_USB3_NUM_SSIC_PORTS != 0 && DWC_USB3_EN_PWROPT==2 **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_NUM_SSIC_PORTS!=0 && DWC_USB3_EN_PWROPT==2) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
23	IgnStrmPP	R/W	IgnoreStreamPP This bit only affects stream-capable bulk endpoints. When this bit is set to '0' and the controller receives a Data Packet with the Packet Pending (PP) bit set to 0 for OUT endpoints, or it receives an ACK with the NumP field set to 0 and PP set to 0 for IN endpoints, the controller attempts to search for another stream (CStream) to initiate to the host. However, there are two situations where this behavior is not optimal: ■ When the host is setting PP=0 even though it has not finished the stream, or ■ When the endpoint on the device is configured with one transfer resource and therefore does not have any other streams to initiate to the host. When this bit is set to '1', the controller ignores the Packet Pending bit for the purposes of stream selection and does not search for another stream when it receives DP(PP=0) or ACK(NumP=0, PP=0). This can enhance the performance when the device system bus bandwidth is low or the host responds to the controller's ERDY transmission very quickly. **Value After Reset:** 0x0 **Exists:** Always
22	LPMCAP	R/W	LPM Capable The application uses this bit to control the LPM capabilities of the DWC_usb3 controller. If the controller operates as a non-LPM-capable device, it cannot respond to LPM transactions. ■ 1'b0: LPM capability is not enabled. ■ 1'b1: LPM capability is enabled. **Value After Reset:** 0x0 **Exists:** Always
21:17	NUMP	R/W	Number of Receive Buffers. This bit indicates the number of receive buffers to be reported in the ACK TP. The DWC_usb3 controller uses this field for non-control endpoints if GRXTHRCFG.UsbRxPktCntSel is set to '0'. The application can program this value based on RxFIFO size, buffer sizes programmed in descriptors, and system latency. For an OUT endpoint, this field controls the number of receive buffers reported in the NumP field of the ACK TP transmitted by the controller. Note: This bit is used in host mode when Debug Capability is enabled. **Value After Reset:** {DWC_USB3_DEVDBC_OUT_NUMP} **Exists:** Always

Bits	Name	Memory Access	Description
16:12	INTRNUM	R/W	Interrupt number Indicates interrupt/EventQ number on which non-endpoint-specific device-related interrupts (see DEVT) are generated. **Value After Reset:** 0x0 **Exists:** Always
11:10	reserved_10_11	R	Reserved **Value After Reset:** 0x2 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Write Constraint:** writeAsRead
9:3	DEVADDR	R/W	Device Address. The application must perform the following: ■ Program this field after every SetAddress request. ■ Reset this field to zero after USB reset. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
2:0	DEVSPD	R/W	Device Speed. Indicates the speed at which the application requires the controller to connect, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the controller is connected. ■ 3'b100: SuperSpeed (USB 3.0 PHY clock is 125 MHz or 250 MHz) ■ 3'b000: High-speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) ■ 3'b001: Full-speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) **Values:** ■ 0x4 (SuperSpeed): SuperSpeed (USB 3.0 PHY clock is 125 MHz or 250 MHz) ■ 0x0 (HighSpeed): High-speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) ■ 0x1 (FullSpeed): Full-speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) **Value After Reset:** {DWC_USB3_SSPHY_INTERFACE==1 ? 0x4 : 0x0} **Exists:** Always **Testable:** {(DWC_USB3_EN_USB2_ONLY==1 || DWC_USB3_NUM_SSIC_PORTS!=0) ? "untestable" : "unconstrained"}

1.3.2 DCTL

- **Description:** Device Control Register
  Note:
  When Hibernation is not enabled using GCTL.GblHibernationEn field,
  - you can write any value to CSS, CRS, L1HibernationEn, and KeepConnect fields
  - L1HibernationEn, and KeepConnect fields always return 0 when read in this hibernation-disabled state
- **Size:** 32 bits
- **Offset:** 0xc704
- **Exists:** Always

31	30	29	28:24	23	22	23:20	19	18	17	16	15:13	12	11	10	9	8:5	4:1	0
RUN_STOP	CSFTRST	reserved_29	HIRDTHRES	APPL1RES	Rsvd	LPM_NYET_thres	KeepConnect	L1HibernationEn	CRS	CSS	reserved_15_13	INITU2ENA	ACCEPTU2ENA	INITU1ENA	ACCEPTU1ENA	ULSTCHNGREQ	TSTCTL	reserved_0

Table 1-79 Fields for Register: DCTL

Bits	Name	Memory Access	Description
31	RUN_STOP	R/W	Run/Stop The software writes 1 to this bit to start the device controller operation. To stop the device controller operation, the software must remove any active transfers and write 0 to this bit. When the controller is stopped, it sets the DSTS.DevCtrlHlt bit when the controller is idle and the lower layer finishes the disconnect process. The Run/Stop bit must be used in following cases as specified: ■ After power-on reset and CSR initialization, the software must write 1 to this bit to start the device controller. The controller does not signal connect to the host until this bit is set. ■ The software uses this bit to control the device controller to perform a soft disconnect. When the software writes 0 to this bit, the host does not see that the device is connected. The device controller stays in the disconnected state until the software writes 1 to this bit. The minimum duration of keeping this bit cleared is specified in the Note below. If the software attempts a connect after the soft disconnect or detects a disconnect event, it must set DCTL[8:5] to 5 before reasserting the Run/Stop bit. ■ When the USB or Link is in a lower power state and the Two Power Rails configuration is selected, software writes 0 to this bit to indicate that it is going to turn off the Core Power Rail. After the software turns on the Core Power Rail again and re-initializes the device controller, it must set this bit to start the device controller. For more details, see "Low Power Operation" section of the Databook. **Note:**
31...(cont.)	RUN_STOP.	R/W	The following is the minimum duration under various conditions for which the soft disconnect (SftDiscon) bit must be set for the USB host to detect a device disconnect: *30ms:* ■ For SuperSpeed, when the device state is Suspended, Idle, Transmit, or Receive. *10ms:* ■ For high-speed, when the device state is Suspended, Idle, or not Idle/Suspended (performing transactions) ■ For full-speed/low-speed, when the device state is Suspended, Idle, or not Idle/Supended (performing transactions) To accommodate clock jitter, it is recommended that the application add extra delay to the specified minimum duration. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
30	CSFTRST	R/W1S	Core Soft Reset Resets the all clock domains as follows: ■ This bit clears the interrupts and all the CSRs except GSTS, GSNPSID, GGPIO, GUID, GUSB2PHYCFGn registers, GUSB3PIPECTLn registers, DCFG, DCTL, DEVTEN, and DSTS registers. ■ All module state machines (except the SoC Bus Slave Unit) are reset to the IDLE state, and all the TxFIFOs and the RxFIFO are flushed. ■ Any transactions on the SoC bus Master are terminated as soon as possible, after gracefully completing the last data phase of a SoC bus transfer. Any transactions on the USB are terminated immediately. The application can write this bit at any time to reset the controller. This is a self-clearing bit; the controller clears this bit after all necessary logic is reset in the controller, which may take several clocks depending on the current state of the controller. Once this bit is cleared, the software must wait at least 3 PHY clocks before accessing the PHY domain (synchronization delay). Typically, software reset is used during software development and also when you dynamically change the PHY selection bits in the USB configuration registers listed above. When you change the PHY, the corresponding clock for the PHY is selected and used in the PHY domain. Once a new clock is selected, the PHY domain must be reset for proper operation. Note: Programming this field with random data causes side effect . Bit Bash register testing is not recommended. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Volatile:** true
29	reserved_29	R	Reserved1 **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
28:24	HIRDTHRES	R/W	HIRD Threshold (HIRD_Thres) The controller asserts output signals utmi_l1_suspend_n and utmi_sleep_n (see "LPM Interface Signals" table in the Databook) on the basis of this signal: The controller asserts utmi_l1_suspend_n to put the PHY into Deep Low-Power mode in L1 when both of the following are true: ■ HIRD value is greater than or equal to the value in DCTL.HIRD_Thres[3:0] ■ HIRD_Thres[4] is set to 1'b1. The controller asserts utmi_sleep_n on L1 when one of the following is true: ■ If the HIRD value is less than HIRD_Thres[3:0] or ■ HIRD_Thres[4] is set to 1'b0. **Note:** This field must be set to '0' during SuperSpeed mode of operation. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
23	APPL1RES	R/W	LPM Response Programmed by Application When LPM Errata is disabled: Bit [23]: LPM Response Programmed by Application (AppL1Res) Handshake response to LPM token specified by device application. Response depends on DCFG.LPMCap. ■ DCFG.LPMCap is 1'b0 . The controller always responds with Timeout (that is, no response). ■ DCFG.LPMCap is 1'b1 . The controller response is based on the value of this bit: ■ 1'b0: The controller responds with an ACK upon a successful LPM transaction, which requires all of the following are satisfied. There are no PID/CRC5 errors in both the EXT token and the LPM token (if not true, inactivity results in a timeout ERROR). A valid bLinkState = 0001B (L1) is received in the LPM transaction (else STALL). No data is pending in the Transmit FIFO and OUT endpoints not in flow controlled state (else NYET). ■ 1'b1: The controller responds with an ACK upon a successful LPM, independent of transmit FIFO status and OUT endpoint flow control state. The LPM transaction is successful if all of the following are satisfied. ■ There are no PID/CRC5 errors in both the EXT token and the LPM token (else ERROR) ■ A valid bLinkState = 0001B (L1) is received in the LPM transaction (else STALL) Bits [22:20]: Reserved **Value After Reset:** 0x0 **Exists:** DWC_USB3_EN_LPM_ERRATA == 0
22			**Reserved Field:** Yes

Bits	Name	Memory Access	Description
23:20	LPM_NYET_thres	R/W	LPM NYET Threshold When LPM Errata is enabled: Bits [23:20]: LPM NYET Response Threshold (LPM_NYET_thres) Handshake response to LPM token specified by device application. Response depends on DCFG.LPMCap. ■ DCFG.LPMCap is 1'b0 - The controller always responds with Timeout (that is, no response). ■ DCFG.LPMCap is 1'b1 - The controller responds with an ACK on successful LPM transaction, which requires that all of the following are satisfied: ■ There are no PID or CRC5 errors in both the EXT token and the LPM token (if not true, inactivity results in a timeout ERROR). ■ No data is pending in the TxFIFO and RxFIFO is empty (else NYET). ■ The BESL value in the LPM token is less than or equal to LPM_NYET_thres[3:0] **Value After Reset:** 0xf **Exists:** Always
19	KeepConnect	R/W	Keep Connect When '1', this bit enables the save and restore programming model by preventing the controller from disconnecting from the host when DCTL.RunStop is set to '0'. It also enables the Hibernation Request Event to be generated when the link goes to U3 or L2. The device controller disconnects from the host when DCTL.RunStop is set to '0'. This bit indicates whether to preserve this behavior ('0'), or if the controller must not disconnect when RunStop is set to 0 ('1'). This bit also prevents the LTSSM from automatically going to U0/L0 when the host requests resume from U3/L2. **Note:** If Hibernation is disabled, that is, GCTL[1].GblHibernationEn = 0, this bit is tied to zero. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
18	L1HibernationEn	R/W	L1HibernationEn When this bit is set along with KeepConnect, the device controller generates a Hibernation Request Event if L1 is enabled and the HIRD value in the LPM token is larger than the threshold programmed in DCTL.HIRD_Thres. The controller does not exit the LPM L1 state until software writes Recovery into the DCTL.ULStChngReq field. This prevents corner cases where the device is entering hibernation at the same time the host is attempting to exit L1. **Note:** If Hibernation is disabled, that is, GCTL[1].GblHibernationEn = 0, this bit is tied to zero. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
17	CRS	R/W	Controller Restore State (CRS) This command is similar to the USBCMD.CRS bit in host mode and initiates the restore process. When software sets this bit to '1', the controller immediately sets DSTS.RSS to '1'. When the controller has finished the restore process, it sets DSTS.RSS to '0'. **Note:** When read, this field always returns '0'. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
16	CSS	R/W	Controller Save State (CSS) This command is similar to the USBCMD.CSS bit in host mode and initiates the save process. When software sets this bit to '1', the controller immediately sets DSTS.SSS to '1'. When the controller has finished the save process, it sets DSTS.SSS to '0'. **Note:** When read, this field always returns '0'. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:13	reserved_15_13	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7 **Write Constraint:** writeAsRead

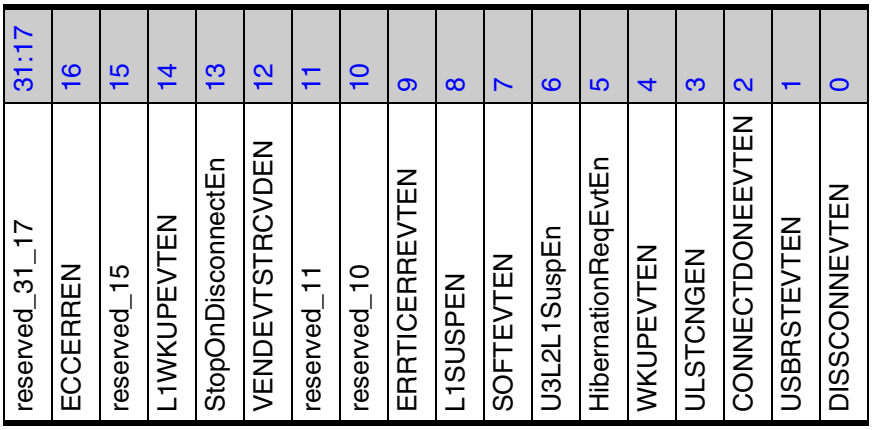
Bits	Name	Memory Access	Description
12	INITU2ENA	R/W	Initiate U2 Enable ■ 1'b0: May not initiate U2 (default) ■ 1'b1: May initiate U2 On USB reset, hardware clears this bit to 0. Software sets this bit after receiving SetFeature(U2_ENABLE), and clears this bit when ClearFeature(U2_ENABLE) is received. If DCTL[11] (AcceptU2Ena) is 0, the link immediately exits U2 state. **Value After Reset:** 0x0 **Exists:** Always
11	ACCEPTU2ENA	R/W	Accept U2 Enable ■ 1'b0: Reject U2 except when Force_LinkPM_Accept bit is set (default) ■ 1'b1: Controller accepts transition to U2 state if nothing is pending on the application side. On USB reset, hardware clears this bit to 0. Software sets this bit after receiving a SetConfiguration command. **Value After Reset:** 0x0 **Exists:** Always
10	INITU1ENA	R/W	Initiate U1 Enable ■ 1'b0: May not initiate U1 (default); ■ 1'b1: May initiate U1. On USB reset, hardware clears this bit to 0. Software sets this bit after receiving SetFeature(U1_ENABLE), and clears this bit when ClearFeature(U1_ENABLE) is received. If DCTL[9] (AcceptU1Ena) is 0, the link immediately exits U1 state. **Value After Reset:** 0x0 **Exists:** Always
9	ACCEPTU1ENA	R/W	Accept U1 Enable ■ 1'b0: Controller rejects U1 except when Force_LinkPM_-Accept bit is set (default) ■ 1'b1: Controller accepts transition to U1 state if nothing is pending on the application side. On USB reset, hardware clears this bit to 0. Software sets this bit after receiving a SetConfiguration command. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
8:5	ULSTCHNGREQ	W	ULSTCHNGREQ Software writes this field to issue a USB/Link state change request. A change in this field indicates a new request to the controller. If software wants to issue the same request back-to-back, it must write a 0 to this field between the two requests. The result of the state change request is reflected in the USB/Link State in DSTS. These bits are self-cleared on the MAC Layer exiting suspended state. If software is updating other fields of the DCTL register and not intending to force any link state change, then it must write a 0 to this field. SS Compliance mode is normally entered and controlled by the remote link partner. Refer to the USB 3.0 specification. Alternatively, you can force the local link directly into compliance mode, by resetting the SS link with the RUN/STOP bit set to zero. If you then write '10' to the USB/Link State Change field and '1' to RUN/STOP, the link goes to compliance mode. Once you are in compliance, you may alternately write zero and '10' to this field to advance the compliance pattern. In SS mode: ■ Value Requested Link State Transition/Action ■ 0 No Action ■ 4 SS.Disabled ■ 5 Rx.Detect ■ 6 SS.Inactive ■ 8 U3 exit request ■ 10 Compliance ■ Others: Reserved In HS/FS/LS mode: ■ ValueRequested USB state transition ■ 8 Remote wakeup request ■ Others: Reserved The Remote wakeup request must be issued 2us after the device goes into suspend state (DSTS[21:18] is 3 - refer to Table "Fields for Register: DSTS"). **Note:** After coming out of hibernation, software must write 8 (Recovery) into this field to confirm exit from the suspended state. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
4:1	TSTCTL	R/W	Test Control ■ 4'b000: Test mode disabled ■ 4'b001: Test_J mode ■ 4'b010: Test_K mode ■ 4'b011: Test_SE0_NAK mode ■ 4'b100: Test_Packet mode ■ 4'b101: Test_Force_Enable ■ Others: Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead
0	reserved_0	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead

1.3.3 DEVTEN

- **Description:** Device Event Enable Register
  This register controls the generation of device-specific events (see "Event Buffer Content for Device-Specific Events (DEVT)" section). If an enable bit is set to 0, the event will not be generated.
- **Size:** 32 bits
- **Offset:** 0xc708
- **Exists:** Always



31:17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved_31_17	ECCERREN	reserved_15	L1WKUPEVTEN	StopOnDisconnectEn	VENDEVTSTRCVDEN	reserved_11	reserved_10	ERRTICERREVTEN	L1SUSPEN	SOFTEVTEN	U3L2L1SuspEn	HibernationReqEvtEn	WKUPEVTEN	ULSTCNGEN	CONNECTDONEEVTEN	USBRSTEVTEN	DISSCONNEVTEN

Table 1-80 Fields for Register: DEVTEN

Bits	Name	Memory Access	Description
31:17	reserved_31_17	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7fff **Write Constraint:** writeAsRead
16	ECCERREN	* Varies	ECC Error Enable. If this bit is set to 1, the controller reports an ECC error to the software when an uncorrectable ECC occurs internally. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_EN_ECC==1) ? "read-write" : "read-only"}

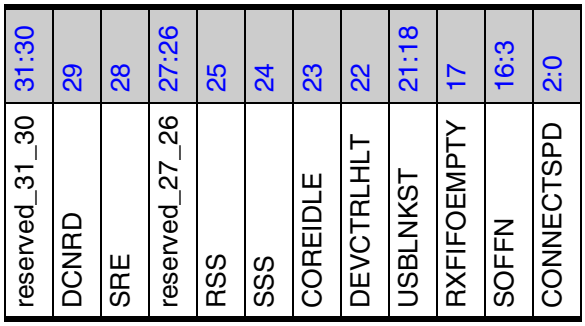
Bits	Name	Memory Access	Description
15	reserved_15	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
14	L1WKUPEVTEN	R/W	L1 Resume Detected Event Enable. **Note:** If GUCTL1[DEV_DECOUPLE_L1L2_EVT] is enabled, then this bit is for L1 Resume Detected Event Enable. **Value After Reset:** 0x0 **Exists:** Always
13	StopOnDisconnectEn	* Varies	Reserved **Value After Reset:** 0x0 **Exists:** DWC_USB3_NUM_SSIC_PORTS != 0 && DWC_USB3_EN_PWROPT==2 **Write Constraint:** writeAsRead **Memory Access:** {(DWC_USB3_NUM_SSIC_PORTS!=0) ? "read-write" : "read-only"}
12	VENDEVTSTRCVDEN	R/W	Vendor Device Test LMP Received Event (VndrDevTstRcvedEn) **Value After Reset:** 0x0 **Exists:** Always
11	reserved_11	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
10	reserved_10	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
9	ERRTICERREVTEN	R/W	Erratic Error Event Enable **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
8	L1SUSPEN	R/W	L1 Suspend Event Enable **Note:** Only if GUCTL1[DEV_DECOUPLE_L1L2_EVT] is enabled, this bit is for L1 Suspend Event Enable. **Value After Reset:** 0x0 **Exists:** Always
7	SOFTEVTEN	R/W	Start of (u)frame **Value After Reset:** 0x0 **Exists:** Always
6	U3L2L1SuspEn	R/W	U3/L2 or U3/L2L1 Suspend Event Enable. **Note:** ■ If GUCTL1[DEV_DECOUPLE_L1L2_EVT] is enabled, then this bit is for U3/L2 Suspend Event Enable. ■ If GUCTL1[DEV_DECOUPLE_L1L2_EVT] is not enabled, then this bit is for U3/L2L1 Suspend Event Enable. **Value After Reset:** 0x0 **Exists:** Always
5	HibernationReqEvtEn	R/W	This bit enables/disables the generation of the Hibernation Request Event. **Value After Reset:** 0x0 **Exists:** Always
4	WKUPEVTEN	R/W	U3/L2 or U3/L2L1 Resume Detected Event Enable. **Note:** ■ If GUCTL1[DEV_DECOUPLE_L1L2_EVT] is enabled, then this bit is for U3/L2 Resume Detected Event Enable. ■ If GUCTL1[DEV_DECOUPLE_L1L2_EVT] is not enabled, then this bit is for U3/L2L1 Resume Detected Event Enable. **Value After Reset:** 0x0 **Exists:** Always
3	ULSTCNGEN	R/W	USB/Link State Change Event Enable **Value After Reset:** 0x0 **Exists:** Always
2	CONNECTDONEEVTEN	R/W	Connection Done Enable **Value After Reset:** 0x0 **Exists:** Always
1	USBRSTEVTEN	R/W	USB Reset Enable **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
0	DISSCONNEVTEN	R/W	Disconnect Detected Event Enable **Value After Reset:** 0x0 **Exists:** Always

1.3.4 DSTS

- **Description:** Device Status Register
  This register indicates the status of the device controller with respect to USB-related events.
  Note:
  When Hibernation is not enabled, RSS and SSS fields always return 0 when read.
- **Size:** 32 bits
- **Offset:** 0xc70c
- **Exists:** Always



31:30	29	28	27:26	25	24	23	22	21:18	17	16:3	2:0
reserved_31_30	DCNRD	SRE	reserved_27_26	RSS	SSS	COREIDLE	DEVCTRLHLT	USBLNKST	RXFIFOEMPTY	SOFFN	CONNECTSPD

Table 1-81 Fields for Register: DSTS

Bits	Name	Memory Access	Description
31:30	reserved_31_30	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Volatile:** true **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
29	DCNRD	R	Device Controller Not Ready The bit indicates that the controller is in the process of completing the state transitions after exiting from hibernation. To complete the state transitions, it takes 256 bus clock cycles from the time DCTL[31].Run/Stop is set. During hibernation, if the UTMI/ULPI PHY is in suspended state, then the 256-bus clock cycle delay starts after the PHY exited suspended state. Software must set DCTL[31].Run/Stop to '1' and wait for this bit to be de-asserted to zero before processing DSTS.USBLnkSt. This bit is valid only when DWC_USB3_EN_PWROPT is set to 2 and GCTL[1].GblHibernationEn =1. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead
28	SRE	R/W1C	Save Restore Error. Currently not supported. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x1 **Volatile:** true
27:26	reserved_27_26	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3 **Volatile:** true **Write Constraint:** writeAsRead
25	RSS	R	RSS Restore State Status This bit is similar to the USBSTS.RSS in host mode. When the controller finishes the restore process, it completes the command by setting DSTS.RSS to '0'. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
24	SSS	R	SSS Save State Status This bit is similar to the USBSTS.SSS in host mode. When the controller has finished the save process, it completes the command by setting DSTS.SSS to '0'. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead
23	COREIDLE	R	Core Idle The bit indicates that the controller finished transferring all RxFIFO data to system memory, writing out all completed descriptors, and all Event Counts are zero. Note: While testing for Reset values, mask out the read value. This bit represents the changing state of the controller and does not hold a static value. **Value After Reset:** 0x1 **Exists:** Always **Testable:** untestable **Volatile:** true **Write Constraint:** writeAsRead
22	DEVCTRLHLT	R	Device Controller Halted This bit is set to 0 when the Run/Stop bit in the DCTL register is set to 1. The controller sets this bit to 1 when, after SW sets Run/Stop to 0, the controller is idle and the lower layer finishes the disconnect process. When Halted=1, the controller does not generate Device events. **Note:** ■ The controller does not set this bit to 1 if GEVNTCOUNTn has some valid value. Software needs to acknowledge the events that are generated (by writing to GEVNTCOUNTn) while it is waiting for this bit to be set to 1. ■ When Interrupt Moderation is enabled, there could be delay in raising the interrupt line when the event count is non-zero. Software should read the GEVNTCOUNT register directly and acknowledge them. **Value After Reset:** 0x1 **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
21:18	USBLNKST	R	USBLNKST. USB/Link State *In SS mode:* LTSSM State ■ 4'h0: U0 ■ 4'h1: U1 ■ 4'h2: U2 ■ 4'h3: U3 ■ 4'h4: SS_DIS ■ 4'h5: RX_DET ■ 4'h6: SS_INACT ■ 4'h7: POLL ■ 4'h8: RECOV ■ 4'h9: HRESET ■ 4'ha: CMPLY ■ 4'hb: LPBK ■ 4'hf: Resume/Reset *In HS/FS/LS mode:* ■ 4'h0: On state ■ 4'h2: Sleep (L1) state ■ 4'h3: Suspend (L2) state ■ 4'h4: Disconnected state (Default state) ■ 4'h5: Early Suspend state (valid only when Hibernation is disabled, GCTL[1].GblHibernationEn = 0) ■ 4'he: Reset (valid only when Hibernation is enabled, GCTL[1].GblHibernationEn = 1) ■ 4'hf: Resume (valid only when Hibernation is enabled, GCTL[1].GblHibernationEn = 1) The link state Resume/Reset indicates that the controller received a resume or USB reset request from the host while the link was in hibernation. Software must write '8' (Recovery) to the DCTL.ULStChngReq field to acknowledge the resume/reset request. When Hibernation is enabled, GCTL[1].GblHibernationEn = 1, this field USBLnkSt is valid only when DCTL[31].Run/Stop set to '1' and DSTS[29].DCNRD = 0. The Early Suspend link state is an early indication of device suspend in HS/FS. The link state changes to Early Suspend after detecting bus idle for 3ms.

Bits	Name	Memory Access	Description
21:18...(cont.)	USBLNKST.	R	■ In HS operation, this is an indication that the USB bus (that is, LineState) has been in idle (SE0) for 3ms. However, it does not confirm whether the next process is Suspend or Reset. The device checks the bus again after pull up enable delay and if the line state indicates Suspend (full speed J), then the device waits for an additional time (~3ms) to indicate the actual Suspend state. ■ In FS operation, this is an indication that the USB bus (that is, LineState) has been in idle (J) for 3ms. The device waits for an additional time (~3ms of Idle) to indicate the actual Suspend state. **Value After Reset:** 0x4 **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead
17	RXFIFOEMPTY	R	RxFIFO Empty. **Value After Reset:** 0x1 **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead
16:3	SOFFN	R	Frame/Microframe Number of the Received SOF. When the controller is operating at SuperSpeed, ■ [16:3] indicates the uframe/ITP number When the controller is operating at high-speed, ■ [16:6] indicates the frame number ■ [5:3] indicates the microframe number When the controller is operating at full-speed, ■ [16:14] is not used. Software can ignore these 3 bits ■ [13:3] indicates the frame number **Note:** After power-on reset, the controller generates the microframe number internally for every 125us if the USB host has not issued SOF/ITP yet. During P3 state, the duration of SOFFN is based on the suspend_clk frequency. **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
2:0	CONNECTSPD	R	Connected Speed (ConnectSpd) Indicates the speed at which the DWC_usb3 controller has come up after speed detection through a chirp sequence. ■ 3'b100: SuperSpeed (PHY clock is running at 125 or 250 MHz) ■ 3'b000: High-speed (PHY clock is running at 30 or 60 MHz) ■ 3'b001: Full-speed (PHY clock is running at 30 or 60 MHz) Low-speed is not supported for devices using a UTMI+ PHY. **Values:** ■ 0x4 (SuperSpeed): SuperSpeed (PHY clock is running at 125 or 250 MHz) ■ 0x0 (HighSpeed): High-speed (PHY clock is running at 30 or 60 MHz) ■ 0x1 (FullSpeed): Full-speed (PHY clock is running at 30 or 60 MHz) **Value After Reset:** {DWC_USB3_EN_USB2_ONLY==1 ? 0x1 : 0x4} **Exists:** Always **Testable:** writeAsRead **Volatile:** true **Write Constraint:** writeAsRead

1.3.5 DGCMDPAR

- **Description:** Device Generic Command Parameter Register
  This register indicates the device command parameter. This must be programmed before or along with the device command. The available device commands are listed in DGCMD register.
- **Size:** 32 bits
- **Offset:** 0xc710
- **Exists:** Always

Table 1-82 Fields for Register: DGCMDPAR

Bits	Name	Memory Access	Description
31:0	PARAMETER	R/W	PARAMETER **Value After Reset:** 0x0 **Exists:** Always

1.3.6 DGCMD

- **Description:** Device Generic Command Register
  This register enables software to program the controller using a single generic command interface to send link management packets and notifications. This register contains command, control, and status fields relevant to the current generic command, while the DGCMDPAR register provides the command parameter.
- **Size:** 32 bits
- **Offset:** 0xc714
- **Exists:** Always

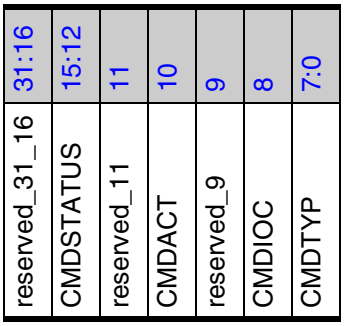


Table 1-83 Fields for Register: DGCMD

Bits	Name	Memory Access	Description
31:16	reserved_31_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffff **Write Constraint:** writeAsRead
15:12	CMDSTATUS	R	Command Status ■ 1: CmdErr: Indicates that the device controller encountered an error while processing the command. ■ 0: Indicates command success **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
11	reserved_11	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
10	CMDACT	R/W1S	Command Active The software sets this bit to 1 to enable the device controller to execute the generic command. The device controller sets this bit to 0 after executing the command. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
9	reserved_9	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
8	CMDIOC	R/W	Command Interrupt on Complete When this bit is set, the device controller issues a Generic Command Completion event after executing the command. Note that this interrupt is mapped to DCFG.IntrNum. Note: This field must not set to '1' if the DCTL.RunStop field is '0'. **Value After Reset:** 0x0 **Exists:** Always
7:0	CMDTYP	R/W	CMDTYP Generic Command Type Specifies the type of generic command the software driver is requesting the controller to perform. ■ 02h: Set Periodic Parameters ■ 04h: Set Scratchpad Buffer Array Address Lo ■ 05h: Set Scratchpad Buffer Array Address Hi ■ 07h: Transmit Device Notification ■ 09h: Selected FIFO Flush ■ 0Ah: All FIFO Flush ■ 0Ch: Set Endpoint NRDY ■ 10h: Run SoC Bus LoopBack Test ■ 11h: Restart After Disconnect All other values are reserved. **Value After Reset:** 0x0 **Exists:** Always

1.3.7 DALEPENA

- **Description:** Device Active USB Endpoint Enable Register.
  This register indicates whether a USB endpoint is active in a given configuration or interface.
- **Size:** 32 bits
- **Offset:** 0xc720
- **Exists:** Always

Table 1-84 Fields for Register: DALEPENA

Bits	Name	Memory Access	Description
31:0	USBACTEP	R/W	USBACTEP USB Active Endpoints (USBActEP) This field indicates if a USB endpoint is active in the current configuration and interface. It applies to USB IN endpoints 0.15 and OUT endpoints 0.15, with one bit for each of the 32 possible endpoints. Even numbers are for USB OUT endpoints, and odd numbers are for USB IN endpoints, as follows: ■ Bit[0]: USB EP0-OUT ■ Bit[1]: USB EP0-IN ■ Bit[2]: USB EP1-OUT ■ Bit[3]: USB EP1-IN The entity programming this register must set bits 0 and 1 because they enable control endpoints that map to physical endpoints (resources) after USBReset. Hardware clears these bits for all endpoints (other than EP0-OUT and EP0-IN) after detecting a USB reset event. After receiving SetConfiguration and SetInterface requests, the application must program endpoint registers accordingly and set these bits. For more information, see "Flexible Endpoint Mapping" section in the Databook. **Value After Reset:** 0x0 **Exists:** Always

1.3.8 Rsvd[0:31]

- **Description:** Reserved
- **Size:** 32 bits
- **Offset:** 0xc724 + (*i* * 0x4), where *i* = 0..31
- **Exists:** Always

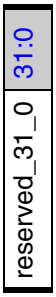


Table 1-85 Fields for Register: Rsvd[0:31]

Bits	Name	Memory Access	Description
31:0	reserved_31_0	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffffffff **Write Constraint:** writeAsRead

1.3.9 DEPCMDPAR2[0:7]

- **Description:** Device Physical Endpoint-n Command Parameter 2 Register (DEPCMDPAR2n)
  This register indicates the physical endpoint command Parameter 2. It must be programmed before issuing the command.
- **Size:** 32 bits
- **Offset:** 0xc800 + (*i* * 0x10), where *i* = 0..=[DWC_USB3_NUM_EPS]-1
- **Exists:** Always

Table 1-86 Fields for Register: DEPCMDPAR2[0:7]

Bits	Name	Memory Access	Description
31:0	PARAMETER	R/W	PARAMETER **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffffffff **Write Constraint:** writeAsRead

1.3.10 DEPCMDPAR1[0:7]

- **Description:** Device Physical Endpoint-n Command Parameter 1 Register (DEPCMDPAR1n)
- **Size:** 32 bits
- **Offset:** 0xc804 + (*i* * 0x10), where *i* = 0..=[DWC_USB3_NUM_EPS]-1
- **Exists:** Always

Table 1-87 Fields for Register: DEPCMDPAR1[0:7]

Bits	Name	Memory Access	Description
31:0	PARAMETER	R/W	PARAMETER **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffffffff **Write Constraint:** writeAsRead

1.3.11 DEPCMDPAR0[0:7]

- **Description:** Device Physical Endpoint-n Command Parameter 0 Register (DEPCMDPAR0n)
- **Size:** 32 bits
- **Offset:** 0xc808 + (*i* * 0x10), where *i* = 0..=[DWC_USB3_NUM_EPS]-1
- **Exists:** Always

Table 1-88 Fields for Register: DEPCMDPAR0[0:7]

Bits	Name	Memory Access	Description
31:0	PARAMETER	R/W	PARAMETER **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffffffff **Write Constraint:** writeAsRead

1.3.12 DEPCMD[0:7]

- **Description:** Device Physical Endpoint-n Command Register
  This register enables software to issue physical endpoint-specific commands. This register contains command, control, and status fields relevant to the current generic command, while the DEPCMDPAR[2:0]n registers provide command parameters and return status information.
  Several fields (including Command Type) are write-only, so their read values are undefined. After power-on, prior to issuing the first endpoint command, the read value of this register is undefined. In particular, the CmdAct bit may be set after power-on. In this case, it is safe to issue an endpoint command.
- **Size:** 32 bits
- **Offset:** 0xc80c + (*i* * 0x10), where *i* = 0..=[DWC_USB3_NUM_EPS]-1
- **Exists:** Always

31:16	15:12	11	10	9	8	7:4	3:0
COMMANDPARAM	CMDSTATUS	HIPRI_FORCERM	CMDACT	reserved_9	CMDIOC	reserved_7_4	CMDTYP

Table 1-89 Fields for Register: DEPCMD[0:7]

Bits	Name	Memory Access	Description
31:16	COMMANDPARAM	R/W	Command Parameters or Event Parameters Command Parameters (CommandParam), when this register is written: For Start Transfer command: ■ [31:16]: StreamID. The USB StreamID assigned to this transfer For Start Transfer command applied to an isochronous endpoint ■ [31:16]: StartMicroFramNum: Indicates the (micro)frame number to which the first TRB applies. For Update Transfer, End Transfer, and Start New Configuration commands ■ [22:16]: Transfer Resource Index (XferRscIdx). The hardware-assigned transfer resource index for the transfer, which was returned in response to the Start Transfer command. The application software-assigned transfer resource index for a Start New Configuration command. Event Parameters (EventParam), when this register is read. Refer to bits [31:16] in Table "Device Endpoint-n Events: DEPEVT". **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xffff **Write Constraint:** writeAsRead
15:12	CMDSTATUS	R/W	Command Completion Status (CmdStatus) Additional information about the completion of this command is available in this field. The information is in the same format as bits 15:12 of the Endpoint Command Complete event, see "Device Endpoint-n Events: DEPEVT" table in the Programming Guide. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xf **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
11	HIPRI_FORCERM	R/W	HighPriority/ForceRM (HiPri_ForceRM) ■ HighPriority: Only valid for Start Transfer command ■ ForceRM: Only valid for End Transfer command ■ ClearPendIN: Only valid for Clear Stall command . Software sets this bit to clear any pending IN transaction (on that endpoint) stuck at the lower layers when a Clear Stall command is issued. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
10	CMDACT	R/W	Command Active (CmdAct) Software sets this bit to 1 to enable the device endpoint controller to execute the generic command. The device controller sets this bit to 0 when the CmdStatus field is valid and the endpoint is ready to accept another command. This does not imply that all the effects of the previously-issued command have taken place. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
9	reserved_9	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1 **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
8	CMDIOC	R/W	CMDIOC Command Interrupt on Complete (CmdIOC) When this bit is set, the device controller issues a generic Endpoint Command Complete event after executing the command. Note that this interrupt is mapped to DEPCFG.IntrNum. When the DEPCFG command is executed, the command interrupt on completion goes to the interrupt pointed by the DEPCFG.IntrNum in the current command. **Note:** This field must not set to 1 if the DCTL.RunStop field is 0. **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0x1 **Write Constraint:** writeAsRead
7:4	reserved_7_4	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xf **Write Constraint:** writeAsRead
3:0	CMDTYP	R/W	Command Type Specifies the type of command the software driver is requesting the controller to perform. ■ 00h: Reserved ■ 01h: Set Endpoint Configuration - -64 or 96-bit Parameter ■ 02h: Set Endpoint Transfer Resource Configuration - 32-bit Parameter ■ 03h: Get Endpoint State - No Parameter Needed ■ 04h: Set Stall - No Parameter Needed ■ 05h: Clear Stall (see Set Stall) - No Parameter Needed ■ 06h: Start Transfer - 64-bit Parameter ■ 07h: Update Transfer - No Parameter Needed ■ 08h: End Transfer - No Parameter Needed ■ 09h: Start New Configuration - No Parameter Needed **Value After Reset:** 0x0 **Exists:** Always **Testable:** untestable **Reset Mask:** 0xf **Write Constraint:** writeAsRead

1.3.13 DEV_IMOD[0]

- **Description:** Device Interrupt Moderation Register (DEV_IMOD)
  This register controls the Interrupt Moderation feature that allows the device software to throttle the interrupt rate.
  Key Functions:
  - Interrupt Moderation is enabled only when the IMOD Interval is programmed to a non-zero value.
  - Interrupt is asserted whenever the IMOD (down) counter is 0, EVNT_HANDLER_BUSY is 0, and there are pending events (that is, event count is non-zero)
  - GEVNTCOUNT[EVNT_HANDLER_BUSY] is set by hardware when interrupt is asserted, and cleared by software when interrupt processing is completed.
  - The Interrupt line is de-asserted after the first write to the event count.
  - IMOD counter is loaded with IMOD interval whenever the Interrupt line is de-asserted.
- **Size:** 32 bits
- **Offset:** 0xca00 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_DEVICE_NUM_INT]-1
- **Exists:** Always

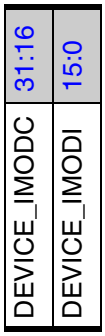


Table 1-90 Fields for Register: DEV_IMOD[0]

Bits	Name	Memory Access	Description
31:16	DEVICE_IMODC	R/W	Interrupt Moderation Down Counter Loaded with the DEVICE_IMODI value, whenever the hardware interrupt(n) line is de-asserted from the asserted state, counts down to 0, and stops. The interrupt(n) is signaled whenever this counter is 0, EVNT_HANDLER_BUSY is 0, and there are pending events (that is, event count is non-zero). This counter may be directly written by software at any time to alter the interrupt rate. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0xffff **Volatile:** true

Bits	Name	Memory Access	Description
15:0	DEVICE_IMODI	R/W	Moderation Interval (DEVICE_IMODI) This field holds the minimum inter-interrupt interval between events. The interval is specified in terms of 250ns increments. A value of 0 disables the interrupt throttling logic and interrupts are generated immediately if event count becomes non-zero. In scaledown simulation mode, 4 ram clocks are used to time 250ns. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0xffff **Volatile:** true

1.4 DWC_usb3_block_bc Registers

1.4.1 BCFG

- **Description:** BC Configuration Register
- **Size:** 32 bits
- **Offset:** 0xcc30
- **Exists:** Always

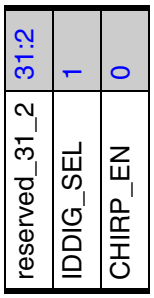


Table 1-91 Fields for Register: BCFG

Bits	Name	Memory Access	Description
31:2	reserved_31_2	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x3fffffff **Write Constraint:** writeAsRead
1	IDDIG_SEL	R/W	This field is not used. **Value After Reset:** 0x0 **Exists:** Always
0	CHIRP_EN	R/W	Chirp Enable When this bit is 1'b1, the controller asserts the bc_chirp_on output signal to indicate an imminent chirp. **Value After Reset:** 0x0 **Exists:** Always

1.4.2 BCEVT

- **Description:** BC Event Register
  Writing 1 to the info bit in this register clears the register bit and the associated interrupt.
- **Size:** 32 bits
- **Offset:** 0xcc38
- **Exists:** Always

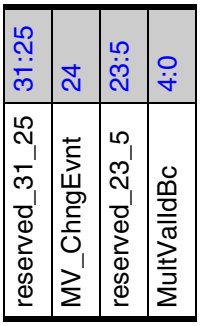


Table 1-92 Fields for Register: BCEVT

Bits	Name	Memory Access	Description
31:25	reserved_31_25	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7f **Write Constraint:** writeAsRead
24	MV_ChngEvnt	R/W1C	Multi-Valued input changed Event: This event indicates that there is a change in the value of at least one ACA pin value. This bit is present only if DWC_USB3_EN_BC = 1, otherwise it is Reserved. **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x1 **Volatile:** true
23:5	reserved_23_5	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7ffff **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
4:0	MultValIdBc	R	Multi Valued ID pin Indicates the Battery Charger ACA inputs. ■ MultValIdBc[4]: bc_rid_float ■ MultValIdBc[3]: bc_rid_gnd ■ MultValIdBc[2]: bc_rid_a ■ MultValIdBc[1]: bc_rid_b ■ MultValIdBc[0]: bc_rid_c **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1f **Volatile:** true **Write Constraint:** writeAsRead

1.4.3 BCEVTEN

- **Description:** BC Event Enable Register
- **Size:** 32 bits
- **Offset:** 0xcc3c
- **Exists:** Always

Table 1-93 Fields for Register: BCEVTEN

Bits	Name	Memory Access	Description
31:25	reserved_31_25	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7f **Write Constraint:** writeAsRead
24	MV_ChngEvntEna	R/W	BCEvtInfoEna[0] Multi-Valued input changed Event Enable (MV_ChngEvntEn) When this bit is set, MV_ChngEvnt in BCEVT register is enabled. **Value After Reset:** 0x0 **Exists:** Always
23:0	reserved_23_0	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffffff **Write Constraint:** writeAsRead

1.5 DWC_usb3_block_link Registers

1.5.1 LU1LFPSRXTIM[0]

- **Description:** U1/U2 LFPS Rx Timer Register
  - Link Layer Register for U1/U2 LFPS RX Timers.
  - This register is common for all SS ports.
- **Size:** 32 bits
- **Offset:** 0xd000 + (*i* * 0x80), where *i* = 0..0
- **Exists:** Always

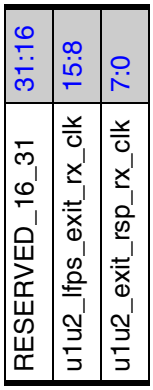


Table 1-94 Fields for Register: LU1LFPSRXTIM[0]

Bits	Name	Memory Access	Description
31:16	RESERVED_16_31	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
15:8	u1u2_lfps_exit_rx_clk	R/W	Programmable U1U2 LFPS EXIT RX CLKS ■ Applicable to Remote Partner initiated Ux exit: Time to recognize valid Ux exit request from the remote partner. ■ This field is encoded as the pipe clk (8ns) count for the LFPS. ❑ 1: 8ns ❑ 2: 16ns ❑ 3: 24ns, and so on **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LU1LFPSRXTIM_INIT" 8 8] **Exists:** Always

Bits	Name	Memory Access	Description
7:0	u1u2_exit_rsp_rx_clk	R/W	Programmable U1U2 EXIT RESP RX CLKS ■ Applicable to locally initiated Ux exit: Minimum LFPS reception from remote to consider Ux exit handshake is successful. ■ This field is encoded as the pipe clk (8ns) count for the LFPS. ❑ 1: 8ns ❑ 2: 16ns ❑ 3: 24ns, and so on **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LU1LFPSRXTIM_INIT" 0 8] **Exists:** Always

1.5.2 LINK_SETTINGS[0]

- **Description:** Link Setting Register
  - Link Layer User Control Register for enabling Link/PHY-specific options.
  - This register is common for all SS ports.
- **Size:** 32 bits
- **Offset:** 0xd020 + (*i* * 0x80), where *i* = 0..0
- **Exists:** Always

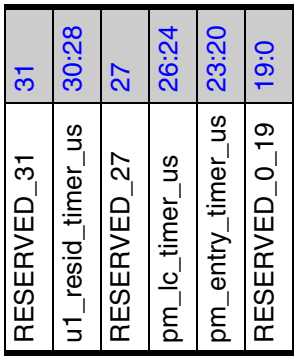


Table 1-95 Fields for Register: LINK_SETTINGS[0]

Bits	Name	Memory Access	Description
31	RESERVED_31	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
30:28	u1_resid_timer_us	R/W	Programmable U1 MIN RESIDENCY TIMER This field specifies U1 MIN RESIDENCY TIMER value in us. Set to 0 to disable the timer. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_SETTINGS_INIT" 28 3] **Exists:** Always
27	RESERVED_27	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
26:24	pm_lc_timer_us	R/W	Programmable PM_LC_TIMER This field specifies PM_LC_TIMER value in us. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_SETTINGS_INIT" 24 3] **Exists:** Always

Bits	Name	Memory Access	Description
23:20	pm_entry_timer_us	R/W	Programmable PM_ENTRY_TIMER This field specifies PM_ENTRY_TIMER value in us. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_SETTINGS_INIT" 20 4] **Exists:** Always
19:0	RESERVED_0_19	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0

1.5.3 LLUCTL[0]

- **Description:** Link User Control Register
  - Link Layer User Control Register for enabling Link/PHY-specific options.
  - This register is common for all SS ports.
- **Size:** 32 bits
- **Offset:** 0xd024 + (*i* * 0x80), where *i* = 0..0
- **Exists:** Always

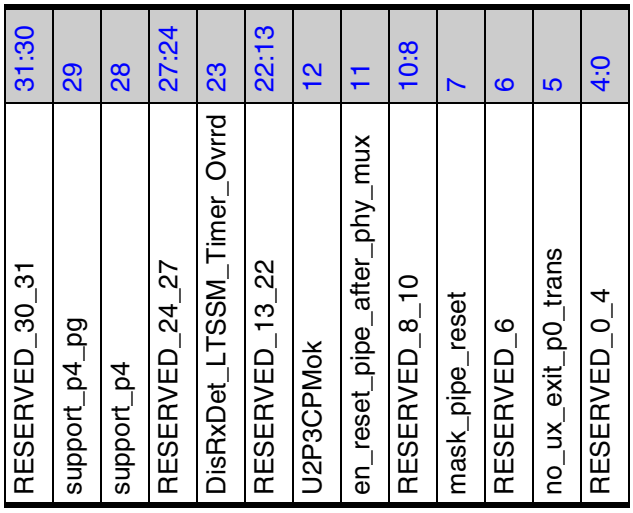


Table 1-96 Fields for Register: LLUCTL[0]

Bits	Name	Memory Access	Description
31:30	RESERVED_30_31	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
29	support_p4_pg	R/W	PHY P4 Power gate mode (PG) is enabled. Set this bit if the PHY supports PG mode in P4. This bit is used only for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LUCTL_INIT" 29 1] **Exists:** Always
28	support_p4	R/W	Support PHY P3.CPM and P4 Power States. When this bit is set, the controller puts the PHY in P3.CPM or P4 in certain states. This bit is used only for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LUCTL_INIT" 28 1] **Exists:** Always

Bits	Name	Memory Access	Description
27:24	RESERVED_24_27	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
23	DisRxDet_LTSSM_Timer_Ovrrd	R/W	DisRxDet_LTSSM_Timer_Ovrrd. When DisRxDetU3RxDet is asserted in Polling or U1, the timeout expires immediately. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LUCTL_INIT" 23 1] **Exists:** Always
22:13	RESERVED_13_22	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
12	U2P3CPMok	R/W	P3CPM OK for U2/SSInactive (U2P3CPMok) ■ 0: During link state U2/SS.Inactive, put PHY in P2 (Default) ■ 1: During link state U2/SS.Inactive, put PHY in P3CPM. **Note:** For a port, if both GUCTL1[25]=1 and LUCTL[12]=1, LUCTL[12]=1 takes priority. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LUCTL_INIT" 12 1] **Exists:** Always
11	en_reset_pipe_after_phy_mux	R/W	en_reset_pipe_after_phy_mux. The controller issues USB 3.0 PHY reset after DisRxDetU3RxDet is de-asserted. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LUCTL_INIT" 11 1] **Exists:** Always
10:8	RESERVED_8_10	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
7	mask_pipe_reset	R/W	Mask pipe reset. If this bit is set, controller blocks pipe_reset_n from going to the PHY when DisRxDetU3RxDet=1. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LUCTL_INIT" 7 1] **Exists:** Always

Bits	Name	Memory Access	Description
6	RESERVED_6	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
5	no_ux_exit_p0_trans	R/W	no_ux_exit_p0_trans. Link LTSSM detects Ux_exit LFPS when P0 transition is on-going by default. If this bit is set, Link LTSSM may miss Ux_exit LFPS when P0 transition is happening. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LUCTL_INIT" 5 1] **Exists:** Always
4:0	RESERVED_0_4	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0

1.5.4 LPTMDPDELAY[0]

- **Description:** Link Datapath Delay Register
  - Link Layer Timer Register for P3CPM/P4 Residency.
  - This register is common for all SS ports.
- **Size:** 32 bits
- **Offset:** 0xd028 + (*i* * 0x80), where *i* = 0..0
- **Exists:** Always

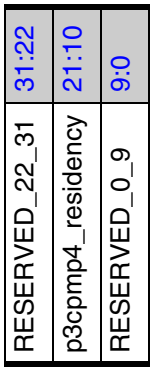


Table 1-97 Fields for Register: LPTMDPDELAY[0]

Bits	Name	Memory Access	Description
31:22	RESERVED_22_31	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0
21:10	p3cpmp4_residency	R/W	p3cpmp4 residency timer value. Minimum number of suspend_clk periods that the controller needs to stay in P3.CPM or P4 before exiting P3.CPM or P4. This field is used only for Synopsys PHY. **Value After Reset:** =mpformat "0x%x" [get_field_val "DWC_USB3_LINK_LPTMDPDELAY_INIT" 10 12] **Exists:** Always
9:0	RESERVED_0_9	R/W	Reserved **Value After Reset:** 0x0 **Exists:** Always **Reset Mask:** 0x0

1.6 DWC_usb3_block_ecc Registers

1.6.1 BRSERRCNT

- **Description:** Block RAM Single Bit Error Count.
  8 bit count for each of the 3 RAMs.
- **Size:** 32 bits
- **Offset:** 0xd850
- **Exists:** Always

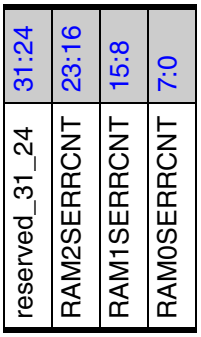


Table 1-98 Fields for Register: BRSERRCNT

Bits	Name	Memory Access	Description
31:24	reserved_31_24	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead
23:16	RAM2SERRCNT	R	RAM2 Single bit Error Count **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:8	RAM1SERRCNT	R	RAM1 Single bit Error Count **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:0	RAM0SERRCNT	R	RAM0 Single bit Error Count **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.6.2 BRMERRCNT

- **Description:** Block RAM Multiple Bit Error Count.
  8 bit count for each of the 3 RAMs.
- **Size:** 32 bits
- **Offset:** 0xd854
- **Exists:** Always

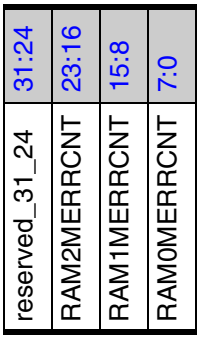


Table 1-99 Fields for Register: BRMERRCNT

Bits	Name	Memory Access	Description
31:24	reserved_31_24	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xff **Write Constraint:** writeAsRead
23:16	RAM2MERRCNT	R	RAM2 Multiple bit Error Count **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
15:8	RAM1MERRCNT	R	RAM1 Multiple bit Error Count **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
7:0	RAM0MERRCNT	R	RAM0 Multiple bit Error Count **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.6.3 BRMECCERR

- **Description:** Block RAM ECC Error Vector Register
- **Size:** 32 bits
- **Offset:** 0xd858
- **Exists:** Always

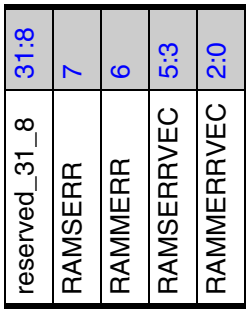


Table 1-100 Fields for Register: BRMECCERR

Bits	Name	Memory Access	Description
31:8	reserved_31_8	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffffff **Write Constraint:** writeAsRead
7	RAMSERR	R	RAM Single bit Error Status **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
6	RAMMERR	R	RAM Multiple bit Error Status **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead
5:3	RAMSERRVEC	R	RAM Single Error Vector **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

Bits	Name	Memory Access	Description
2:0	RAMMERRVEC	R	RAM Multiple Error Vector **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.6.4 BRERRCTL

- **Description:** Block RAM ECC Error Control Register
- **Size:** 32 bits
- **Offset:** 0xd85c
- **Exists:** Always

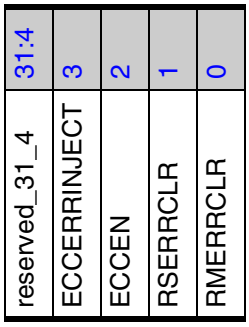


Table 1-101 Fields for Register: BRERRCTL

Bits	Name	Memory Access	Description
31:4	reserved_31_4	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1fffffff **Write Constraint:** writeAsRead
3	ECCERRINJECT	R/W	This bit should be used for testing purposes only. ■ 1: Inject ECC error ■ 0: No ECC error injected **Value After Reset:** 0x0 **Exists:** Always
2	ECCEN	R/W	ECC Enable (1) or Disable (0) **Value After Reset:** DWC_USB3_EN_ECC **Exists:** Always
1	RSERRCLR	W	RAM Single bit Error Clear **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RMERRCLR	W	RAM Multiple bit Error Clear **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

1.6.5 BRAM0ADDRERR

- **Description:** Block RAM Address of RAM0 having uncorrectable error
- **Size:** 32 bits
- **Offset:** 0xd860
- **Exists:** Always

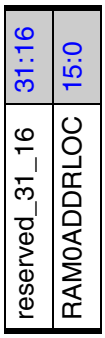


Table 1-102 Fields for Register: BRAM0ADDRERR

Bits	Name	Memory Access	Description
31:16	reserved_31_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffff **Write Constraint:** writeAsRead
15:0	RAM0ADDRLOC	R	RAM0 Address Location **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.6.6 BRAM1ADDRERR

- **Description:** Block RAM Address of RAM1 having uncorrectable error
- **Size:** 32 bits
- **Offset:** 0xd864
- **Exists:** Always

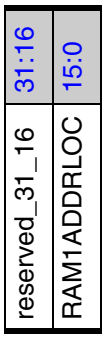


Table 1-103 Fields for Register: BRAM1ADDRERR

Bits	Name	Memory Access	Description
31:16	reserved_31_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffff **Write Constraint:** writeAsRead
15:0	RAM1ADDRLOC	R	RAM1 Address Location **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.6.7 BRAM2ADDRERR

- **Description:** Block RAM Address of RAM2 having uncorrectable error
- **Size:** 32 bits
- **Offset:** 0xd868
- **Exists:** Always

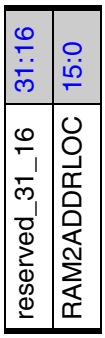


Table 1-104 Fields for Register: BRAM2ADDRERR

Bits	Name	Memory Access	Description
31:16	reserved_31_16	R	Reserved **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0xffff **Write Constraint:** writeAsRead
15:0	RAM2ADDRLOC	R	RAM1 Address Location **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Write Constraint:** writeAsRead

1.7 DWC_usb3_block_debug Registers

1.7.1 BU3RHBDBG(#n) (for n = 0; n <= DWC_USB3_NUM_U3_ROOT_PORTS-1)

- **Description:** U3 Root Hub Debug Register
- **Size:** 32 bits
- **Offset:** 0xd800 + (*i* * 0x4), where *i* = 0..=[DWC_USB3_NUM_U3_ROOT_PORTS]-1
- **Exists:** [<functionof> DWC_USB3_GENERATION] == 30

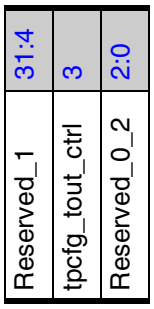


Table 1-105 Fields for Register: BU3RHBDBG(#n) (for n = 0; n <= DWC_USB3_NUM_U3_ROOT_PORTS-1)

Bits	Name	Memory Access	Description
31:4	Reserved_1	R	Reserved_1 **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x1fffffff **Write Constraint:** writeAsRead
3	tpcfg_tout_ctrl	R/W	tpcfg_tout_ctrl This bit controls the USB 3.0 port configuration timeout duration. ■ 1: The port configuration timeout counter resets when the link is not in U0. ■ 0: The port configuration timeout counter does not reset if the link enters recovery or exits U0. **Value After Reset:** 0x0 **Exists:** Always
2:0	Reserved_0_2	R	Reserved_0_2 **Value After Reset:** 0x0 **Exists:** Always **Testable:** writeAsRead **Reset Mask:** 0x7 **Write Constraint:** writeAsRead

2

Additional Register Information

This chapter includes the following topics:

2.1 Controller CSR Memory Map

Table 2-1 shows the controller's CSR memory map. Address bits A0 to A19 are used by the controller (1 MB memory space). For an access to an unused address within the 1 MB memory space, a read completes successfully with all-zero read data, and a write completes successfully with write data being discarded. Use address bits A63:20 as the address decoder; this becomes the base address.

The tables in this section provide a high-level summary of the register address map.

Unused Offsets:

- A read to any unused location successfully completes with "don't care" read data.
- A write to any unused location successfully completes with write data being discarded.

Table 2-1 Register Address Map (Continued)

Address Index	Register Type
0x0_0000 to 0x0_7FFF ■ 0 to CAPLENGTH-1 ■ CAPLENGTH to RTSOFF - 1 ■ RTSOFF to DBOFF - 1 ■ DBOFF to (xECP*4-1) ■ xECP*4 to 0x0_7FFF	xHCI Registers ■ eXtensible Host Controller Capability Registers ■ Host Controller Operational Registers ■ Controller Runtime Registers ■ Doorbell Register ■ xHCI Extended Capabilities
0x0_8000 to 0x0_C0FF	Unused Offset
0x0_C100 to 0x0_C6FF	Global Register Block
0x0_C700 to 0x0_CBFF	Device Register Block
0x0_CC00 to 0x0_CCFF	Battery Charger Register Block
0x0_CD00 to 0x0_CFFF	Unused Offset
0x0_D000 to 0x0_D7FF	Link Register Block
0x0_D800 to 0x0_D93F	Debug/ECC Register Block
0x0_D940 to 0x3_FFFF	Unused Offset
0x4_0000 to 0x7_FFFF	Internal RAM 0 - Debug Access (256KB)
0x8_0000 to 0xB_FFFF	Internal RAM 1 - Debug Access (256KB)
0xC_0000 to 0xF_FFFF	Internal RAM 2 - Debug Access (256KB)

2.2 Register Resets

Table 2-2 summarizes the list of resets for all registers.

Note Registers implemented in RAM are cleared to “0” after a hardware reset or any software reset.

Table 2-2 Resets for Registers (Continued)

Register	Hardware Reset Vcc_reset_n	Device Reset DCTL[30] Controller Soft Reset (CSftRst)	Global Reset GCTL[11] Controller Soft Reset (CoreSoftReset)	Host Reset Host Controller Reset (HCRST)	Light Host Reset Light Host Controller Reset (LHCRST)[a]
GSBUSCFG0	Yes	No	No	No	No
GSBUSCFG1	Yes	No	No	No	No
GTXTHRCFG	Yes	No	No	No	No
GRXTHRCFG	Yes	No	No	No	No
GCTL	Yes	No	No	No	No
GPMSTS	Yes	No	No	No	No
GGPIO	Yes	No	No	No	No
GUID	Yes	No	No	No	No
GUCTL	Yes	No	No	No	No
GPRTBIMAP[31:0]	Yes	No	No	No	No
GPRTBIMAP[63:32]	Yes	No	No	No	No
GPRTBIMAP_HS[31:0]	Yes	No	No	No	No
GPRTBIMAP_HS[63:32]	Yes	No	No	No	No
GPRTBIMAP_FS[31:0]	Yes	No	No	No	No
GPRTBIMAP_FS[63:32]	Yes	No	No	No	No
GUSB2PHYCFGn	Yes	No	No	No	No
GUSB2I2CCTL	Yes	No	No	No	No
GUSB2PHYACCn	Yes	No	No	No	No
GUSB3PIPECTLn	Yes	No	No	No	No
GTXFIFOSIZn	Yes	No	No	No	No
GRXFIFOSIZn	Yes	No	No	No	No
GDBGFIFOSPACE	Yes	No	No	No	No
GDBGLSPMUX	Yes	No	No	No	No

Register	Hardware Reset	Device Reset	Global Reset	Host Reset	Light Host Reset
	Vcc_reset_n	DCTL[30] Controller Soft Reset (CSftRst)	GCTL[11] Controller Soft Reset (CoreSoftReset)	Host Controller Reset (HCRST)	Light Host Controller Reset (LHCRST)[a]
GDMAHLRATIO	Yes	No	No	No	No
GEVNTSIZn	Yes	No	No	No	No
GFIFOPRIDBC	Yes	No	No	No	No
GTXFIFOPRIDEV	Yes	No	No	No	No
GTXFIFOPRIHST	Yes	No	No	No	No
GRXFIFOPRIHST	Yes	No	No	No	No
GFLADJ	Yes	No	No	No	No
DCFG	Yes	No	No	No	No
DCTL	Yes	No	No	No	No
DEVTEN	Yes	No	No	No	No
DSTS_COREIDLE	Yes	No	No	No	No
DGCMDPAR	Yes	Yes	Yes	Yes	Yes
DGCMD	Yes	Yes	Yes	Yes	Yes
DALEPENA	Yes	Yes	Yes	Yes	Yes
BCEVT	Yes	No	Yes	No	No
BCEVTEN	Yes	No	Yes	No	No
USBCMD	Yes	Yes	Yes	Yes	Yes
USBSTS	Yes	Yes	Yes	Yes	Yes
PORTSC	Yes	Yes	Yes	Yes	No
PORTPMSC_SS and PORTPMSC_20	Yes	Yes	Yes	Yes	No
PORTHLPMC_SS2 and PORTHLPMC_20	Yes	Yes	Yes	Yes	No
MFINDEX	Yes	Yes	Yes	Yes	Yes
IMAN[i]	Yes	Yes	Yes	Yes	Yes
IMOD[i]	Yes	Yes	Yes	Yes	Yes
ERSTSZ[i]	Yes	Yes	Yes	Yes	Yes
USBLEGSUP	Yes	Yes	Yes	Yes	No
USBLEGCTLSTS	Yes	Yes	Yes	Yes	No

a. For details on light reset, refer to “LHCRST Behavior” on page 381.

2.3 xHCI Host Registers

For register definitions, refer to the xHCI specification.

2.3.1 Host Controller Capability Registers

For details on Host Controller Capability Registers, see "DWC_usb3_block_eXtensible_Host_Cntrl_Cap_Regs Registers" on page 26.

2.3.2 xHCI Extended Capabilities

The controller supports the following extended Capabilities in host mode:

- USB Legacy Support Capability
- Supported Protocol Capability:
  - USB 2.0
  - USB 3.0 (only when USB 2.0-only mode is not enabled)
- Debug Capability (if you select "Yes" for "Enable xHCI Debug Capability?" in coreConsultant, that is, DWC_USB3_EN_DBC is set to 1)

The addresses are automatically calculated by the controller. For register definitions, refer to sections 7.1, 7.2, and 7.6 of the *xHCI Specification*.

At Power-on, the value of Port Link State in PORTSC (Port Status and Control Register) is 4 (Disable state). After power-on reset, it defaults to 5 (RxDetect state).

2.3.2.1 xHCI Supported Protocol Capability – Data Word 2 (SUPTPRT2_DW2)

Offset: xECP * 4 + USBLEGSUP.Next Capability Pointer * 4 + 0x0008h

Table 2-3 xHCI Supported Protocol Capability – Data Word 2 (SUPTPRT2_DW2)

Field	Description	Reset	Access
20[a]	BESL LPM Capability (BLC) When this bit is set to, ■ 1: The ports described by this xHCI Supported Protocol Capability applies BESL timing to the BESL and BESLD fields of the PORTPMSC and PORTHLPMC registers. ■ 0: The ports described by this xHCI Supported Protocol Capability applies HIRD timing to the BESL and BESLD fields of the PORTPMSC and PORTHLPMC registers.	DWC_USB3_EN_LPM_ERRATA	RO

a. For other register field definitions, refer to section 7.2 of the *xHCI Specification.*

2.3.3 Address Calculations for Host Registers

Table 2-4 shows the address calculations for the host registers.

Table 2-4 Address Calculations for Host Registers

Host Register Type	Starting Address
eXtensible Host Controller Capability Registers	`Base address = 0x0000`. Refer to Additional Register Information.
Host Controller Operational Registers	`Base address + CAPLENGTH`, where `CAPLENGTH` is `DWC_USB3_HOST_CAP_REG_LEN` whose default value is `20h`.
Host Controller Port Register Set	`Base address + CAPLENGTH + 400h`
Host Controller Runtime Registers	`Base address + RTSOFF` where `RTSOFF` is `DWC_USB3_HC_RUNTIME_REGISTER_SPACE_OFFSET`. This parameter is defined as `((DWC_USB3_HOST_CAP_REG_LEN + 'h400 + ((DWC_USB3_HOST_NUM_U3_ROOT_PORTS + DWC_USB3_HOST_NUM_U2_ROOT_PORTS) * 'h10) + 'h10) >> 5) << 5`, where DWC_USB3_HOST_NUM_U3_ROOT_PORTS and DWC_USB3_HOST_NUM_U2_ROOT_PORTS are coreConsultant parameters.
Interrupter Registers	`Base address + RTSOFF + 20h`
Doorbell Register	`Base address + DBOFF` where `DBOFF` is `DWC_USB3_HC_DOORBELL_ARRAY_OFFSET`. This parameter is defined as `(DWC_USB3_HC_RUNTIME_REGISTER_SPACE_OFFSET) + 'h20 + ('h20 * (DWC_USB3_HOST_NUM_INTERRUPTER_SUPT))`, where `USB3_HOST_NUM_INTERRUPTER_SUPT` is a coreConsultant parameter.
xHCI Extended Capabilities	
USB Legacy Support Capability	`Base address + xECP * 4`, where `xECP` is `` `DWC_USB3_HC_XECP ``. This parameter is defined as `DWC_USB3_HC_EXTCAP_USBLEGSUP_OFFSET/4`, where `DWC_USB3_HC_EXTCAP_USBLEGSUP_OFFSET` is `DWC_USB3_HC_DOORBELL_ARRAY_OFFSET + (256*4)`. Refer to Table 23 of xHCI specification for further details.
xHCI Supported Protocol Capability(USB 2.0)	`Addr1` which is calculated as `Base address + xECP * 4 + USBLEGSUP.Next Capability Pointer * 4`. Refer to xHCI specification for more details.
xHCI Supported Protocol Capability(USB 3.0)	`Addr2` which is calculated as `Addr1 + SUPTPRT2_DW0.Next Capability Pointer * 4`. Refer to xHCI specification for more details.

For more details on the parameters used in the address calculations, refer to *<workspace>/src/DWC_usb3_params.v* file.

2.4 Considerations for Register Write and Read Testing

In Host and DRD modes, the DWC_usb3 controller initializes the RAM0 cache after `vcc_reset_n` is removed. The register testing should wait for this to complete. The RAM0 initialization completion is indicated by `Controller Not Ready` field of xHCI `USBSTS` register. This field going low indicates when it is safe to do slave CSR accesses.

For simulation purpose, RAM0 initialization can be bypassed by setting, "`+define+DWC_USB3_SKIP_RTL_RAM_INIT`".

In the VTB, this parameter is defined in the <*workspace*>*/sim/SoC_sim/vtb/com/tb_defines.v* file.

It is recommended not to connect USB Device Verification IP or USB Host Verification IP when doing non-functional random CSR write and read testing. For example, the device mode random CSR read/write can cause USB-Reset scenario from the USB Host VIP, which would initialize some of the internal registers.

2.4.1 CSR Timeout Mechanism

If software attempts to read or write a CSR in the controller which is not immediately accessible (for example, accessing a RAM-based CSR when the `ram_clk` is not running, or a `PORTSC` register when the mac3_clk is not running), the controller accepts the CSR operation on the system bus after `DWC_USB3_CSR_ACCESS_TIMEOUT` bus clock cycles (default is `h1FFFF`).

The `DWC_USB3_CSR_ACCESS_TIMEOUT` defined is in the <*workspace*>*/src/pwrm/DWC_usb3_pwrm_csr.v* file.

If the CSR access was a READ, the read data returned will be unknown (residue from a previous read). If the CSR access was a WRITE, the data to be written may be lost.

After the controller accepts this CSR access due to the timeout, the `CSRTimeout` field of the `GSTS` register will be set to '1'. This indicates the software that a Timeout occurred. Software can clear this field by writing '1' to it.

Table 2-5 gives the list of considerations while performing register write and read testing.

Table 2-5 Register Access Considerations

Register Name	Access Considerations
Global Registers	
Global Common Register (`GCTL`)	`PrtCapDir`: Programming this bit with random data causes the controller to keep toggling between the host mode and the device mode. This is not recommended for register testing. `CoreSoftReset`: This bit will reset the internal logic of the host controller. Under soft reset, some CSR accesses may fail (Timeout) `RAMClkSel`: If RAM clock is changed from its default `bus_clk` to pipe or pipe/2, and the pipe clock is not running, then all CSR accesses to any of the CSR's in the `ram_clk` domain will fail (Timeout).

Register Name	Access Considerations
Device Registers	
Device Control Register (DCTL)	CSftRst: This bit will clear the interrupts and all the CSRs except the following registers: ■ GSTS ■ GSNPSID ■ GGPIO ■ GUID ■ GUSB2PHYCFGn registers ■ GUSB3PIPECTLn registers ■ DCFG ■ DCTL ■ DEVTEN ■ DSTS
Device Configuration Register (DCFG)	DCFG.DevSpd unsupported value will cause device attach; hence not recommended.
Host Registers	
USB Command Register (USBCMD)	The following bits will reset the internal logic of the host controller. Under soft reset, some CSR accesses may fail (timeout). ■ HCRST ■ LHCRST
Port Status and Control Register (PORTSC)	The PORTSC register access will fail (timeout) if pipe clock (for SS), or utmi/ulpi clock (for HS/FS/LS) is not running.
Port PM and Control Register (PORTPMSC)	The PORTPMSC register access will fail (timeout) if pipe clock (for SS) or utmi/ulpi clock (for HS/FS/LS) is not running, or reset is asserted.
Port Hardware LPM Control Register (PORTHLPMC)	The PORTHLPMC register access will fail (timeout) if utmi/ulpi clock is not running or reset is asserted.
Port Link Info Register (PORTLI)	The PORTLI register access will fail (timeout) if pipe clock is not running or reset is asserted.
Doorbell Register (DB)	The Doorbell register access will fail (timeout) when DB FIFO is full.

2.4.1.1 Explanation for PORT CSR Access (PORTSC/PORTPMSC/PORTLI)

The Port CSR access needs to do the handshake with MAC clock domain. Therefore, when the MAC clock is not running or reset is asserted, the synchronizers stop functioning and the port access will timeout. If the port clock and the MAC clock is running and reset is not asserted, the port access behaves like other CSR accesses.

2.4.1.2 CSR Access Under Software Reset

When the controller is under software reset (controlled by GCTL/USBCMD/DCTL), the accesses to the GCTL/USBCMD/DCTL is safe but accesses to other CSRs may fail (timeout).

2.4.1.3 Doorbell CSR Access

All the “Doorbell Register Write” data is pushed inside a FIFO and the LSP module pops the Doorbell register write data. When the FIFO is not full, the Doorbell Register Write timing is same as other CSR writes.

If the LSP is not running, for example, when the run/stop bit in `USBCMD` Register is not set, it would not be able to pop the “Doorbell Register Write Data” from the FIFO. Upon multiple Doorbell writes to the FIFO, it gets full and hence the FIFO will not be able to accept more Doorbell Writes and the CSR access to Doorbell Register will timeout.

If the LSP is running, but the speed of “LSP Data Pop” is slower than “Doorbell Write Push”, then after sometime, the FIFO may get full as well. In this case, a new “Doorbell Write Push” will not be accepted by the FIFO until the LSP pops some data out from the FIFO. This timing is not fixed as the speed of Doorbell Pop handling of the LSP is affected by many factors.

2.5 Usage of Global SoC Bus Configuration Register 0 (GSBUSCFG0)

This section describes the usage of the GSBUSCFG0 register. For register description, see "Register Descriptions" on page 17.

Burst Length

For a given DMA transfer, the burst length is set according to the largest enabled burst length. Note that the Undefined Length INCR burst, if enabled, has priority over all other burst lengths.

Cache Type

For a given DMA transfer, the cache type is set according to the 4-bit cache type register field corresponding to the transfer type:

- Data read
- Descriptor read
- Data write
- Descriptor write

The definition of the 4-bit cache type register field corresponds to the cache type definition of the configured master bus type (AHB, AXI3, Native) as defined below:

Table 2-6 Cache Type Bit Assignments (Continued)

MBUS_TYPE	Cache Type Output	[3]	[2]	[1]	[0]
AHB	hprot[3:0]	Cacheable	Bufferable	Privileged	Data
AXI3	ar/awcache[3:0]	Write Allocate	Read Allocate	Cacheable	Bufferable
Native	gmr/gmw_ mcache_reqinfo[3:0]	Same as AXI	Same as AXI	Same as AXI	Same as AXI

Cache Type in AHB/AXI

- AHB: CSR cache type refers to AHB protection control (hprot[3:0]), which indicates access permissions (privileged, data) as well as cacheable and bufferable.
- AXI3: CSR cache type refers to AXI cache type (awcache[3:0] and arcache[3:0]); or in ACE (AXI coherency extensions) to AXI memory type.

Equivalent Signal Names

The AHB, AXI3, and PCIe buses use different names for certain signals, which have the same meaning:

- Bufferable = Posted
- Cacheable = Modifiable = Snoop (negation of No Snoop)

Posted Requests

The native interface DMA read request gmr_mcmd[2:0] indicates posted vs. non-posted requests depending on the bufferable/posted bit of the cache type.

Table 2-7 Cache Type Bit Definitions and Usage (Continued)

Signal Name	Definition	Usage
Bufferable, Posted	The transaction response can be returned quickly, but the transaction itself can take an arbitrary number of clock cycles to reach the final destination.	■ Improves bus utilization of DMA writes with a high response latency, such as off-chip memory or bridges. ■ Allows the controller to start a new DMA write transaction while a DMA write is in progress. ■ For descriptor write transactions, "Bufferable/Posted" must be zero to avoid a DMA write race condition with a software interrupt or subsequent DMA read. When the controller writes back a descriptor or event, it waits for the DMA response to indicate that it can safely re-fetch that descriptor or set the controller's interrupt. If the descriptor DMA write is posted, the data integrity is not ensured for future transactions.
Cacheable, Modifiable, Snoop (negation of `No Snoop`)	The characteristics of the transaction at the destination may not match the original.	■ For DMA writes, multiple write transactions may be merged. ■ For DMA reads, a location can be pre-fetched or fetched just once for multiple read transactions. ■ For AXI, this bit should be used in conjunction with the Read Allocate / Write Allocate to indicate upon a cache miss whether the transaction should be cached, or Other Allocate / Allocate bits to indicate how to handle cache misses.

Connections from GSBUSCFG0 to Native Master Cache Type

In device mode, the `GSBUSCFG0` cache type fields are connected to the two (write and read) system bus master cache type ports through a pair of 4-bit wide 2x1 multiplexers; the MUX select signal is the descriptor access indication (as opposed to data).

In host mode, the `GSBUSCFG0` cache type fields are connected as in device mode with these exceptions:

- For data accesses, the controller sets Cacheable to the inverse of the xHCI TRB's `No Snoop` flag; the Cacheable bit in `GSBUSCFG0` is disregarded.
- For descriptor accesses, the controller sets Cacheable to the inverse of the internal DMA command's `No Snoop` flag. When the LSP performs a Scratchpad DMA write, it sets the internal DMA command's `No Snoop` flag thereby ensuring Cacheable=0, as required by the *xHCI Specification*.

Depending on whether the master bus type is AHB or AXI/Native, the `GSBUSCFG0` to Cache Type connections are reflected by one of the two following diagrams (the Cacheable bit is bit[3] for AHB but bit[1] for AXI/Native).

Figure 2-1 GSBUSCFG0 Cache Type Connections for AHB

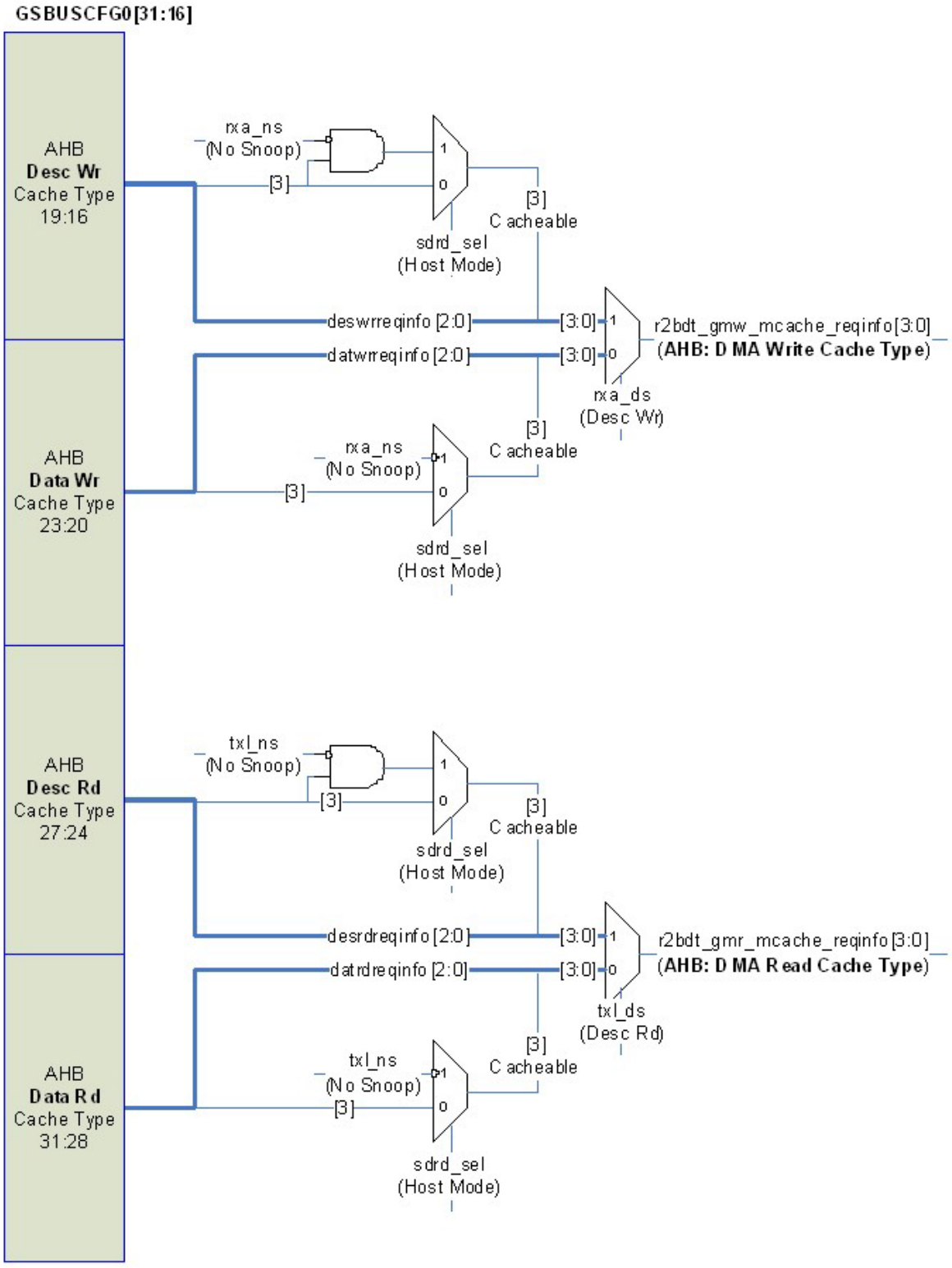
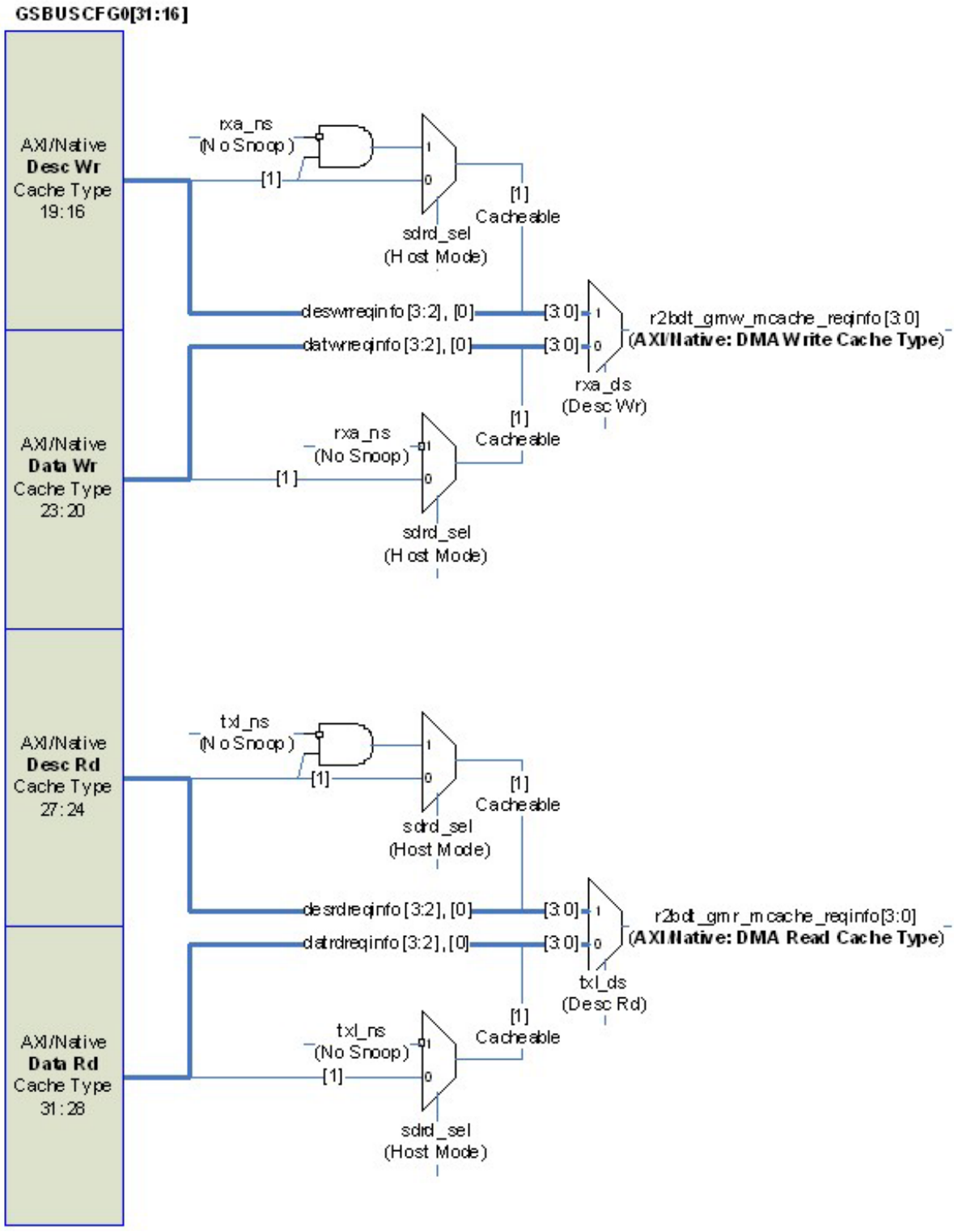


Figure 2-2 GSBUSCFG0 Cache Type Connections for AXI/Native



For more information, refer to the “Integrating the Controller” chapter in the *DesignWare Cores SuperSpeed USB 3.0 Controller User Guide*.

3

Device Data Structures

This chapter describes the USB 3.0 device data structures. The following topics are discussed:

3.1 Device Descriptor Structures

This section describes the USB 3.0 device descriptor structure.

3.1.1 Definitions

Device mode Transfer Request Blocks (TRBs) are small (4 DWORDs) and at the same time provide a rich set of features so that the software can efficiently manage the USB controller, memory buffers, and MIPS requirements.

The following is a list of Device mode TRB characteristics:

- Scattered data structures

  The scattered data buffers can be zero bytes to 16 MB in length.
- System memory aligned to a 16-byte boundary

  To prevent race condition of same TRB DWORDs being written/read by hardware/software, it is highly recommended to have a design that prevents descriptor reads and writes from being split into separate transfers on AXI/AHB.
- Linear memory to enhance descriptor caching performance

  If the data buffers are small (as in an Ethernet application), the controller can efficiently collect the scattered data to build USB packets without wasting bus efficiency that collecting scattered descriptors requires.
- TRB linking

  This allows software to set up a circular ring of TRBs with the last TRB linking to the first one.
- 64-bit addressing

  The address must not cross the 32-bit (4 GB) boundary within a single list.
- Byte-aligned buffers for each TRB of a transfer

  This feature prevents the need for buffer copying in cases where the USB device driver receives unaligned data buffers from other applications (for example, Ethernet). Whenever the application controls buffer allocation, it must allocate SoC bus width and burst-aligned buffers to facilitate efficient bus and memory usage. For transmitting, the controller supports byte-aligned buffers on all TRBs.

  Example: If you plan to use 16 bursts in a 64-bit system, you must try to allocate buffers that are 16 * 8 = 128 bytes aligned. This is the normal buffer structure because Linux-like OSs allocate 4 KB buffers. SDR/DDR memory controllers also provide better performance when requests are burst aligned.
- Software queuing of multiple USB transfers (`LST` bit for IN/OUT transfers and `CSP` bit for OUT transfers control this function)
- Interrupt moderation capability, allowing software to selectively enable events on TRB completion, as controlled by the `IOC` bit

  The IOC event is also used by software to reallocate the released buffers, allowing software to re-use just a few buffers in a circular fashion. This helps when your memory is limited, but there are enough MIPS to process interrupts. You can allocate larger buffers to reduce the number of interrupts.

  In USB 3.0, larger transfers are recommended because the raw transfer rate is almost 10 times faster, unlike USB 2.0, where drivers set up only 64 KB or 128 KB transfers. For example, a 64 KB transfer that takes 1.3 ms in USB 2.0 requires only 164 µS in USB 3.0. If you set up buffers of 64 KB and enable the transfer completion event, then you receive an interrupt every 164 µS.
- Streaming (Stream ID field used for this purpose)

3.1.2 Structures

Figure 3-1 shows the control and status fields of a Transfer Request Block.

Figure 3-1 TRB Control and Status Fields

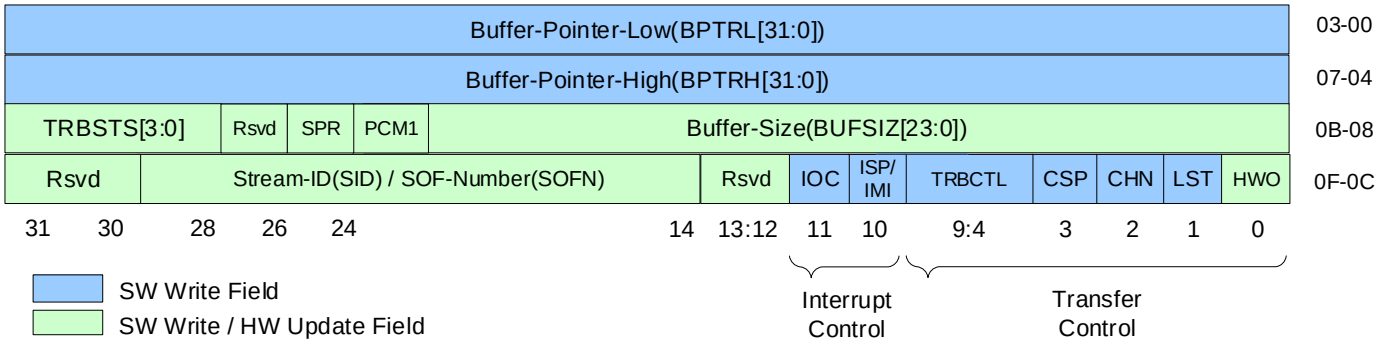


Table 3-1 defines the TRB fields shown in Figure 3-1. For more information on the effect of TRB control bits on a transfer, see "Transfer and Buffer Rules" on page 342.

Note

In Table 3-1, hardware access types are:

- R_W = Read and Write
- R = Read Only

Table 3-1 Device Descriptor Structure Field Definitions (Continued)

Field	Description	Hardware Access
DW 03-00		
31:0	`Buffer Pointer Low (BPTRL)` Data buffer pointer to low 32 bit address (`BPTR[31:0]`). Hardware may also update this field (implementation specific).	R_W
DW 07-04		
31:0	`Buffer Pointer High (BPTRH)` Data buffer pointer to high 32-bit address (`BPTR[63:32]`).	R_W
DW 0B-08		
23:0	`Buffer Size (BUFSIZ)` If `CHN`=0 and `HWO`=0 for the TRB, this field represents the total remaining Buffer Descriptor buffer size in bytes. Valid Range: 0 bytes to (16 MB - 1 byte). The hardware decrements this field to represent the remaining buffer size after data is transferred. For a Link TRB, the buffer size should be "0". For details about how to set the buffer size in different scenarios, see "Buffer Size Rules and Zero-Length Packets" on page 345.	R_W

Field	Description	Hardware Access
25:24	`Packet Count M1 (PCM1)` For High-Speed, High Bandwidth isochronous IN endpoints, this field in an Isoc-First TRB represents the total number of packets in the Buffer Descriptor minus 1.	R_W
27	`Reserved`. Hardware may update these bits for internal usage.	R_W
26	`SPR/Reserved` ■ If `GUCTL1[DEV_TRB_OUT_SPR_IND]` is 1'b1, this field indicates the short packet received for the last processed TRB for the transfer. If a short packet has been received, this TRB field is set during an OUT transfer TRB write back if this is the last TRB used for that transfer descriptor. ■ If `GUCTL1[DEV_TRB_OUT_SPR_IND]` is 1'b0, this field is reserved. Hardware may update these bits for internal usage.	R_W
31:28	`TRB Status (TRBSTS)` Hardware updates this field with transfer status information before releasing the TRB. ■ 4'h0: OK ■ 4'h1: MissedIsoc: Isochronous interval missed or incomplete ■ 4'h2: SetupPending – During the current control transfer data/status phase, another SETUP was received. ■ 4'h4: TransferInProgress – During the current transfer, an end transfer command was received. ■ 4'hF: ZLP_PENDING – This is used only for TRBs with type NORMAL-ZLP. This status indicates that the zero-length-packet for an IN transfer is still pending to be completed. This status is for controller internal operation only, and is not written out during normal operation. When hibernation entry happens with a zero-length-packet pending, this status gets written out and is restored back during hibernation restore.	R_W
DW 0F-0C		
0	`Hardware Owner of Descriptor (HWO)` Indicates that hardware owns the TRB. Software sets this bit to 1 when it creates the TRB, and cannot modify it until hardware resets this bit to 0. However, there are exceptions for short packets on OUT endpoints and Link TRBs. For more information, see "TRB Control Bit Rules" on page 344". Because the hardware autonomously checks this bit to determine if the entire TRB is valid, software must set this field to '1' after preparing the other three DWORDs of the TRB with valid information.	R_W
1	`Last TRB (LST)` Indicates this is the last TRB in a list. After completing the transfer for the associated buffer, the controller stops the transfer for the endpoint / bulk-stream and issues an XferComplete event. The stream is automatically removed by the hardware.	R
2	`Chain Buffers (CHN)` Applicable to IN and OUT endpoints. Set to 1 by software to associate this TRB with the next TRB. A Buffer Descriptor is defined as one or more TRBs. The `CHN` bit is used to identify the TRBs that comprise a Buffer Descriptor. The `CHN` bit is always 0 in the last TRB of a Buffer Descriptor and when the `LST` field is set to 1.	R

Field	Description	Hardware Access
3	`Continue on Short Packet (CSP)` Applicable to OUT endpoints only when a short packet is received. ■ If this bit is 1, the controller continues to the next Buffer Descriptor. This setting is required for isochronous endpoints. ■ If this bit is 0, the controller generates an XferComplete event and remove the stream.	R
9:4	`TRB Control (TRBCTL)` Indicates the type of TRB: ■ 1: Normal (Control-Data-2+ / Bulk / Interrupt) – Set `TRBCTL` to 1 for all TRBs used in data stage except the first TRB ■ 2: Control-Setup ■ 3: Control-Status-2 – Set `TRBCTL` to 3 for a SETUP request without data stage ■ 4: Control-Status-3 – Set `TRBCTL` to 4 for a SETUP request with data stage ■ 5: Control-Data – Set `TRBCTL` to 5 for the first TRB of a data stage ■ 6: Isochronous-First – Set `TRBCTL` to 6 for the first TRB of a Service Interval ■ 7: Isochronous ■ 8: Link TRB ■ 9: Normal-ZLP (Bulk-IN) – Set `TRBCTL` to 9 for a BULK IN TRB where a zero-length packet termination is required when completing the TRB chain[1] (when the transfer size of the buffer descriptor is MPS-aligned) ■ Others: Reserved	R
10	`Interrupt on Short Packet / Interrupt on Missed ISOC (ISP/IMI)` Applicable to OUT endpoints when a short packet is received, and `CSP=1` and `LST=0`. If this bit is 1, the controller generates an XferInProgress event. For Isochronous endpoints: If this bit is 1, the controller generates an XferInProgress event when the interval represented by the Buffer Descriptor completes with a “Missed Isoc” status.	R
11	`Interrupt on Complete (IOC)` When `IOC` is set in a TRB, and once the transfer for this buffer is completed, the controller issues XferInProgress event with `IOC` bit set in the event’s status. This indicates the buffer is available for software to reuse or release.	R
13:12	`Reserved`. Hardware may update these bits for internal usage.	R_W
29:14	`Stream ID / SOF Number` ■ For stream-based bulk endpoints: The Stream ID of the transfer that this TRB is associated with. Stream ID must be the same in all TRBs (R_W). For isochronous endpoints: The (micro)frame number in which the last packet of this TRB’s buffer was transmitted or received (debug purposes only) (RO).	R_W
31:30	`Reserved`. Hardware may update these bits for internal usage.	R_W

1. The ZLP type must be used only on the last TRB of a chain to send a ZLP packet when the transfer size is an exact multiple of MaxPacketSize (MPS). Do not use this type when the transfer size is not MPS-aligned;

Use this type only when the host expects ZLP to complete an IN transfer.

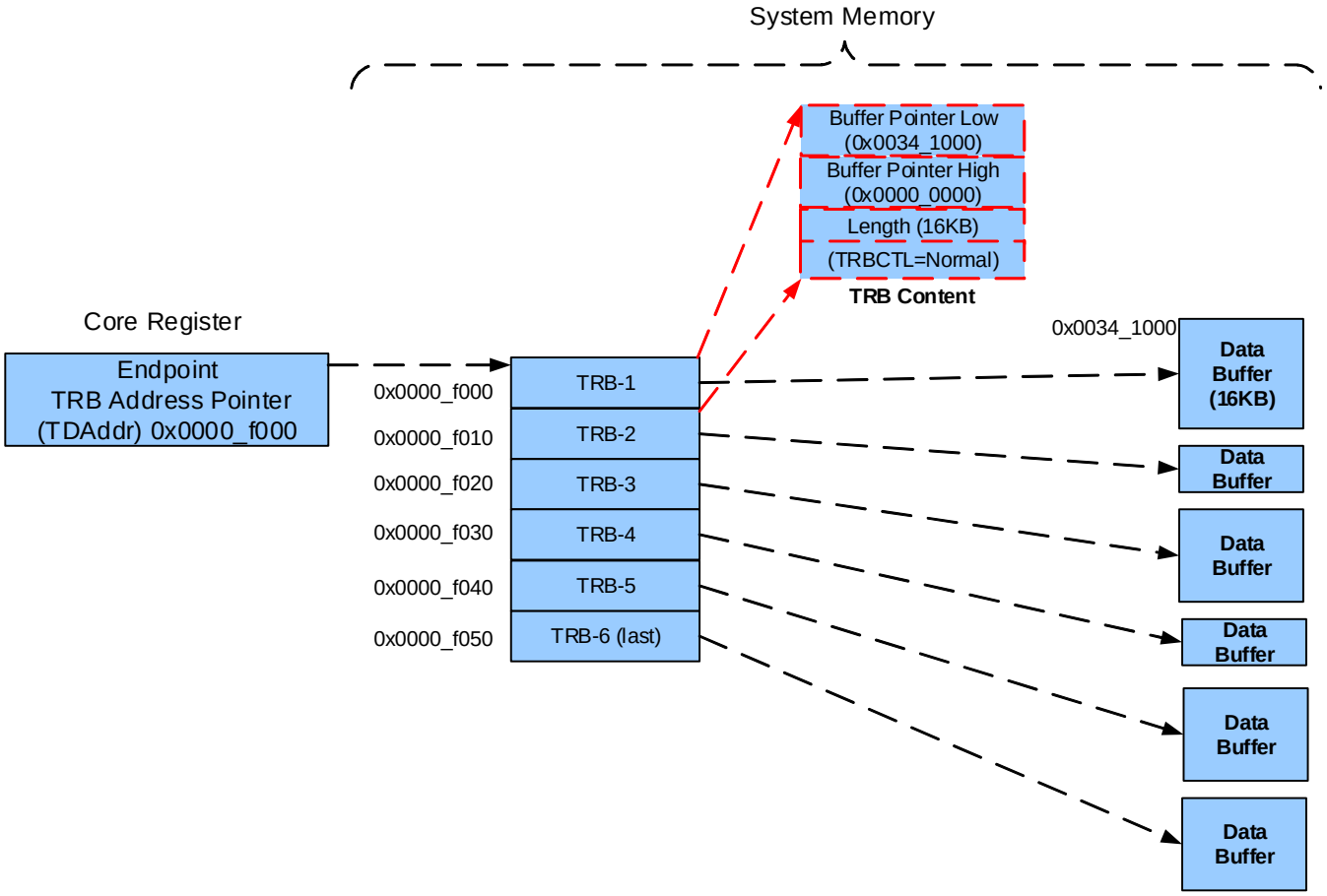
Do not use this type when creating a single TRB for a zero-byte transfer size. In such a scenario, use a Normal TRB with a zero byte buffer size.

3.1.2.1 Normal (Control-Data/Bulk/Interrupt), Isochronous, and Status Transfer Request Block Structure

The Normal TRB is used for Bulk/Control-Data/Interrupt endpoint transfers. The data buffers can be scattered anywhere and each may have different sizes. The TRB Buffer Pointer and the Buffer Size fields point to buffer address and size respectively. The Stream ID for bulk endpoints is programmed by the application.

For isochronous endpoints, the first TRB in a service interval must have the Isoc-First type, the last TRB in a service interval must have `CHN=0`, and any other TRBs have `CHN=1`. The starting (micro)frame time is communicated through the Start Transfer command, and the controller tracks the (micro)frame times of the subsequent Buffer Descriptors.

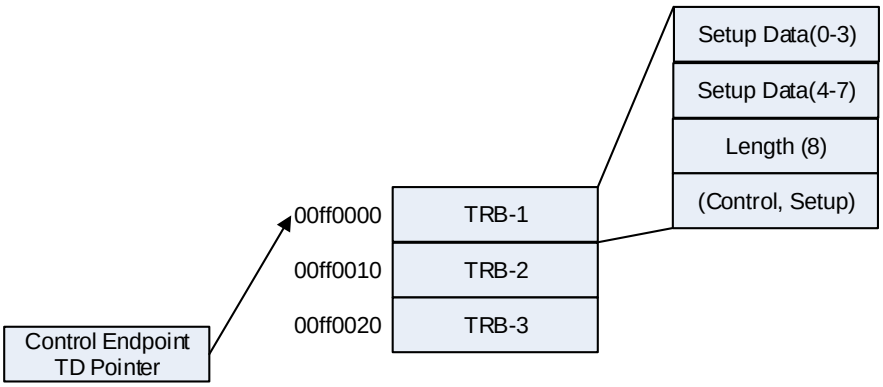
Figure 3-2 Normal (Control-Data/Bulk/Interrupt) Descriptor Structure



3.1.2.2 Setup and Status TRB Structure

To receive a SETUP packet, the driver queues up a single Setup TRB, whose buffer pointer value may be set to any address, including the address of the TRB. The buffer size must be set to 8. The controller writes the 8 bytes of the received SETUP to the address requested. If the address of the TRB is used, there is no need for a separate data buffer to receive a SETUP packet. After completing Setup stage, the driver schedules Data stage and Status stage transfers. For more information, see "Control Transfer Programming Model" on page 364.

Figure 3-3 SETUP Descriptor Structure with Buffer Pointing to Setup TRB



After interpreting the SETUP bytes, the software determines if the next stage of the control transfer is a data stage or status stage.

If the SETUP bytes require a 3-stage control transfer, the TRB type used in the Data stage must be Control-Data for the first TRB of the Buffer Descriptor. When the host moves on to the Status stage, the TRB Type must be Control-Status-3.

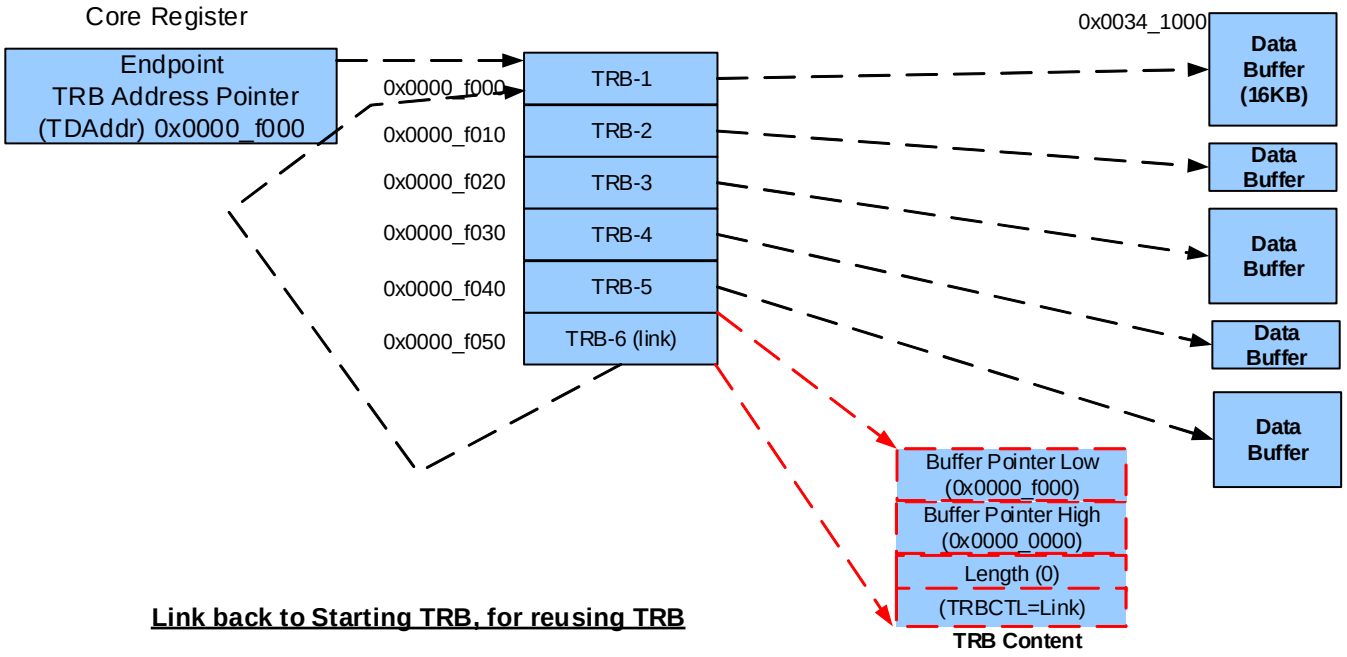
If the SETUP bytes require a 2-stage control transfer, the TRB Type must be Control-Status-2.

The Status TRB is a zero-length TRB, with an unspecified Buffer Pointer value and a Buffer Size of zero. There is no data buffer associated with a Status TRB.

3.1.2.3 Link TRB Structure

The Link TRB is used to link back to the starting TRB for reusing TRBs in a circular fashion.

Figure 3-4 Link TRB Structure



If software prepares a circular TRB list that is shorter than the number of cached TRBs (DWC_USB3_CACHE_TRBS_PER_TRANSFER), the controller automatically detects the loop and does not re-fetch a TRB that has already been fetched. For example, if DWC_USB3_CACHE_TRBS_PER_TRANSFER is equal to four, and software has setup three TRBs in the following way:

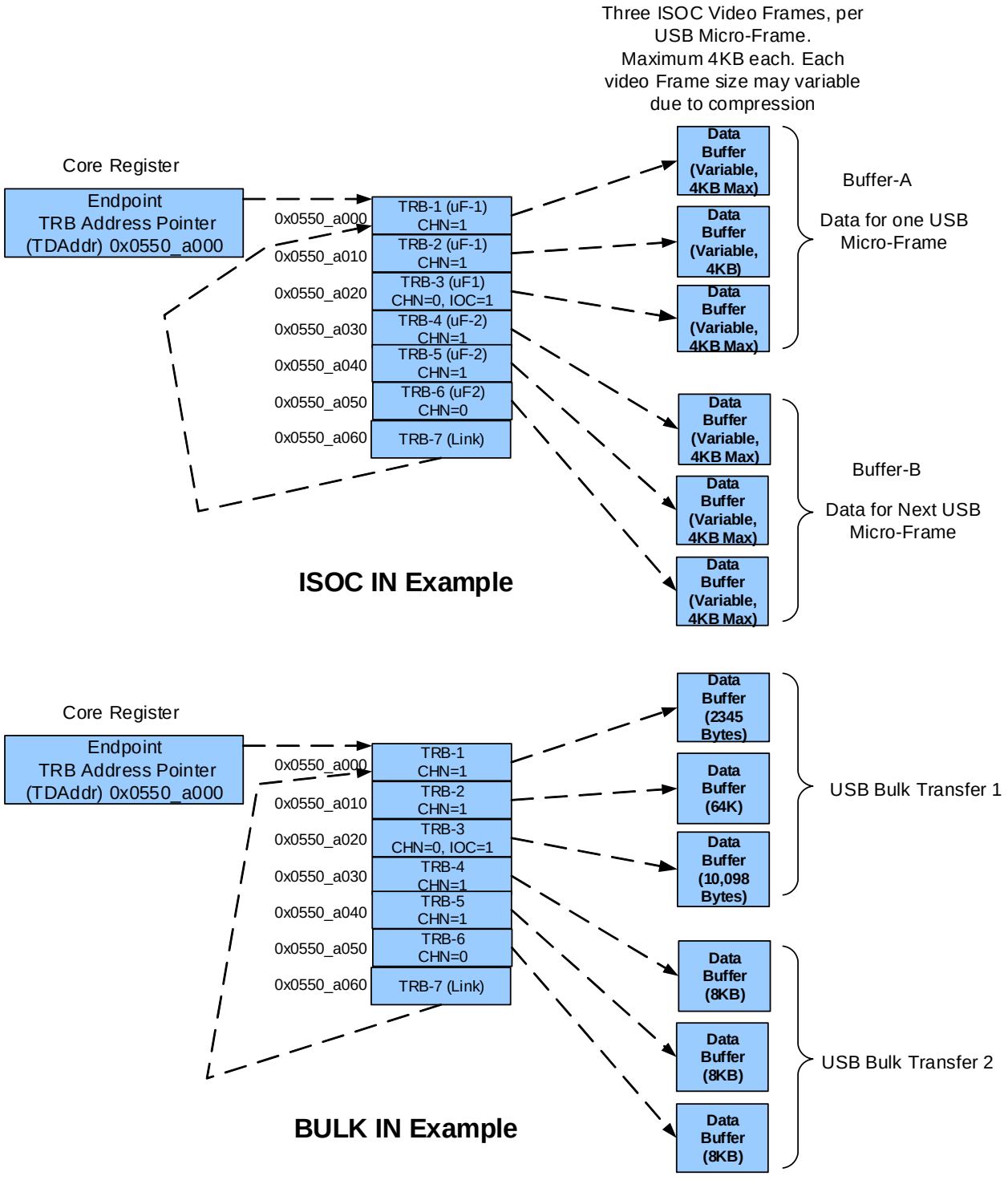
- TRB-1: Normal, HWO=1
- TRB-2: Normal, HWO=1
- TRB-3: Link to TRB 1

The controller fetches TRBs 1, 2, and 3. Then the controller follows the link to TRB-1, see that its address is the same as the TRB-1 that has already been fetched into the cache, and temporarily stops fetching TRBs. When TRB-1 completes due to traffic on the USB, the controller writes TRB-1 back to memory with the HWO field set to '0', generates a XferInProgress event if necessary, and automatically attempts to fetch TRB-1 again. In most cases, the controller sees the HWO field set to '0' and stops fetching until software updates the TRB, sets the HWO field back to '1', and issues an Update Transfer command.

3.1.2.4 Chaining Buffers (CHN) and Interrupt On Completion (IOC) Usage

Figure 3-5 shows a chaining buffer example for an isochronous IN and a bulk IN transaction.

Figure 3-5 Chaining Buffers for Isochronous and Bulk IN



In the isochronous IN application shown in Figure 3-5, there are three video frames to be sent to the host in each microframe. Each video buffer size is maximum of 4 KB and the size may vary for each microframe depending on the compression. The CHN bit in the TRB-3 is 0, which indicates this is the last buffer of a transfer for the given microframe. The device schedules TRB-1, 2, and 3 in the first bInterval. Similarly, TRB-6 indicates (CHN=0) microframe boundary, and TRB-4, 5, and 6 is sent during the next bInterval.

For the last packet of last transfer in a microframe, the device internally sets last packet flag when responding at SuperSpeed for isochronous IN transfers.

The application can also use the IOC bit to receive an interrupt when the Buffer-A transfer is completed, so it can use buffer-A to send data on the bInterval following the next bInterval. As long as the driver can service the interrupt and set up data before another bInterval, the USB transfers can continue without interruption. Depending upon the system interrupt latency, you can adjust the buffer size.

If interrupt latency is high occasionally, the software may not have time to set up the linked TRB before the next bInterval. In this case, the controller, on seeing HWO bit set to 0 in the TRB, stops processing further TRBs of this endpoint. When hardware receives the Update Transfer command from software, it re-fetches the TRB. In the case of bulk application, the controller issues the NRDY signal. In isochronous transfers, zero-length packets are sent to the host until software enables the transfer again.

In the bulk IN example in Figure 3-5 on page 303, the CHN bit indicates USB Transfer boundary. During a TRB transfer, CHN indicates whether to send the bytes from next TRB buffer as part of the current transfer. For example, even though TRB-1 has 2,345 bytes, the last 297 (2345 – 2 * 1024 = 297) bytes is combined with the data in TRB-2 and sent as a 1,024 byte packet on the USB since CHN=1. On the other hand, when there is a short packet left in TRB-3, the short packet is sent separately because CHN=0.

Figure 3-6 on page 305 and Figure 3-7 on page 306 show examples of chaining buffers, with the fetch on the SoC bus and the transfer on the USB.

Figure 3-8 on page 307 shows a bulk OUT application where two transfers are set up. The device writes data into TRB-1 and TRB-2 for the first transfer. The CHN bit (CHN=0) in TRB-2 indicates that this is the last buffer of the transfer, then TRB-3 and TRB-4 are written for the second transfer. Similarly, the CHN bit (CHN=0) in TRB-4 indicates the second transfer boundary.

If an interrupt latency is high, software may not have time to set up the TRB. In this case, the controller, on seeing the HWO bit set to 0 in the TRB, stops processing further TRBs for the endpoint. Hardware re-fetches the TRB when software issues the Update Transfer command. In the case of bulk, control, or interrupt endpoints, the controller issues NRDY. In the case of isochronous endpoints, packets are dropped until software enables the transfer again.

In the bulk OUT example below, when a short packet is received for TRB-1, TRB-3 is used for the next OUT transfer and TRB-2 is closed.

Figure 3-6 Chaining Buffers, Example 1

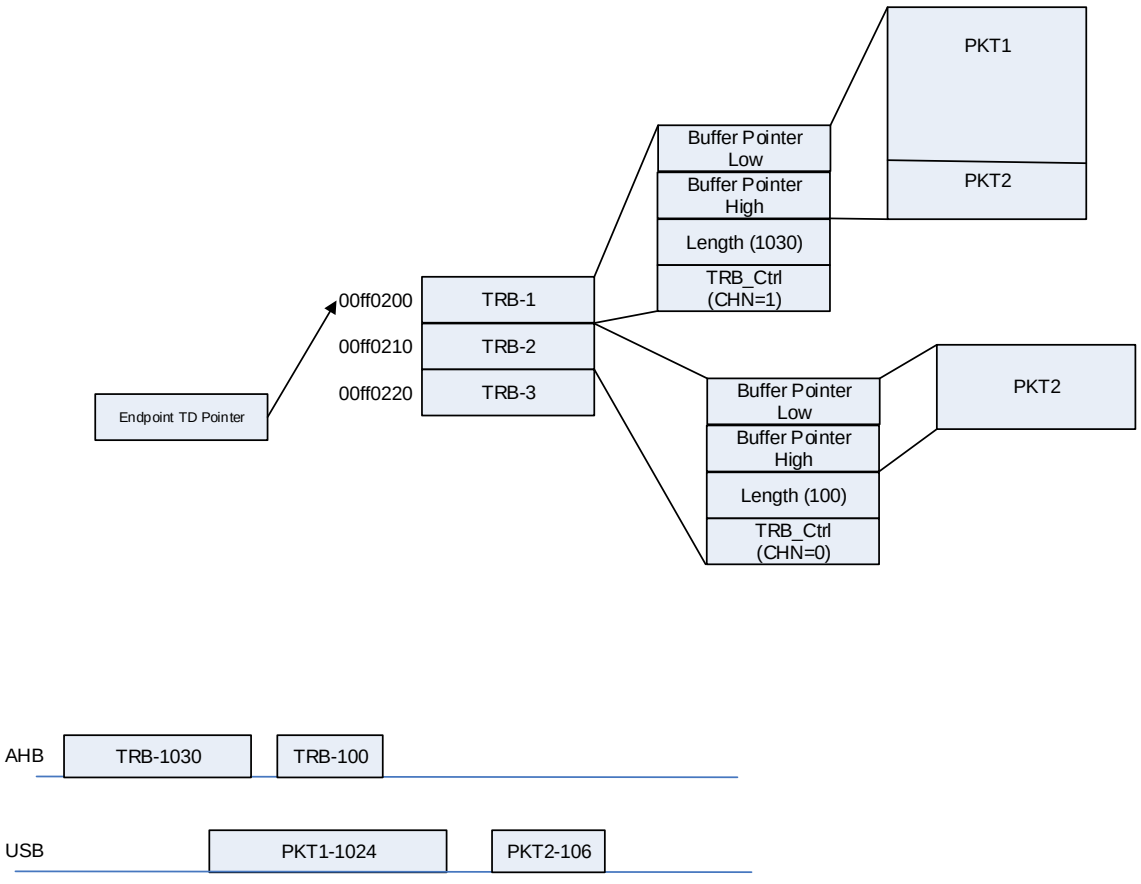


Figure 3-7 Chaining Buffers, Example 2

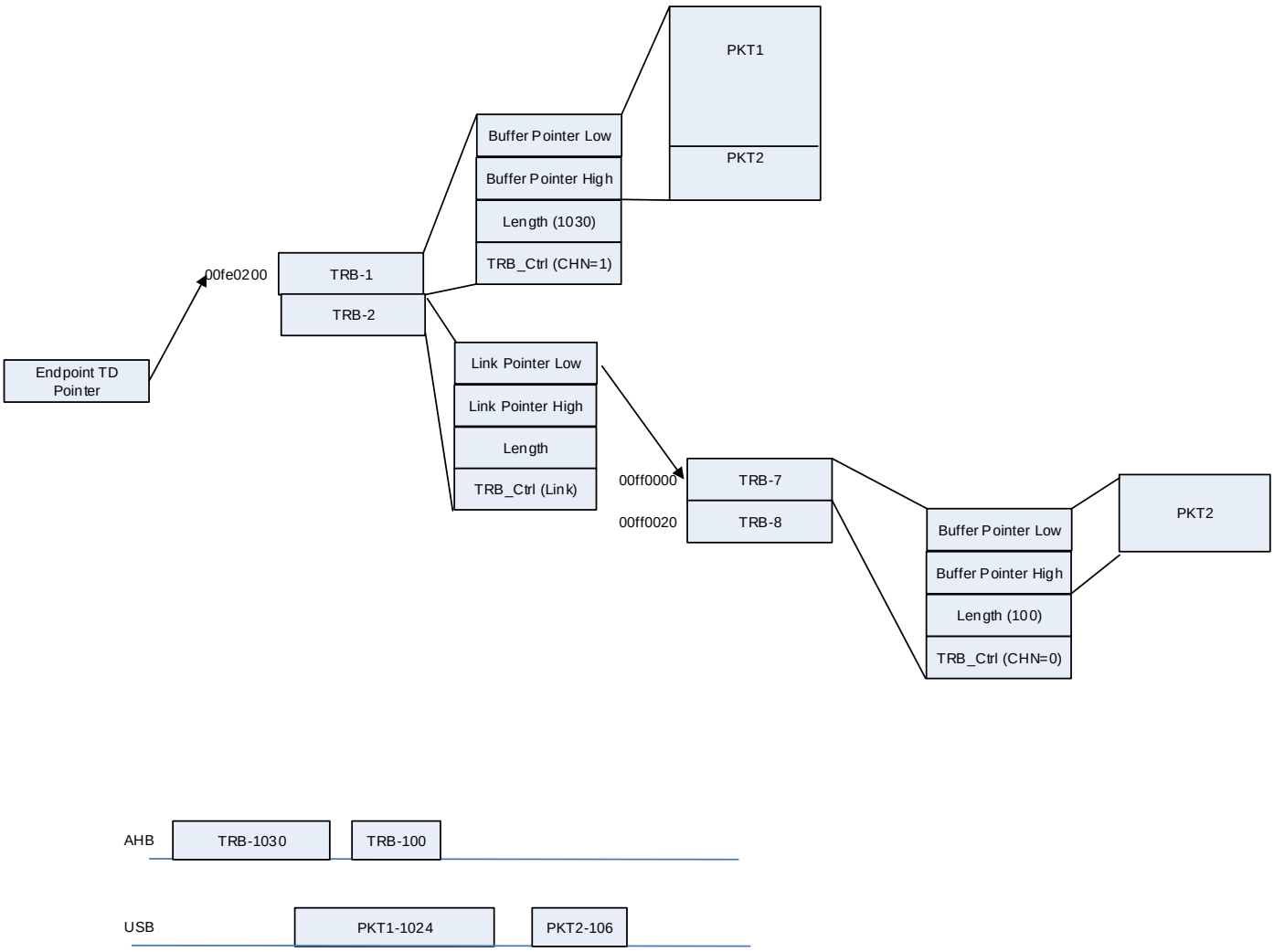
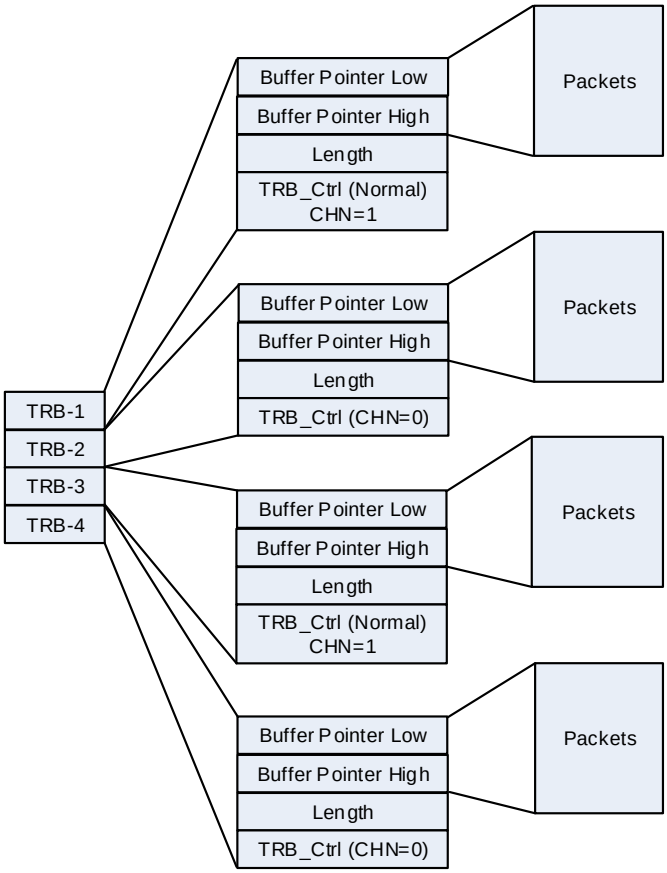


Figure 3-8 Bulk OUT with Two Transfers



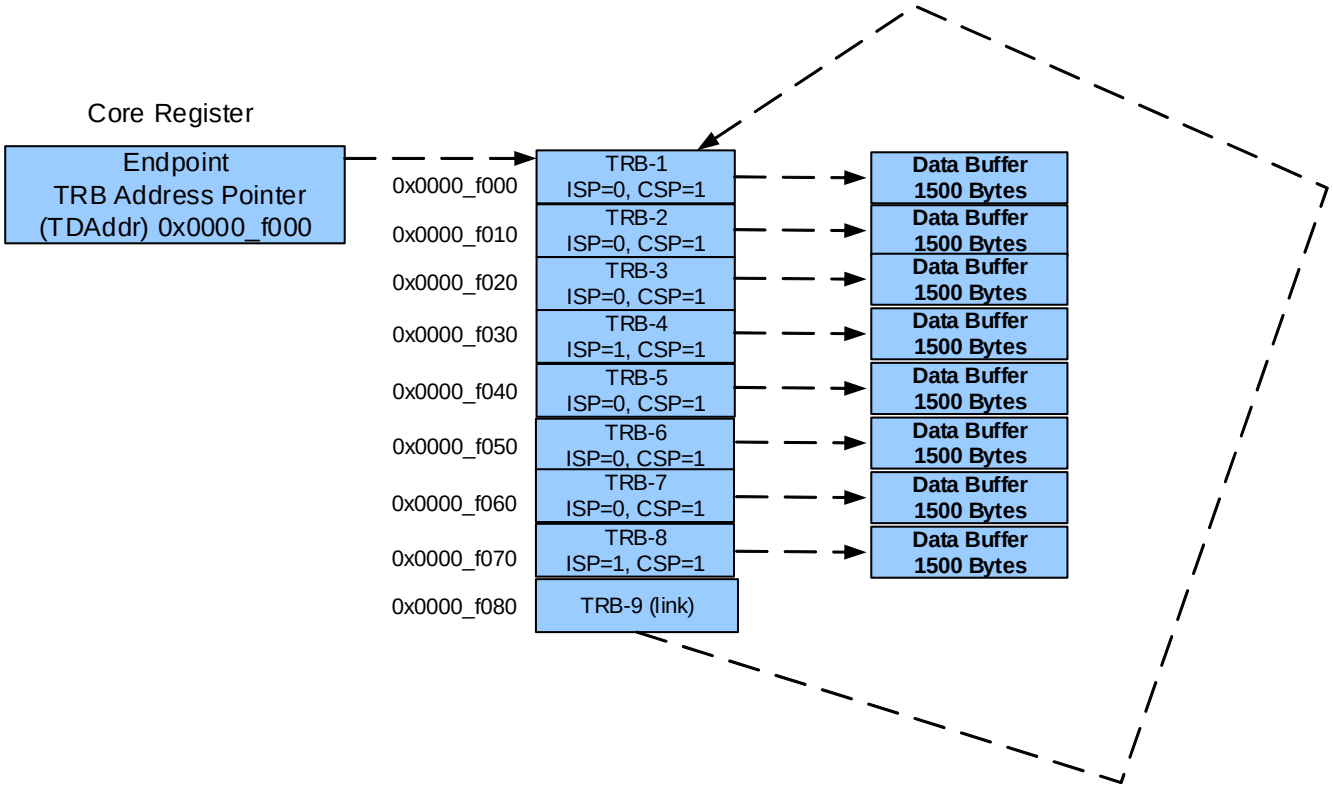
3.1.2.5 Interrupt on Short Packet (ISP) and Continue on Short Packet (CSP) Usage

`ISP` and `CSP` bits are used to schedule single or multiple OUT transfer. In most applications where only one OUT transfer per endpoint is scheduled, the software always sets `ISP=1` and `CSP=0`. If the device receives a short packet, then it indicates a USB transfer completion through XferComplete events.

In an Ethernet-over-USB application, where software knows it is going to receive short packets, it can set up multiple transfer size buffers in one step by setting `CSP=1` in all the TRBs. On a short packet, the device updates the byte count and moves to the next TRB. The software can also set ISP once in *n* number of TRBs to receive an XferInProgress event so it can process the previous short packets.

The following example shows software setting multiple 1500-byte Ethernet transfers and enabling the XferInProgress event once for four Ethernet packets to reduce the number of interrupts.

Figure 3-9 ISP and CSP Usage

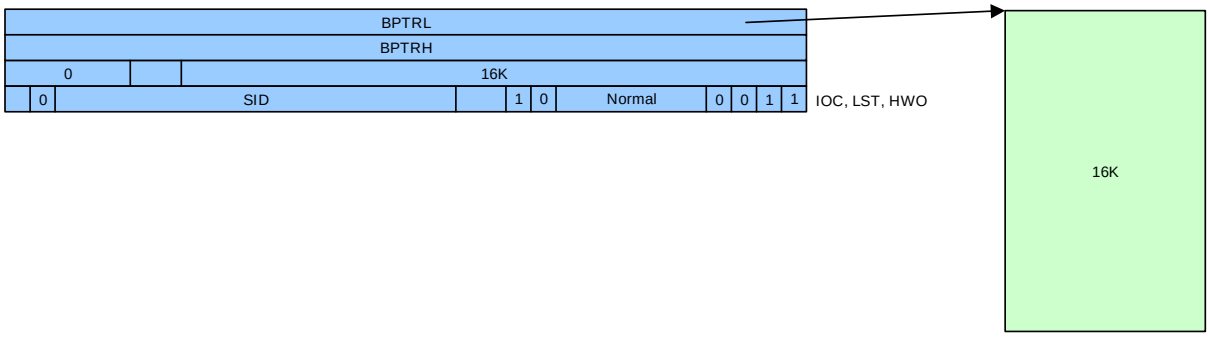


3.1.2.6 Example of Setting Up TRBs

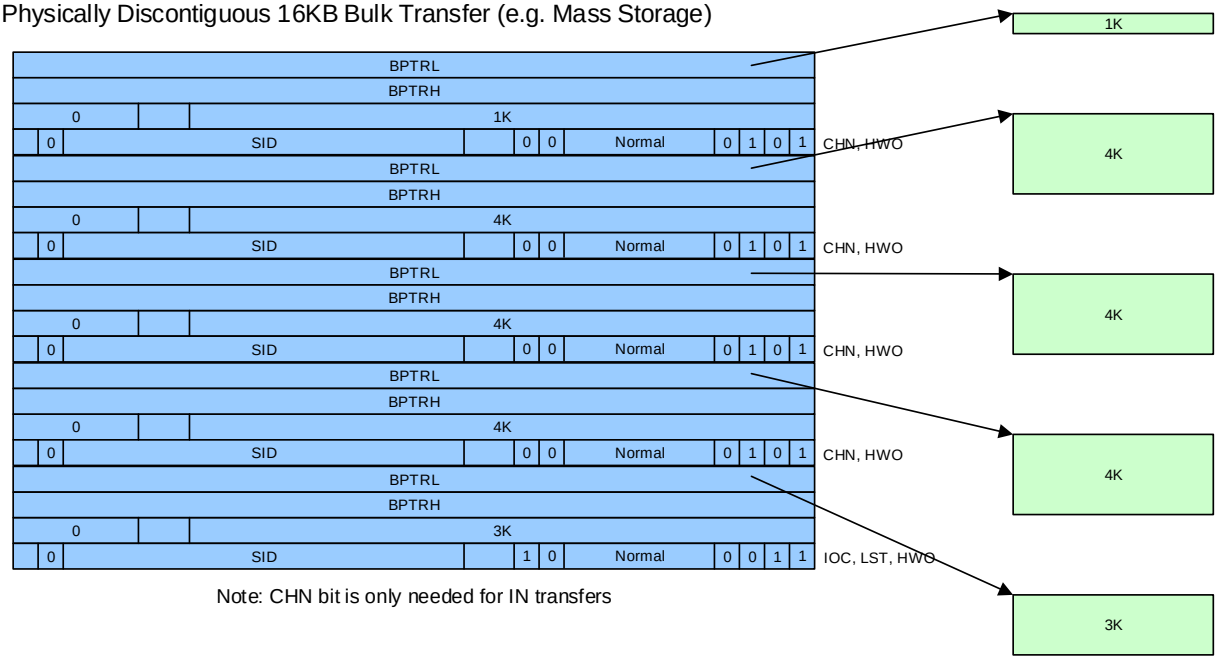
This section shows examples of how to setup TRBs.

Figure 3-10 Bulk IN TRB Examples

Physically Contiguous 16KB Bulk Transfer

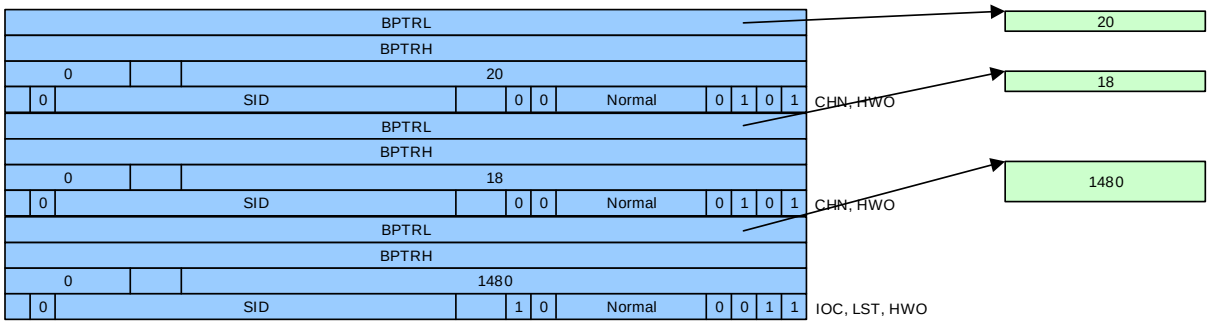


Physically Discontiguous 16KB Bulk Transfer (e.g. Mass Storage)



Note: CHN bit is only needed for IN transfers

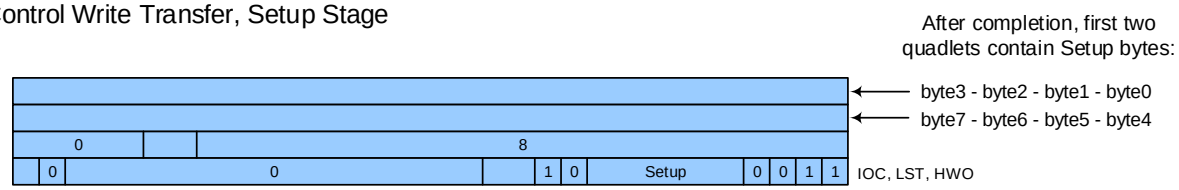
Physically Discontiguous 1518 Byte Bulk Transfer (e.g. Ethernet)



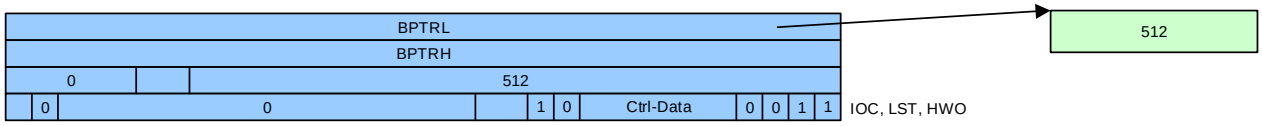
Note: CHN bit is only needed for IN transfers

Figure 3-11 Setup/Control/Status TRB Examples

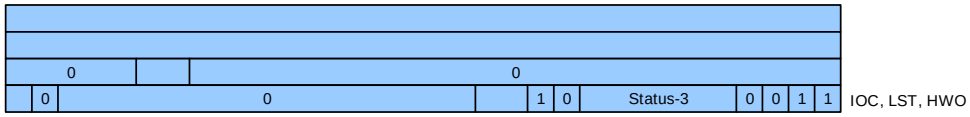
Control Write Transfer, Setup Stage



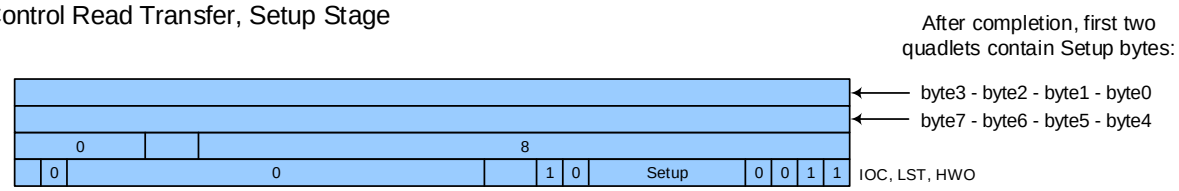
Control Write Transfer, Data Stage (Optional)



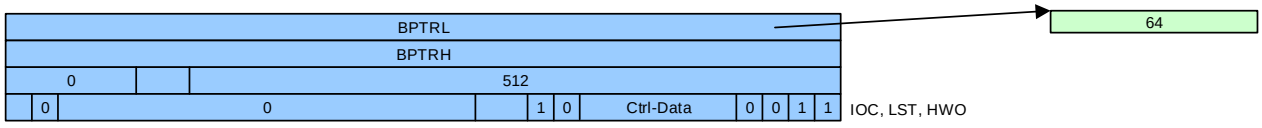
Control Write Transfer, Status Stage



Control Read Transfer, Setup Stage



Control Read Transfer, 64B Data Stage



Control Read Transfer, Status Stage

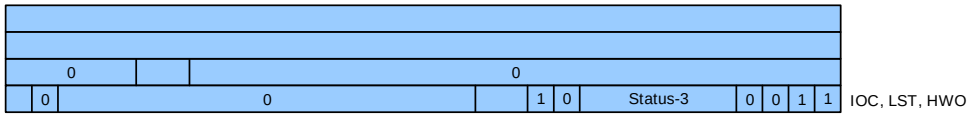


Figure 3-12 Isochronous IN TRB, Example 1

Eight Isochronous Transfers, Physically Contiguous Buffers, IOC on the 4th and 8th Transfers, Circular Linked

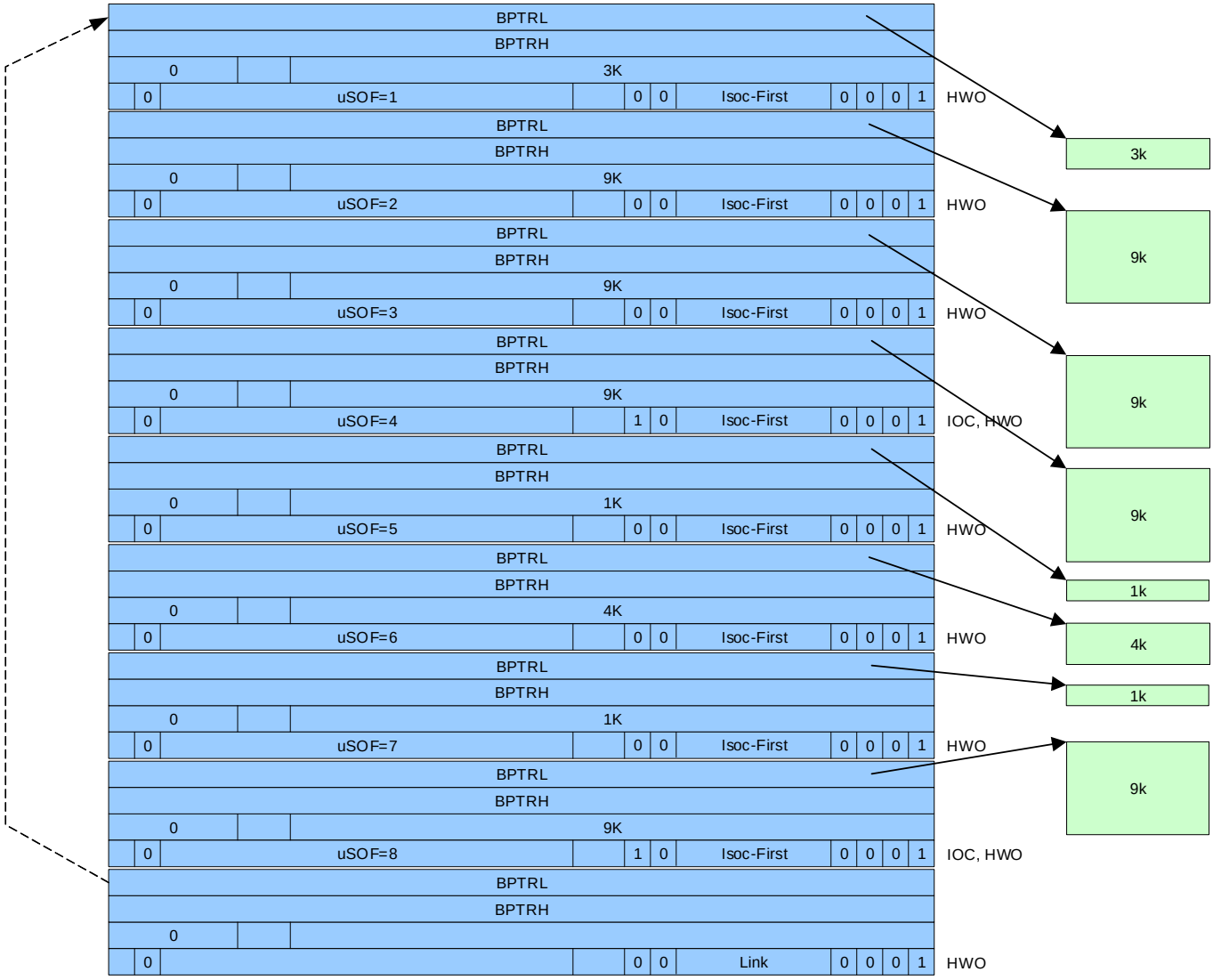
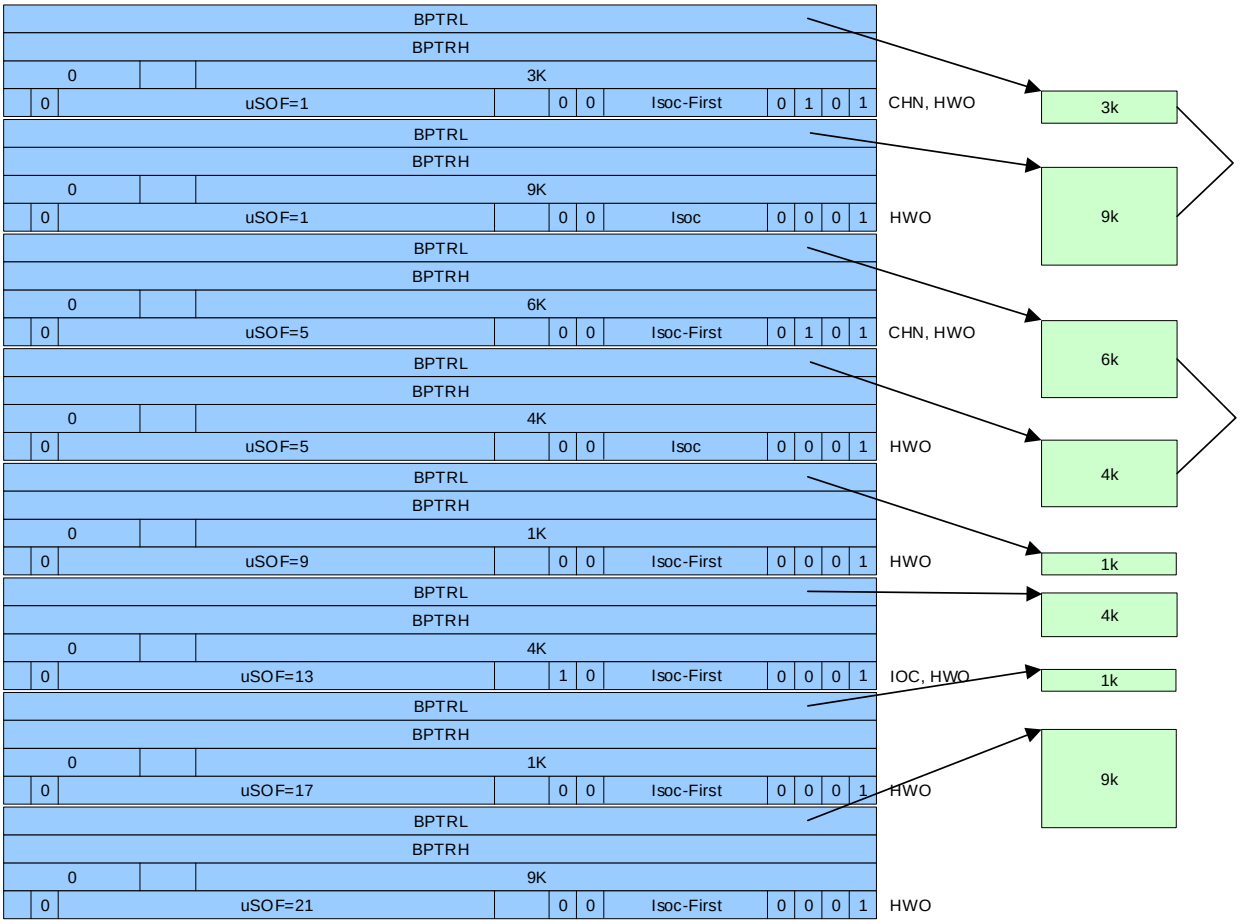


Figure 3-13 Isochronous IN TRB, Example 2

Six Isochronous Transfers, One Transfer Every Four uFrames, Some Physically Discontiguous Buffers, IOC on the 4th Transfer



3.2 Device Command Structure

This section describes the following device command structures.

- "Device Generic Command Structure"
- "Device Physical Endpoint-Specific Command Structure" on page 315

3.2.1 Device Generic Command Structure

Software can program the controller using DGCMD and DGCMDPAR registers to send link management packets and notifications. For register descriptions, see "Register Descriptions" on page 17.

DGCMD[7:0] specifies the type of generic command that the software needs the controller to perform. This section discusses the following commands:

- "Set Periodic Parameters"
- "Set Scratchpad Buffer Array Address Lo"
- "Set Scratchpad Buffer Array Address Hi"
- "Transmit Device Notification"
- "Selected FIFO Flush"
- "All FIFO Flush"
- "Set Endpoint NRDY"
- "Run SoC Bus LoopBack Test"

Any command that is not present in the Table 3-2 is considered Reserved.

Table 3-2 Device Generic Command Types (Continued)

Command	Description
02h	Set Periodic Parameters Parameter[9:0] (SystemExitLatency): Software should set this to the same value programmed by the host through the Set SEL device request, in microseconds. The Set SEL control transfer has six bytes of data and contains four values; Refer to Section 9.4.12 of the *USB 3.0 Specification*. Offset: Name (Meaning) ■ 0: U1SEL (Time in µs for U1 System Exit Latency) ■ 1: U1PEL (Time in µs for U1 Device to Host Exit Latency) ■ 2: U2SEL (Time in µs for U2 System Exit Latency) ■ 4: U2PEL (Time in µs for U2 Device to Host Exit Latency) If the device is enabled for U1 and U2, then the U2PEL should be programmed. If the device is enabled only for U1, then U1PEL should be programmed into this parameter. If the value is greater than 125 µs, then the software must program a value of zero into this register. coreConsultant configuration provides the default value, assuring the register contains a valid setting when software does not set the register value. The coreConsultant parameter is DWC_USB3_PERIODIC_PARAMS. **Note**: Currently, the controller does not use the programmed value.

Command	Description
04h	`Set Scratchpad Buffer Array Address Lo` This command sets bits [31:0] of the external address of the scratchpad buffer array used for save/restore. If either this command or command 05h is issued while the controller is stopped (`RunStop=0`), the `CmdIOC` bit must be set to '0'. The device scratchpad buffer array has the same format as the xHCI scratchpad buffer array; it contains an array of 64-bit pointers to data buffers that will be used to save the controller state.
05h	`Set Scratchpad Buffer Array Address Hi` This command sets bits [63:32] of the external address of the scratchpad array buffer used for save/restore. If either this command or command 04h is issued while the controller is stopped (`RunStop=0`), the `CmdIOC` bit must be set to '0'. The device scratchpad buffer array has the same format as the xHCI scratchpad buffer array; it contains an array of 64-bit pointers to data buffers that will be used to save the controller's state.
07h	`Transmit Device Notification` This command allows any device notification to be transmitted, using the notification type and notification parameters specified in the `DGCMDPAR` register. `DGCMDPAR[3:0]` = Notification Type `DGCMDPAR[31:4]` = Notification parameters, depends on the notification type For example, to transmit a Function Wake, software sets `DGCMDPAR[3:0]` to 1, and `DGCMDPAR[10:4]` to the Interface Number. This field relates to the “Notification Type Specific” field in a Device Notification Transaction Packet as described in Section 8.5.6 of the *USB 3.0 Specification*. The following bits of the `DGCMDPAR` register have been put into the corresponding DWORD described in Section 8.5.6 of the *USB 3.0 Specification*: `DGCMDPAR[3:0]` into DWORD 1[7:4] (Notification Type) `DGCMDPAR[27:4]` into DWORD 1[31:8] (Notification Type Specific) `DGCMDPAR[31:28]` into DWORD 2[3:0] (Notification Type Specific) There is one exception for the Bus Interval Adjustment Device Notification: `DGCMDPAR[19:4]` represents the Bus Interval Adjustment field; however, in the *USB 3.0 Specification*, the Bus Interval Adjustment field is actually at 31:16 of DWORD 1. In the case of Host Role Request, `DGCMDPAR[3:0]` = 4, and `DGCMDPAR[5:4]` = RSP Phase.
09h	`Selected FIFO Flush` ■ Parameter[4:0] = FIFO Number ■ Paramer[5] = ‘1’ for TX FIFO or ‘0’ for RX FIFO
0Ah	`All FIFO Flush` No Parameter
0Ch	`Set Endpoint NRDY` Issuing this command will make the controller think that the given endpoint is in an NRDY state. If there are buffers available in that endpoint, the controller will immediately transmit an ERDY. Parameter[4:0]: Physical Endpoint Number

Command	Description
10h	`Run SoC Bus LoopBack Test` When enabled, executes an SoC Bus Loopback test, which allows the data flow from transmit to receive to be tested without any connection to a PHY. 1. Configure Physical EP0 as a non-stream capable Bulk OUT endpoint (`USB Endpoint Number` field set to 5'h0) with the desired MaxPacketSize. 2. Configure Physical EP1 as a non-stream capable Bulk IN endpoint (`USB Endpoint Number` field set to 5'h1) assigned to TxFIFO 0 with the desired MaxPacketSize. 3. Issue this command with Parameter[0] set to '1', enabling Loopback mode. 4. Issue Start Transfer to EP0 (Command 6: Start Transfer (DEPSTRTXFER)). 5. Issue Start Transfer to EP1. 6. The controller reads data from the IN buffers and writes it back to the OUT buffers. The Tx-FIFO 0 default value of ((512/'DWC_USB3_MBYTES) + 2) should be changed to ((1024/'DWC_USB3_MBYTES) + 2) for loopback mode. Parameter[0]: When 1, enables Loopback mode. When 0, disables Loopback mode.

3.2.2 Device Physical Endpoint-Specific Command Structure

Software can issue physical endpoint-specific commands using `DEPCMDn`, `DEPCMDPAR0n`, `DEPCMDPAR1n`, and `DEPCMDPAR2n` registers. For register descriptions, see "Register Descriptions" on page 17.

`DEPCMDn[3:0]` specifies the type of endpoint-specific command that the software needs the controller to perform. This section discusses the following commands in detail:

- "Command 1: Set Endpoint Configuration: DEPCFG"
- "Command 2: Set Endpoint Transfer Resource Configuration (DEPXFERCFG)" on page 319
- "Command 3: Get Endpoint State (DEPGETSTATE)" on page 319
- "Commands 4 and 5: Set Stall and Clear Stall (DEPSSTALL, DEPCSTALL)" on page 320
- "Command 6: Start Transfer (DEPSTRTXFER)" on page 320
- "Command 7: Update Transfer (DEPUPDXFER)" on page 322
- "Command 8: End Transfer (DEPENDXFER)" on page 322
- "Command 9: Start New Configuration (DEPSTARTCFG)" on page 324

3.2.2.1 Command 1: Set Endpoint Configuration: DEPCFG

This command sets the physical endpoint configuration information.

> **Note** When operating in USB 2.0 speeds (HS/FS), if `GUSB2PHYCFG[6]` or `GUSB2PHYCFG[8]` is set to ‘1’, it must be set to '0' prior to issuing this command and may be set to ‘1’ after the command completes.

Table 3-3 Command 1: Set Endpoint Configuration Parameters: DEPCFG (Continued)

Field	Description
Parameter 2	
31:0	Set to endpoint state when the Config Action field of Parameter 0 is 1. Otherwise, this is Reserved.
Parameter 1	
31	FIFO-based Set to ‘1’ if this isochronous endpoint represents a FIFO-based data stream where TRBs have fixed values and are never written back by the controller.
30	Reserved
29:25	USB Endpoint Number ■ bit[29:26]: Endpoint number ■ bit[25]: Endpoint direction: ❑ 0: OUT ❑ 1: IN Physical endpoint 0 (EP0) must be allocated for control endpoint 0 OUT. Physical endpoint 1 (EP1) must be allocated for control endpoint 0 IN.
24	Stream Capable (StrmCap) Indicates this endpoint is stream-capable (MaxStreams != 0).
23:16	bInterval_m1 Set to the bInterval value minus 1. The valid values for this field are 0 through 13. The bInterval value is reported in the endpoint descriptor. When the controller is operating in Full-Speed mode, this field must be set to 0.
15	Set to '1' if this bulk endpoint utilizes the External Buffer Control (EBC) mode. This limits the number of outstanding DMA transfers to one read and one write. This is a requirement for the EBC to operate properly. Using EBC without setting this bit to '1' results in undefined behavior.
14	Set this bit only if the field 15 is also set. Otherwise, do not set this field. ■ When set to '1', the controller does not write back the TRB descriptor (does not update `HWO` bit in the TRB). ■ When set to '0', the controller writes back the TRB descriptor (updates `HWO` bit in the TRB).

Field	Description
13:8	Bits 13:8 are used for Device Endpoint Specific Event Enable (DEPEVTEN). If an enable bit is set to 0, the corresponding event is not generated. ■ Bit 13: Stream Event Enable (StreamEvtEn) ■ Bit 12: Reserved ■ Bit 11: Reserved ■ Bit 10: XferNotReady Enable (XferNRdyEn) ■ Bit 9: XferInProgress Enable (XferInProgEn) ■ Bit 8: XferComplete Enable (XferCmplEn)
7:5	Reserved
4:0	Interrupt number (IntrNum) Applies to IN and OUT endpoints. Indicates interrupt/Event Buffer number on which endpoint related interrupts for this endpoint are generated. If multiple Interrupter configuration is not selected, this must be 0.
Parameter 0	
31:30	Config Action ■ 0 – Initialize endpoint state: This action is used when an endpoint is configured the first time. It will cause the data sequence number and flow control state to be reset. DEPCMDPAR2 will be ignored. The encoding of this action is backward compatible with software that previously set "Ignore Sequence Number" to 0. ■ 1 – Restore endpoint state: This action is used when reconfiguring an endpoint with saved state after hibernation. It will cause the data sequence number and flow control state to be restored from DEPCMDPAR2. ■ 2 – Modify endpoint state: This action is used when modifying an existing endpoint configuration, such as changing the DEPEVTEN event enable bits, interrupt number, or MaxPacketSize. The data sequence number and flow control state will be unchanged, and DEPCMDPAR2 will be ignored. The encoding of this action is backward compatible with software that previously set "Ignore Sequence Number" to 1. When issuing a Endpoint Configuration command with Config Action=Restore, Parameter 2 must be filled in with the same value returned by the Get Endpoint State command prior to hibernation.
29:26	Reserved

Field	Description
25:22	Burst Size (BrstSiz) When field is set to: ■ 0: Burst length = 1 ■ 1: Burst length = 2, and so on, up to, ■ 15: Burst length = 16 For IN transfers, this value represents the maximum length of the burst that the device attempts when the Host initiates an IN transfer with a TP_ACK. ■ If BrstSiz < the NumP value in the initiating TP_ACK, then the device controller limits the burst length to BrstSiz. ■ If BrstSiz >= the NumP value in the initiating TP_ACK, then the device controller attempts a burst length of NumP. For OUT transfers, this value represents the NumP value utilized in the response TP_ACK from the device controller. The NumP value indicates (to the host) the burst length the device desires. Note: In the special case of BrstSiz=0 (burst length = 1), the TP_ACK from the device controller has NumP=0; which is a flow-control condition. If this value is utilized, then the endpoint enters flow control for each DP received and a subsequent ERDY is transmitted (according to the USB Specification). However, this may have an undesirable impact on the system throughput. To avoid this impact, it is recommended to set BrstSize=1 to prevent the flow-control condition. The trade-off is that the host could potentially burst two OUT data packets to the device, resulting in an ACK for the first packet, and an NRDY for the second packet.
21:17	FIFO Number (FIFONum) Indicates which transmit FIFO is assigned to this endpoint. For control endpoints, the FIFONum value in the OUT direction must be programmed to the same value as the IN direction. This field should be set to 0 for all other OUT endpoints. Even though there may be more than 16 TxFIFOs in DRD mode, the device mode must use lower 16 TxFIFOs.
16:14	Reserved
13:3	Maximum Packet Size (MPS) Applies to IN and OUT endpoints. The application must program this field with the maximum packet size (in bytes) for the current USB endpoint. USB 3.0 supports up to 1024 bytes.
2:1	Endpoint Type (EPType) This is the transfer type supported by this USB endpoint. ■ 2'b00: Control ■ 2'b01: Isochronous ■ 2'b10: Bulk ■ 2'b11: Interrupt
0	Reserved

3.2.2.2 Command 2: Set Endpoint Transfer Resource Configuration (DEPXFERCFG)

There must be only one transfer resource allocated per endpoint. Start Transfer causes the use of the transfer resource. End Transfer or an XferComplete event releases the transfer resource. If software attempts to allocate more transfer resources than have been configured in the hardware, this command will return an error in the CmdStatus/EventStatus field.

> **Note** When operating in USB 2.0 speeds (HS/FS), if GUSB2PHYCFG[6] or GUSB2PHYCFG[8] is set to '1', it must be set to '0' prior to issuing this command and may be set to '1' after the command completes.

Table 3-4 Command 2: Set Endpoint Transfer Resource Configuration Parameters: DEPXFERCFG (Continued)

Field	Description
Parameter 2	
31:0	Reserved
Parameter 1	
31:0	Reserved
Parameter 0	
31:16	Reserved
15:0	Number of Transfer Resources (NumXferRes) Defines the number of Transfer Resources allocated to this endpoint. This field must be set to 1.

3.2.2.3 Command 3: Get Endpoint State (DEPGETSTATE)

After Get Endpoint State (DEPGETSTATE) command completes, the 32 bits in DEPCMDPAR2 represent the endpoint state to be saved. This includes the current data sequence number, flow control state, and control transfer state (for control endpoints).

This command is only used when the controller enters hibernation (refer to "Entering Hibernation in Device Mode While Connected" on page 419). The controller requires the 32 bits of state information when exiting hibernation as part of the DEPCFG command with the Config Action field set to "Restore endpoint state".

> **Note** When operating in USB 2.0 speeds (HS/FS), if GUSB2PHYCFG[6] or GUSB2PHYCFG[8] is set to '1', it must be set to '0' prior to issuing this command and may be set to '1' after the command completes.

3.2.2.4 Commands 4 and 5: Set Stall and Clear Stall (DEPSSTALL, DEPCSTALL)

Applies to non-isochronous IN and OUT endpoints only.

The application issues a Set Stall command to stall all tokens from the USB host to this endpoint. If the endpoint is in the NRDY state, the STALL state takes priority.

If a transaction is currently in progress when the software issues Set Stall, the behavior depends on the direction of the endpoint:

- For OUT endpoints, the current packet will complete. The controller will respond with STALL to the next OUT DP or token.
- For IN endpoints in Super Speed, the current burst transaction will complete, and the controller will respond with STALL to the next ACK TP. For other speeds, the controller will complete the current packet and respond STALL to the next IN token.

For non-control endpoints, the application is responsible for both setting and clearing STALL via the Set Stall/Clear Stall commands. When the application clears the STALL, the endpoint's data sequence number is reset to zero.

For control endpoints, the application issues only the Set Stall command, and only on the OUT direction of the control endpoint. The controller automatically clears the STALL when it receives a SETUP token for the endpoint. The application must not issue the Clear Stall command on a control endpoint.

If the endpoint is in flow control (NRDY) and also in the STALL state, it responds with a STALL for any packet. The only exception is that the controller always responds to SETUP data packets with an ACK handshake, independent of the STALL state.

Note

- When operating in USB 2.0 speeds (HS/FS), if `GUSB2PHYCFG[6]` or `GUSB2PHYCFG[8]` is set to ‘1’, it must be set to '0' prior to issuing this command and may be set to ‘1’ after the command completes.
- When issuing Clear Stall command for IN endpoints in SuperSpeed mode, the software must set the “ClearPendIN” bit to ‘1’ to clear any pending IN transactions, so that the device does not expect any ACK TP from the host for the data sent earlier.

3.2.2.5 Command 6: Start Transfer (DEPSTRTXFER)

Applies to IN and OUT endpoints.

A Transfer Descriptor (TD) in Device mode is a list of TRBs that comprises one or more transfers. The pointer to a TD is added to the controller’s cache by the Start Transfer command and removed by an XferComplete event or by the End Transfer command.

Software issues this command indicating that a descriptor is ready to be processed and DMA can start for this endpoint/stream combination. For IN endpoints, this causes the descriptor to be processed, and data is moved from the corresponding memory buffer to corresponding transmit FIFO when the transfer is started. For OUT endpoints, this causes the descriptor to be processed and data is moved from the receive FIFO to corresponding memory buffer.

In response to the Start Transfer command, the hardware assigns this transfer a resource index number (XferRscIdx) and returns the index in the `DEPCMDn` register and in the Command Complete event. This index must be used in subsequent Update and End Transfer commands.

It is illegal to issue a Start Transfer command for the same TD if it is already present in the controller’s cache.

Non-stream capable endpoints rules:

- The Stream ID field must be set to 0
- The HighPri field is reserved

Stream-capable endpoint rules:

- The Stream ID field must be set to non-zero and match the Stream ID passed into the Start Transfer endpoint command associated with this transfer. In all the TRBs of a transfer, the Stream ID fields must be the same.
- The HighPri field is reserved.

- When operating in USB 2.0 speeds (HS/FS), if GUSB2PHYCFG[6] or GUSB2PHYCFG[8] is set to '1', it must be set to '0' prior to issuing this command and may be set to '1' after the command completes.
- Before issuing a start transfer command, software needs to verify whether the link is in U0. If the link is not in U0, software needs to bring the link to U0 by performing a remote wakeup. This is applicable to both SS and USB 2.0 speeds.
- When an IN transfer is in progress, (that is, started and not completed yet), and the software sees a link state change event to L1, then the software can use the GDBGFIFOSPACE register (write followed by a read) to determine if the TxFIFO is empty or not. If the TxFIFO is not empty, it can initiate a remote wakeup.
- If software keeps polling the command status (bit 10, Command Active (CmdAct) of the Device Physical Endpoint command register) of a Start Transfer command for a stream capable Bulk endpoint without serving the events generated by the controller, the command will get stuck because the internal event queue buffer is full. This behavior occurs because internally, the command handler state machine inside the cch module is waiting for the stream state machine in the smu module to acknowledge that a new stream has already been added. At the same time, the stream state machine is waiting for the stream event to be stored in the event queue buffer.

  Software must always service the event interrupts generated by the controller. To prevent this deadlock situation from occurring, software is either processing the events while it is polling the Command Active bit or waiting for the command complete interrupt (by setting the bit 8, Command Interrupt on Complete (CmdIOC)) while it is processing the events.

Table 3-5 Command 6: Start Transfer Parameters: DEPSTRTXFER (Continued)

Field	Description
Parameter 2	
31:0	Reserved
Parameter 1	
31:0	Transfer Descriptor Address (TDAddr Low) Indicates the lower 32 bits of the external memory's start address for the transfer descriptor. Because TRBs must be aligned to a 16-byte boundary, the lower 4 bits of this address must be 0.
Parameter 0	
31:0	Transfer Descriptor Address (TDAddr High) Indicates the higher 32 bits of the external memory's start address for the transfer descriptor.

3.2.2.6 Command 7: Update Transfer (DEPUPDXFER)

If software uses circular TRB buffers and updates a TRB, whose Hardware Owner (HWO) bit was 0, by setting HWO=1, it must execute the Update Transfer command, specifying the transfer resource index of the TRB in the DEPCMD register. The hardware uses this information to re-cache the TRB.

Software may issue a special "No Response Update Transfer" command by setting CmdAct=0 and CmdIOC=0. In this case, the hardware does not generate a Command Complete event, does not set the CmdAct bit to '0' (because it will be '0'), and software may immediately issue another command to the same endpoint following this one. This special type of Update Transfer may not be used when software depends on the XferNotReady event to setup TRBs (see "On Demand" transfers in "Transfer Setup Recommendations" on page 345).

Software may issue an Update Transfer command for a transfer resource that has already completed (either due to XferComplete event or an End Transfer command), and the controller will detect that the Update Transfer is unnecessary. However, software must not issue an Update Transfer command for a transfer resource index that has never been started.

When operating in USB 2.0 speeds (HS/FS), if GUSB2PHYCFG[6] or GUSB2PHYCFG[8] is set to '1', it must be set to '0' prior to issuing this command and may be set to '1' after the command completes.

3.2.2.7 Command 8: End Transfer (DEPENDXFER)

Applies to IN and OUT endpoints. Software issues this command requesting DMA to stop for the endpoint/stream specifying the transfer resource index of the TRB and the ForceRM parameter to be set to 1 in the DEPCMD register.

When issuing an End Transfer command, software must set the CmdIOC bit (field 8) so that an Endpoint Command Complete event is generated after the transfer ends. This is necessary to synchronize the conclusion of system bus traffic before the End Transfer command is completed.

If GUCTL2[Rst_actbitlater] is set, Software can poll the completion of the End Transfer command by polling the command active bit to be cleared to 0.

For IN endpoints, this command causes descriptor processing to stop and the controller stops fetching new data for the endpoint and stops transfers on the USB. The controller may truncate a packet being transmitted with the DPPABORT ordered set, does not wait for any pending ACKs from the USB, and does not update the TRB status.

For OUT endpoints, this command causes descriptor processing to stop. If there is currently data, it is moved from the receive FIFO to corresponding memory buffer and completed at packet boundary. The controller does not update the TRB status.

Use this command under the following conditions:

- When handling USBReset or SetConfiguration, endpoints are closed using this command.
- After receiving a ClearFeature (STALL) control transfer, software issues End Transfer followed by Clear Stall, followed by Start Transfer.
- After an XferInProgress event when the TRB after the one that caused the XferInProgress event has its HWO bit set to '0', this command can be used.
- For isochronous endpoints, if the host stops moving data for many intervals, software may force the end of the transfer and wait for the host to restart.

The hardware does not issue a XferComplete event on End Transfer, but only issues a CommandComplete event.

When operating in USB 2.0 speeds (HS/FS), if GUSB2PHYCFG[6] or GUSB2PHYCFG[8] is set to ‘1’, it must be set to '0' prior to issuing this command and may be set to ‘1’ after the command completes.

Table 3-6 Command 8: End Transfer Parameters: DEPENDXFER (Continued)

Field	Description
Parameter 2	
31:0	Reserved
Parameter 1	
31:0	Reserved
Parameter 0	
31:0	Reserved

End Transfer Command With Hibernation Enabled

The following applies only when hibernation is enabled. Prior to hibernation, software will set ForceRM to '0' when it issues End Transfer commands for active transfers. This command may cause a TRB to be written back with intermediate state

When software issues an End Transfer command with the ForceRM bit set to 0, and the controller has at least one valid current TRB, it will perform the following:

- Write back the TRB updating the Buffer Pointer and Buffer Size fields to reflect the current state of the TRB.
- Write the value '4' (TRBInProgress) into the TRBSTS field to indicate that this TRB may have been partially used and is still active. This is an indication to software that the original buffer pointer field may have been changed by hardware.
- Set the HWO bit to '0' unless it is a Link TRB. The HWO bit in a Link TRB will be unchanged.

After the state is restored, software may restart a transfer from a partially completed TRB by setting TRBSTS to '0', setting the HWO bit back to '1', and issuing the Start Transfer command.

Software is not required to perform any special manipulation on partially completed TRBs. When the controller completes a restarted partially complete TRB, it will set TRBSTS back to '0'.

3.2.2.8 Command 9: Start New Configuration (DEPSTARTCFG)

Software issues this command under the following conditions:

- After power-on-reset with `XferRscIdx=0` before starting to configure Endpoints 0 and 1.

  `CmdIOC` must be set to '0' and software must poll the `CmdAct` bit to determine when the command is complete because Endpoint 0 is not yet configured with a valid interrupt number.
- With `XferRscIdx=2` when it receives SetConfiguration before starting to configure Endpoints > 1.

  `CmdIOC` may be set to '0' or '1'.

This command should always be issued to Endpoint 0 (`DEPCMD0`).

Hardware resets the transfer resource allocation to the value in the `XferRscIdx` parameter (must be 0 or 2) upon receiving this command.

When operating in USB 2.0 speeds (HS/FS), if `GUSB2PHYCFG[6]` or `GUSB2PHYCFG[8]` is set to ‘1’, it must be set to '0' prior to issuing this command and may be set to ‘1’ after the command completes.

3.3 Device Event Buffer Structure

This section describes the content of the Event Buffers.

When an event occurs within the controller, the hardware checks the enable bit that corresponds with the event to determine if the event will be written to the Event Buffer. The Event Buffer contains one of the following types of information, depending on the value of the lower bits of the event:

- Device Endpoint-Specific Event (DEPEVT) (Event[0] = 0x0)
  - The DEPCFG endpoint-specific command specifies the bits that are enabled and the Event Buffer used for these events.
- Device-Specific Event (DEVT) (Event[0] = 0x1, Event[7:1] = 0x00)
  - The Generic Command Complete event is enabled through the DGCMD.CmdIOC field when the command was issued.
  - The Event Buffer Overflow Event cannot be disabled and is written to the Event Buffer that encounters the overflow.
  - The rest of the Device-Specific Events are enabled via the DEVTEN register.
  - Except for the Event Buffer Overflow Event, these events are written to the Event Buffer specified in the DCFG.IntrNum field.

For details on the registers that define the beginning, end, and number of valid events in the Event Buffer, refer to the following sections in the Registers chapter:

- GEVNTADR(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)
- GEVNTSIZ(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)
- GEVNTCOUNT(#n) (for n = 0; n <= DWC_USB3_DEVICE_NUM_INT-1)

3.3.1 Event Buffer Content for Device Endpoint-Specific Events (DEPEVT)

Table 3-7 provides the event buffer content for device endpoint events.

Table 3-7 Device Endpoint-n Events: DEPEVT (Continued)

Field	Description
31:16	Event Parameters (EventParam) ■ For XferNotReady, XferComplete, and Stream events on Bulk Endpoints: ❑ [31:16]: StreamID. Applies only to bulk endpoints that support streams. This indicates the Stream ID of the transfer for which the event is generated ■ For XferInProgress: ❑ [31:16]: Isochronous Microframe Number (IsocMicroFrameNum): Indicates the microframe number of the beginning of the interval that generated the XferInProgress event (debug purposes only) ■ For XferNotReady events on Isochronous Endpoints: ❑ [31:16]: Isochronous Microframe Number (IsocMicroFrameNum). Indicates the microframe number during which the endpoint was not ready ❑ Note that DWC_usb3 controller represents USB bus time as a 14-bit value on the bus and also in the DSTS register (DSTS.SOFFN), but as a 16-bit value in the XferNotReady event. Use the 16-bit value to interact with Isochronous endpoints via the StartXfer command. The extra two bits that the DWC_usb3 controller produces will be necessary for handling wrap-around conditions in the interaction between software and hardware. ■ EPCmdCmplt events For all EPCmdCmplt events ■ [27:24]: Command Type. The command type that completed (Valid only in a DEPEVT event. Undefined when read from the DEPCMD.EventParam field). For EPCmdCmplt event in response to Start Transfer command: ■ [22:16]: Transfer Resource Index (XferRscIdx). The internal hardware transfer resource index assigned to this transfer. This index must be used in all Update Transfer and End Transfer commands.

Field	Description
15:12	Event Status (EventStatus) Within an XferNotReady event: ■ [15]: Indicates the reason why the XferNotReady event is generated: ❑ 1'b0: XferNotActive: Host initiated a transfer, but the requested transfer is not present in the hardware ❑ 1'b1: XferActive: Host initiated a transfer, the transfer is present, but no valid TRBs are available ■ [14]: Not Used ■ [13:12]: For control endpoints, indicates what stage was requested when the transfer was not ready: ❑ 2'b01: Control Data Request ❑ 2'b10: Control Status Request Within an XferComplete or XferInProgress event: ■ [15]: LST bit of the completed TRB (XferComplete only) ■ [15]: MissedIsoc: Indicates the interval did not complete successfully (XferInProgress only) ■ [14]: IOC bit of the TRB that completed ■ [13]: Indicates the TRB completed with a short packet reception or the last packet of an isochronous interval ■ [12]: Reserved If the host aborts the data stage of a control transfer, software may receive a XferComplete event with the EventStatus field equal to '0'. This is a valid event that must be processed as a part of the "Control Transfer Programming Model" on page 364. Within a Stream Event: ■ [15:12]: ❑ 4'h2: StreamNotFound: This stream event is issued when the stream-capable endpoint performed a search in its transfer resource cache, but could not find an active and ready stream. ❑ 4'h1: StreamFound: This stream event is issued when the stream-capable endpoint found an active and ready stream in its transfer resource cache, and initiated traffic for that stream to the host. The ID of the selected Stream is in the EventParam field. In response to a Start Transfer command: ■ [15:12]: ❑ 4'h2: Indicates expiry of the bus time reflected in the Start Transfer command. ❑ 4'h1: Indicates there is no transfer resource available on the endpoint. In response to a Set Transfer Resource (DEPXFERCFG) command: ■ [15:12]: ❑ 4'h1: Indicates an error has occurred because software is requesting more transfer resources to be assigned than have been configured in the hardware. In response to a End Transfer command: ■ [15:12]: ❑ 4'h1: Indicates an invalid transfer resource was specified.
11:10	Reserved
9:6	4'h7: Endpoint Command Complete (EPCmdCmplt) Indicates software may issue another Device Endpoint command to the endpoint. ■ When issued in response to an End Transfer command, indicates that DMA stopped for the endpoint. ■ For all other commands, this event does not imply that all effects of the command took place. The DEPCMD register contains the same status information present in the Event Status Bits field.

Field	Description
9:6	4'h6: Stream Event (StreamEvt) Indicates that a stream-capable endpoint initiated a search within its transfer resource cache. The result of the search is in the EventStatus field (Found or NotFound).
9:6	4'h5: Reserved
9:6	Reserved
9:6	4'h3: XferNotReady Event (XferNotReady) Indicates receipt of a transaction when no TRBs are available for the endpoint. For isochronous IN endpoints, a zero-length packet is automatically sent by hardware and NRDY for non-isochronous endpoints. The application must enable this event if it plans to issue Start Transfer on demand. This event can happen when software issues a Start Transfer or Update Transfer. In this case, software must ignore this event because it has already issued the Start Transfer or Update Transfer. XferNotReady is generated when the controller responds NRDY to the host on the USB. It is useful in the beginning of a transfer when software wants to wait for the host to start polling the endpoint before setting up TRBs, but it is not efficient to use this event to determine when the controller has run out of TRBs. For determining when the controller has processed TRBs and needs software to setup more, use the IOC field in a TRB along with the XferInProgress event that is generated when the controller completes a TRB with IOC=1. For non-stream-capable endpoints, the hardware filters multiple events if the host continues transactions, even after the controller responds with NRDY. This internal filter is reset after software issues any endpoint command to this endpoint. For stream-capable endpoints, this event is generated each time the host attempts a transaction, even if the controller responds with NRDY. For isochronous endpoints, this event is generated only once prior to the Start Transfer command to communicate the current bus time. For additional information, see the Event Status field.
9:6	4'h2: XferInProgress Event (XferInProgress) Applies to IN and OUT endpoints. Indicates an EP/Stream specific event happened and it is continuing the transfer. For additional information, see "Programming DWC_usb3 in Device Mode" on page 333".
9:6	4'h1: XferComplete Event (XferComplete) Applies to IN and OUT endpoints. Indicates that a EP/Stream transfer completed and the controller stopped the transfer. For additional information, see "Programming DWC_usb3 in Device Mode" on page 333.
9:6	4'h0: Reserved
5:1	Physical Endpoint Number (0–31)
0	1'b0: Indicates that this is an endpoint-specific event.

3.3.2 Event Buffer Content for Device-Specific Events (DEVT)

Table 3-8 Device-Specific Events: DEVT (Continued)

Field	Description
31:25	Reserved
24:16	Event Information Bits (EvtInfo) For a USB/Link State change Event, this field indicates the state of the link: ■ EvtInfo[8:5] – HIRD value received from the LPM token (valid for a Hibernation Request Event) ■ EvtInfo[4] – SuperSpeed event. Set to 1 for SS; Set to 0 for non-SS. ■ EvtInfo[3:0] – Link State. Indicates link state at the time of the event. Follows same encoding as DSTS link state bits.
15:13	Reserved
12:8	16: ECC Error This event is generated when a multiple-bit ECC error is internally detected.Note: This event is not generated only if DWC_USB3_EN_ECC=1 and BRERRCTL.ECCEN=1
	15: Reserved
	14: L1 Resume/RemoteWake Event This event is generated when the host initiates a L1 resume on the USB bus. It is not generated by the device initiating a L1 remote wakeup to the host.Note: This value is applicable only when GUCTL1[31] (DEV_DECOUPLE_L1L2_EVT) is set.
	13: Reserved
	12: Vendor Device Test LMP Received Event (VndrDevTstRcved) The controller writes this event to indicate that it received a Vendor Device Test LMP from the link partner. Bits 23:16 in this event indicates the Vendor-Specific Device Test field of the received Vendor Device Test LMP. The two DWORDs of Vendor Defined Data follow this event in the Event Buffer.
	11: Event Buffer Overflow Event (EvntOverflow) The controller writes this event to indicate there is no space in the event buffer, and one or more Device-specific events may have been dropped after this event. Endpoint-Specific events will not be dropped, but they will be delayed which can cause a drop in performance on the USB. Software uses this event as a warning that the Event Buffer is too small and the controller should be reconfigured with a larger Event Buffer or software needs to process events more quickly. In order to avoid repeated Event Buffer Overflow Events, software must free up space in the Event Buffer by acknowledging more than 1 event (writing a value greater than 4 to the GEVNTCOUNTn register). This event cannot be disabled, and is always written to the Event Buffer that encounters the overflow.

Field	Description
11:8 (Contd)	10: Generic Command Complete Event (CmdCmplt) The controller writes this event to indicate the generic command is complete.
	9: Erratic Error Event (ErrticErr) The controller writes this event to report erratic errors (phy_rxvalid_i/phy_rxvldh_i or phy_rxactive_i is asserted for at least 2 ms, due to PHY error) seen on the UTMI+. Due to erratic errors, the DWC_usb3 controller goes into Suspended state and a USB/Link state change event (ULStChng) is generated to the application. If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover. For SuperSpeed operation, the PHY erratic error event indicates that the PIPE is not responding to the PHY command (for example, `pipe3_PowerDown` change or receiver detection). After the controller requested any PHY command, if it did not receive pipe3_PhyStatus within 100ms, the controller will timeout and generate the erratic error event. After generating the erratic error, the controller assumes that the PHY command is completed successfully and proceeds with the next pending PHY command. Software must reset the controller on receiving the erratic error.
	8: L1 Suspend Event (L1SUSP) This event provides a notification that the link has gone to a L1 suspend state. This event is not generated when L1 hibernation is enabled. Note: This value is applicable only when `GUCTL1[31] (DEV_DECOUPLE_L1L2_EVT)` is set.
	7: Start of (micro)Frame (Sof)
	6: USB Suspend Entry Event ■ When `GUCTL1[DEV_DECOUPLE_L1L2_EVT]=1`: This event provides a notification that the link has gone to a suspend state (L2, U3). This event is generated when hibernation mode is disabled. The existing Link State Change event (3) provides the same information, but is generated for every link state change. ■ When `GUCTL1[DEV_DECOUPLE_L1L2_EVT]=0`: This event provides a notification that the link has gone to a suspend state (L2, U3, or L1). This event is generated when hibernation mode is disabled. The existing Link State Change event (3) provides the same information, but is generated for every link state change.

Field	Description
11:8 (Contd)	5: Hibernation Request Event ■ This event provides a notification that the link has gone to a suspend state where hibernation is supported (L2, U3, or L1) and requires software intervention ❑ When `DCTL.KeepConnect=1` and `L1HibernationEn=0`, this event is generated only when the link goes to L2 or U3 ❑ When `DCTL.KeepConnect=1`, `L1HibernationEn=1`, and the controller is configured to support LPM (`DCTL.LPMCap=1` and `DCTL.HIRD_Thres[4]=1`), this event is generated when the link goes to L2, U3, or L1 with an HIRD value greater than the `DCTL.HIRD_Thres[3:0]` value ■ The existing Link State Change event (3) provides the same information, but is generated for every link state change. If both events are enabled, only the Hibernation Request event will be generated if the conditions are met. ■ When this event occurs, software is required to start the hibernation process ■ EvtInfo[4:0] indicates the Link State and Speed which caused the Hibernation Event (using the same encoding as the Link State Change event. Suspend is 3, and L1 is 2) ■ EvtInfo[8:5] provides the HIRD value from the host's LPM token ■ For L2 and U3, this event is only generated when `DCTL.KeepConnect=1` ■ For L1, this event is only generated when `DCTL.L1HibernationEn=1` and `DCTL.KeepConnect=1` ■ In addition to the suspend states that trigger this event, hibernation is also supported when the link is disconnected. See ‘Device-Initiated Disconnect and Low Power While Disconnected’ for details.
	4: Resume/Remote Wakeup Detected Event (WkUpEvt) ■ When `GUCTL1[DEV_DECOUPLE_L1L2_EVT]=1`: L2/U3 Resume/Remote Wakeup detected Event (WkUpEvt).This event is generated when the host initiates a resume condition on the USB bus from L2/U3 state. It is not generated by the device initiating a remote wakeup to the host. ■ When `GUCTL1[DEV_DECOUPLE_L1L2_EVT]=0`: This event is generated when the host initiates a resume (L1/L2/U3) condition on the USB bus. This event is not generated by the device initiating a remote wakeup to the host.
	3: USB/Link State Change (ULStChng) The controller writes this event to indicate a change of USB or Link state. Bits 23:16 (EvtInfo) of this event indicate the Link Status. The event is generated in SuperSpeed when the link LTSSM state changes, except when LTSSM exits from HOT_RESET, POLL or LTSSM enters into RECOVERY.
	2: Connection Done (ConnectDone)
	1: USB Reset (USBRst)
	0: Disconnect Detected Event (DisconnEvt) Note: This event is generated only if Vbus is off.
7:1	7’h00 - Device Specific Event
0	1’b1: Non-Endpoint-Specific Event

4

Programming DWC_usb3 in Device Mode

This chapter describes the programming requirements for the DWC_usb3 controller in device mode. The following topics are discussed:

4.1 Initializing Registers

The USB 3.0 controller contains global registers (prefixed by G*) and device registers (prefixed by D*) that are programmed to start operation and handle certain events. This section describes which registers must be accessed depending on the event that software is attempting to handle:

- Power-On or Soft Reset
- USB Reset Event
- Connect Done Event
- SetAddress Device Request
- SetConfiguration Device Request
- Disconnect Event
- Device-Initiated Disconnect and Reconnect
- Device U3 Exit

4.1.1 Device Power-On or Soft Reset

This section explains device controller initialization after power-on or soft reset. The application must follow this initialization sequence for device mode operation. Some registers are initialized according to the configuration parameters that you selected during coreConsultant configuration. For information on the global registers and device registers, see Registers chapter.

When the controller is first powered on, software initializes the following registers. The order of operations is not important, except for the first and last steps (`DCTL.CSftRst=1` and `DCTL.RunStop=1`).

Table 4-1 Power-On or Soft Reset Register Initialization (Continued)

Register	Description
DCTL	Set the CSftRst field to ‘1’ and wait for a read to return ‘0’. This resets the device controller.
GSBUSCFG0/1	Leave the default values if the correct power-on values were selected during coreConsultant configuration.
GTXTHRCFG/ GRXTHRCFG	This is required only if you are planning to enable thresholding. Leave the default values if the correct power-on values were selected during coreConsultant configuration.
GSNPSID	The software must read the Synopsys ID register to find the controller version and configure the driver for any version-specific features.
GUID	Optionally, the software can program the User ID GUID register, if this register is selected for implementation in coreConsultant.
GUSB2PHYCFG	Program the following PHY configuration fields: USBTrdTim, FSIntf, PHYIf, TOUTCal, or leave the default values if the correct power-on values were selected during coreConsultant configuration. Note: The PHY must not be enabled for auto-resume in device mode. Therefore, the field GUSB2PHYCFG[15] (ULPIAutoRes) must be written with '0' during the power-on initialization in case the reset value is '1'.
GUSB3PIPECTL	Program the following PHY configuration fields: DatWidth, PrtOpDir, or leave the default values if the correct power-on values were selected during coreConsultant configuration.

Register	Description
GTXFIFOSIZn	Write these registers to allocate prefetch buffers for each Tx endpoint. Unless the packet sizes of the endpoints are application-specific, it is recommended to use the default value. For details, see Programming DWC_usb3 in Device Mode“Device TxFIFO Data Allocation” on page 218.
GRXFIFOSIZ0	Write this register to allocate the receive buffer for all endpoints. Unless the packet sizes of the endpoints are application-specific, it is recommended to use the default value. For details, see Programming DWC_usb3 in Device Mode“Device RxFIFO Data Allocation” on page 217.
GEVNTADRn/ GEVNTSIZn/ GEVNTCOUNTn	Depending on the number of interrupts allocated, program the Event Buffer Address and Size registers to point to the Event Buffer locations in system memory, the sizes of the buffers, and unmask the interrupt. Note: USB operation stops if the Event Buffer memory is insufficient, because the controller stops receiving/transmitting packets.
GCTL	Program this register to override scaledown, RAM clock select, and clock gating parameters
DCFG	Program device speed and periodic frame interval Note: If HWPARAMS3.`DWC_USB3_SSPHY_INTERFACE is zero, then program the DCFG.DevSpd to USB 2.0-only (HS/FS) speeds.
DEVTEN	At a minimum, enable USB Reset, Connection Done, and USB/Link State Change events.
DEPCMD0	Issue a DEPSTARTCFG command with DEPCMD0.XferRscIdx set to 0 and CmdIOC set to 0 to initialize the transfer resource allocation. Poll CmdAct for completion.
DEPCMD0/ DEPCMD1	Issue a DEPCFG command for physical endpoints 0 & 1 with the following characteristics, and poll CmdAct for completions: ■ USB Endpoint Number = 0 or 1 (for physical endpoint 0 or 1) ■ FIFONum= 0 ■ XferNRdyEn and XferCmplEn = 1 ■ Maximum Packet Size = 512 ■ Burst Size = 0 ■ EPType = 2’b00 (Control) Note: The command has to be issued for EP0 first, followed by EP1.
DEPCMD0/ DEPCMD1	Issue a DEPXFERCFG command for physical endpoints 0 & 1 with DEPCMDPAR0_0/1 set to 1, and poll CmdAct for completions Note: The command has to be issued for EP0 first, followed by EP1.
DEPCMD0	Prepare a buffer for a setup packet, initialize a setup TRB, and issue a DEPSTRTXFER command for physical endpoint 0, pointing to the setup TRB. Poll CmdAct for completion. Note: The controller attempts to fetch the setup TRB via the master interface after this command completes.
DALEPENA	Enable physical endpoints 0 & 1 by writing 0x3 to this register.
DCTL	Set DCTL.RunStop to ‘1’ to allow the device to attach to the host. At this point, the device is ready to receive SOF packets, respond to control transfers on control endpoint 0, and generate events.

After the controller has been started, wait for the following events:

1. Wait for a `DEVT.USBReset` event. This indicates that a reset is detected on the USB. On this event, the application needs to perform the steps listed in Initialization on USB Reset.
2. Wait for a `DEVT.ConnectionDone` event. This event indicates the end of the reset on the USB. On this event, read the `DSTS` register to get the connection speed. On this event, the application needs to perform the steps listed in section Initialization on Connect Done.

4.1.2 Initialization on USB Reset

To initialize the controller as a device, during USB Reset, the application must perform the following steps:

Table 4-2 Initialization on USB Reset (Continued)

Register	Description
DEPCMD0	If a control transfer is still in progress, complete it and get the controller into the “Setup a Control-Setup TRB / Start Transfer” state (see Control Transfer Programming Model)
DEPCMDn	Issue a DEPENDXFER command for any active transfers (except for the default control endpoint 0)
DEPCMDn	Issue a DEPCSTALL (ClearStall) command for any endpoint in STALL mode prior to the USB Reset (excluding control endpoints)
DCFG	Set DevAddr to ‘0’

Special Reset Considerations for Default Control Endpoint 0: The default control endpoint is not affected by a USB Reset, so software must continue following the software flow for control transfers explained in Control Transfer Programming Model even across a USB Reset event. Resources can only be assigned to the default control endpoint once after a power-on or soft reset.

4.1.3 Initialization on Connect Done

When this event is received, software must perform the following steps:

Table 4-3 Initialization on Connect Done (Continued)

Register	Description
DSTS	Read this register to obtain the connection speed.
GCTL	Program the RAMClkSel field to select the correct clock for the RAM clock domain. This field is reset to 0 after USB reset, so it must be reprogrammed each time on Connect Done.
DEPCMD0/ DEPCMD1	Issue a DEPCFG command (with Config Action set to “Modify”) for physical endpoints 0 & 1 using the same endpoint characteristics from Power-On Reset, but set MaxPacketSize to 512 (SuperSpeed), 64 (High-Speed), 8/16/32/64 (Full-Speed), or 8 (Low-Speed).
GTXFIFOSIZn	(optional) Based on the new MaxPacketSize of IN endpoint 0, software may choose to re-allocate the TX FIFO sizes by writing to these registers.

4.1.4 Initialization on SetAddress Request

When the application receives a `SetAddress` request in a SETUP packet, it performs the following steps:

Table 4-4 Initialization on SetAddress (Continued)

Register	Description
DCFG	Program the DCFG register with the device address received as part of the SetAddress request when SETUP packet is decoded.
DEPCMD1	After receiving the XferNotReady(Status) event, acknowledge the status stage by issuing a DEPSTRTXFER command pointing to a Status TRB. This step must be done after the DCFG register is programmed with the new device address.

At this point, the device is ready to receive micro-SOF/ITP and is configured to receive control transfers on control endpoint 0 with a new address assigned.

4.1.5 Initialization on SetConfiguration or SetInterface Request

When the application receives a `SetConfiguration` or `SetInterface` request in a SETUP packet, it performs the following steps:

Table 4-5 Initialization on SetConfiguration or SetInterface Request (Continued)

Register	Description
DALEPENA	Set this register to 0x3 to disable all endpoints other than the default control endpoint 0.
DEPCMDn	Issue a DEPENDXFER command for any active transfers (except for the default control endpoint 0).
DEPCMD1	Issue a DEPCFG command (with Config Action field set to “Modify”) for physical endpoint 1 using the current endpoint characteristics to re-initialize the TX FIFO allocation.
DEPCMD0	Issue a DEPSTARTCFG command with DEPCMD0.XferRscIdx set to 2 to re-initialize the transfer resource allocation.
DEPCMDn	Issue a DEPCFG command (with the Config Action field set to “Initialize”) for each endpoint that is present in the new configuration (except for the default control endpoint). Note: Control endpoints are bi-directional and must use consecutive even/odd physical endpoint numbers for the OUT/IN direction (such as 2/3, or 4/5), and the FIFONum must be configured to the same value in both directions.
DEPCMDn	Issue a DEPXFERCFG command for each endpoint that is present in the new configuration (except for the default control endpoint 0).
GTXFIFOSIZn	(optional) Based on the new configuration of IN endpoints, the software may choose to re-allocate the TX FIFO sizes by writing to these registers.
DALEPENA	Enable the logical endpoints that are active in the new configuration.
DEPCMD1	After receiving the XferNotReady(Status) event, acknowledge the status stage by issuing a DEPSTRTXFER command pointing to a Status TRB. This step must be done after the previous steps to ensure the host does not access any other endpoints that are being set up.

At this point, the controller is prepared to accept Start Transfer commands for the newly-configured endpoints. For information on how to set up transfers, see the Operational Model.

4.1.6 Alternate Initialization on SetInterface Request

When handling a `SetInterface` device request, another possibility is that software may reconfigure existing endpoints instead of starting the configuration from the beginning. This is only possible if no TX FIFOs need to be reassigned. This flow also assumes that the endpoints that are being removed or reconfigured in the new interface have halted their traffic before the host issuing the `SetInterface` request.

Table 4-6 Alternate Initialization on SetInterface Request (Continued)

Register	Description
DEPCMDn	Make sure there are no transfers still active on the endpoints that are changing in the new interface. This is normally ensured by the host prior to issuing the SetInterface, but if not, issue an End Transfer command for the transfer on each endpoint that is changing.
DEPCMDn	Issue a DEPCFG command (with the Config Action field set to “Initialize”) for each endpoint that is changing in the new configuration. The side effect of the DEPCFG command is that the endpoint's sequence number is automatically set to 0. If a new endpoint is being added, issue a DEPXFERCFG command.
DALEPENA	Write this register with all the endpoints that are enabled (including the ones that are not changing).
DEPCMDn	Using the StartXfer command, acknowledge the Status stage response for the SetInterface request.

4.1.7 Initialization on Disconnect Event

When the application receives a Disconnect event, it must set `DCTL[8:5]` to 5. Other than this, the controller does not require any initialization. Because the `DCTL.RunStop` bit is still ‘1’, the device attempts to reconnect to the host, at which time a USB Reset and Connect Done event occurs. However, if the application does not want to attempt to reconnect to the host, it must perform the steps in the next section.

When DCFG.DevSpd is programmed for 2.0 only mode (such as, High-Speed or Full-Speed), if the application wants to issue any commands to clear any pending transfers during a Disconnect interrupt, then it has to disable gusb2phycfg[SusPHY] before issuing any commands and re-enable it after the commands have completed.

4.1.8 Device-Initiated Disconnect

If the application wants to disconnect from the host, it must perform the following actions:

Table 4-7 Initialization on Device-Initiated Disconnect (Continued)

Register	Description
DEPCMD0	If a control transfer is still in progress, complete it and get the controller into the “Set up a Control-Setup TRB / Start Transfer” state (see Control Transfer Programming Model).
DEPCMDn	Issue a DEPENDXFER command for any active transfers (except for the default control endpoint 0).
DCTL	Set DCTL.RunStop to ‘0’ to disconnect from the host.
DSTS	Poll DevCtrlHlt until it is ‘1’.

At this point, the device is disconnected from the host and does not attempt to reconnect.

4.1.9 Reconnect after Device-Initiated Disconnect

If the application decides that it wants to reconnect to the host, it must follow the steps in “Device Power-On or Soft Reset” on page 334.

4.1.10 Initialization after U3 Exit

When a device exits from U3 to U0, if there is an active transfer on an endpoint prior to the entry into U3, then software needs to issue a ‘Set Endpoint NRDY’ command. This makes the device endpoint send ERDY.

The Set Endpoint NRDY command should be used only for device initialization after U3 exit. It should not to be used in any other scenario.

Typically, the host does not transition the device to U3 during transfers. However, when the following conditions occur, the host may send LGO_U3 request:

- Endpoint enters flow control because transfer is not ready, and
- Host decides to enter a low power state due to user interaction

In such a scenario, the device accepts the LGO_U3 request and enters the U3 state. Because SS is full duplex, the device may have sent ERDY while receiving the LGO_U3 request. As a result,

- On the host side, this ERDY is not processed
- On device side, there is no information stored that this corner case occurred and the host did not process ERDY

Therefore, when coming out of U3, the device needs to send ERDY again so that the host continues the transfer for that endpoint. This is achieved by the device software driver issuing Set Endpoint NRDY command.

When the device software receives a link state change event with the link indicating that the device is entering U3, then the device software tracks and waits for the link state change event with the link indicating U0 entry. When the device software detects this condition and also detects that there are transfers still active on the endpoint, it issues the Set EndPoint NRDY command.

4.2 Operational Model

The following sections describe the processes and data structures that the controller uses to implement the USB 3.0 specification.

4.2.1 USB and Physical Endpoints

Endpoints are referred to in two ways:

- As a USB endpoint number: USB endpoints are defined in the USB specification.
- As a physical endpoint resource number: The hardware has a fixed (coreConsultant-configurable) number of physical endpoint resources. Each resource is unidirectional; you can configure any USB endpoint to refer to either direction.

The software always works on physical endpoints and it knows how physical endpoint corresponds to USB endpoints (see “Flexible Endpoint Mapping” in the databook). `DALEPENA` is the only register that has one enable bit per USB endpoint.

During `SetConfiguration`, software maps physical endpoint resources to the required USB endpoints.

When a USB request comes in, the USB endpoint number gets converted to a physical endpoint number in the controller. Similarly, when a USB packet is sent out, the physical endpoint number is converted to the USB endpoint and sent out.

A USB control endpoint requires two physical endpoints. One physical endpoint is mapped to the OUT direction of the Control endpoint, and the other one is mapped to the IN direction of the Control endpoint. Specifically, the two physical endpoints are used as the following:

- The Setup stage of any control transfer uses the OUT direction physical endpoint.
- For a control write or 2-stage transfer, the Data stage (if present) uses the OUT direction physical endpoint and the Status stage uses the IN direction physical endpoint.
- For a control read transfer, the Data stage uses the IN direction physical endpoint and the Status stage uses the OUT direction physical endpoint.

4.2.2 Event Buffers

Hardware passes command completion, transfer progress, and asynchronous events to software through one or more Event Buffers.

To configure an Event Buffer, the software performs the following steps:

1. Sets up an empty buffer in system memory.
2. Writes the address of the beginning of the buffer into `GEVNTADRn`. This address must be aligned to the Event Buffer size.
3. Writes the size of the buffer and interrupt mask into `GEVNTSIZn`. Depending on your system interrupt latency, enough Event Buffer space must be allocated to avoid lost interrupts or reduced performance.
4. Write a 0 into the `GEVNTCOUNTn` register. This must be the last step, as it enables the Event Buffer.

After the Event Buffer has been configured, software must not change the size or address.

There is one interrupt line per Event Buffer that indicates there are one or more events present. Software reads one or more events out of the buffer and indicates to hardware how many events it processed by writing the byte count to the `GEVNTCOUNTn` register.

Clock crossing delays may result in the interrupt's continual assertion after software acknowledges the last event. Therefore, when the interrupt line is asserted, software must read the GEVNTCOUNT register and only process events if the GEVNTCOUNT is greater than 0.

The first event produced by the controller after the Event Buffer is configured is written to the address specified in GEVNTADRn. Most events are 32 bits, and subsequent events are written to the address (PreviousEventAddress + 4). When that address exceeds the sum of GEVNTADR and GEVNTSIZ, the controller wraps around to the first GEVNTADR value. In this way, the Event Buffer operates like a circular buffer with hardware writing to the "tail" of the buffer and software reading from the "head."

Most events are exactly four bytes in size, but there is one exception: The Vendor Device Test LMP Received Event (VndrDevTstRcved) is a 12-byte event that includes a header in the first four bytes and the contents of the LMP in the following eight bytes.

Using Vendor Device Test feature during normal operation of the link results in undefined behavior. Therefore, do not use Vendor Device Test feature when there is normal traffic on the USB.

When an event occurs within the controller, hardware checks the enable bit that corresponds to the event to decide whether the event needs to be written to the Event Buffer or not. The Event Buffer contains one of the following types of information:

- Endpoint-Specific Event (DEPEVT)
  - The DEPCFG endpoint-specific command specifies the enables and which Event Buffer to use for these events.
- Device-Specific Event (DEVT)
  - The Generic Command Complete event is enabled through the DGCMD.CmdIOC field when the command is issued.
  - The rest of the device-specific events are enabled through the DEVTEN register.
  - These events are written to the Event Buffer specified in the DCFG.IntrNum field.

The controller always leaves one entry free in each Event Buffer. When the Event Buffer is almost full, hardware writes the Event Buffer Overflow event and the USB eventually gets stalled when endpoints start responding NRDY or the link layer stops returning credits (in SuperSpeed). This event is an indication to software that it is not processing events quickly enough. During this time, events are queued up internally. When software frees up Event Buffer space, the queued up events are written out and the USB returns to normal operation.

4.2.3 Transfer and Buffer Rules

Buffers that are used to transfer data to and from an endpoint are defined using a Buffer Descriptor, which consists of one or more Transfer Request Blocks (TRBs). A CHN flag in the TRB is used to identify the TRBs that comprise a Buffer Descriptor. Therefore, a Buffer Descriptor refers to a consecutive set of TRB data structures where the CHN flag is set in all TRBs, except the last. Note that a Buffer Descriptor may consist of a single TRB, whose CHN flag will not be set.

Software communicates the location of the first TRB by using the Start Transfer command on an endpoint. Even endpoints that are not stream-capable use this command.

The controller fetches TRBs from external memory starting at the address provided in the Start Transfer command parameters, continuing in a linear fashion and following the Link TRB until a TRB is encountered that has its `LST` bit set to '1' or `HWO` bit set to '0'. Therefore, software must ensure that it has valid TRBs prepared before issuing the Start Transfer command to prevent the controller from reading uninitialized memory.

Descriptor fetch requests are buffered within the hardware and handled separately from the progress of the current transfer. Therefore, it is possible that the controller completes a transfer with `XferComplete` but still continues reading TRBs that it has not cached yet. If software is immediately de-allocating the memory for TRBs based on the `XferComplete` event, it is recommended that software issue an End Transfer command for the endpoint/transfer resource before de-allocating the memory. The completion of the End Transfer command flushes out any pipelined descriptor fetches and avoids a potential bus error.

While processing TRBs, two conditions may cause the controller to write out an event and raise an interrupt line:

- TRB Complete:
  - For OUT endpoints, a packet is received which reduces the remaining byte count in the TRB buffer to zero.
  - For IN endpoints, an acknowledgment is received for a transmitted packet which reduces the remaining byte count in the TRB buffer to zero.
- Short Packet Received:

  For OUT endpoints only. While writing to a TRB buffer, the endpoint receives a packet that is smaller than the endpoint's `MaxPacketSize`.

Table 4-8 describes the action taken by the controller when the these conditions occur.

Table 4-8 DWC_usb3 Controller Actions Based on TRB Control Bits (Continued)

Direction	TRB Complete	Short Packet	ISP	IOC	CHN	LST	CSP	Action
IN	Yes			X	0	1		XferComplete event
IN	Yes			0	X	0		No event
IN	Yes			1	X	0		XferInProgress event
OUT	Yes	No		X	0	1		XferComplete event
OUT	Yes	No		0	X	0		No event
OUT	Yes	No		1	X	0		XferInProgress event
OUT	X	Yes	X	X	X	X	0	XferComplete event[a]
OUT	X	Yes	X	X	0	1	1	XferComplete event
OUT	X	Yes	X	X	1	0	1	Search for CHN=0, accumulate IOC and ISP, then follow CHN=0 rules[b]
OUT	X	Yes	X	1	0	0	1	XferInProgress event
OUT	X	Yes	0	0	0	0	1	No event
OUT	X	Yes	1	0	0	0	1	XferInProgress event

a. When a TRB receives whose CSP bit is 0 and CHN bit is 1 receives a short packet, the chained TRBs that follow it are not written back (for example, the BUFSIZ and HWO fields remain the same as the software-prepared value)

b. In the case of an OUT endpoint, if the CHN bit is set (and CSP is also set), and a short packet is received, the controller retires the TRB in progress and skip past the TRB where CHN=0, accumulating the ISP and IOC bits from each TRB. If ISP or IOC is set in any TRB, the controller generates an XferInProgress event. Hardware does not set the HWO bit to 0 in skipped TRBs. If the endpoint type is isochronous, the CHN=0 TRB will also be retired and its buffer size field updated with the total number of bytes remaining in the BD.

On XferComplete and XferInProgress events, status bits in the event indicate LST, IOC, Short Packet Received, or Bus Error status. In the “fast-forward” case, the IOC status is accumulated from the skipped TRBs, not just the TRB that received the short packet.

When the hardware writes back the TRBs, it updates the BUFSIZ field to represent the remaining unused buffer.

4.2.3.1 Number of TRBs Rule

- Software must set up only one TRB for a control setup or status stage.
- If software is preparing multiple transfers for an IN endpoint, it may be necessary to place a 0-length TRB between transfers that are MaxPacketSize aligned to indicate transfer boundaries to the host. This is not necessary, if the class driver and the host can handle bursting between transfers.
- If software is preparing multiple transfers for an OUT endpoint, it needs to place a MaxPacketSize TRB between transfers, if it expects the host to transmit a 0-length packet between transfers.

4.2.3.2 TRB Control Bit Rules

Transfer control bits (see "Structures" on page 297) must conform to the following restrictions:

- For OUT endpoints, the CSP bit must be the same in every TRB within a Buffer Descriptor (either set or clear).
- The controller autonomously checks the HWO field of a TRB to determine if the entire TRB is valid. Therefore, software must ensure that the rest of the TRB is valid before setting the HWO field to '1'. In most systems, this means that software must update the fourth DWORD of a TRB last. Every time software validates a TRB by setting HWO=1, it must also issue an Update Transfer command to the controller.
- Software sets the HWO bit to 1 when it creates the TRB, and cannot modify it until hardware resets it to 0. However,
  - Software must detect when a "fast-forward" occurs on an OUT endpoint that receives a short packet, since some TRBs in a chain may still have their HWO bit set to 1 while belonging to software.
  - Hardware does not clear the HWO bit of a Link TRB. Therefore, software can only modify a Link TRB if the TRB prior to the Link TRB has its HWO bit set to 0.
- The LST bit must not be set to 1 for isochronous endpoints.
- For a Setup or Status TRB, set CHN=0, LST=1, and CSP=0.
- For the data stage of a control transfer, set CSP=0 in all TRBs and LST=1 in the last TRB of the data stage.
- For Link TRBs, the LST, CHN, IOC, ISP, CSP, and Stream ID fields are ignored and must be set to 0.
- The Link TRB's chain bit is implicitly equal to the chain bit of the TRB before it. If the TRB before it has CHN=1, then the Link acts as if it's CHN=1. If the TRB before it has CHN=0, then the Link acts as if it's CHN=0. A Link TRB cannot be the last TRB in a Buffer Descriptor.
- When CSP=1 and the controller receives a short packet, it searches for the end of the Buffer Descriptor by finding the Normal TRB that has CHN=0. Therefore, it is illegal to setup a circular buffer with all Normal TRBs with CHN=1.

4.2.3.3 Buffer Size Rules and Zero-Length Packets

The hardware contains a cache that holds a fixed number of TRBs per transfer, configured through the coreConsultant parameter DWC_USB3_CACHE_TRBS_PER_TRANSFER or 15, whichever is smaller.

For IN endpoints, the following rules apply:

- The number of chained TRBs necessary to construct a single packet must never exceed (DWC_USB3_CACHE_TRBS_PER_TRANSFER – 1). A maximum of one Link TRB can be present in the chain.
- If software wants to indicate a transfer completion to the host by sending a zero-length packet after a multiple of MaxPacketSize, it must set up a zero-length TRB following the last TRB in the transfer.

For OUT endpoints, the following rules apply:

- The BUFSIZ field must be ≥ 1 byte.
- The total size of a Buffer Descriptor must be a multiple of MaxPacketSize.
- For setup stage of control transfer, BUFSIZ field must be 8 bytes.
- A received zero-length packet still requires a MaxPacketSize buffer. Therefore, if the expected amount of data to be received is a multiple of MaxPacketSize, software should add MaxPacketSize bytes to the buffer to sink a possible zero-length packet at the end of the transfer.

For IN and OUT endpoints, the following rule applies:

- The BUFSIZ field in a Link TRB must be set to 0.

4.2.4 Transfer Setup Recommendations

The software can either set up transfers before the host attempts to move data on an endpoint (“preset” transfers) or can set up transfers on demand (“on-demand” transfers). When using preset transfers, software can safely disable the XferNotReady event in the endpoint configuration. However, when using on-demand transfers, the XferNotReady event must be enabled and software may not use the “No Response” variant of the Update Transfer command. The XferNotReady event is issued when the host attempts to move data on an endpoint when one of the following conditions is present:

- No previous transfer was started with Start Transfer.
- Not enough hardware-owned (HWO=1) TRBs are available to handle the requested data movement.

The XferNotReady event must not be disabled for control endpoints because the event is an integral part of control transfer handling.

When RX packet threshold feature is enabled, do not use the “on-demand” mode of transfer for SS OUT endpoints. This restriction is because, if the last packet of the transfer ends with an ACK TP (NumP=0), then the USB is in flow control and the host waits for an ERDY, but the “on-demand” application does not setup any packet until the host polls, creating a dead-lock situation.

Although there are many valid ways to set up transfers, it is recommended that you choose one of three general mechanisms:

- When software wants to set up one transfer at a time and has the entire buffer available for transfer, it must set up TRBs that point to the data buffers and in the last TRB it must set the LST bit and issue Start Transfer, which points to the first TRB location.

- When the USB transfer completes, the controller notifies the software through the `XferComplete` event. The `LST` bit is set in the status field of the event. The `XferComplete` also releases the Transfer Resource.
- A premature `XferComplete` event can happen before all the data buffers are exhausted, if there is a Bus Error or, in an OUT transfer a short packet has been received and `CSP=0`. During these conditions, the TRB has an updated `BUFSIZ` field which represents the amount of buffer remaining after the successful part of data transfer.
- Software can also set up an `IOC` bit TRB, so that it gets notified when data buffers are used up and can free them up sooner than waiting for the entire transfer to complete. On completing a TRB with `IOC` set, and not the last one, the controller will issue an `XferInProgress` event with `IOC` set in the status field of the event.

- When software wants to set up one transfer at a time, but it has fewer data buffers available than the full transfer size, it must set up circular TRBs (using a Link TRB), and also set up `IOC` bits in the TRBs. Depending on buffer allocation and interrupt frequency it can set `IOC` for once in “x” number of TRBs.
  - As soon as the USB transfer for a TRB is completed, and if `IOC` is set, the controller generates an XferInProgress event with `IOC` set in the status field.
  - On seeing the event, software reuses the TRB and updates it with the next data buffer. It also issues an Update Transfer command, indicating it has updated a TRB.
  - If software is slow and hardware finds a TRB with `HWO` reset to 0, it waits for an Update Transfer command. Upon seeing the Update Transfer, it prefetches the TRB and continues the transfer.
  - When software reaches the end of the transfer, it sets the TRB LST bit. When hardware completes the TRB, it issues an `XferComplete` event and releases the Transfer Resource.
- When the Device software has multiple transfers to set up, it must set up circular TRBs and also set up `CHN` bits in all TRBs, except the last of each transfer. For OUT endpoints, multiple transfers can be supported when the `CSP` field is set to '0' if each transfer has a multiple of `MaxPacketSize` bytes, because a short packet ends the transfer. If multiple transfers can contain short packets, the CSP field must be set to '1' to enable the next transfer to continue even if a short packet is received. Depending on buffer allocation and interrupt frequency, it can either set the `IOC` for once in "x" number of TRBs or in the last TRB of each transfer.
  - For OUT endpoints, the transfer can finish prematurely due to a short packet from the host. In this case, the controller processes the remaining TRBs, skipping the updates to these TRBs until it reaches TRB of the next transfer. While skipping these TRBs, if any of the TRB it encounters the interrupt setting of `ISP` or `IOC`, it generates the corresponding event once and ignore the interrupt settings of the remaining TRBs until a TRB of next transfer is reached.
  - Device software can reclaim these skipped TRBs even though the `HWO` still indicates 1 (hardware-owned).
  - For IN endpoints, software cannot stop providing transfers while it is ending transfers with `CHN=0` and `LST=0`, otherwise, it is possible that the endpoint is left in a flow-controlled state on the USB.

4.2.5 Transfer Resource Usage and Transfer State

The software allocates one Transfer Resource for an endpoint during initialization. When software issues a Start Transfer command, this Transfer Resource is used. When an `XferComplete` event happens, the Transfer Resource is released back to software. Similarly, when End Transfer completes, the Transfer Resource is released.

Following is a typical transfer usage summary (not complete usage model):

- Software sets up data buffer(s) and TRB(s) in external memory with the TRB(s) pointing to the buffer(s).
- Software issues Start Transfer with the pointer to the first TRB in the command parameters.
- If software sets up all the buffers and TRBs at the same time (the host and device software negotiates the transfer size), then just one Start Transfer is enough, and software waits for a `XferComplete` event.
- If software did not set up all the TRBs, then every time software adds a new TRB (by setting `HWO=1`) it must issue an Update Transfer command. The hardware fetches the TRB again and continues the transfer.
- The End Transfer command is used only during error conditions and not used during normal transfers. For example, if software has set up multiple transfers and if a USB Reset event happens, it must remove all the transfers from the queue using End Transfer with the `ForceRM` bit set 1.
- A transfer is completed when all the data has been transferred or a short packet has been received. If software is queuing in only single transfers at a time (normal method for mass storage – for Ethernet over USB, software can set up multiple transfers using the Chain bit and Continue on Short Packet bit), once XferComplete interrupt has happened, the transfer is completed. When starting a new transfer, software now needs to issue the Start Transfer command.
- An OUT transfer's transfer size (total TRB buffer allocation) must be a multiple of `MaxPacketSize` even if software is expecting a fixed non-multiple of `MaxPacketSize` transfer from the Host.

  The following is the usage flow when the device does not know the transfer size and the host is not expected to send short packets to end the transfer. Note that this is not a normal scenario, but some device applications require this flow.

  a. The device sets up the TRB(s) with the maximum transfer size expected, and enables all of them (`HWO=1`).
  b. The host sends the OUT transactions, and completes the transfer without sending any short or zero-length packet.
  c. The device controller keeps updating the TRBs as required but does not give `XferComplete` event.
  d. The device software implements a timer to count inactivity on the OUT endpoint to detect the end of transfer.
  e. When the timer expires, software issues End Transfer command with `ForceRM=1`, and waits for the command to complete.
  f. Software reads the RAM0 cache (stream scratchpad bytecount location) through the slave interface to know the remaining byte count and calculates the received byte count.

g. Software calculates the byte address using the following formula:

```
dut_bytecount_addr = dut_ram0_base + `DWC_USB3_ADDR_CACHE_STREAMS +
(stream_id *(32 + (`DWC_USB3_CACHE_TRBS_PER_TRANSFER * 16))) + (7*4);
```

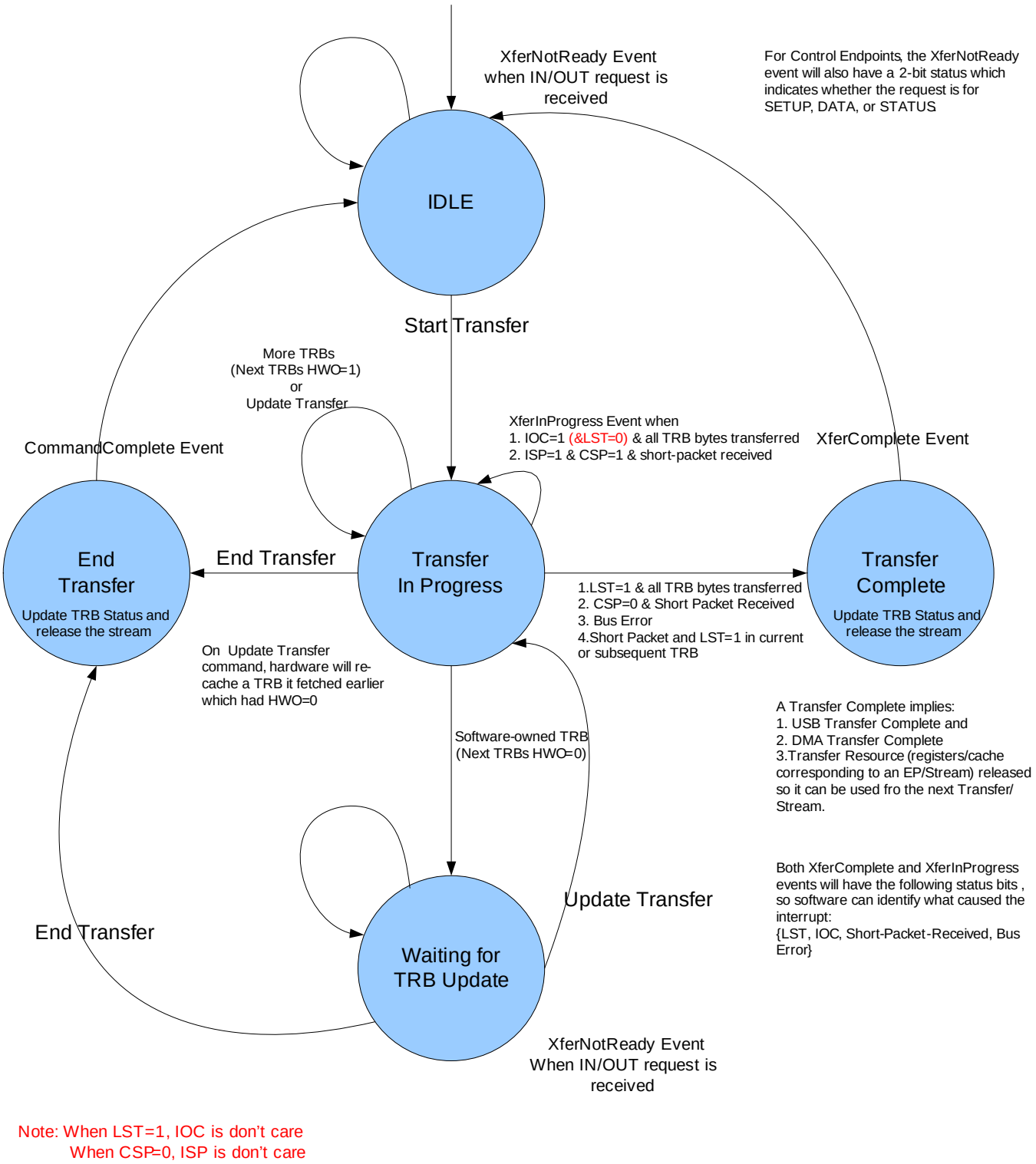
where, stream_id is the Transfer Resource Index (XferRscIdx) of the OUT endpoint, and

```
dut_ram0_base = `DUT_REGISTER_BASE_ADDRESS+32'h4_0000; // RAM0 starting
address
```

h. Software sets up the new transfer.

Figure 4-1 on page 349 displays the Transfer Usage State Machine.

Figure 4-1 Transfer Usage State Diagram



Note: When LST=1, IOC is don't care
When CSP=0, ISP is don't care

4.2.6 Transfer Descriptions

The following sections describe typical transfer processes.

4.2.6.1 Non-Isochronous OUT Transfers

The sequence below describes a regular, non-isochronous OUT transfer (control-data/bulk/interrupt).

Table 4-9 Non-Isochronous OUT Transfer Sequence (Continued)

Steps	Host	Device Controller	Device Software
Software sets up data buffers before a host request			
Step 1			Software sets up Normal TRBs and enables DMA by issuing Start Transfer.
Step 2		Device fetches the TRB and caches it.	
Step 3	Host sends data packets		
Step 4		Controller receives the packets in RxFIFO and stores them in to TRB data buffers through DMAs. While the RxFIFO has space and the BUS DMA keeps up with the host data transfers, the device receives all the packets. Once all the data is received, the controller updates the TRB status and then generates a XferComplete event. In SS mode, if GRXTHRCFG.USBRxPktCntSel is set to, ■ 0: the NumP sent depends on the DCFG.NumP and bMax-BurstSize ■ 1: the NumP sent depends on the RX FIFO space available when the packet is received for non-control OUT When DMA is slow or CRC fails: ■ If the RxFIFO overruns or CRC error is detected, device times out in Non-SS mode or asks for the same data packet (through ACK/SeqNumber/Retry Bit) in SS mode. ■ In HS mode, on NAK the host will issue PING and device will ACK it (if there is enough space in the RxFIFO) and the host resumes with data again. In SS mode, the host will send the requested data. If the device receives any packet with other than the requested data sequence, it will ignore it.	
Step 5			Software processes the XferComplete event.
Software sets up data buffers after the host request.			
Step 6	Host sends data packets.		

Steps	Host	Device Controller	Device Software
Step 7		Since DMA is not set up, the device will send NAK in Non-SS mode and will send NRDY in SS mode to the host. It also generates an XferNotReady event.	
Step 8			On seeing the XferNotReady event, Software sets up TRBs and enables DMA by issuing Start Transfer.
Step 9		Steps 2–5 are followed.	
Step 10		If the controller encounters a short packet before the transfer size data programmed in the TRB is received, it skips the remaining TRBs of the transfer. If the controller encounters control bits LST, IOC, or ISP it will generate events and, in the case of an LST TRB, the stream will be completed. Software has to reclaim any TRBs skipped with HWO=1.	On detecting a skipped TRB condition, software reclaims the TRBs with HWO=1.

4.2.6.2 Isochronous OUT Transfers

This section describes the difference between isochronous and non-isochronous OUT transfers. Referring to the sequence in "Non-Isochronous OUT Transfers" on page 350:

- In Step 2, if the TRB is not fetched in time, the controller discards the packet. In addition subsequent packets in the service interval also will be dropped.
- In Step 4, if DMA is enabled and if RxFIFO overrun happens or CRC fails, the controller discards the packet. In addition subsequent packets in the service interval also will be dropped.
- In Step 7, if DMA is not enabled, the controller drops the data and generates `XferNotReady` event

In general, the TRB chain of a service interval will be released if the controller receives DATA0 PID in HS USB 2.0 mode or receives a packet with LPF set in SuperSpeed mode.

Special scenarios are:

- If the controller misses this last packet indication, it releases TRB at the service interval boundary.
- If the controller encounters a short packet before the transfer size data programmed in TRB is received, it skips the remaining TRBs of the transfer.
- If the controller encounters control bits `LST` or `IOC` or `ISP` it will generate event and, in the case of `LST` TRB, the stream will be completed, but software has to reclaim TRBs if any TRBs skipped with `HWO=1`.

4.2.6.3 Non-Isochronous IN Transfers

The sequence in Table 4-10 describes a regular, non-isochronous IN data transfer (control-data/bulk/interrupt).

Table 4-10 Non-Isochronous IN Transfers (Continued)

Steps	Host	Device Controller	Device Software
	Software sets up data buffers before a host request		
Step 1			Software sets up TRBs and enables DMA by issuing Start Transfer.
Step 2		Device fetches the TRB, caches it, and prefetches the data into TxFIFO. Only in SS mode, the controller sends ERDY (Async ERDY, even before host requesting data provided endpoint is in flow controlled state. LSP flow is same regardless of if the host asked for it.)	
Step 3	Non-SS host issues a token request, SS host issues ACK TP to request data.		
Step 4		If DMA fetches keep up with host requests, the device sends all the data, updates the TRB status, and then generates an XferComplete event when ACKs for all the packets are received (see “DWC_usb3 Controller Actions Based on TRB Control Bits” on page 343). When DMA is slow: ■ If the TxFIFO becomes empty on a host data request, the device sends a NAK in Non-SS mode. In SS mode, the device sends NRDY, and when data is available in TxFIFO it sends ERDY and waits for the host request again. If the host retries a packet: ■ In Non-SS mode, a transmitted packet is kept in the TxFIFO until the ACK arrives (this is immediate in Non-SS mode). If the host retries a packet, it is immediately sent out from the TxFIFO.	
Step 4 Cont’d		■ In SS mode, before ACK can be received, the controller could have sent additional packets in a burst. When a retry happens, NRDY is sent to the host and the TxFIFO is flushed. A DMA command is issued on the master bus to prefetch data from where the host is asking. Once data is fetched into the TxFIFO, the device sends ERDY and waits for host request again. ■ In Threshold mode, when the TX threshold amount of data is fetched, the device sends ERDY to the host.	
Step 5			Software processes XferComplete event.

Steps	Host	Device Controller	Device Software
	Software sets up data buffers after a host request.		
Step 6	Non-SS host issues a token request. SS host issues ACK TP to request data.		
Step 7		Since DMA is not set up, the device sends s NAK in Non-SS mode, or NRDY in SS mode, to the host. The controller also generates an XferNotReady event	
Step 8			On seeing the XferNotReady event, software sets up Normal TRBs and enables DMA by issuing Start Transfer.
Step 9		Steps 2–5 are followed.	
Step 10		When software knows it is talking to a host that always ends transfers with short packets, and therefore always requires a zero-byte packet to end a transfer, it includes a zero-byte TRB (CHN=0) after the last TRB which is an even multiple of MaxPacketSize. When software does not know if the host is going to do another IN token after receiving the exact number of bytes it expected (and it was a multiple of MPS), it does not include a zero-byte TRB, and sets LST in the last TRB. If the host returns with another IN for a zero-byte packet, it uses the XferNotReady/Start Transfer mechanism to add one zero-byte TRB. This is called “On-Demand.”	

4.2.6.4 Isochronous IN Transfers

This section describes the difference between isochronous and non-isochronous IN transfers. Refer to “Non-Isochronous IN Transfers” on page 352:

- In Step 4, if the Tx FIFO is empty when host IN request arrives, the controller returns a zero length packet in both Non-SS and SS modes.
- In Step 7, the device returns zero length packet and generates an `XferNotReady` event.

Special scenarios:

- When data is fetched and there is no request from the host (possibly due to a missing request or corrupted request) the data will be flushed at the end of the service interval (corresponding Micro-Frame) boundary and the controller moves onto fetching next service interval data. If there are some TRBs that are not serviced in the service interval, the controller will skip these TRBs.
- If the controller encounters control bits `LST` or `IOC`, it will generate an event, and in the case of LST TRB, the stream will be completed, but has to reclaim TRBs if any TRBs skipped with `HWO=1`.

4.2.7 Handling ENDPOINT_HALT

On receiving a SetFeature (ENDPOINT_HALT) control transfer, software issues a Set Stall command on the endpoint. Because of the delay from when software issues the command and when it is recognized by the controller, other transfer events (XferInProgress, XferComplete) may be received prior to the endpoint being halted. For a description of the Set Stall endpoint command and its effects, see section "Commands 4 and 5: Set Stall and Clear Stall (DEPSSTALL, DEPCSTALL)" on page 320.

On ClearFeature (ENDPOINT_HALT), software must first remove all pending transfers for the endpoint through the End Transfer command. It may then issue a Clear Stall command on the endpoint followed by Start Transfer to start transfers again. For a description of the Clear Stall endpoint command and its effects, see section "Commands 4 and 5: Set Stall and Clear Stall (DEPSSTALL, DEPCSTALL)" on page 320.

4.2.8 Handling L1 Event During a Transfer

This section discusses the scenario in which the software already issued the Start Transfer command but the LPM occurred before or during the data transfer.

When an IN transfer is in progress, (i.e., started and not completed yet), and the software sees a link state change event to L1, then the software can use the `GDBGFIFOSPACE` register (write followed by a read) to determine if the TxFIFO is empty or not. If the TxFIFO is not empty, it can initiate a remote wakeup.

- The GDBGFIFOSPACE register is a debug register which gives the 'Space Available" for the endpoint selected with a write prior to the read. If this space available is less than the particular TxFIFO's maximum size, then the TxFIFO is not empty. For more details on GDBGFIFO-SPACE, refer to the Registers chapter.
- Software can enable the Link State change event for USB 2.0 as there are not a lot of events compared to SS.
- This remote wakeup by software is needed only when GUCTL1[DEV_L1_EXIT_BY_HW] is not enabled.

4.3 Isochronous Transfer Programming Model

This section provides details about the programming model of isochronous transfers.

4.3.1 Overview

Isochronous endpoints get guaranteed service by the host during a “Service Interval”. The service interval for an endpoint is communicated to the host via the endpoint descriptor, and it is configured in the hardware through the DEPCFG command. However, unexpected behavior on the bus may prevent the host from servicing the endpoint as frequently as expected.

Although the endpoint receives “guaranteed” service during the interval, there are no bus-level acknowledgments, which means that there is no guarantee that the host (IN endpoints) or the device (OUT endpoints) receives correct data.

Therefore, the isochronous programming model differs from the bulk programming model to accommodate these two factors:

- No bus-level acknowledgment of packet transmission or reception
- No guarantee that all the data setup for an interval has been transmitted or received

For isochronous endpoints, software has the following responsibilities:

- Set up enough TRBs to keep up with the rate of data transmission or reception
- For FIFO-based IN endpoints, guarantee the validity of the TX data at least 1 uF before the beginning of the interval that the data will be transmitted

Hardware has the following responsibilities:

- Transmit or receive data at the appropriate bus time
- For FIFO-based IN endpoints, delay fetching until 1 uF before the beginning of the interval that the data will be transmitted
- Maintain a 1-to-1 correspondence between Buffer Descriptors and the intervals on the bus
- Report missed isoc intervals and update the buffer sizes in TRBs to reflect the amount of data moved

4.3.2 Definitions

- bInterval: The field in an endpoint descriptor used to communicate the service interval of the device. Its value ranges from 1 to 16, however, the USB 3.0 controller supports a range of 1 to 14.
- Service Interval: An integral number of microframes (2^[bInterval-1]) during which the host will poll the device to move data.
- Beginning of interval: The interval begins when the least significant (bInterval-1) bits of the bus time are all 0's.
- End of interval: The interval ends when the least significant (bInterval-1) bits of the bus time are all 1's.
- Data payload: The number of bytes to be moved during the service interval. It will be transmitted using MaxPacketSize packets.
- Buffer Descriptor: A set of one or more TRBs with CHN=1 and the last one having CHN=0. The group of TRBs represents the data buffers for one service interval.
- Last TRB of a Buffer Descriptor: The TRB within a Buffer Descriptor that has CHN=0. When an interval has completed, this TRB will contain the total remaining buffer size of the Buffer Descriptor and also the completion status of the interval.
- Microframe (uF): A unit of time on the USB that lasts 125us.
- Start of Frame (SOF): The beginning of a microframe.
- FIFO-Based Isoc IN: A sub-type of Isoc IN endpoints that has the characteristic of always assuming the HWO bit in a TRB is '1' and delays fetching of transmit data.
- Prefetch Delta: For FIFO-based IN endpoints, TX data may be prefetched as early as 1 uF before the beginning of a Service Interval, but never earlier.
- Retire a TRB: When HW sets the HWO bit to 0 in a TRB and writes it back.
- Retire a Buffer Descriptor: When HW retires at least the first and last TRB of a Buffer Descriptor.
  - Other TRBs of the Buffer Descriptor may not be retired if an interval was missed (IN or OUT) or if a short packet/last packet is received (OUT). In this case, software may reclaim those TRBs even though HWO=1.
- Missed Interval: When the device does not move all the data in an interval. This may occur when the host does not poll for all the data, or because of application-side delays that prevent all the data from being moved.
  - For IN endpoints, this occurs when the host does not request all the data prepared in the Buffer Descriptor, or not all the data is fetched from the system bus in time.
  - For OUT endpoints, this occurs when the host does not send a packet, a packet is dropped due to CRC error, or the system bus is too busy to accept the data.

4.3.3 Endpoint Configuration

An isochronous endpoint is setup in much the same way as a bulk endpoint, with the following exceptions:

- The bInterval_m1 value in DEPCFG Parameter 1 must be set to the value reported in the endpoint descriptor minus 1. When the controller is operating in Full Speed, bInterval_m1 must be set to 0.
- The MaxPacketSize value in DEPCFG Parameter 0 must be set to the same value reported in the endpoint descriptor.
- The "FIFO-based" bit in DEPCFG Parameter 1, bit 31, must be set for FIFO-based isochronous endpoints (see "FIFO-based isochronous IN Endpoints" on page 362 and "FIFO-based isochronous OUT Endpoints" on page 363).
- For the best performance, set the TXFIFOs corresponding to Isochronous IN endpoints to a high priority.

After the endpoint is configured, it can be enabled by setting the appropriate bit in the DALEPENA register.

4.3.4 Transfer Configuration

Software describes isochronous transfers through a series of Buffer Descriptors (see "Transfer Setup Recommendations" on page 345 for a definition of Buffer Descriptors). Each Buffer Descriptor corresponds directly to one service interval of data. The fields within an isochronous TRB are the same as bulk TRBs with the following exceptions:

- The `SOF` field in an IN endpoint TRB is a don't care. For an OUT endpoint, the controller will write this field with the timestamp of the last packet received into the TRB's buffer. This information is not needed for normal operation, only for debug purposes.
- For High-Speed, High-Bandwidth IN endpoints, a maximum of three packets can be sent during an interval. The `PktCntM1` field ([25:24] of the third DWORD) must be set to the (number of packets in the Buffer Descriptor - 1). For example, if third packets are to be transmitted during the interval, this field must be set to 2. For Super-Speed and OUT endpoints, this field is a don't care.
- The first TRB in a Buffer Descriptor must have the `TRBCTL` field set to the "Isochronous-First" type while all others have this field set to "Isochronous".
- The ISP bit is renamed IMI (Interrupt on Missed Interval) and should be set if software wants to receive an XferInProgress event when at least one packet is missed within an interval.
- The `IMI` bit should be set to the same value in all TRBs of a Buffer Descriptor.
- The `CSP` bit must be set to 1 (short packets cause the hardware to move to the next Buffer Descriptor, they do not end an isochronous transfer).
- The `LST` bit should be set to 0 (isochronous transfers normally continue until the endpoint is removed entirely, at which time an End Transfer command is used to stop the transfer).

All other bits and fields (`IOC, HWO, CHN, BPTR, BUFSIZ`) retain the same behavior as they have for bulk endpoints. If software needs to receive an interrupt after every service interval, it should set the `IOC` bit to '1' in each TRB. However, if software wants to reduce the interrupt frequency and the application can tolerate some latency, the `IOC` bit can be set to '0' and the controller will not generate a XferInProgress event when the TRB is completed.

For OUT endpoints, the Buffer Descriptor must describe a total buffer size of:

```
(Mult+1)*MaxBurst*MaxPacketSize
```

4.3.5 Starting a Transfer

The hardware will report the bus time that the host starts polling the endpoint inside the XferNotReady event. Software will use this value as a reference when issuing the Start Transfer command to serve as time synchronization with the host.

1. Set up TRBs and data buffer for the endpoint.
2. After configuring and enabling the endpoint, wait for an XferNotReady event.

   The XferNotReady event will contain the time that the host started polling the endpoint in the upper 16 bits.
3. Issue the Start Transfer command to the endpoint with a future microframe time written into the upper 16 bits of the `DEPCMD` register.

   The future microframe time must be a value that is an integral multiple of intervals after the time reported in the XferNotReady event and aligned to the beginning of an interval. For example, if bInterval is 3 (4 microframes), and the XferNotReady time is 2, the value can be 4, 8, 12, and so on. The future microframe time must also be no greater than 4 seconds past the time reported in the XferNotReady event.
4. If the future microframe time has already passed when the command is received, the controller will respond with an error (bit 13 in the Command Complete event).

   In this case, software must issue End Transfer, then wait for another XferNotReady event and attempt the command again with a time that is further in the future.

   Otherwise, the first Buffer Descriptor will be used for the interval starting with the microframe time specified in the Start Transfer command.

4.3.6 Controller Behavior During an Interval

After a transfer has been started, the hardware will perform the following functions for IN endpoints:

1. Fetch TX data as early as one interval prior to the beginning of the interval (call it A) if the HWO bit is set to one in the TRB.
2. Decrement the buffer size of each TRB as packets are transmitted.
3. Retire TRBs when their buffer size has reached 0, issuing an XferInProgress event if the IOC bit is set.
4. If the next interval (B) starts before all the packets have been transmitted for interval A:
   a. Flush the TxFIFO.
   b. Retire the Buffer Descriptor of interval A with a “Missed Isochronous” status.
   c. Retire the Buffer Descriptor of interval B with a “Missed Isochronous” status.
   d. See “Checking Interval Status” on page 360 for a description of how software can determine that an interval ended unexpectedly.
   e. Go to step 1 to prepare for interval C
5. Otherwise, if all the TRBs of interval A are completed, the hardware will prepare for interval B.
6. If the host completes an interval by polling for all the data, but then it polls the endpoint again during the same interval, the hardware will respond with a zero-length packet and no interrupt will be made to software.
7. If the host polls the endpoint prior to the time specified in the Start Transfer command, the hardware will respond with a zero-length packet and no interrupt will be made to software.

8. If the host polls the endpoint during the expected interval and the hardware has no data prepared, the controller will respond with a zero-length packet, but no XferNotReady event will be generated because the transfer is active.

   When the next interval starts, the XferInProgress event is generated (based on ISP and IMI) with the "Missed Isochronous" status, which is the way software finds out that this occurred.

For OUT endpoints, the hardware performs the following functions:

1. If a packet is received for the correct interval represented by the Buffer Descriptor (call it A).
   a. Write the packet into the buffer.
   b. Decrement the buffer size of the TRB.
   c. Write the timestamp of the received packet into the TRB.
   d. Retire a TRB when its buffer size reaches 0, issuing an XferInProgress event if the `IOC` bit is set.
2. If a short packet or packet with the last packet flag (lpf) is received for the correct interval.
   a. Write the packet into the buffer.
   b. Retire the Buffer Descriptor of interval A with the `TRBSTS` set to 0.
   c. If the `IOC` bit is set in the last TRB of the Buffer Descriptor, issue an XferInProgress event with bit [13] set.
   d. Go to step 1 to prepare for interval B.
3. If a packet is received for the correct interval but it has an error (CRC error, RxFIFO overflow), the controller will not increment its expected sequence number value which causes future packets within the same interval to be dropped.
4. If a packet is received at a bus time prior to the time specified in the Start Transfer command, it will be dropped.
5. If a packet is received for the correct interval (A), but not enough buffer space is available for the controller to write the packet (due to the HWO bit still '0' in one or more of the TRBs of the Buffer Descriptor), no XferNotReady event will be generated. The controller will behave as follows:
   a. Wait until software sets the HWO bit to '1' and issues an Update Transfer command
   b. Retire the Buffer Descriptor of interval A with a "Missed Isochronous" status
   c. Go to step 1 to prepare for interval B.
6. If a packet is received for the next interval (B) before all the packets have been received for interval A:
   a. Retire the Buffer Descriptor of interval A with a "Missed Isochronous" status.
   b. Retire the Buffer Descriptor of interval B with a "Missed Isochronous" status.
   c. See the next section "Checking Interval Status" for a description of how software can determine that an interval ended unexpectedly.
   d. Go to step 1 to prepare for interval C.

Special Considerations for Isochronous OUT Endpoints

For OUT isochronous endpoints, the hardware detects a missed interval when the host sends a data packet in a future interval. It does not detect the missed interval at the exact interval boundary. Therefore, if the host abruptly stops sending isochronous OUT packets, there will be no interrupt or event indicating this to software. Software should use one of the following mechanisms to detect the interruption of isochronous OUT traffic:

1. The ultimate consumer of the isochronous OUT data will detect an underflow.
2. The device driver can use a timer detect that no XferInProgress events have been received for multiple intervals.

If this occurs, software should issue an End Transfer command to the endpoint and wait for a XferNotReady event which signals that the host is ready to resume isochronous traffic.

4.3.7 Checking Interval Status

As packets are transmitted or received during an interval, the buffer size of each TRB will be decremented. When the host is operating normally and polling for all the interval data, the buffer size of all the TRBs of an IN endpoint Buffer Descriptor will be zero after the interval has completed. For OUT endpoints, if the host sends less data than the Buffer Descriptor was setup for, the remaining buffer size may be greater than 0.

When the interval completes, the final remaining buffer size and missed isoc status is written to the last TRB of the Buffer Descriptor. If MissedIsoc is set, then it means the BUFSIZ is not accurate and may indicate that more data was transmitted or received than in reality. If the MissedIsoc is not set, it means the BUFSIZ field is correct.

When hardware detects the end of an interval (including normal and abnormal ends), it performs the following:

- If it has not already been retired, the first TRB of the Buffer Descriptor will be retired.
- Any non-first TRBs with CHN=1 that had not already been retired will not be written back.

  HWO will still be 1, and software can reclaim them for another transfer.
- The last TRB of the Buffer Descriptor will be retired with:
  - HWO = 0.
  - BUFSIZ = The total remaining buffer size of the Buffer Descriptor.
  - TRBSTS = “Missed Isoc” if any packets were missed, zero otherwise.
- If the IOC bit is set in the last TRB, or the IMI bit is set and packets were missed, hardware will issue an XferInProgress event

Software can get notification of an interval completing by setting the IOC bit in the last TRB of the Buffer Descriptor. The IOC bit may also be set in any other TRB of the Buffer Descriptor if software wants earlier notification that a TRB has completed.

Software can also get only a notification of an interval completing unexpectedly by setting the IMI (Interrupt on Missed Isoc) bit in the last TRB of the Buffer Descriptor. When an interval completes unexpectedly and either the IOC or IMI bit is set in the TRB, the MissedIsoc bit (15) of the XferInProgress event will be set. The following table shows which events are generated in each scenario:

Table 4-11 TRB Event Generation Events (Continued)

Scenario	IOC	IMI	Action
TRB completed with a MaxPacketSize packet	0	0	No interrupt
	0	1	No interrupt
	1	0	XferInProgress(IOC=1)
	1	1	XferInProgress(IOC=1)
TRB completed with a short packet	0	0	No interrupt
	0	1	No interrupt
	1	0	XferInProgress(Short=1,IOC=1)
	1	1	XferInProgress(Short=1,IOC=1)
Interval completed due to missed packet (missed isoc)	0	0	No interrupt
	0	1	XferInProgress(MissedIsoc=1,IOC=0)
	1	0	XferInProgress(MissedIsoc=1,IOC=1)
	1	1	XferInProgress(MissedIsoc=1,IOC=1)

It is normal to lose two intervals at a time when an error occurs during one interval. One interval is lost because of the host (which may not be polling enough) and the second interval is dropped because the device needs time to synchronize up to the next (third) interval.

4.3.8 Adding Intervals to a Transfer

Because isochronous endpoints represent a stream of data, the TRBs of an isochronous endpoint will normally be setup in a circular list, such as:

- TRB 1 (CHN=0, IOC=1)
- TRB 2 (CHN=0, IOC=1)
- Link (to TRB 1)

Example: Assume TRB 1 represents the data for the first interval and TRB 2 represents the data for the second interval. Because IOC=1, when the controller completes the first interval, it will issue an XferInProgress event which indicates to software that TRB 1 can be analyzed to retrieve the results from interval 1. TRB 1 can the be re-used to represent the data for the third interval by setting HWO=1 and issuing an Update Transfer command.

Each time software sets up TRBs for a new interval, it must follow the guidelines in the Transfer Configuration section. Software will set up TRBs for future intervals when it obtains those buffers from another software layer or when TRBs are retired by the controller.

To prevent hardware from getting stuck on a stale TRB transfer associated with an old micro-frame (hardware will be sending zero-length packets), software must set up at least two separate TRB transfers in a circular buffer such that if the micro-frame associated to that TRB transfer has already expired, the hardware must move on to the next TRB transfer.

4.3.9 Moderating Events

During a transfer, software may enable or disable any endpoint-specific event by re-issuing the DEPCFG command with the “Config Action” set to “Modify” set to ‘1’ and a modified DEPEVTEN field. All other field. All other fields in all other parameters must remain the same as the initial endpoint configuration.

4.3.10 Other Types of Isochronous Endpoints

The above description of the isochronous endpoint programming model assumes that software is setting up buffers in external memory that apply to certain intervals of data. This is called a “buffer-based” isochronous endpoint.

However, there is another mode that the hardware supports to accommodate other mechanisms of sourcing or sinking isochronous data, called “FIFO-based” and the types are defined by bits [31:30] of DEPCFG Parameter 1. Table 4-12 illustrates the differences between the models:

Table 4-12 Isochronous Endpoints Models

Type	DEPCFG Parameter 1[31] (FIFO-Based)	TRB Fetch	TRB Write Back	TX DMA	Retire Buffer Descriptor at Passed Interval
Buffer-based	0	When HWO=1	Yes	When HWO=1, no earlier than one interval ahead of time.	Yes
FIFO-based	1	When internal cache space available	No	No earlier than 1 µF before the interval	Yes

4.3.10.1 FIFO-based isochronous IN Endpoints

The software of some applications is unable to keep up with the short latency and high bandwidth of isochronous traffic and require the data to be sourced and done a sync through external FIFOs. One example is that there are two external TxFIFOs, one for each interval, and an address-to-FIFO-pop logic is implemented so that when the controller attempts to read from interval 1's address, the translation logic converts this into a pop signal for interval 1's TxFIFO. When the controller attempts to read from interval 2's address, the translation logic converts this into a pop signal for interval 2's TxFIFO.

To describe a FIFO-based implementation, software sets up an endpoint with the “FIFO-based” configuration bit set. This type of endpoint ignores the HWO bit in all TRBs, assuming that the TRB is always valid, and that the controller should never write it back. Software chooses the buffer pointers within the TRBs to correspond to the address needed by the translation logic to specify which FIFO should be popped. By also using the CHN bit, headers and payload can be concatenated from different FIFOs. For example, if there are 4 external FIFOs: HA, PA, HB, PB, where HA/HB contain the 4 byte header for 2 intervals and PA/PB contain the 512 byte payload for two intervals, this can be described by using the following five TRBs:

- BUFPTR=HA, IOC=0, CHN=1, BUFSIZ=4
- BUFPTR=PA, IOC=1, CHN=0, BUFSIZ=512
- BUFPTR=HB, IOC=0, CHN=1, BUFSIZ=4
- BUFPTR=PB, IOC=1, CHN=0, BUFSIZ=512
- Link to (1)

Every interval, the controller will be creating a 516 byte packet that consists of 4 bytes from the header FIFO and 512 bytes from the payload FIFO. However, if the host does not poll for the packet, the controller will skip 1 or 2 intervals of popping, depending on whether it has already started reading the next interval. External logic (or software) is responsible for flushing and refilling alternate FIFOs so that the controller is always reading the correct data for the next interval. In normal intervals, the FIFOs will be empty, but when an interval is missed, there may still be data present in the FIFOs.

Software Requirements:

- The "FIFO-based" bit must be set in the DEPCFG when configuring the endpoint.
- No field within any TRB may be changed after the Start Transfer command is issued.
- Data must be valid in the external FIFO at least 1 uF before the beginning of the interval for which the data is intended.

Controller Behavior:

- The earliest the controller will read from the FIFO will be 1 uF before the beginning of the interval for which the data is intended.
- The controller will not write back the TRB after it has completed it.
- The controller will re-read the TRB from external memory even though it is not allowed to change.
- The only indication of missed intervals is the XferInProgress event if IOC or IMI is set in the TRBs.

No indication will be made of how much data was actually transmitted.

4.3.10.2 FIFO-based isochronous OUT Endpoints

The FIFO-based model is also supported for OUT endpoints. In this case, controller writes to specific addresses will be translated into external FIFO pushes. The same example from above (4 byte headers and 512 byte payloads) can be applied to OUT endpoints where the headers are split from the payload as the controller receives 516 byte packets. The same software requirements and controller behavior apply (for example, no writebacks and no indication of how much data was received). External logic assumes that if less than the expected amount of data is pushed into an interval's FIFO, the interval is invalid.

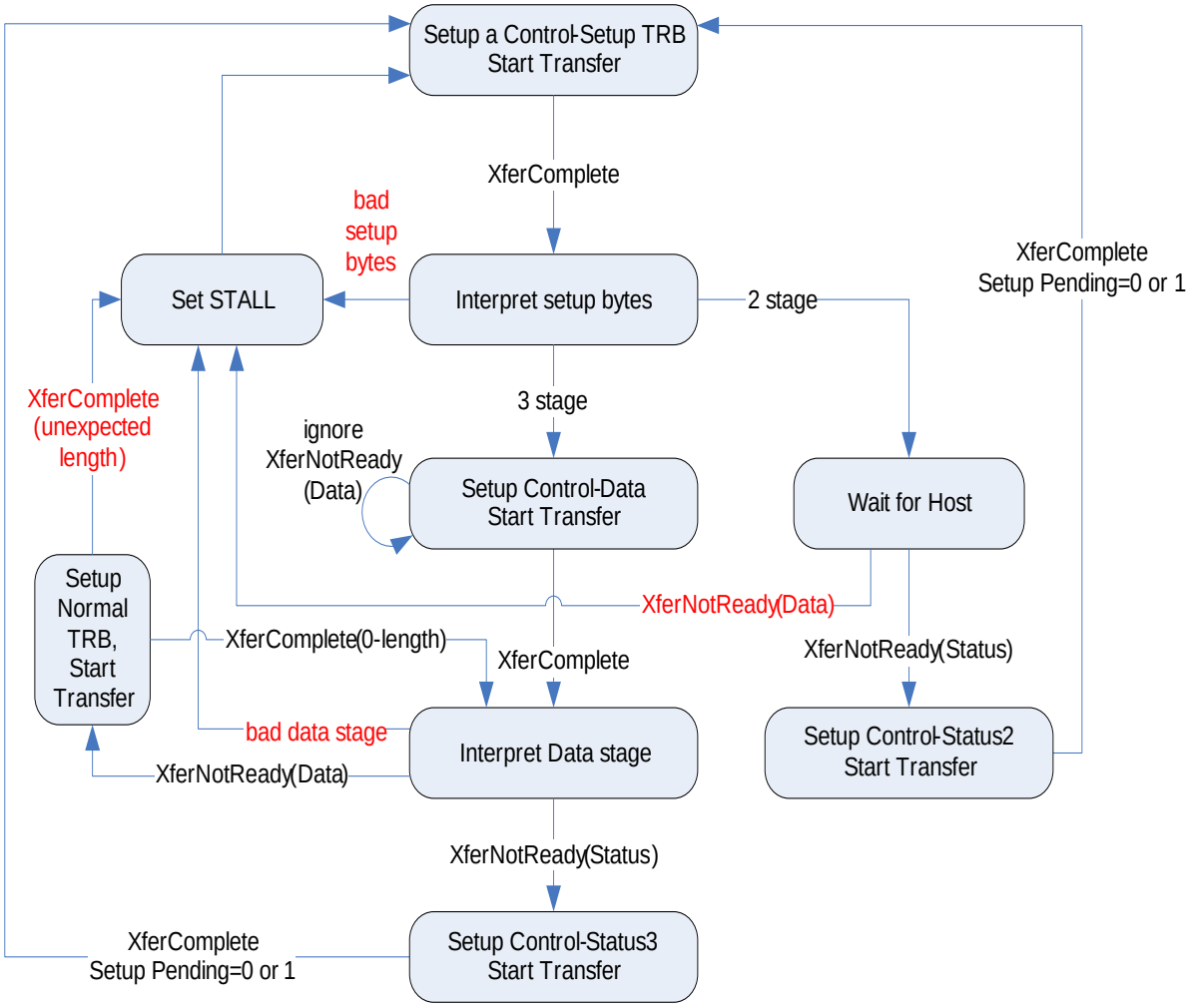
4.3.11 Ending a Transfer

Because isochronous endpoints have no bus-level acknowledgments, it is not necessary to set LST=1 in a TRB to gracefully end an isochronous transfer. Software can issue an End Transfer command to end an isochronous transfer. The controller will wait until it can complete operations for the endpoint before returning the Command Complete event in response.

4.4 Control Transfer Programming Model

Control transfers follow a flow of setup/data/status. On the USB bus, there are many error scenarios that can disrupt this flow. The hardware has a special mechanism that automatically recovers from these scenarios as long as software follows this single control transfer programming model. The following figure illustrates the required software flow for control transfers:

Figure 4-2 Software Flow for Control Transfers



If hardware does not receive a setup packet, it discards the unexpected data and status stages. It does not issue XferNotReady to the application. The hardware will also not generate a XferNotReady event if it receives a setup packet before a TRB is prepared for it.

As control endpoints are bi-directional, the IN and OUT directions of the endpoint are used for the different stages as follows:

Control read:

- EP0 - SETUP OUT
- EP1 - Data IN
- EP0 - STATUS OUT/STATUS TP

Control write or two-stage transfer:

- EP0 - SETUP OUT
- EP0 - Data OUT (if present)
- EP1 - STATUS IN/STATUS TP

Set STALL is always issued on EP0, and the XferComplete/XferNotReady events are generated on the correct direction.

For other control endpoints, substitute “EP0” with the OUT direction (2, 4, 6, etc.) and “EP1” with the IN direction (3, 5, 7, and so on).

The same programming model is used for USB 2.0 and USB 3.0 control transfers.

4.4.1 Two-Stage Control Transfer Programming Model

1. Software sets up a Setup TRB and issues Start Transfer on EP0 pointing to the Setup TRB.
2. After XferComplete is received, software interprets the setup bytes. If they are bad, issue Set Stall on EP0 and go back to Step 1.

   If a XferNotReady (Data/Status) event is received before the XferComplete event for the Setup stage, issue Set Stall. This is an error case where the host is attempting to move data or start the status stage for a previous control transfer that has already completed.
3. Wait for an XferNotReady event for the Status stage which will occur on EP1. If an XferNotReady event for the Data stage is received (either direction), issue Set Stall on EP0 and go back to Step 1. This is an error case where the host is attempting to start the data stage when the setup bytes did not indicate a data stage was present.
4. After XferNotReady (Status) is received, start the status stage by issuing Start Transfer on EP1 pointing to a valid Status-2 TRB.
5. After XferComplete is received on EP1, go back to Step 1.

   Although not required, software may look at the Status TRB and examine the “Setup Pending” bit in the TRB status field. If it is set, it means that this control transfer was aborted on the USB bus and that the host did not receive the Status stage ACK.

4.4.2 Three-Stage Control Transfer Programming Model

1. Software sets up a Setup TRB and issues Start Transfer on EP0 pointing to the Setup TRB.
2. After XferComplete is received, software interprets the setup bytes. If they are bad, issue Set Stall on EP0 and go back to Step 1.

   If a XferNotReady (Data/Status) event is received before the XferComplete event for the Setup stage, issue Set Stall. This is an error case where the host is attempting to move data or start the status stage for a previous control transfer that has already completed.
3. Start the data stage by issuing Start Transfer on EP0 (control write) or EP1 (control read) pointing to a valid Control-Data TRB.
   a. If an XferNotReady (Data) event is received for the incorrect direction, software must issue an End Transfer for the data stage it has already started, then issue Set Stall. This is an error case where the host is attempting to move data in the wrong direction.
   b. If a XferNotReady (Data) event is received for the correct direction, ignore the event and continue to wait for a XferComplete event.

4. After the XferComplete event is received, software waits for an XferNotReady event.

   Although not required, software may look at the completed Data TRB(s) and examine the “Setup Pending” bit in the TRB status field and the BUFSIZ. If the Setup Pending bit is set or the BUFSIZ is non-zero, it means that this control transfer was aborted on the USB bus and that the host did not complete the data stage.
5. If a XferNotReady (Data) event is received after the XferComplete for the Data stage, it could mean one of two things:

   a. This host is trying to complete the data stage by moving a 0-length packet. This can occur if the data stage was an exact multiple of max packet size. If this is the case, software sets up an extra TRB (with a BUFSIZ of max packet size for control writes, or 0 for control reads), issues Start Transfer, and goes back to step 4.

   b. This host is trying to move more data than specified in the wLength field of the setup bytes. In this case, software issues Set Stall on EP0 and goes back to Step 1.
6. When the XferNotReady (Status) event is received, software interprets the data stage (e.g. number of bytes transferred, content of the control write buffer) and decides if the data stage was successful. If not, it issues Set Stall on EP0 and goes back to Step 1.
7. Start the status stage by issuing Start Transfer on EP1 (control write) or EP0 (control read), pointing to a valid Status-3 TRB.
8. After XferComplete is received, go back to Step 1.

   Although not required, software may look at the Status TRB and examine the “Setup Pending” bit in the TRB status field. If it is set, it means that this control transfer was aborted on the USB bus and that the host did not actually receive the Status stage ACK.

When device software sets STALL with the Set Stall command, it is possible that this command can be delayed on the SoC bus due to latency or the SoC bus servicing other agents. In this case, software can receive an XferNotReady event if the controller detects no TRB active before it detected the Set Stall. Software must service these events knowing that it stalled the control command.

4.4.3 Handling Fewer Requests than wLength

In a control IN transfer, if the host asks for less data than wLength, the controller skips the TRBs of the Data stage and generates an XferComplete event after encountering the TRB that has LST=1. Hardware will set HWO=0 in those TRBs that it skipped and in the last TRB.

In a control OUT transfer, if the host sends less data than wLength, the controller treats this like a short packet received into a TRB with CSP=0. It will immediately generate a XferComplete event, not setting the HWO bit in the remaining TRBs to ‘0’. Software will need to reclaim those TRBs that still have HWO=1.

4.4.4 Control OUT Transfer Examples

The sequence in the following table describes two- and three-stage control OUT transfers.

Table 4-13 Control OUT Transfer

Steps	Host	Device Controller	Device Software
	Setup stage		
1			Software sets up the Setup TRB and enables DMA by issuing Start Transfer to EP0 (endpoint 0, OUT direction).
2		Device fetches the TRB and caches it.	
3	Host sends SETUP packet		
4		SETUP packet is received in the Rx FIFO. There are 24 bytes allocated for each control endpoint, which guarantees reception of the three back-to-back SETUP packets. An ACK handshake is returned in non-SS mode. An ACK TP with NumP=1 is returned in SS mode. After SETUP, bytes are transferred through the DMA, and the controller issues an XferComplete event.	
5		If the host starts a Data/Status stage, the controller returns NAK in Non-SS mode or NRDY TP in SS mode. If the SETUP request is invalid, controller returns STALL (when software sets STALL) in non-SS mode. In SS mode the controller returns ERDY, and when the Data/Status stage is received it returns STALL.	Software processes the XferComplete event and decodes the SETUP bytes. If a Data stage is required, it will set up one Control-Data TRB and possibly subsequent Normal TRBs, then issue Start Transfer to EP0 (endpoint 0, OUT direction). If a Data stage is not needed, it will skip to the Status stage. If the SETUP request is invalid, software instead sets the STALL bit by issuing Set Stall command for EP0 (OUT direction).
	Data stage:		
6		Device fetches the TRB and caches it. In SS mode only, the controller will send ERDY.	
7	Host sends data packets		

Steps	Host	Device Controller	Device Software
8		This step is similar to Step 4 of "Non-Isochronous OUT Transfers" on page 350. The NumP for control transfer will be 1. In addition, if another SETUP is received before the data transfer happens on USB, the controller will skip the data transfer that software sets up without waiting for it to happen on the USB bus, and send XferComplete (when LST=1 TRB encountered) with the SetupPending status bit set. Software has to reclaim the TRBs with HWO=1 in the skipped TRBs.	
9		If the host starts a Status stage, the controller returns NAK in Non-SS mode or NRDY TP in SS mode.	On seeing the event: ■ Software decodes the control write data. If the Data stage is OK, then it will wait for Xfer-NotReady with "Control Status Request" set, set up a Control-Status-3 TRB (0-bytes), and enable DMA by issuing Start Transfer to EP1 (endpoint 0, IN direction). ■ If the Data stage is not OK, then software sets STALL by issuing Set Stall command to EP0.
	Status stage:		
10		Device fetches the TRB and caches it. In SS mode only, the controller will send ERDY	Software sets up a Status stage TRB only after the Data stage of a three-stage control transfer is completed and an XferNotReady event occurred.
11	Host requests status packets		
12		In Non-SS mode, the device will send a zero-length packet if STALL is not set; otherwise, it will send STALL. In SS mode, if STALL is not set, then it sends ACK TP with NUMP=1 else it sends STALL. If another SETUP is received before the status transfer on USB, the controller will send XferComplete with the SetupPending status bit set. The STALL bit will be cleared by the controller whenever it receives a SETUP packet. SETUPs are always accepted.	
13			On seeing the event: ■ If there are no other errors, then the transfer has completed successfully.

4.4.5 Control IN Transfer Examples

The sequence in the following table describes two- and three-stage control IN transfers.

Table 4-14 Control IN Transfer

Steps	Host	Device Controller	Device Software
1–5	Setup stage: Steps 1–5 are same as "Control OUT Transfer Examples" on page 367, except that software sets up Tx buffers and Start Transfer is issued to EP1 (endpoint 0, IN direction).		
	Data stage:		
6		Device fetches the TRB and caches it. It prefetches the data and puts it in the TxFIFO. In SS mode only, the controller will send ERDY.	
7	Host requests data packets		
8		This step is similar to Step 4 of "Non-Isochronous IN Transfers" on page 352. The NumP for control transfer is 1. In addition, if another SETUP is received before the data transfer happens on USB, the controller will skip the data transfer software has set up without waiting for it to happen on the USB bus and send XferComplete (when LST=1 TRB encountered) with the SetupPending status bit set. Software has to reclaim the TRBs with HWO=1 in the skipped TRBs and flush the TxFIFO.	
9		If the host starts the Status stage, the controller returns NAK in Non-SS mode or NRDY TP in SS mode.	On seeing the event: ■ If the Data stage is OK, then it sets up Control-Status-3 TRB (zero-bytes) and enables DMA by issuing Start Transfer to EP0 (endpoint 0, OUT direction). ■ If the Data stage is not OK, then software will set a STALL by issuing Set Stall command to EP0 (OUT direction).
	Status stage: Same as in "Control OUT Transfer Examples" on page 367.		

4.5 Stream Handling in SuperSpeed

The SuperSpeed stream protocol consists of a negotiation between the host and device concerning the selection of a stream, followed by data movement on the selected stream. The controller automatically handles the stream protocol by initiating stream selection and also taking into account streams initiated by the host.

Note This section is not applicable for USB 2.0-only mode.

4.5.1 Stream IDs and Transfer Resources

A "Stream ID" is a 16-bit value which represents an individual flow of data between host and device. For stream-capable endpoints, the Stream ID must be non-zero and is used in:

- The Command Parameter field in the DEPCMD register used when issuing a Start Transfer command.
- The Stream ID field in a TRB.
- The EventParam field in an endpoint-specific event (DEPEVT): XferComplete, XferInProgress, XferNotReady, and StreamEvt.

You need to allocate 1 Transfer Resource per endpoint. The cost of each transfer resource is (32 + N_TRBS_PER_XFER * 16) bytes of memory in the descriptor cache, where DWC_USB3_CACHE_TRBS_PER_XFER is a coreConsultant parameter specifying the number of TRBs the controller caches per transfer.

After the Start Transfer command completes, the hardware returns the 7-bit Transfer Resource Index in the command complete event (and DEPCMD register). This index is used only for the Update Transfer and End Transfer commands. In all other places, the Stream ID is used.

4.5.2 Stream Selection and Stream Programming Model

Referring to the USB 3.0 Device Stream Protocol State Machine, the Idle state is the state where a device does not have a current stream to work on. When the hardware detects that it is in the Idle state, it will attempt to initiate and move data on a stream by transmitting ERDY to the host.

In order for this to occur, the endpoint must be "primed" and the stream must be both "active" and "ready."

- Primed: After a stream-capable endpoint is configured, it is in the "Disabled" state, which prevents the device from initiating any stream selection. After the host performs a Prime transaction, the endpoint is moved into the "Primed" state where it is eligible to initiate stream selection.
- Active: Software makes a stream active by issuing a Start Transfer command for the stream. A stream becomes inactive after an XferComplete event or after software issues an End Transfer command.
- Ready: After a stream is added via the Start Transfer command, its default state is "ready." The hardware will label the stream as "not ready" if the host gives a NoStream rejection to the hardware's attempt to initiate the stream. If the host performs a Prime, all currently active streams are automatically set to the "ready" state by hardware.

In the Idle state, the hardware selects a stream from its cache to initiate.

Once the controller has chosen an active stream, it attempts to initiate and move data on that stream until the endpoint enters the Idle state, which occurs when the host rejects the stream selection (NoStream) or the

stream is terminated (PP=0) by the host or device. Therefore, software is required to prepare enough HWO=1 TRBs for at least one packet prior to issuing the StartXfer command.

In device-oriented stream selection class drivers (such as UASP), the host will not reject the device's stream selection. However, other class drivers may be developed in which the host initiates stream selection. In this situation, software will not start any transfers and will wait for a XferNotReady event. When that event is received, software will look at the StreamID in the event and start the transfer associated with the given Stream ID. If for some reason the class driver allows the host to reject a stream that has already been initiated by the host or device, software will receive a Stream(NotFound) event. If this occurs, software may replace the existing stream by issuing End Transfer for it and then issuing Start Transfer for another stream.

Some host implementation scenarios exist where the host and device become out of sync in the stream protocol, creating a deadlock condition where the device waits for the host to issue a Prime transaction while the host waits for the device to issue an ERDY. To resolve potential deadlock conditions, software should perform the following:

1. Implement a timeout for stalled transfers on stream-capable endpoints.
2. When the timeout elapses with no data transfer for any stream after a StreamEvent (NotFound) event, restart one of the streams by issuing an End Transfer command followed by a Start Transfer command.

   This places the stream in the Ready state and causes the controller to transmit an ERDY to the host.

4.5.3 Data Movement Within a Stream

Data movement within a stream is the same as data movement using the Bulk endpoint type (Start Transfer, Update Transfer, and updating TRBs). The XferInProgress, XferComplete, and XferNotReady events are issued as data is transferred between host and device. The Stream ID field in the events indicate which stream within the endpoint is generating the events.

If software receives an XferNotReady event with the EventStatus field indicating “XferNotActive,” it means that the host is attempting to initiate a stream that has not been added to the hardware's cache through Start Transfer. In response to this event, software should add the requested stream if it is available.

4.6 Interrupts

Note

- This section is for applicable only for non-SSIC device configurations.
- For register descriptions, see “Register Descriptions” on page 17.
- For command and event buffer structure, see “Device Data Structures” on page 295.

The DWC_usb3 controller generates the interrupt[(DWC_USB3_NUM_INT-1):0] signal to indicate the occurrence of an event needing application intervention.

While configuring the controller using coreConsultant, you can select the number of Event Buffers by choosing the “Number of Device Mode Event Buffers” (DWC_USB3_NUM_INT). Each Event Buffer is associated with one interrupt line.

Software allocates memory for the Event Buffers, to receive interrupts, using the GEVNTADR and GEVNTSIZ registers. The GEVNTCOUNT register holds the number of valid bytes in the Event Buffer.

When an event occurs within the controller, the hardware checks the enable bit that corresponds with the event to determine if the event needs to be generated. If it is enabled, the controller writes the event to the Event Buffer, increments the GEVNTCOUNT[15:0] (Event Count) register field in the GEVNTCOUNT register, and generates the hardware interrupt (if not masked and is the correct time to assert the interrupt based on the DEV_IMOD register).

For information on the structure of the Event Buffers, see “Programming DWC_usb3 in Device Mode” on page 333“Device Event Buffer Structure” on page 882. By reading the Event Buffer, software can decode the type of event that occurred. For information on how to use multiple interrupts, see “Multiple Device Interrupt Support” on page 375.

The DWC_usb3 controller generates following types of interrupt events:

- “Command Interrupt Event” on page 373
- “Device-Specific Interrupt Event” on page 373
- “Device Physical Endpoint-Specific Interrupt Event” on page 373
- “Interrupt Based on TRB Settings” on page 374
- “Device Interrupt (GSTS[6])” on page 374
- “Battery Charger Interrupt Event” on page 374

4.6.1 Command Interrupt Event

Command interrupt event are generated based on completion of device generic commands or endpoint-specific commands. For more information on these two types of commands, see "Device Command Structure" on page 313.

Based on whether the command is issued to the generic device or a specific endpoint, there are two types of command interrupt events:

- Device generic interrupt events

  If you set DGCMD[8] (CMDIOC) register bit while issuing a device generic command, you get Generic Command Complete event for that command.

  In the Event Buffer, this event shows up with DEVT[11:8] bits set to 10 (Refer to Table 3-8 on page 329).

- Device endpoint-specific interrupt events

  If you set DEPCMD[8] (CMDIOC) register bit while issuing a device endpoint-specific command, you get Endpoint Command Complete event for that command.

  In the Event Buffer, this event shows up with DEPEVT[9:6] bits set to 7 (Refer to Table 3-7 on page 326).

4.6.2 Device-Specific Interrupt Event

You can use DEVTEN register fields to enable/disable the device-specific events. (Refer to the "DEVTEN" section of the Registers chapter).

If the last bit of the event in the Event Buffer is 1, it indicates a device-specific event. Bits [11:8] indicate the type of device-specific event that occurred. For different types of device-specific events, see bits [11:8] in Table 3-8 on page 329.

4.6.3 Device Physical Endpoint-Specific Interrupt Event

Unlike the case of device-specific commands, for device endpoint-specific commands, you need to issue a DEPCFG command with field [13:8] (DEPEVTEN) set to enable/disable the device endpoint-specific events (Refer bits [13:8] in Table 3-3 on page 316).

If the last bit of the event in the Event Buffer is 0, it indicates a device endpoint-specific event. Bits [9:6] indicate the type of endpoint-specific event that occurred. For the different types of device endpoint-specific events, see Table 3-7 on page 326.

4.6.4 Interrupt Based on TRB Settings

Based on the TRB control fields (DW 0F-0C), the DWC_usb3 controller generates events, and writes them to the Event Buffer. Note that you must have enabled the required events using the DEPCFG command prior to this step.

For example, when you set the Interrupt on Complete (IOC) bit in a TRB, once the transfer indicated by that TRB (buffer) is completed, the controller issues XferInProgress event with IOC bit set in the event status field of the DEPEVT event. This indicates the buffer is available for software to reuse or release. This helps in freeing up the buffer sooner than waiting for the entire transfer to complete. Note that you must have set XferInProgEn to 1 using DEPCFG command for that specific endpoint for XferInProgress event to occur.

For more information, see the following sections:

- "Device Descriptor Structures" on page 296
- "Operational Model" on page 340
- "Isochronous Transfer Programming Model" on page 355
- "Stream Handling in SuperSpeed" on page 370

4.6.5 Device Interrupt (GSTS[6])

The GSTS[6] (Device_IP) field indicates that there is a pending interrupt pertaining to the device operation in the Device event queue. This can be used to know if any interrupt is pending before halting the controller.

4.6.6 Battery Charger Interrupt Event

Battery charger events are captured in the BCEVT register if the corresponding BC event enable bits in the BCEVTEN register is set accordingly.

The GSTS[9] (BC_IP) register bit indicates that there is a pending interrupt pertaining to BC in the BCEVT register.

The DWC_usb3 controller also drives the bc_interrupt signal to indicate a BC event has occurred and it needs application intervention. This signal is driven high as long as any event is set in BCEVT register. Once the application services the interrupt/event, it can clear it by writing a 1 to the corresponding event bit in the BCEVT register.

Unlike other interrupt events, all battery charger-related events (those captured in the BCEVT register) are not mapped to the "interrupt" output signal. They are mapped to a separate interrupt line called "bc_interrupt".

For more information, see "Battery Charger Operation" section in the databook.

4.7 Multiple Device Interrupt Support

The DWC_USB3_DEVICE_NUM_INT parameter allows you to instantiate up to 32 interrupt lines. Each interrupt line is associated with an event buffer described by the GEVNT*n registers. The interrupt number and the event buffer number are the same. When the endpoint is configured with the DEPCFG command, it is mapped to a specific interrupt such that all endpoint-specific events are written to the event buffer selected.

One time static interrupt mapping during the device endpoint configuration is recommended. In some circumstances, software may want to change the interrupt number of an endpoint between transfers. For example, the Bulk Only Transport (BOT) class driver uses a single endpoint for command (CBW), Data IN/OUT, and status (CSW), and you may be interested in routing CBW and CSW to one interrupt and Data IN/OUT to another interrupt. Software can issue a DEPCFG command with the “Config Action” set to “Modify” to change the IntrNum field without affecting the flow control status of the endpoint. This ability to dynamically map interrupt comes with the following restrictions:

- The XferNotReady event must always be disabled on the endpoint
- The interrupt number can only be changed on Non-Stream Bulk endpoints
- The interrupt number can only be changed between transfers (i.e. after a XferComplete event or before issuing a StartXfer command)
- The EPCmdComplete event for the DEPCFG command changing the interrupt number will come on the new interrupt number
- All other fields of the DEPCFG command must match the original configuration of the endpoint, except the event enable bits

4.8 Worst-Case Response Time

The DWC_usb3 controller, as a high-speed device, needs a minimum of three PHY (UTMI/ULPI) clocks to complete processing of the OUT data packet. Therefore, it cannot accept new token for these three clocks. However, for an isochronous transaction, there is no handshake and the next token could come within three PHY clocks after the previous RxActive de-assertion[1]. If this worst-case condition occurs, then the controller sends NAK for bulk and interrupt tokens and drops isochronous and SETUP tokens. The host handles this as a timeout condition for SETUP packet and retries the SETUP packet, but the ISOC OUT token is dropped.

1. This can happen theoretically when operating in UTMI 16-bit mode only with worst-case timing/delays with five hubs/cables. The host provides a minimum of 88 HS bit-times (typically, more than that) of Inter Packet Gap between packets and generally will not result in less than three clocks on the device-end UTMI PHY.

4.9 Low Power Operation

This section discusses the low power operation of the DWC_usb3 controller.

4.9.1 Low Power Operation of USB

The USB 3.0 protocol allows a SuperSpeed host and device to negotiate with each other and decide when to bring their link into a lower power state (U1 or U2). This is accomplished by each side making individual decisions about whether they have any traffic currently pending or if they expect any traffic soon. The device requests low power entry by transmitting an LGO_U1 (or LGO_U2) Link Command and exits low power by using an LFPS handshake.

The Synopsys USB 3.0 Device Controller uses information from the host (bus side) and from software (application side) to decide when the best time to request low power entry is and when to exit from low power. The same calculation is used when the controller decides whether to accept a low power request from the host. The controller maintains state information about each endpoint and whether it is “active” or “paused”. If all enabled endpoints are paused from either the application side or the bus side, low power entry is allowed. If at least one endpoint is not paused, low power entry is not allowed.

To indicate that it does not have any pending traffic and allow the link to go into a low power state, the software may pause all enabled endpoints using one of the following mechanisms:

- Allow transfers to complete and not start any new ones
- Not setting the HWO bit in the TRBs to ‘1’

Exception: Software may prepare a Setup TRB for control endpoints without affecting the ability of the link going into low power

Otherwise, an endpoint is considered active on the application side. For these endpoints, the host indicates that it is paused in one of the following ways:

- Setting the Packet Pending (PP) flag to ‘0’ in the DPH
- Setting the PP flag to ‘0’ and NumP to ‘0’ in the ACK TP
- For interrupt endpoints, by not polling the endpoint for two service intervals
- Note: After an endpoint is configured using the DEPCFG command and before the host sends any packets to it, the endpoint is considered to be paused on the bus side

Exception: If the data packet payload has a CRC error, the controller responds with an ACK (Retry=1), and the endpoint is not paused.

Otherwise, if PP=1 or Deferred=1, the endpoint is considered to be active on the bus side.

Control endpoints have an additional condition as follows:

The controller does not allow low power entry during a control transfer (starting from the reception of setup packet and ending at the completion of the status stage).

If the controller is configured for isochronous support, the following additional rules apply:

- A PING causes every isochronous endpoint to wake up and be considered active, preventing low power entry.
- An isochronous endpoint is paused automatically when it transmits or receives the last packet of its interval (lpf=1) or if two intervals pass and the endpoint has not transmitted or received its last packet.

In this way, after the PING, the controller does not enter low power until all isochronous endpoints have moved their last packet or two intervals pass without another PING.

If all enabled endpoints are paused by software or the host, the controller issues an LGO_U1 (or LGO_U2) link command or it responds with an LAU link command to the low power request of the host or hub. Otherwise, the controller rejects the low power requests with an LXU link command.

Other related information:

- For bulk endpoints, the USB 3.0 specification says that the device may go to U1/U2 500ms (tERDY-Timeout) after transmitting ERDY. It is the responsibility of the software to detect no activity on a bulk endpoint for 500ms or more and issue End Transfer if there is no activity. When the transfer is removed, the endpoint allows U1/U2 entry again.
- Because U1/U2 entry is based on the presence of packets in the TX FIFO and not based on the TRBs that are available to the controller, it is possible that an IN endpoint responds with a NRDY, goes to U1, then U0, then transmits an ERDY if the timing and TX FIFO size works out such that the controller cannot keep at least one packet in the TX FIFO at all times.
- In addition, before the final decision made to either initiate or accept low power state, device checks the state of DCTL.InitU1Ena/RejectU1Dis/initU2Ena/RejectU2Dis. If the device receives Set Link Function LMP with Force_LinkPM_Accept bit asserted, the device accepts low power request from its link partner independent of endpoints state and DCTL Power control settings.

4.9.2 Low Power Operation of Controller

When the link is in the U1, U2, U3, SS.Inactive, SS.Rxdetect, or SS.Disabled state (for USB 3.0) or the USB is in Sleep, Suspend, or Vbus-off mode (for USB 2.0), the controller can be put into a low power mode to save power.

The controller supports the following low-power modes:

- Clock-Gating mode: In this mode, the clocks connected to most of the controller modules are gated off. Modules that still get clocks in the low power mode detect wakeup conditions and remove clock gating to other modules when a wakeup condition is detected. Clock-Gating mode can be used when the USB is in any low power state. Clock-Gating mode is enabled by setting the Disable Clock Gating bit in GCTL to 0.the following low power mode:
- Hibernation: When this feature is enabled, the controller saves some of its internal state to external modules named “Power Management Units,” or PMUs, when directed by software and the customer-supplied power controller. For more information, see the following sections:
  - “Hibernation” in the databook
  - “Programming DWC_usb3 for Hibernation” on page 407

5

Programming DWC_usb3 in Host Mode

This chapter describes the programming requirements for the DWC_usb3 controller in host mode.

Note The DWC_usb3 controller in host mode is fully compliant with the xHCI Specification.

This chapter includes the following sections:

5.1 Initializing Host Registers

To initialize the controller as host, the application must perform the steps described in the xHCI Specification. If intermediate firmware needs to override a coreConsultant parameter or reset value, it can write to the controller-specific global (G*) registers at power-on.

Table 5-1 Overriding Global Registers in Host Configuration

Register	Description
GSBUSCFG0/1	Leave the default values if the correct power-on values were selected during coreConsultant configuration.
GTXTHRCFG/ GRXTHRCFG	This is required only if you are planning to enable thresholding. Leave the default values if the correct power-on values were selected during coreConsultant configuration.
GSNPSID	The software can read the Synopsys ID register to find the controller version and configure the firmware for any version-specific features.
GUID	Optionally, the software can program the User ID GUID register, if this register is selected for implementation in coreConsultant.
GUSB2PHYCFG	Program the following PHY configuration fields: USBTrdTim, FSIntf, PHYIf, TOUTCal, or leave the default values if the correct power-on values were selected during coreConsultant configuration.
GUSB3PIPECTL	Program the following PHY configuration fields: DatWidth, PrtOpDir, or leave the default values if the correct power-on values were selected during coreConsultant configuration.
GTXFIFOSIZn/ GRXFIFOSIZn	Program these registers based on the speed used for the FIFO. For details, see the “Architecture” chapter in the *DWC SuperSpeed USB 3.0 Controller Databook*. Unless the packet sizes of the endpoints are application-specific, it is recommended that you use the default value.
GCTL	Program this register to override scaledown, RAM clock select, and clock gating parameters.
GUCTL	If you want to improve the interoperability with different devices, program this register to override the behavior of the controller in Host mode.
DCTL	If you want to improve interoperability as a Debug Capability Target, program the InitU1/U2Ena and AcceptU1/U2Ena fields to '0' to disable U1/U2 entry.

5.2 Host Controller Capability Registers

For register definitions, refer to the host controller capability registers in “DWC_usb3_block_eXtensible_Host_Cntrl_Cap_Regs Registers” on page 26.

5.3 Host Programming Model

For more details on Host Programming Model, refer to the xHCI Specification.

5.4 xHCI Implementation Details

This section discusses xHCI implementation details specific to the DWC_usb3 controller.

5.4.1 LHCRST Behavior

The xHCI Specification does not describe the programming model of light reset. It only specifies that the Operational and Runtime Registers that are not contained in the Aux Power well are at their default values.

Light reset must only be applied when `USBCMD.Run/Stop` is equal to '0' to ensure that there is no discrepancy between hardware and software states. If light reset is applied during a transfer, the behavior on the USB is undefined, and may cause a packet to be terminated abruptly.

On light reset, the hardware resets all non-Aux registers to their default values which means that the port state (which is contained in `PORTSC`) does not change, but `USBCMD.Run/Stop` is immediately set to '0' and all the context information about connected devices is lost. It is the responsibility of software to re-initialize the controller.

5.4.2 ENT Requirements

In the xHCI Specification, the rules about when the ENT (Evaluate Next TRB) flag in a TRB must be set or cleared are not very strict. However, it states that the ENT flag must be set to '1' in the last Normal TRB when a TD ends with an Event Data TRB.

The controller requires software to follow these rules:

1. Regardless of whether the endpoint is stream-capable or not, if a TD ends with an Event Data TRB, the Normal TRB that precedes it must have ENT set to '1'. This Normal TRB may be separated from the Event Data TRB by a Link TRB
2. In all other cases, the ENT flag must be set to '0'

Failure to follow these rules results in undefined behavior.

5.4.3 Behavior on Babble Error

Section 4.10.2.4 of the xHCI Specification states “If a Babble Error is detected and the received data passes all integrity checks, the host controller may write the received data (up to the expected data length) to the data buffer, and the value of the TRB Transfer Length field in the Babble Detected Error Transfer Event shall be consistent with the number of data bytes written to the buffer.”

The controller does not write the received data to the data buffer. The entire packet is discarded and the residual byte count is written to the TRB Transfer Length field.

5.4.4 Max_exit_latency_too_large Message

If periodic transfers are on every microframe (Binterval 1), the controller reports `max_exit_latency_too_large` if PING scheduling is enabled. The xHCI scheduler must prefetch data from the system memory ahead of next microframe, manage non-periodic transfer, and schedule PING before ISOC. Because the system memory access and USB responses are not predictable, Quality of Service (QoS) is not guaranteed if the xHCI host needs to perform all of the above for every microframe when U1/U2 is enabled. Therefore, the expectation is to disable U1/U2 if periodic data needs to be scheduled every microframe. In addition, because the U1 exit latency is in the order of 10us, only very low system-level power saving is achieved even if the scheduler allows U1 transition by reducing QoS.

U1, U2, and U3 link states and exit time observed during lab testing:

- U1: Lowest power saving
  - In this state the SoC is active, USB controller is inactive, only the Tx transmitter of the PHY is inactive.
  - U1 exit time per link partner: ~10 - 19 us
- U2: Higher power saving than U1
  - In this state the SoC is active, USB controller is inactive, both the Rx and Tx transceivers of the PHY are inactive.
  - U2 exit time per link partner: ~85 - 115 us
- U3: Highest power saving
  - In this state, the SoC and USB controller/PHY are inactive.
  - U3 exit time per link partner: 200 - 450 us

The following are recommendations for the driver to handle periodic endpoints and the `max_exit_latency_too_large` message:

- If Binterval of any endpoint is '1', then disable U1/U2 and do not schedule PING (`max_exit_latency` set to 0).
- If periodic scheduling is expected on every microframe, then disable U1/U2 and do not schedule PING.
- If periodic scheduling is expected only on every other microframe, then disable U2 and use U1 exit latency for `max_exit_latency` calculation.
- If ((number of hubs +1) * U2 exit latency) >= (2**(Binterval-1) * 125 us), then disable U2.
- For all other periodic scheduling use U2 exit latency for `max_exit_latency` calculation.
  - If the `max_exit_latency_too_large` error is sent to software, then disable U2 and re-issue `ep_config` with U1 exit latency.
  - If `max_exit_latency_too_large` still happens, then disable U1 and re-issue `ep_config` with `max_exit_latency` set to 0 (no PING).

5.4.5 Disabled Checks

The xHCI specification requires the host controller to perform some checks to ensure that the xHCI driver does not inadvertently set a parameter to an incorrect value. However, some of those checks are incompatible with existing software and so they are currently disabled.

1. Disabled check for EP State in Input Context

   Section 4.6.6 “Configure Endpoint” in the *xHCI Specification* states:

   “else // Not all Input Endpoint Contexts identified by Add Context flag fields = ‘1’ are valid Completion Code =Parameter Error.”

The xHC must check that all referenced contexts are valid before executing the command. If aninvalid context is detected, the state of the Output Device Context shall not change and the a Command Completion Event is generated with the Completion Code set to Parameter Error."

   Section 6.2.3.2 “Configure Endpoint Command Usage” in the *xHCI Specification* states:

   “An Input Endpoint Context is considered “valid” by the Configure Endpoint Command if the Add Context flag is ‘1’ and: 1) the values of the Max Packet Size, Max Burst Size, and the Interval are considered within range for endpoint type and the speed of the device, 2) if MaxPStreams > 0, then the TR Dequeue Pointer field points to an array of valid Stream Contexts, or if MaxPStreams = 0, then the TR Dequeue Pointer field points to a Transfer Ring, 3) the EP State field = Disabled, and 4) all other fields are within their valid range of values.”

   Therefore, it suggests the controller must verify the EP State field is 'Disabled'. However, the xHCI CV fails on this check and so it is currently disabled.

2. Disabled check for DCS field in Input Context

   Section 4.6.5 “Address Device” in the *xHCI Specification* states:

   “The xHC must check that all referenced contexts are valid before executing the command. If an invalid context is detected, the state of the Output Device Context shall not change and the a Command Completion Event shall be generated with the Completion Code set to Parameter Error.”

   Section 6.2.3.1 “Address Device Command Usage” in the *xHCI Specification* states:

   “The Input Endpoint 0 Context is considered “valid” by the Address Device Command if: 1) the EP Type field = Control, 2) the values of the Max Packet Size, Max Burst Size, and the Interval are considered within range for endpoint type and the speed of the device, 3) the TR Dequeue Pointer field points to a valid Transfer Ring, 4) the DCS field = ‘1’, 5) the MaxPStreams field = ‘0’, and 6) all other fields are within the valid range of values.”

   Therefore, it suggests the controller must verify the DCS field is '1'. However, it was found in lab testing that some drivers do not guarantee DCS is always '1', so this check is disabled.

5.5 xHCI Debug Capability

Note

- The controller supports byte-aligned buffers for each TRB of a transfer. This feature prevents the need for buffer copying in cases where the USB device driver receives unaligned data buffers from another application (for example, Ethernet). Whenever the application controls buffer allocation, however, it must allocate SoC bus width and burst-aligned buffers to facilitate efficient bus and memory utilization.
- The Host controller does not support zero-length TRB in a Transfer Descriptor (TD) that has multiple chain TRBs. However, if the TD has only a single transfer TRB, then it is supports zero-length TRB.

The xHCI Debug Capability (DbC) allows one of the host ports to negotiate as an upstream (device) port. When this happens, the debug port is called the "Debug Target," and the connected host is called the "Debug Host."

Note

- To implement the xHCI Debug Capability, select "Yes" for "Enable xHCI Debug Capability?" in coreConsultant (parameter DWC_USB3_EN_DBC = 1)
- For details on the supported configurations and limitations, refer to "Supported Configurations" on page 400.

Section 7.6 of the xHCI Specification explains the functionality of the Debug Capability. However, it explains mostly the software interface, the operation of the DbC at the lower layers, and the handling of the control transfers. The operation of the bulk endpoints for a Debug Target is explained in the rest of the specification because the TRB structure, TD descriptions, Event Ring descriptions, and Endpoint Context are described in other parts of the xHCI Specification.

"Programming DWC_usb3 in Device Mode" on page 333 describes the operational model for a device. Even though the operational model for a device is not applicable to a Debug Target, the hardware actually performs some of the tasks that are normally performed by a device driver. For example, DbC hardware handles control transfers that are normally handled by a device driver.

The DbC feature merges the host (xHCI Specification) software-side programming model with the device USB-side operations. Within the DbC hardware, this translates to a logic that interprets the DbC/xHCI programming model at the upper layers and interacts with the lower layers in the same way the device hardware interacts.

This section discusses the following:

5.5.1 Debug Support in Multi-Port and One-Port Configurations

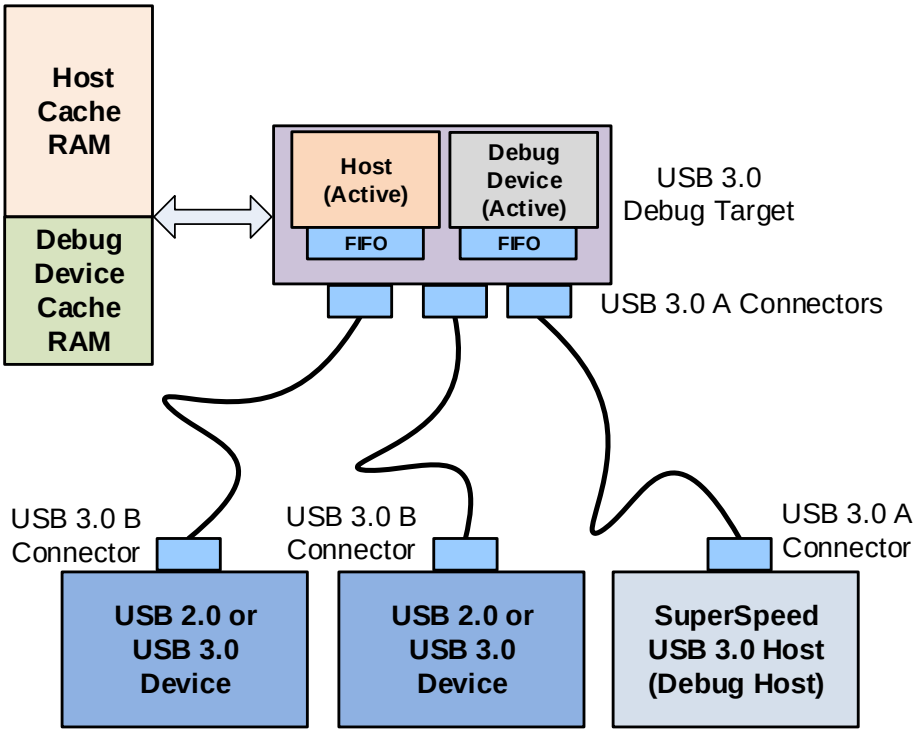
This section discusses the following topics:

- "Debug Support in a Multi-Port Configuration"
- "Debug Support in a One-Port Configuration Supporting Either Host or Debug Operation" on page 386
- "Debug Support in a One-port Configuration Supporting Concurrent Host and Debug Operation" on page 387

5.5.1.1 Debug Support in a Multi-Port Configuration

Figure 5-1 shows a multi-port xHCI USB 3.0 host controller in which any one of the ports can be used for debugging. A multi-port xHCI host controller supports concurrent host and debug operation.

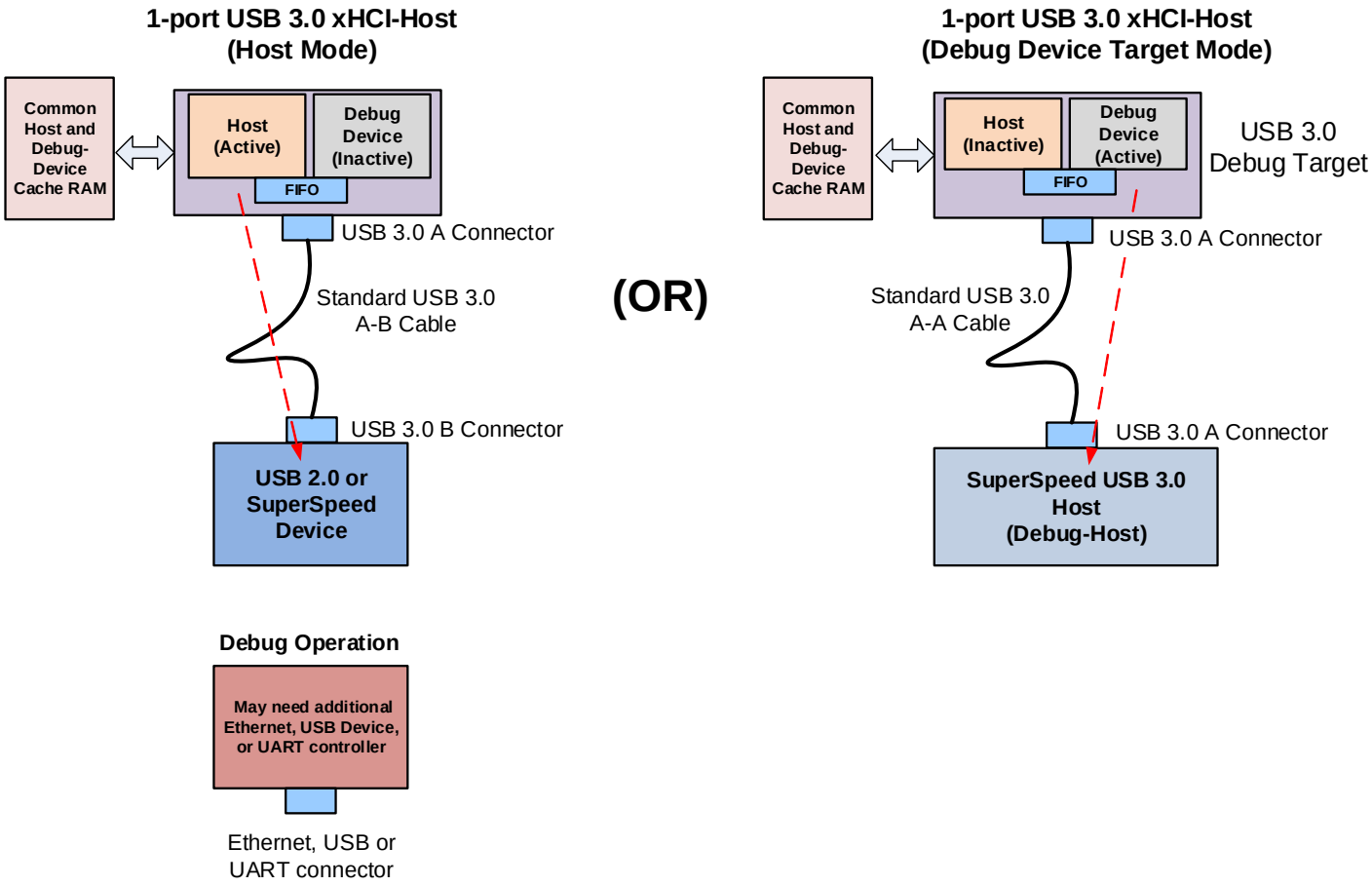
Figure 5-1 Multi-Port USB3.0 xHCI-Host (Host and Debug-Device Concurrent Mode)



5.5.1.2 Debug Support in a One-Port Configuration Supporting Either Host or Debug Operation

Figure 5-2 illustrates the operation of a normal one-port xHCI host controller that supports either host or debug operation at any given time. You may need an additional Ethernet, or UART, or USB device controller to support debug operation to support concurrent host and debug operation if you have only a one-port host controller. This adds additional development cost and silicon cost. Also, an additional Ethernet, UART, or USB connector is not desirable in small devices such as mobile phones.

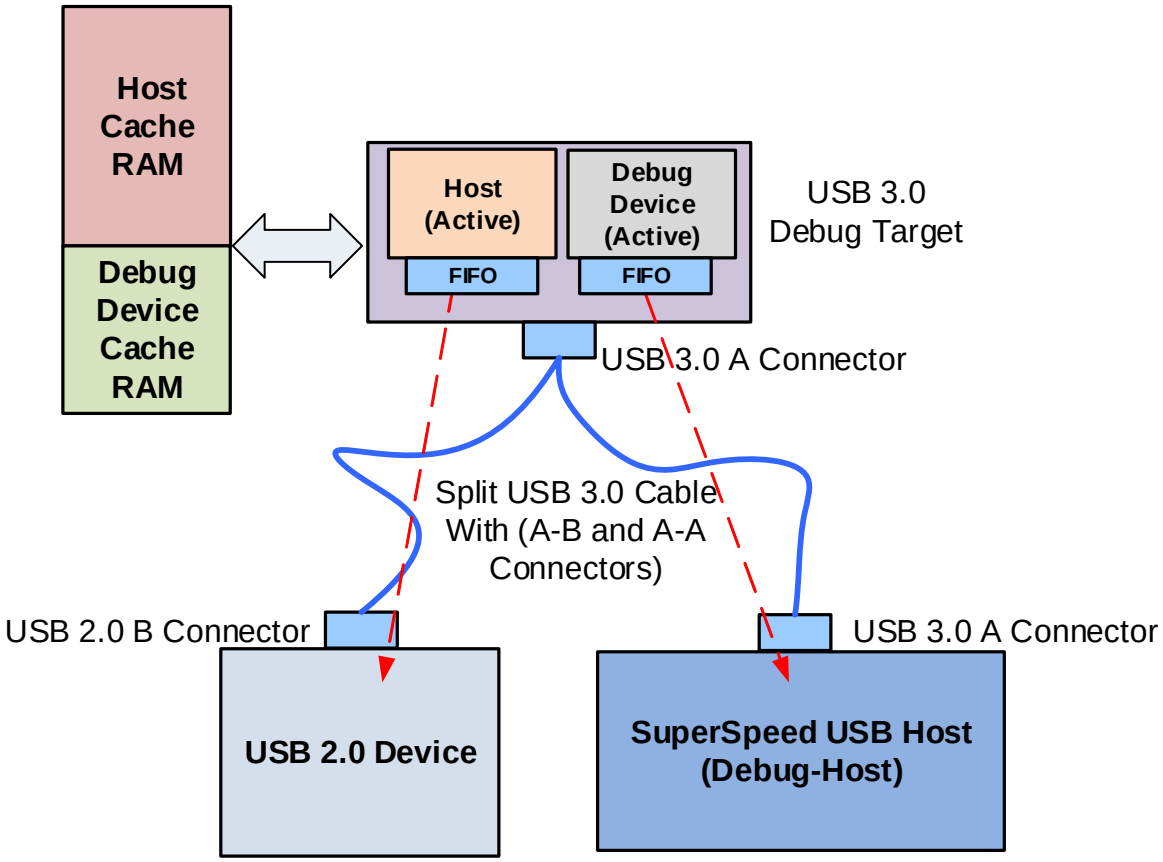
Figure 5-2 One-Port USB3.0 xHCI-Host (Host or Debug Mode)



5.5.1.3 Debug Support in a One-port Configuration Supporting Concurrent Host and Debug Operation

The Synopsys xHCI host controller supports concurrent host and debug operation even in a one-port configuration (patent pending). Figure 5-3 illustrates a one-port configuration in which a split USB cable mechanism is used for concurrent host and debug support. The Synopsys xHCI controller supports separate data paths (FIFOs and DMA) for host and debug operation even in a one-port configuration. Similarly, the cache memory also has separate host and debug data structures. The increase in area because of this additional logic is much smaller than adding a separate Ethernet, UART, or USB device controller.

Figure 5-3 One-Port USB 3.0 xHCI-Host (Host and Debug-Device Concurrent Mode) – Patent Pending



The split cable includes three plugs and two cables.

The details of the plugs are as follows:

- The first plug is a USB 3.0 A plug,
- The second plug is a USB 2.0 B plug, and
- The third plug is a USB 3.0 A plug

The details of the cables are as follows:

- The first cable couples the first and second plugs, and includes VBUS, D-, D+, and GND wires.
- The second cable couples the first and third plugs, and includes GND, StdA_SSRX-, StdA_SSRX+, GND_DRAIN, StdA_SSTX-, and StdA_SSTX+ wires.

The first plug is connected to the Synopsys xHCI host controller and the second plug is connected to a USB 2.0 or USB 3.0 device, and the third plug is connected to a USB 3.0 debug host. This provides concurrent USB 2.0 traffic between the Synopsys xHCI host controller and the USB device and SuperSpeed traffic between the Synopsys xHCI host controller and the USB 3.0 debug host.

5.5.2 Operational Model

The operational model of the Debug Capability is described in the xHCI Specification. However, further details about the actual implementation of the operational model and its affect on the hardware are detailed in this section.

5.5.2.1 Capability Discovery

The software discovers that the DWC_usb3 controller supports the Debug Capability feature by reading through the extended capabilities list. ID 10 is used to indicate that the Debug Capability extended feature is supported, and the software then knows the location of the DbC registers. It can then initialize the DbC registers as described in the xHCI Specification.

5.5.2.2 Port Training

The software enables the Debug Capability by setting the `DCCTRL.DCE` field to '1'. This causes the lower layer to advertise that it supports being an upstream port in its Port Capability LMP's. If there are multiple ports exposed, the controller does this across all of the USB 3.0 ports. If the user plugs a host (the Debug Host) into any of these ports, the Debug Host gets trained as a downstream port, and the port that is enabled for Debug Capability gets trained as an upstream port.

When the debug port is selected, the BMU reconfigures TX/RX FIFOs to match the sizes required by the Debug Capability. In addition, the TR switches to select the correct Bus Interfaces to connect the upper and lower layers.

When a port status change occurs in the `DCPORTSC` register for the debug port, a Port Status Change Event is written to the DbC Event Ring, and the `DCST.ER` field is set to '1' to indicate that an event is present.

5.5.2.3 Standard USB Request Handling

Once the port is trained as an upstream port, the Debug Host initializes the Debug Target (as it normally does for any other device) by issuing a USB reset, `SetAddress`, `GetDescriptor(s)`, and `SetConfiguration`. The Debug Capability LSP module handles each of the control transfers without involving the debug capability driver. After the `SetConfiguration` request, the Debug Target exposes EP1 with an IN and OUT endpoint. The response of the DWC_usb3 controller to each of the possible control transfers is discussed in “Control Transfers” on page 401.

5.5.2.4 Transfer Handling

The debug capability driver on the Debug Target prepares the TRBs for the OUT endpoint, expecting to receive commands from the host. It rings the "IN TR" doorbell (which corresponds to the OUT endpoint), and this enables the DWC_usb3 controller to receive packets on the OUT endpoint and generate Transfer Events. The software polls the `DCST.ER` field to determine whether there are events present in the DbC Event Ring. When it sees a transfer event, it knows that the buffers pointed at by the TRBs are now valid. It can interpret the commands sent, and prepare a data or status response on the IN endpoint.

For the IN endpoint, the debug capability driver prepares the TRBs on the "OUT TR", and then rings the OUT TR doorbell (which corresponds to the IN endpoint). The DWC_usb3 controller starts prefetching the data to be transmitted and transmits ERDY to the host if necessary (that is, if the endpoint is flow controlled on the USB). Retries are automatically handled by the controller which re-fetches the data to be transmitted. There is no communication to the debug capability driver if a retry happens on the USB.

The DWC_usb3 controller supports two transfer rings and assumes that only the following types of TRBs can be on the transfer rings:

- Normal
- Link
- No-Op
- Vendor-Defined: The cores ignores these TRBs as described in section 4.11.6 of the xHCI Specification

The controller generates a "TRB Error" transfer event for any other TRB type.

The controller inherits the following transfer behaviors from the xHCI operational model:

- Waits for a doorbell before fetching TRBs
- Generates Transfer Events when IOC=1
- Fast-forwards to the next TD when the OUT endpoint receives a short packet
- Skips and ignores Vendor-Defined TRBs

5.5.2.5 Endpoint Context State

Figure 5-4 shows the Endpoint Context state diagram as illustrated in section 4.8.3 of the xHCI Specification.

Figure 5-4 xHCI Endpoint State Transitions

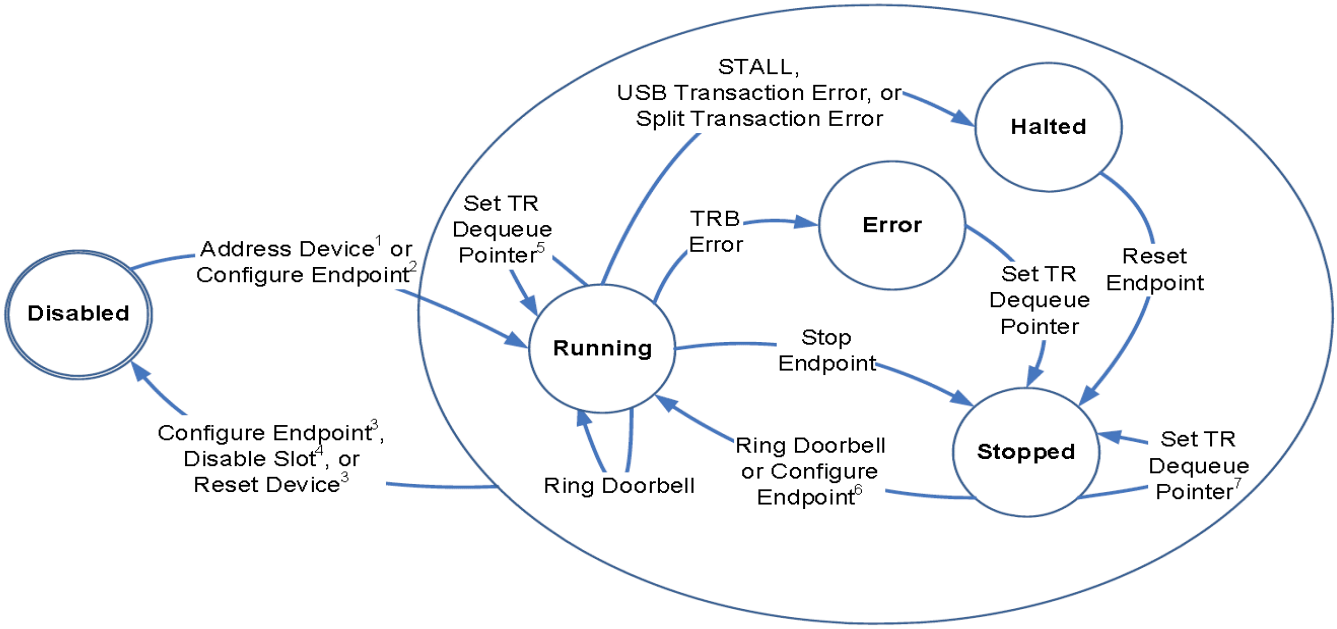
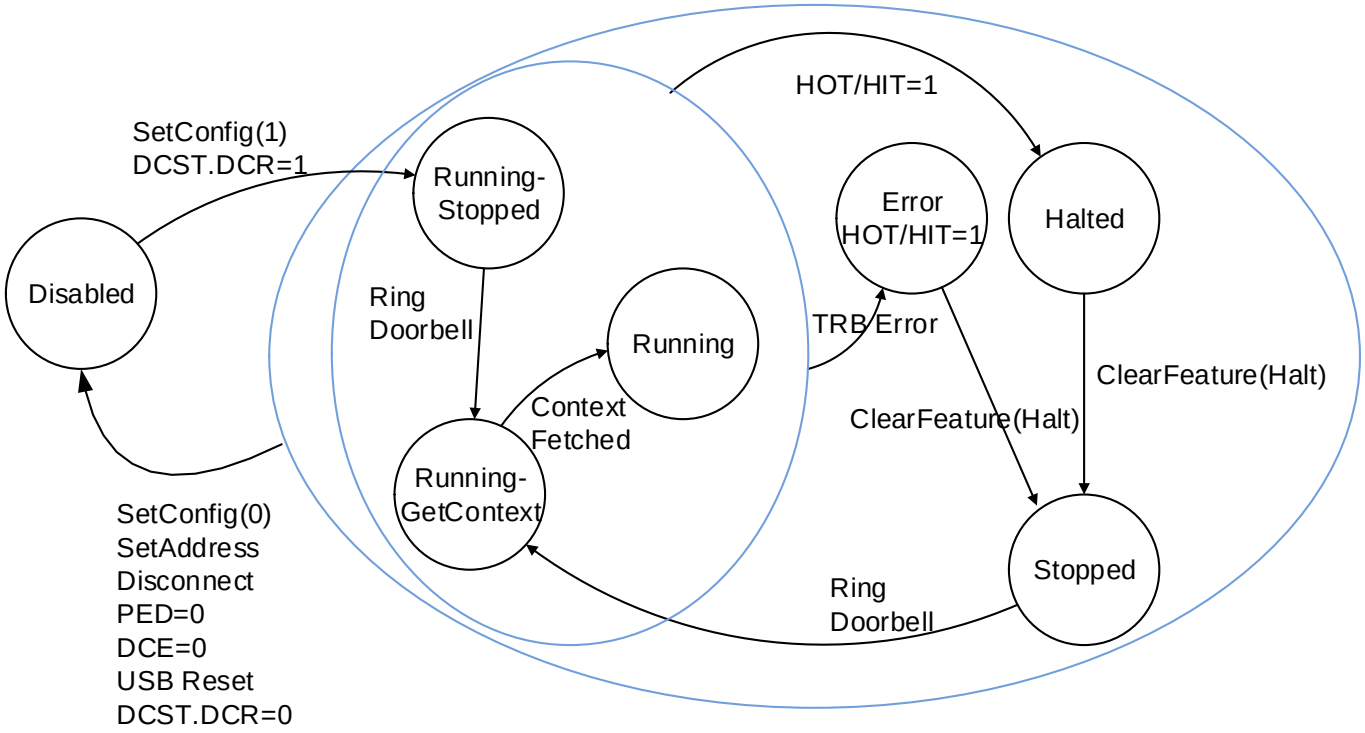


Figure 5-4 is applicable to the endpoints as referenced by the host controller, but for debug capability, the endpoint state diagram is different (see Figure 5-5).

Figure 5-5 DbC Bulk Endpoint State Transitions



The main differences are as follows:

- DbC does not have any “Set TR Dequeue Pointer,” “Stop Endpoint,” or “Reset Endpoint” command
- Address Device and Configure Endpoint are commands issued by the hardware when it receives the `SetAddress` or `SetConfiguration` control transfer. They are not issued by the software
- The Running state has four substates: Stopped, GetContext, Paused, and Fetching. These substates are not visible to the software, but they are used internally to determine whether the endpoint needs to fetch the Endpoint Context, and whether it is fetching TRBs or not.

The DbC writes the endpoint state to the Endpoint Context during the transitions between the Disabled, Running, Halted, Stopped, and Error states. Even when the transition to the Disabled state is caused by the software setting `DCCTRL.DCE` to '0', the controller still writes the new state to the Endpoint Context despite the fact that the context is not owned by the controller.

5.5.2.6 Halting Endpoints

The software can halt one of the bulk endpoints by setting the `DCCTRL.HOT` or `HIT` field to '1'. `HOT` applies to the OUT transfer ring, which is the IN endpoint. `HIT` applies to the IN transfer ring, which is the OUT endpoint.

When the software writes HOT or HIT with ‘1’, the controller performs the following operations:

1. Propagates the STALL endpoint setting request to the USB clock domain, and waits for the end of the current transaction on the USB

   Meanwhile, if the software reads the `DCCTRL` register, it sees the `HOT` or `HIT` field is still '0' which indicates that the halt request is not propagated to the USB yet.
2. Starts returning STALL on the USB to any transaction to that endpoint

3. Sets the DCCTRL.HOT or HIT read value to '1'
4. Writes out the Endpoint Context with the current value of the TR Dequeue Pointer (after the current transaction is completed on the USB), and "Halted" in the EP State field
5. Generates a Transfer Event if there is an incomplete Normal TRB present in the transfer ring
   a. The TRB pointer references the TRB that the STALL occurred on.
   b. The Transfer Length field may indicate that the TRB is partially completed.
   c. The Completion Code is "Stall Error."
   d. This event is generated regardless of the value of IOC in the error TRB.
   e. If the TRB ring is empty, or does not contain any Normal TRBs, a "Stall Error" transfer event is not generated because it does not have any TRB to reference.

If the controller is in the process of responding to a transfer on the USB when the software sets the HOT or HIT field to '1,' it completes the current transaction before performing the previous-listed operations. For example, if the controller already responded with an ACK to an OUT packet, that packet is written to the system memory before the transfer event is generated, but any transaction after the HOT/HIT setting receives a STALL response.

When the host sends a ClearFeature(Halt) request to clear the halt on an endpoint, the controller performs the following operations:

1. Clears the STALL condition, and resets the data sequence number to '0'.

   After this point, any new token on the USB does not receive a STALL response. If the host attempts a transaction to the endpoint before the software prepares a new transfer, the host receives an NRDY response.
2. Writes the Endpoint Context with "Stopped" in the EP State field.

   This indicates that the endpoint is not halted on the USB, and the software can update the TR Dequeue Pointer, and ring the doorbell to restart the transfer.
3. Sets the DCCTRL.HOT or HIT read value to '0'.

When the software needs to resume transfers on the endpoint, it must wait for the EP State in the Endpoint Context to be "Stopped," then program the new TR Dequeue Pointer in the Endpoint Context before ringing the doorbell. The controller fetches the new TR Dequeue Pointer from the Endpoint Context the first time the doorbell is rung after the endpoint halted.

The USB endpoint state can be determined by the EP State in the Endpoint Context and the value of the HOT/HIT field as shown in Table 5-2.

Table 5-2 USB Endpoint State Vs. HOT/HIT

EP State	HOT/HIT	USB Endpoint State
Running	0	Not Stalled
Running	1	Stalled
Halted	0	<Invalid>
Halted	1	Unknown (maybe Not Stalled if the host just sent a ClearFeature(Halt))
Stopped	1	Not Stalled
Stopped	0	Not Stalled

5.5.2.7 TRB Error

If an unknown TRB type is encountered on the transfer ring, the controller handles this similar to the HOT/HIT setting described in "Halting Endpoints" on page 391. In particular, the controller performs the following operations:

1. Propagates the STALL endpoint setting request to the USB clock domain, and waits for the end of the current transaction on the USB

   Meanwhile, if the software reads the DCCTRL register, it sees the HOT or HIT field is still '0' which indicates that the halt request is not propagated to the USB yet.
2. Starts returning STALL on the USB to any transaction to that endpoint
3. Sets the DCCTRL.HOT or HIT read value to '1'
4. Writes out the Endpoint Context with the current value of the TR Dequeue Pointer (after the current transaction is completed on the USB), and "Error" in the EP State field
5. Generates a Transfer Event
   a. The TRB pointer references to the TRB with the unsupported TRB type.
   b. The Transfer Length field is '0'.
   c. The Completion Code is "TRB Error".
   d. This event is generated regardless of the value of IOC in the unsupported TRB.

Upon receiving the TRB Error transfer event, the software can reset the controller by toggling the DCE from 1->0->1, or can wait for the host to send a ClearFeature(Halt) request. When the host sends a ClearFeature(Halt) request to clear the halt on the endpoint, the controller performs the following operations:

1. Clears the STALL condition and resets the data sequence number to 0

   After this point, any new tokens on the USB do not receive a STALL response. If the host attempts a transaction to the endpoint before the software prepares a new transfer, the host receives an NRDY response.
2. Writes the Endpoint Context with "Stopped" in the EP State field

   This indicates that the endpoint is not halted on the USB, and the software can update the TR Dequeue Pointer and ring the doorbell to restart the transfer.
3. Sets the DCCTRL.HOT or HIT read value to '0'

When the software needs to resume transfers on the endpoint, it must wait for the EP State in the Endpoint Context to be "Stopped," then program the new TR Dequeue Pointer in the Endpoint Context before ringing the doorbell. The controller fetches the new TR Dequeue Pointer from the Endpoint Context the first time the doorbell is rung after the endpoint halted.

5.5.2.8 Event Ring

The Debug Capability has its own Event Ring where it places transfer events and port status change events. However, there is no interrupt line that is asserted when there is an event on the Event Ring. The software polls the DSTS.ER field to determine when there is an event to be processed.

When there is one location left in the Event Ring before it is full, the controller writes the "Event Ring Full" transfer event to indicate that the Event Ring is too full to write additional events. However, this does not

mean that the events are lost, only that the events are piling up inside the controller and possibly causing back pressure to the Debug Host which eventually causes the controller to stop releasing credits on the USB 3.0 link.

5.5.2.9 Disabled Endpoint Handling (DCCTRL.DRC/DCR)

The xHCI Specification does not precisely describe the handshake between the hardware and software when an endpoint becomes Disabled and the DbC exits the Configured state, leaving open several different interpretations. The DWC_usb3 controller adheres to the following description of the Endpoint Disabled handling. This text has been proposed as an errata to the xHCI Specification.

There are several conditions which cause an endpoint to transition from the Running to the Disabled state (DbC exits the Configured state). This causes the `DCCTRL.DCR` field to transition from a '1' to '0' and `DCCTRL.DRC` to be set to '1'.

- The Debug host initiates a USB Reset.
- The Debug host issues a SetConfiguration(0) device request.
- Timeout occurs in the Port State Machine.
- The USB cable is disconnected.
- The Debug Capability driver writes '0' to the `DCPORTSC.PED` field.

When the bulk endpoint becomes Disabled, the following occurs:

- If there is a valid Transfer TRB on the Transfer Ring, a Transfer Event is generated.
  - The TRB Pointer field of the Transfer Event references the Transfer TRB that the event occurred on.
  - The TRB Transfer Length field of the Transfer Event indicates that the Transfer TRB is partially completed.
  - The Completion Code field of the Transfer Event indicates a USB Transaction Error.
  - This Transfer Event is generated irrespective of the IOC flag being set or not in the associated Transfer TRB.
  - The DbC increments its TR Dequeue Pointer.
- The Endpoint Context is written with
  - TR Dequeue Pointer set to its current value
  - Endpoint State set to Disabled

The software detects this condition by reading the `DCCTRL.DRC` field as '1' and by reading the Endpoint State as Disabled. In response, the software reads the TR Dequeue Pointer field in the Endpoint Context and updates it to point to the next TRB to be executed after it clears the `DCCTRL.DRC` field and rings the doorbell.

The DbC is not required to advance the Dequeue Pointer of an endpoint to the next TD boundary when a Disabled Endpoint condition occurs.

5.5.2.10 Resets and Asynchronous Operations

This section discusses the several asynchronous operations which affect the debug capability.

Resets

- When the software sets DCE from 1 to 0, it invokes a “DCE Reset.”
  - The controller sets the `DCPORTSC` status change bits to '0.'
  - If the controller is previously in the DbC state, the port exits the DbC state.
- When the Debug Host performs a Hot or Warm Reset, it invokes a “USB Reset.”
  - The controller stops the transfer rings for endpoint 1, and stops responding on this endpoint.
  - The controller does not resume transfers until the Debug Host reconfigures the Debug Target and the software rings the doorbell again.
- When the software sets `HCRST` or `GCTL.CoreSoftReset`, it invokes a “Chip Hardware Reset.”

  This causes all asynchronous resets in the controller to be asserted. `DCPORTSC` is set to 0, `DCCTRL` is set to 0 (DCE=0), all state machines go back to IDLE, the sticky `PORTSC` registers are reset, and the PHYs are reset.
- When the software sets LHCRST, it invokes a “Light Reset.”

  This resets the upper layer, the debug capability, and its registers, but not the sticky `PORTSC` registers. If the debug capability is in an “Upstream Mode” state, this reset causes the DCE bit to go to '0' which in turn causes the root hub (which does not reset on light reset) to go back to downstream mode only.
- `GUSB2PHYCFG.SoftReset`

  This resets only the individual USB 2.0 PHY.
- `GUSB3PIPECTL.SoftReset`

  This resets only the individual USB 3.0 PHY.

Other Asynchronous Operations

- When the software sets DCE from 0 to 1, it invokes a “DCE Enable.”

  The LSP fetches the DbCIC, enables endpoint 0, and allows upstream connections on its ports.
- When the host sends the device a `SetConfiguration(1)`, it invokes a “Device Configure.”

  The LSP configures EP1 IN and OUT by clearing any residual state, such as STALL, Data Sequence Number, and the SPR flag (synchronization between LSP and PTL).
- When the host sends the device a `SetConfiguration(0)`, it invokes a “Device Deconfigure.”

  The controller stops transfer rings, and stops responding on endpoint 1.
- When the software sets `DCPORTSC.PED` to '1', it invokes a “Port Enable.”
- When the software sets `DCPORTSC.PED` to '0', it invokes a “Port Disable.”

  The port still belongs to the debug capability, but it disconnects from the Debug Host if it was already connected.

5.5.2.11 U1/U2/U3 Entry

The Debug Host enables the Debug Target to initiate U1/U2 entry by sending the `SetFeature(U1_ENABLE)` or `SetFeature(U2_ENABLE)` requests. The U1/U2 entry can be forcefully disabled by the software by setting `DCTL.InitU1/U2Ena` and `DCTL.AcceptU1/U2Ena` to '0'. This may be necessary to improve interoperability with the Debug Hosts and Hubs.

The Debug Device, similar to DWC_usb3 device, accepts and initiates U1/U2 entry. This means that the Debug Device keeps track of its endpoints and the Packet Pending (PP) flag from the Debug Host. If all the endpoints are idle, the controller accepts U1/U2 entry. If the endpoints are idle, and the host has enabled the U1_ENABLE/U2_ENABLE feature, then the Debug Target initiates U1/U2 entry on the link.

For a complete description of how the controller determines that endpoints are idle, refer to "Low Power Operation of USB" on page 376. This is based on section C.3 "Device-Initiated Link Power Management Policies" of the USB 3.0 specification.

The USB 3.0 protocol allows a SuperSpeed host and device to negotiate with each other and decide when to bring their link into a lower power state (U1 or U2). This is accomplished by each side making their individual decisions based on whether they have any traffic currently pending or expect any traffic soon. The device requests low power entry by transmitting an LGO_U1 (or LGO_U2) Link Command and exits low power by using a LFPS handshake.

The Debug Capability uses the information from the host (bus side) and the software (the application side) to decide the best time to request a low power entry and exit from the low power state. The same calculation is used by the controller to decide whether to accept a low power request from the host. The controller maintains the state information about each endpoint and whether it is "active" or "paused". If all enabled endpoints are paused from either the application side or the bus side, the low power entry is allowed. If at least one endpoint is not paused, the low power entry is not allowed.

To indicate that there is no pending traffic and allow the link to go into a low power state, the software pauses all enabled endpoints. Otherwise, an endpoint is considered active on the application side. For these endpoints, the host indicates that it is paused using one of the following ways:

- By setting the Packet Pending (PP) flag to '0' in a DPH
- Setting PP to '0' and NumP to '0' in an ACK TP

Exception: If the data packet payload has a CRC error, the DbC responds with an ACK (Retry=1), and the endpoint is not paused. Otherwise, if PP=1 or Deferred=1, the endpoint is considered to be active on the bus side.

Control endpoints have the following additional condition:

The controller does not allow low power entry during a control transfer (starting from the reception of a setup packet and ending at the completion of the status stage).

If all the enabled endpoints are paused by the software or the host, the controller issues an LGO_U1 (or LGO_U2) link command or it responds with an LAU link command to the low power request of the host or hub. Otherwise, the controller rejects the low power requests with an LXU link command.

Other related information:

- For bulk endpoints, the USB 3.0 Specification says that the device may go to U1/U2 500ms (tERDY-Timeout) after transmitting ERDY. However, it is not possible to force the DbC to enter U1/U2 in this scenario where the host is unresponsive to the ERDY.
- Because U1/U2 entry is based on the presence of packets in the TX FIFO and not based on the TRBs that are available to the controller, it is possible that an IN endpoint responds NRDY, goes to U1, then

U0, then transmits an ERDY if the timing and TX FIFO size work out such that the controller cannot keep at least 1 packet in the TX FIFO at all times.

- In addition, before the final decision is made to either initiate or accept a low power state, the DbC checks the state of `DCTL.InitU1Ena/RejectU1Dis/InitU2Ena/RejectU2Dis` bits. If the device receives Set Link Function LMP with `Force_LinkPM_Accept` bit asserted, the DbC accepts a low power request from the host independent of endpoints state and DCTL Power control settings.

For U3, the Debug Device, similar to the normal device, enters and exits U3, as required in the USB 3.0 specification. The Debug Device does not transmit a Function Wake Device Notification because it is not Remote Wakeup capable.

5.5.3 Data Structures

The xHCI data structures that are produced and consumed by the Debug Capability are:

- Debug Capability Info Context Data Structure (DbCIC)
- Endpoint Context
- Transfer TRB
- Event TRB

Each of these data structures are described in the xHCI Specification, but details specific to the implementation are described in this section.

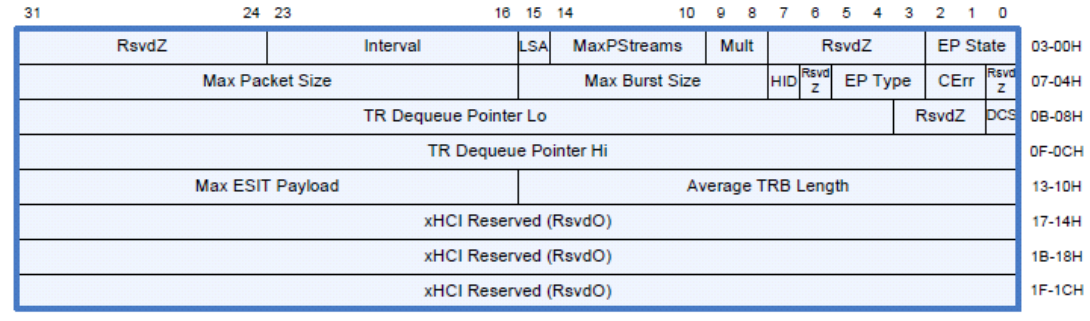
5.5.3.1 Debug Capability Info Context Data Structure

The Debug Target fetches the Debug Capability Info Context data structure when the software sets DCE=1, and uses the information to respond to `GetDescriptor(STRING)` requests from the host.

5.5.3.2 Endpoint Context

In Debug Capability, the Endpoint Contexts are fixed to 64 bytes each. Figure 5-6 shows the Endpoint Context.

Figure 5-6 Endpoint Context



Even though the Endpoint Context format is the same, only EP State, TR Dequeue Pointer, and DCS fields of the EP Context are applicable to a Debug Target. The following fields are ignored:

- Max Packet Size: Always assumed to be 1024 bytes, unless `GCTL.FRMSCLDWN` is set to a different value, in which case the MPS is 1024 << `GCTL.FRMSCLDWN`

- EP Type: Always assumed to be 2 (Bulk)
- Max Burst Size: Assumed to be 15 (`DWC_USB3_DBC_MAX_BURST_SIZE`) for both IN and OUT endpoints. However, for the OUT endpoint the controller fills in the NumP (number of receive buffers) field of the ACK TP based on the value in the DCFG.NumP register, which is 4 (`DWC_USB3_DEVDBC_OUT_NUMP`) by default.

The other fields (`Mult, MaxPStreams, LSA, Interval, CErr, HID, Average TRB Length, and Max ESIT Payload`) are not applicable.

5.5.3.3 Transfer TRBs

The following are the only supported Transfer TRBs types:

- Normal
- Link
- No-Op
- Vendor-Defined

- The Interrupter Target and BEI fields are ignored because the debug capability does not have an interrupt line.
- The IDT flag in Transfer TRBs must not be set to '1.'
- The ENT flag is ignored, as it is primarily used for stream-based endpoints.

5.5.3.4 Event TRBs

The controller produces the following types of Event TRBs:

- Port Status Change Event
- Transfer Event

The Completion Codes that are generated are:

- Success
- TRB Error
- Stall Error
- Short Packet
- Event Ring Full Error
- USB Transaction Error (see “Disabled Endpoint Handling (DCCTRL.DRC/DCR)” on page 394)

The Babble Detected completion code is not generated, because the controller responds with ACK(Retry) to a TD Babble scenario. The Endpoint ID field in transfer events is ‘1’ for OUT Transfer Ring and ‘2’ for IN Transfer Ring.

5.5.4 Requirements, Assumptions, and Limitations

This section discusses the various requirements, assumptions and limitations of the xHCI Debug Capability Table 5-3 gives a summary of the limitations.

Table 5-3 xHCI Debug Capability Limitations (Continued)

Feature	Comments
Incompatible Features	DbC can be configured along with Hibernation and Clock Gating. However, these features are turned off automatically when Debug Capability is enabled by the DbC driver.
U1/U2 Support	The controller supports autonomous entry into U1 or U2, but it is recommended to disable this support via the DCTL register to improve interoperability with the Debug Hosts and Hubs.
Event Data TRB	The controller does not support the use of Event Data TRBs on the Debug Capability Transfer Rings. The Microsoft Debug Capability driver does not use them.
SetFeature(Halt) on EP1	If the debug host issues a SetFeature(ENDPOINT_HALT) request to the debug target's bulk endpoint 1, it receives a STALL response. This request is not supported.
ERDY	The controller transmits an ERDY when it is ready to transmit or receive a packet on the corresponding bulk endpoint, not explicitly when the endpoint's doorbell is rung by the software.
Chained TRBs greater than TRB cache size	The debug capability driver must not create a multi-TRB TD that describes smaller than a 1K packet that spreads across 8 or more TRBs on either the IN TR or the OUT TR.
TD Babble	If the host transmits a packet larger than the TD that the Debug Capability Driver has been setup for it, the debug target respondd repeatedly with ACK(Retry). No Transfer Event is generated for this scenario.
Latency Tolerance Message	The Debug Target does not transmit LTM Device Notifications.
HOT/HIT and Transfer Events	When the Debug Capability Driver sets HOT or HIT to '1', the controller generates a Transfer Event with "Stall Error" only if a Normal TRB is valid in the Transfer Ring. For more information, see “Halting Endpoints” on page 391.
Reading DCCTRL.HOT/HIT	Immediately after setting HOT or HIT to '1', the field returns '0' when read until the STALL condition has internally propagated. When the controller is prepared to return STALL on the USB, the HOT/HIT field returns '1' when read. For more information, see “Halting Endpoints” on page 391.
Residual DMA after DCE=0	The Debug Capability specification has no feedback after the software sets DCE to '0' to know when the DbC has halted (ala USBSTS.HCHalted). Therefore, it is possible that the controller continues DMA on the system bus to and from data buffers, TRBs, or Endpoint Contexts even after the software has set DCE=0. There is no definition for how long the DMA persists.
DCE Toggling	The software must wait at least 1 millisecond after setting DCE to '0' before setting it back to '1'. For more information, see “**DCE Toggling**” on page 405.

Feature	Comments
Overcurrent on Debug Port	If an overcurrent occurs on the debug port, the controller generates an xHCI port status change event and set the OCA and OCC bits in the associated PORTSC register. For more information, see “Overcurrent on Debug Port” on page 406.
Zero Length TD's for the OUT endpoint	Zero-length TD's are not allowed on the IN Transfer Ring (OUT endpoint). If the Debug Capability Driver is expecting a zero-length packet, it must prepare a TD that has at least 1 byte.
HCRST behavior	The xHCI Specification mentions that the DbC updates the Endpoint Context TR Dequeue Pointer if HCRST is set to ‘1’. However, because the debug capability is reset on HCRST, the controller does not update the Endpoint Context TR Dequeue Pointer.

5.5.4.1 Supported Configurations

The DWC_usb3 controller can be configured with the Debug Capability hardware in any configuration supported by a host configuration. In particular, the Debug Capability logic can co-exist with the following other configurations:

- Host Only (`DWC_USB3_MODE=1`)
- Dual Role Device (`DWC_USB3_MODE=2`)
- Clock Gating (`DWC_USB3_EN_PWROPT > 0`)
- Hibernation (`DWC_USB3_EN_PWROPT=2`)

However, when the debug capability is enabled (`DCCTRL.DCE=1`), the following features (if configured) are disabled by the hardware:

- Clock gating (`GCTL.DsblClkGtng=1`)
- Hibernation (`GCTL.GblHibernationEn=0`)

5.5.4.2 RAM Requirements

For details on the RAM requirements for Debug Capability, refer “Additional RAM Requirements for xHCI Debug Capability” in the *DWC SuperSpeed USB 3.0 Controller Databook*.

5.5.4.3 Port State Machine/Root Hub

Multiple Debug Target Ports

- Any port may become a debug port, but only one port can be the debug port at a point of time.
- After one port is selected as the debug port, the others stop reporting upstream capable in their Port Capability LMP's.
- Even though the controller prevents two ports from training as upstream ports, exhaustive testing is not done for this scenario because it is assumed that the user does not attempt to simultaneously plug two Debug Hosts into two different ports on the Debug Target.

Port power

- The `vbus_ctrl` is not driven to 0 when the port becomes a debug port. The `vbus_ctrl` is based on the `PORTSC` register.

Overcurrent

- If an overcurrent occurs on a port that is acting as a Debug Target, the `PORTSC` register associated with that port reports the overcurrent.

5.5.4.4 Control Transfers

When the controller handles control transfers autonomously, it always responds with NRDY for the first time it receives a STATUS TP, then sends ERDY, and ACKs the status phase when the host repeats it. The reason the controller never ACKs the STATUS TP for the first time is to account for the misbehaving hosts that might continue the data phase for more transactions, or might initiate the data phase in the incorrect direction (IN vs. OUT). By waiting for the status phase, the controller is guaranteed that the host has concluded the data phase.

Table 5-4 lists the controller'd responses to the standard device requests.

- As shown in Table 5-4, the Debug Target responds STALL to a SetFeature(ENDPOINT_HALT) request from the host. The debug capability cannot be halted from the host, it can only be halted from the debug capability driver.
- The Debug Target responds STALL to a SetInterface(0) request from the host. The USB 2.0 and USB 3.0 specifications allow this behavior when a device supports only one interface.

Table 5-4 Control Transfer Responses (Continued)

Request	Index/Value	State	Response
SetFeature	ENDPOINT_HALT (EP1)	All states	STALL
	ENDPOINT_HALT (EP0)	All states	ACK
	FUNCTION_SUSPEND	Configured	ACK
	U1_ENABLE	Configured	ACK
	U2_ENABLE	Configured	ACK
	LTM_ENABLE	All states	STALL
	B3_NTF_HOST_REL	All states	STALL
	TEST_MODE	All states	STALL
ClearFeature	ENDPOINT_HALT (EP1)	Configured	ACK
	ENDPOINT_HALT (EP0)	All states	ACK
	FUNCTION_SUSPEND	Configured	ACK
	U1_ENABLE	Configured	ACK
	U2_ENABLE	Configured	ACK
	LTM_ENABLE	All states	STALL
	B3_NTF_HOST_REL	All states	STALL
	TEST_MODE	All states	STALL
GetConfiguration		Addressed/Configured	Return 0 (unconfigured) or 1 (configured) ACK
SetDescriptor		All states	STALL
GetInterface		Configured	Return 0 ACK
SetInterface		All states	STALL&
GetStatus	Device	Addressed/Configured	ACK
	Interface	Configured	ACK
	Endpoint (EP0)	Addressed/Configured	ACK
	Endpoint (EP1)	Configured	ACK

Request	Index/Value	State	Response
GetDescriptor	DEVICE	All states	ACK
	CONFIGURATION	All states	Fixed value + cC fields
	BOS	All states	Fixed value + cC fields
	STRING (supported desc_index 0 to 3)	All states	Read from external memory
	STRING (unsupported desc_index > 3)	All states	STALL
	INTERFACE	All states	STALL
	ENDPOINT	All states	STALL
	INTERFACE_POWER	All states	STALL
	DEBUG	All states	STALL
	INTERFACE_ASSOCIATION	All states	STALL
	DEVICE_CAPABILITY	All states	STALL
	SUPERSPEED_USB_ ENDPOINT_COMPANION	All states	STALL
SetAddress	non-zero	Default/Addressed	ACK
SetConfiguration		Addressed/Configured	ACK
SetIsochDelay		All states	ACK
SetSEL		Configured	ACK
SynchFrame		All states	STALL
*	Wrong Index/Value or wrong wLength	*	STALL
*	non-standard bmRequest Type (such as, Class/Vendor/Reserved)	*	STALL
*	*	Non-supported states	STALL

5.5.4.5 Flow Control

The controller has the following behavior:

- When an IN endpoint is flow controlled, the Debug Device transmits an ERDY when it has pre-fetched a packet into the TX FIFO.
- When an OUT endpoint is flow controlled, the Debug Device transmits an ERDY when it has fetched enough TRBs to receive a `MaxPacketSize` packet or the last packet of a TD.

The controller does not transmit an ERDY on a doorbell ring. Note that this behavior is different from the statement in section 7.6.7.2.1 of the xHCI Specification:

"If a DbC Bulk pipe had previously sent an NRDY, a doorbell ring shall cause the xHC to generate an ERDY."

5.5.4.6 Bursting Behavior for OUT and IN Transfers

When the controller, as a Debug Target, responds to OUT and IN transfers, it uses the following mechanisms to throttle the Debug Host:

- For OUT transfers, setting the NumP field in the ACK TP
  - It is not possible or practical for the controller to predict the exact rate of packets entering and exiting the receive FIFO, so the `NumP` field is set based on the `DCFG.NumP` field, which defaults to the `DWC_USB3_DEVDBC_OUT_NUMP` parameter (a value of 4) on reset.
  - You can change this field during controller configuration based on your system simulations to allow or prevent the host from initiating large bursts.
  - This is the same behavior as the Synopsys USB 3.0 Device controller.
- For IN transfers, setting the EOB field in the DPH
  - The controller sets EOB as 1 during an IN transfer burst if the transmit FIFO does not have another packet after the one that is being transmitted to force flow control on the endpoint.
  - Once there is a full packet in the transmit FIFO, the controller automatically transmits an ERDY to the Debug Host.
  - This is the same behavior as the Synopsys USB 3.0 Device controller.

5.5.4.7 TRB Cache Size and Transfers

The controller contains a TRB cache, one for the IN endpoint and one for the OUT endpoint, that holds a number of TRBs (configured by DWC_USB3_DBC_TRBS_PER_TRANSFER, default is 16).

For the bulk IN endpoint, the controller constructs a full packet with the TRBs that are present in the internal cache. This means that the debug capability driver must not create a multi-TRB TD that describes less than a MaxPacketSize packet (1K) that spreads across DWC_USB3_DBC_TRBS_PER_TRANSFER-1 or more TRBs.

For the bulk OUT endpoint, the controller receives a full packet or a short packet with the TRBs that are present in the internal cache. This means the debug capability driver must not create a multi-TRB TD that describes a buffer less than 1K that spreads across DWC_USB3_DBC_TRBS_PER_TRANSFER-1 or more TRBs.

In each of the previous equations, the "-1" is added to account for at most one Link TRB, which may be present in a TD. If more than one Link TRB is present in a TD, the requirement shrinks by the maximum number of Link TRBs that can be in a TD.

5.5.4.8 DCE=0 Restrictions

When the software sets DCCTRL.DCE=0, several operations take place inside the controller. This section describes some of the assumptions and restrictions when this occurs.

Data Structure Ownership

Section 7.6.9 of the xHCI Specification states "While the Debug Capability Enable (DCE) bit in the Debug Capability Control Register (DCCTRL) is '1', the xHC maintains ownership of the data structures." This also implies that when DCE=0, the controller does not own the data structures (DbC Info Context, OUT EP, and IN EP Context). However, due to residual DMA (see "**Residual DMA**"), the controller can be in the process of writing to these data structures and therefore there is no guarantee that the controller prevents these writes immediately when DCE is set to '0'.

Residual DMA

Because the controller may be busy performing a read or write DMA operation on the system bus when the software sets DCE=0, it is not possible to immediately abort those operations. Therefore, the controller completes the DMA operations on the system bus, and finds a graceful point to finish the DbC halt operation. Unfortunately, there is no defined "Halted" bit in the xHCI Specification, so the software cannot determine when the DMA operations have completed.

DCE Toggling

Because there are a set of operations performed within the controller when the software sets DCE from '1' to '0', and there is no feedback when the controller has completed its residual DbC operations, the software must wait at least 1 millisecond after setting DCE to '0' before setting it back to '1.'

5.5.4.9 TD Babble Response

A “TD Babble” is when the debug capability driver prepares a TD that is smaller than the amount of data sent by the Debug Host to the OUT endpoint. For example, the debug capability driver prepares an 8 byte TRB to receive an 8 byte packet from the host, but the host instead sends a 1K packet.

The xHCI Specification allows several different responses in this situation. The DWC_usb3 controller responds on the USB with an ACK(Retry) if this happens. If the host continues to send the same packet three times, the host will timeout the transfer and likely reset the Debug Target. There is no indication of the retries to the debug capability driver, but when the reset occurs, the `DCCTRL.DCR` field is set to '0' and DRC is set to '1' to indicate that the Debug Target exited the Configured state, and requires the debug driver to restart the bulk transfers.

5.5.4.10 TRB Types

The controller supports Normal, Link, No-Op, and Vendor Defined TRB Types. Event Data TRBs are not supported, as they are not required by the Microsoft Debug Capability driver.

5.5.4.11 Latency Tolerance Message

The controller, when acting as a Debug Target, does not transmit LTM Device Notifications.

5.5.4.12 Overcurrent on Debug Port

Because some systems use a single overcurrent signal for all ports, the controller cannot determine if an overcurrent on the debug port is on the port itself or whether it is from another downstream port. Therefore, if an overcurrent is detected on the debug port, the controller still sets the OCA and OCC bit in the `PORTSC` register associated with the port, and generates a port status change event to be consumed by the xHCI driver.

5.5.4.13 Vendor Device Test LMP

The controller, acting as a Debug Target, has the capability of receiving the Vendor Device Test LMP without error. However, the LMP is dropped with no indication to the software.

5.5.4.14 FIFO Sizes Impact

Enabling xHCI Debug Capability affects the host TX/RX FIFO size selection. For information on FIFO size selection, see section “TX/RX Data FIFO Sizes and TX/RX Threshold Control Register Settings” in the *DWC SuperSpeed USB 3.0 Controller User Guide*.

6

Programming DWC_usb3 for Hibernation

This chapter describes the programming sequence for DWC_usb3 controller when hibernation features is enabled. The following topics are discussed:

6.1 Programming DWC_usb3 for Hibernation in Host Mode

The host operational model is described in the xHCI Specification, but the following sections add the PCI power transitions which are not explicitly mentioned in the hibernation steps.

- The programming steps (also called “programming flow” or “programming model”) described in this Host Operational Model section must be performed by whoever is implementing the xHCI driver.
- This section includes more details about what role the Power Controller plays in the hibernation process.
- Standard xHCI drivers are available from Microsoft, MCCI, and Linux, but you may prefer to write your own driver.

6.1.1 Entering Hibernation in Host Mode

The software initiates hibernation by performing the following sequence of operations:

1. Prepare for hibernation
   a. Read the HCSPARAMS2 register to determine if the controller requires scratchpad buffer space and the number of scratchpad buffers required.
   b. Write the scratchpad buffer array address into the Device Context Base Address Array.
   c. Ensure that the following register bits are set, either by their reset values or by programming them:
      i. Ensure that GUSB3PIPECTL[17] (Suspend SS PHY) is set to '1'.
      ii. Ensure that GUSB2PHYCFG[6] (Suspend 2.0 PHY) is set to '1'.
      iii. Ensure that GCTL[1] (GblHibernationEn) is set to '1'.
2. Stop all USB activity by issuing stop endpoint commands for each endpoint in the running state. This causes the xHC to update the respective Endpoint or Stream Context TR Dequeue pointer and DCS fields.
3. Put all ports into SS.Disabled, RxDetect, L2, or U3 states.
4. Stop the controller by setting Run/Stop (R/S) = '0'.
5. Save the non-sticky Operational and Runtime registers described in the xHCI specification.
6. Set the Controller Save State (CSS) flag in the USBCMD register and wait for the Save State Status (SSS) flag in the USBSTS register to transition to '0'.

   The controller saves its internal state plus the following global registers into the scratchpad:

   - GTXFIFOSIZn
   - GRXFIFOSIZn
   - GTXTHRCFG
   - GRXTHRCFG
   - GPRTBIMAP
   - GPRTBIMAP_HS
   - GPRTBIMAP_FS
   - GUCTL, GUCTL1, GUCTL2
   - GUSB3PIPECTLn

7. Communicate with the power controller to set the power state to D3. Wait until the PMUs confirm that they have entered D3.
8. Remove controller well power.

The DCBAA and the complete tree of data structures that it references (Device Contexts, Transfer Rings, Stream Arrays, etc.) as well as the Command and Event Rings, and Scratchpad Buffers are preserved by system software.

6.1.2 Exiting Hibernation in Host Mode – Software-Initiated

The software initiates a restore from hibernation using the following sequence:

1. Communicates with the power controller to enable controller well power and set the power state to D0. Waits until the PMUs confirm that they have entered D0.
2. If the power on initialization values of the `GSBUSCFG0` and `GSBUSCFG1` registers (DWC_USB3_GSBUSCFG0_INT and DWC_USB3_GSBUSCFG1) are not the same as the normal system operating value, then re-program these registers with the characteristics of the system bus.

   It is recommended that the power on reset values of these registers are set correctly during coreConsultant configuration, so that they do not have to be re-programmed here again. The restore operation does DMA transfers and these registers define DMA characteristics.
3. Restores the operational and runtime registers with their previously-saved state.
4. Sets the Controller Restore State (CRS) flag in the USBCMD register to '1' and wait for the Restore State Status (RSS) flag in the `USBSTS` register to transition to '0'.

   a. If the controller detects that the saved state information is corrupted, it sets `USBSTS.SRE`.

   The controller restores its internal state as well as the global registers listed above.
5. Enables the controller by setting Run/Stop (R/S) = '1'.
6. The software walks the USB topology and initializes each of the xHC `PORTSC`, `PORTPMSC`, and `PORTLI` registers, and external hub ports attached to USB devices.

   This includes resuming any ports that received a remote wakeup request, and also clearing any change bits in the `PORTSC` registers. After the change bits are cleared, the controller stops asserting generation.
7. Restart each of the previously running endpoints by ringing their doorbells.

6.1.3 Exiting Hibernation in Host Mode – PHY-Initiated

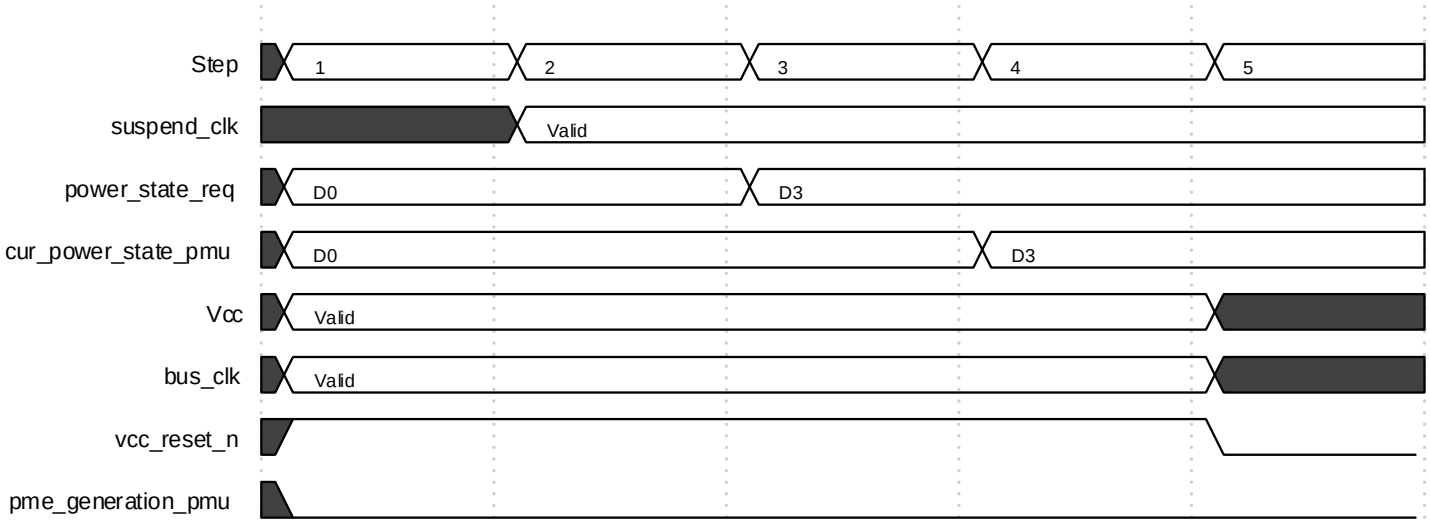
In the case of PHY-initiated wakeup (see "PHY-Initiated Hibernation Exit Event for Host and Device Modes" on page 425), the controller asserts PME to restore power to the controller and wake up the host software.

1. The wakeup logic detects the resume condition on the PHY interface and asserts the `pme_generation_pmu` signal, requesting power to be restored.
2. The power controller communicates the wakeup to the software with its own PME interrupt.
3. Continue with Step 1 of "Exiting Hibernation in Host Mode – Software-Initiated" .

6.1.4 Power Controller Timing Examples

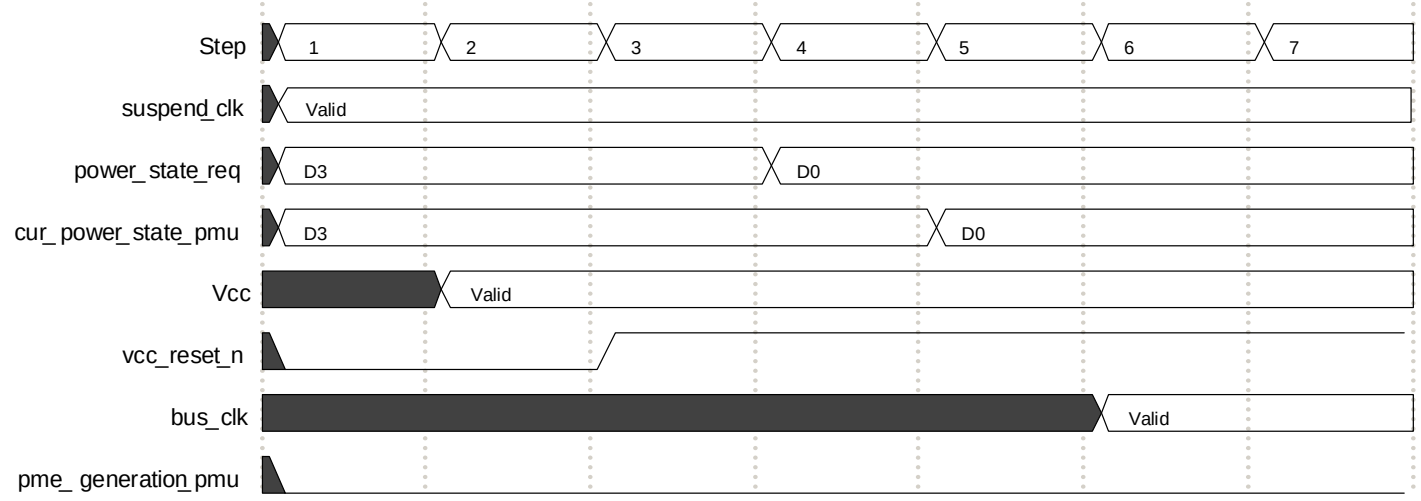
In the following example, the power controller takes the `suspend_clk` valid, puts the controller into D3, and then removes `Vcc`. The `vcc_reset_n` is asserted during this time (the `vaux_reset_n` should not be asserted). These are logical examples, so the number of clock cycles in a single “Cycle” illustrated in these examples is not exactly 1.

Figure 6-1 Power Controller D3cold Entry



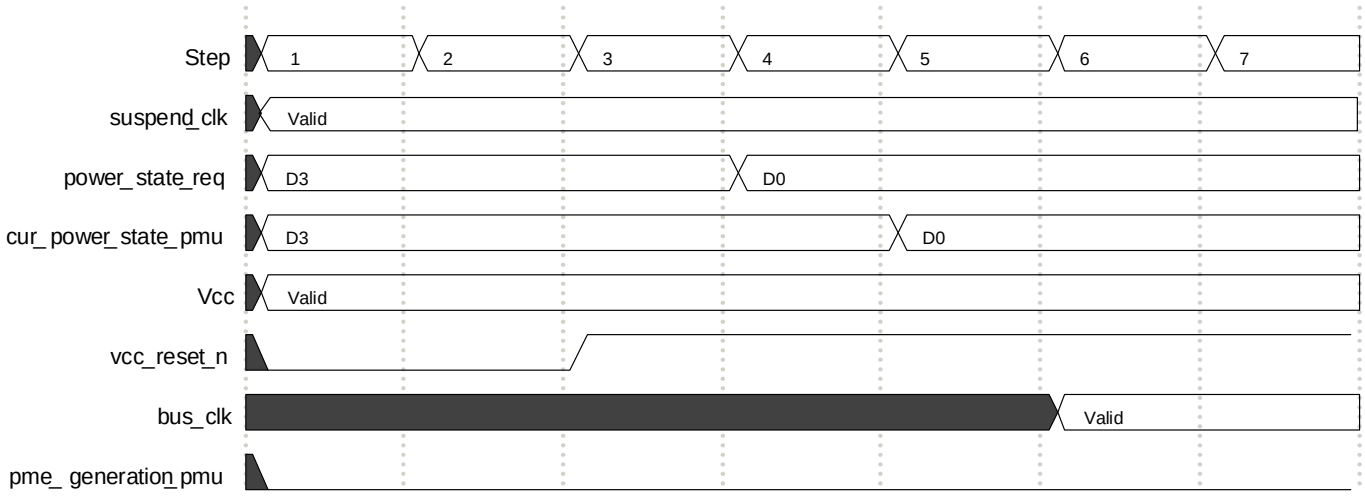
In the following example, the power controller brings the controller out of D3cold by restoring `Vcc` and the clocks and requesting D0.

Figure 6-2 Software Initiated D3cold Exit



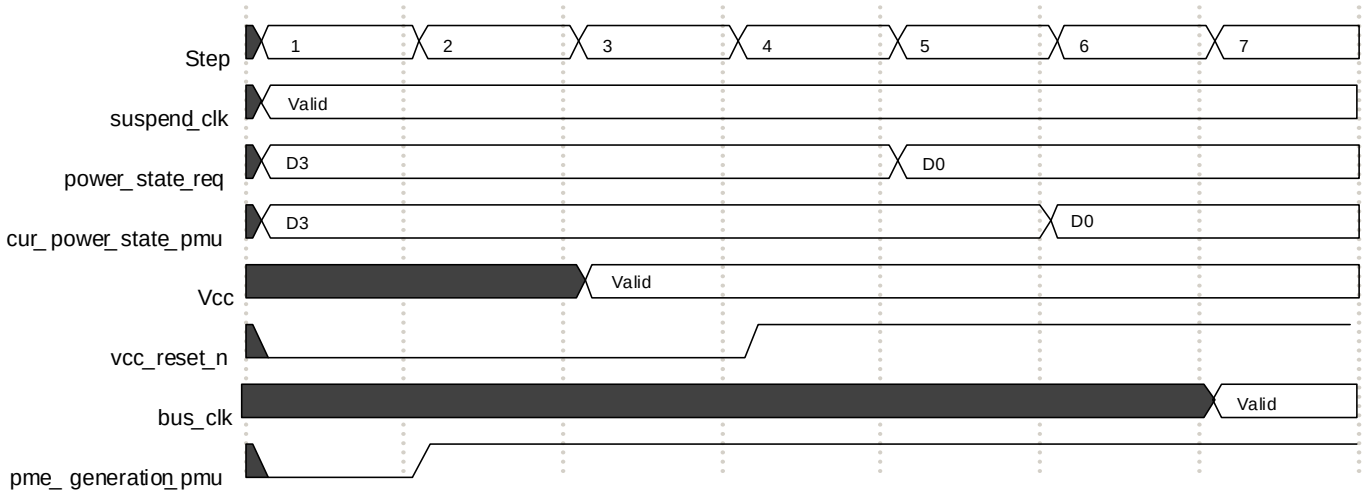
In the following example, Vcc is not removed, so the controller is still in D3hot. However, vcc_reset_n must still be applied in this situation. For that reason, it is recommended that vcc_reset_n be asserted when the controller enters D3 instead of only when Vcc is invalid. Vcc_reset_n is de-asserted when software wants to initiate the D3 exit.

Figure 6-3 Software Initiated D3hot Exit



In the following example, D3 exit is requested by the PMU via `pme_generation`.

Figure 6-4 Power Controller D3cold Exit by PME



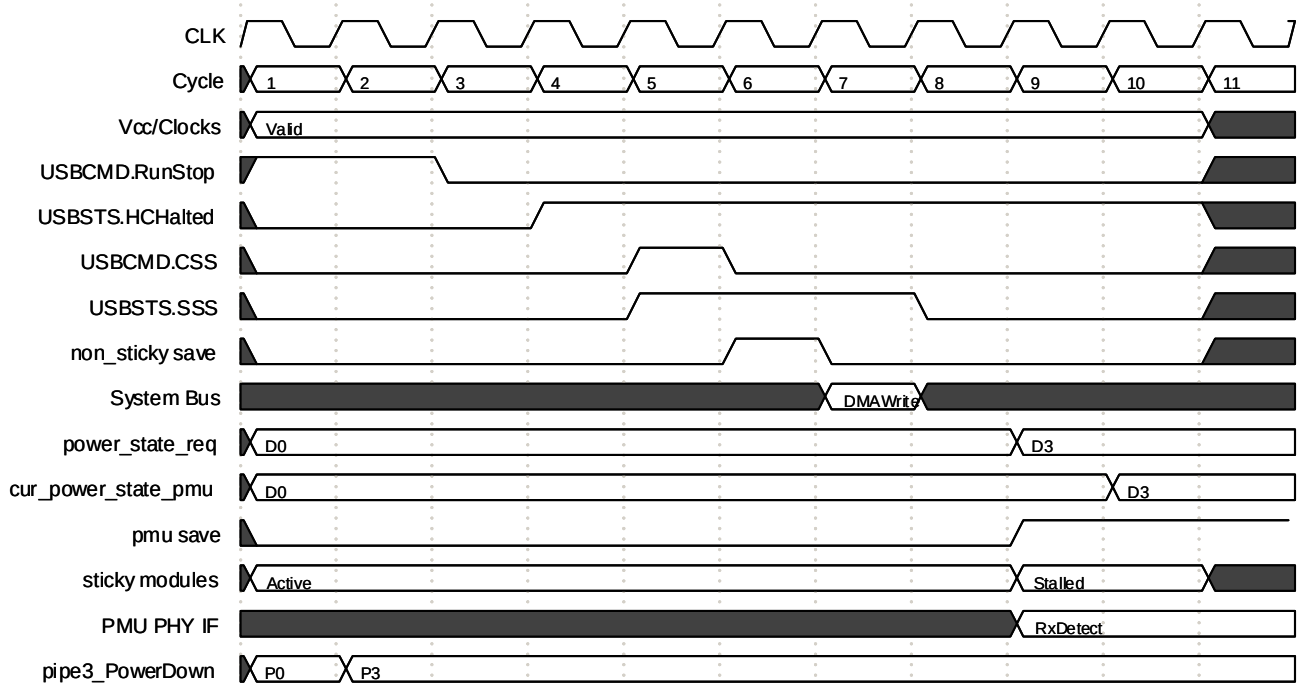
6.1.5 Host Timing Examples

The following are logical examples, so the number of clock cycles in a single “Cycle” in these examples is not exactly 1.

6.1.5.1 Host Hibernation with D3cold Entry

Figure 6-5 shows a waveform depicting hibernation with D3cold entry for the DWC_usb3 controller in host mode.

Figure 6-5 Host Hibernation with D3cold Entry

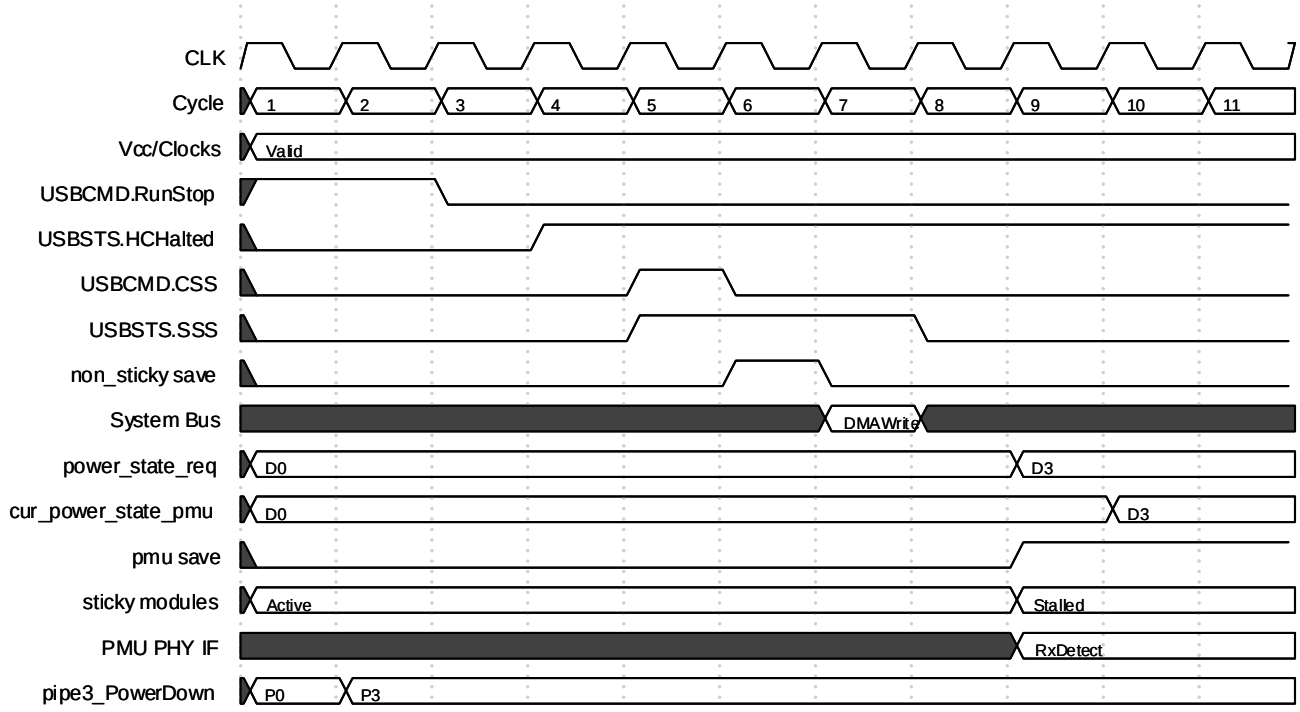


1. RunStop=1, device state is D0, and the host controller is operating normally
2. The software stops USB activity by issuing Stop Endpoint commands, and suspends the port(s), putting them into the P3 state.
3. The software sets `RunStop` to 0.
4. The controller sets `HCHalted` to 1, indicating the controller has stopped. The software then reads internal registers and saves their state.
5. The software sets CSS to 1, requesting that the controller save its state. In response, the controller sets SSS to 1 to indicate that it is in the process of saving its state.
6. The controller writes its internal state to internal memory.
7. The controller performs a DMA write on the system bus to write its internal state to external memory.
8. The controller sets SSS to 0 to indicate that it has completed saving its state.
9. The power controller requests that the controller enter D3hot. The controller saves its sticky state to the PMU, and activates the PMU PHY interface.
10. The controller indicates it is in D3 through the `current_power_state_pmu` signal.
11. The power controller turns off `Vcc` and clocks to the controller, but keeps the PMU powered.

6.1.5.2 Host Hibernation Without D3cold Entry

In this scenario, the power controller decides not to remove Vcc which keeps the controller in the D3hot state. However, the power controller and controller apply a reset when the controller is programmed back into D0, so it is still necessary to restore the state.

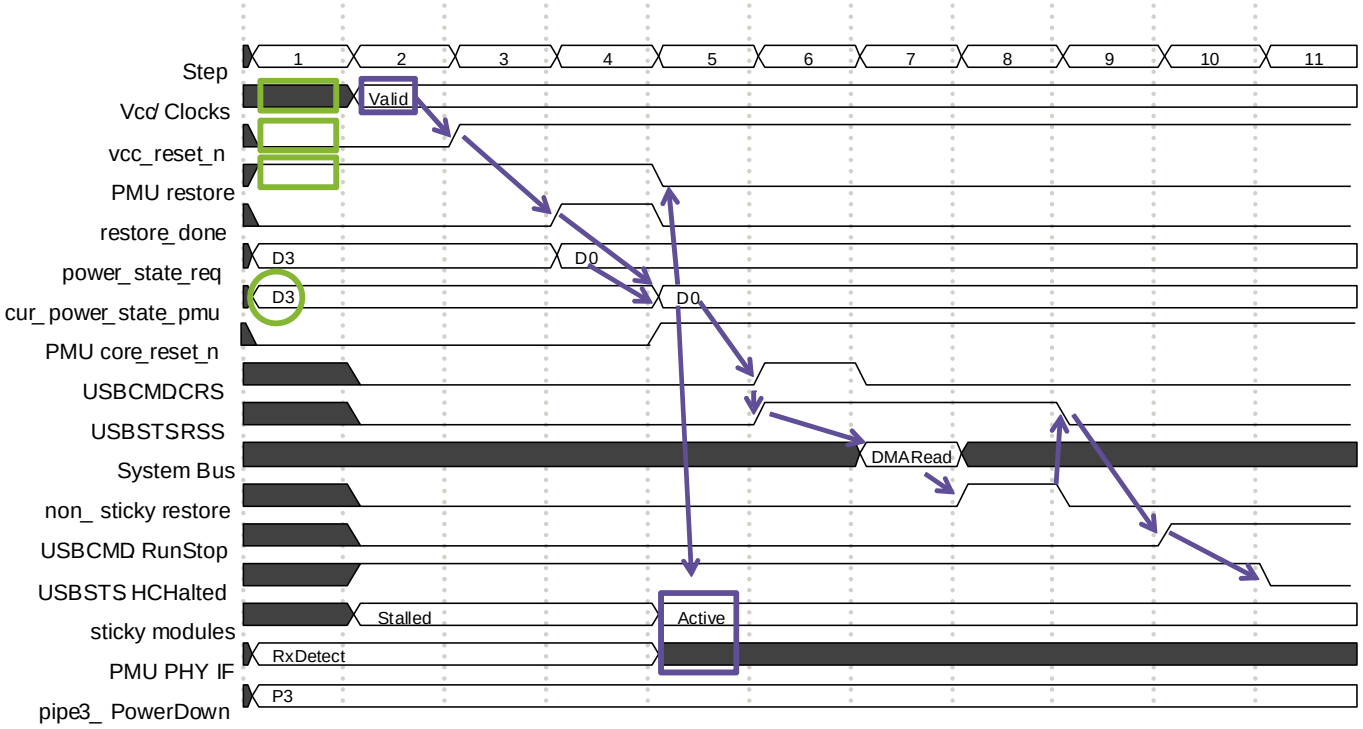
Figure 6-6 Host Hibernation Without D3cold Entry



6.1.5.3 Software Wakeup from D3cold

In this scenario, software decides to wake up the host controller without any activity from the PHY. Because the controller entered D3, the state must be restored.

Figure 6-7 Software Wakeup from D3cold



1. In D3cold, the power controller has turned off `Vcc` and is asserting `vcc_reset_n` to the controller. The PMU is asserting the restore signal in preparation for the restoration of power. The PMU is also controlling the PHY interface.
2. The software tells the power controller to turn `Vcc` back on, but the controller is still being held in reset by the power controller and PMU until clocks are stable.
3. When `Vcc` is valid, the power controller de-asserts `vcc_reset_n` which allows the sticky modules to begin accepting the sticky restore data.
4. When all of the state is restored to the sticky modules, the controller writes to a special location in the PMU to indicate that it is done (shown in the waveform as “restore_done”).

   The power controller also requests the controller to enter the D0 state, but this is not related to the restore/restore_done operation.
5. In this step, the following occurs:

   a. The restore completes.

   b. Because the `power_state_req` is D0 and the restore is completed, the PMU drives `current_power_state_pmu` to D0.

   c. The PMU stops asserting “restore” because the `current_power_state_pmu` is D0

   d. Because the `current_power_state_pmu` is D0, the sticky modules regain control of the PHY interface.

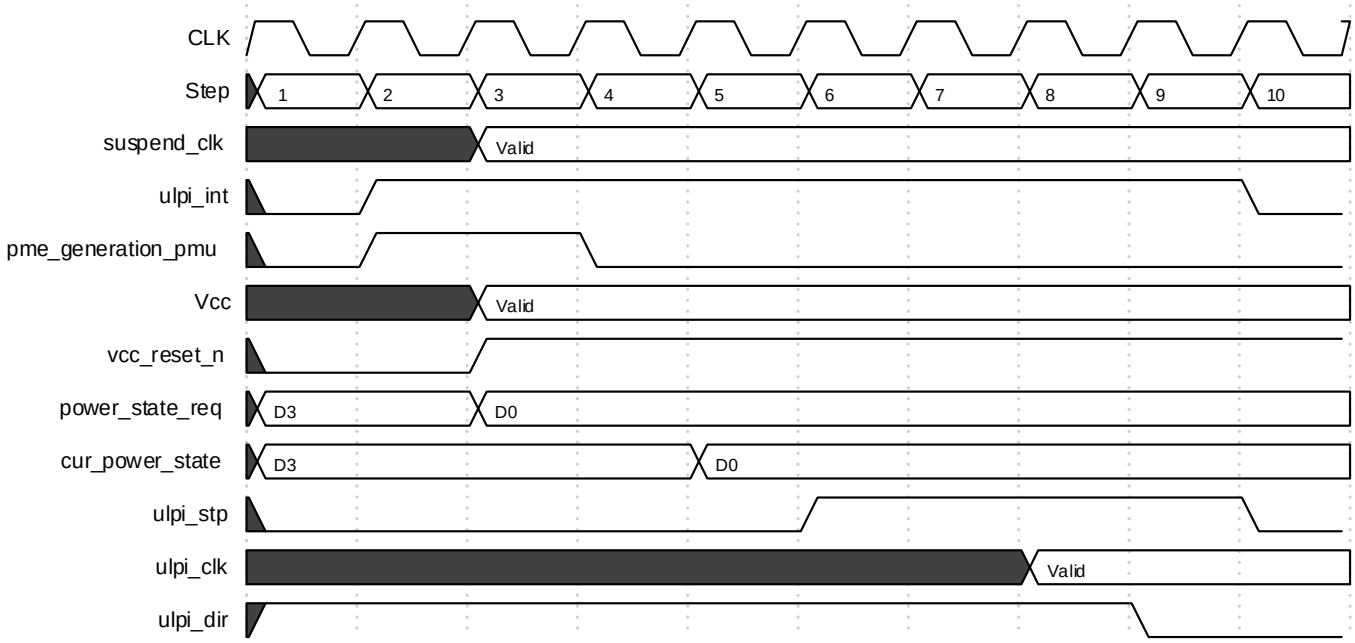
6. The software initiates the non-sticky restore by setting the CRS bit. The controller responds by setting RSS to '1', indicating that it is in the process of restoring its state.
7. The controller initiates DMA reads to external memory to:
   a. Obtain the device context base address array which contains the scratchpad buffer array base address.
   b. Obtain the scratchpad buffer array.
   c. Read the saved state, which is located across 1 or more scratchpad buffers, into a contiguous area of the internal memory.
8. The controller restores its state from internal memory.
9. The controller sets RSS to '0' indicating that it has completed restoring its internal state.
10. Software sets RunStop to '1' to start the controller
11. The controller sets HCHalted to '0' to indicate the controller is no longer halted.

6.1.5.4 PHY Wakeup from D3cold

When the PHY wakes up the controller by asserting PME, the example is the same as the previous one, with the exception that a PME is asserted which causes the wakeup.

Figure 6-8 is a logical waveform that illustrates how the PMU drives the ULPI interface to bring the 2.0 PHY out of suspend.

Figure 6-8 ULPI Wakeup



1. The PMUs and controller are in D3, `Vcc` is removed. If the PMUs are not maintaining a connection (`connect_state_u2pmu=0`), `suspend_clk` can be turned off.
2. The ULPI PHY asserts an interrupt indicating some activity has occurred on the PHY. At the same time, the u2pmu generates a PME asynchronously.
3. The power controllers starts `suspend_clk`, applies `Vcc`, de-asserts `vcc_reset_n`, and requests a power state transition to D0.
4. In this step, the controller is restoring the sticky state information from the PMUs.
5. The sticky state restoration has completed, and the PMUs report that their current power state is D0. The PMU de-asserts its PME signal.
6. The controller asserts the STP signal to the PHY so that the PHY can start its clock.
7. The PHY is attempting to start its clock.
8. The PHY clock is valid.
9. The PHY de-asserts DIR.
10. The controller de-asserts STP.

For a more detailed diagram that also contains applicable timing parameters, see “ULPI Wakeup” on page 428.

6.2 Programming DWC_usb3 for Hibernation in Device Mode

The device software may only start the hibernation process in the following situations:

While the device controller is running (`DCTL.RunStop=1`):

- No connection to the host occurs;
- While connected to the host, software receives a Hibernation Request event;
- While connected to the host, software receives a Disconnect event.

The programming steps described in this Device Programming Model section are required by whoever is implementing the device driver. A reference driver is provided with the product, however, you may write your own device driver or modify the example to your needs.

It is most likely that you will carry out these programming steps by writing C code. The typical process is to combine your C code driver along with the Synopsys IP controller in an FPGA platform to test them together?.

The device hibernation programming model relies on software to save the transfer state and endpoint state from the controller. The transfer state is saved when software ends transfers that are in progress, as the controller writes back the partially completed TRB. Endpoint state is saved when software issues the Get Endpoint State command.

When coming out of hibernation, software resumes those transfers by reconfiguring the endpoints, restoring the saved endpoint state, and starting the transfers up again. This is similar to the host operational model which also requires software to issue a Stop Endpoint command before saving the controller's state and resuming the transfers by ringing their doorbells.

Before entering hibernation, software stops the controller using the DCTL.RunStop and `DCTL.KeepConnect` bits. There are four combinations of these bits that request the controller to take a particular action as summarized in Table 6-1.

Table 6-1 DCTL.RunStop and KeepConnect Effects (Continued)

RunStop	KeepConnect	Description
1	1	Normal operating mode where the controller attempts to connect to the host or maintain a connection while in suspend state. In this mode, the Hibernation Request Event is issued when the link goes into a state that supports hibernation, and software intervention (which writes “Recovery” into the `DCTL.USBLnkSt` field) is required to exit that state, even if the host initiates an exit.
1	0	Normal operating mode where the controller attempts to connect to the host or maintain a connection while in suspend state. In this mode, the Hibernation Request Event is not generated and the controller exits U3, L2, or L1 if the host initiates an exit without software intervention.
0	1	Only applicable after software receives the Hibernation event. This instructs the controller to disable event generation, prepare for subsequent hibernation entry commands, but remain connected to the host. This is the state the controller is in after exiting D3 and entering D0.
0	0	Soft disconnect - This request is applicable after software receives the Disconnect event, decides to forcefully disconnect from a connected host, or decides to enter hibernation before the host has connected.

6.2.1 Initializing DWC_usb3 to Support Hibernation in Device Mode

Software prepares the controller to support hibernation using the following sequence of steps which are performed during “Device Power-On or Soft Reset” Initialization as described in “Device Power-On or Soft Reset”.

1. Read the `DWC_USB3_EN_PWROPT` field in the `GHWPARAMS1` register to determine if the controller supports hibernation. If the value is equal to 2, the controller supports hibernation.
2. Read `GHWPARAMS4` to determine how many scratchpad buffers it requires.

If the scratchpad buffer is implemented in a small SRAM, software may use the following equation to determine how many 32-bit DWORDs are required: 5 + `DWC_USB3_HOST_NUM_U3_ROOT_PORTS

3. Prepare the scratchpad buffer array with pointers to 1 or more scratchpad buffers.
4. Set `GCTL.GblHibernationEn` to 1.
5. To support L1, set `DCTL.HIRD_Thres[4]` to 1, and `DCTL.HIRD_Thres[3:0]` to an appropriate value.
   a. Set `HIRD_Thres` to a value large enough so that hardware and software have enough time to enter and exit hibernation during the host initiated resume duration. This is likely the largest value 4'b1110 (1ms).

      When set to this value, only an LPM token with a HIRD value of 1.2ms causes a hibernation event. The PMUs and the controller have filtering logic which assume that the PHY activity that lasts less than about 61 microseconds is a glitch, so it does not set `HIRD_Thres` to '0' because that represents a HIRD value of 50 microseconds.
6. To support L1 hibernation, set `DCTL.L1HibernationEn` to 1.

   Enabling L1 Hibernation prevents the device from exiting L1 until the software has completed the restore process and sets the link state to U0.
7. Issue a “Set Scratchpad Buffer Array” device generic command to assign the external memory location of the scratchpad buffer array.
8. Ensure that `GUSB2PHYCFG[6]` (Suspend 2.0 PHY) and `GUSB3PIPECTL[17]` (Suspend SS PHY) are set to '1'.

   While the controller is in hibernation, the U3PMU drives the `PowerDown` signal to the SS PHY with '3'. Therefore, the PHY must support the ability to be suspended.
9. Perform the connection to the host and transfers normally as described in this guide. In addition to setting `DCTL.RunStop` to '1', also set DCTL.KeepConnect to '1'.

6.2.2 Entering Hibernation in Device Mode While Connected

After the initialization described in "Initializing DWC_usb3 to Support Hibernation in Device Mode" on page 418, the controller generates an event when the link enters a hibernation-capable state.

1. Software receives a Hibernation Request event when the host puts the link into U3 or L2, or L1 with a large enough HIRD value.
2. Before proceeding to the next step, the software must ensure the following:
   a. Any events that are received before the hibernation event are handled.
   b. If an event is received for the default control endpoint 0, which puts the software into the "Set up a Control-Setup TRB/Start Transfer" state (see "Control Transfer Programming Model" on page 364"), prepares and issues a StartXfer pointing to a new Setup TRB.
3. Issue an End Transfer endpoint command for all active transfers with the `ForceRM` field set to 0, including the default control endpoint 0.

   The controller write backs any TRBs that were in progress at this point, setting the `BUFSIZ` and `BPTR` fields to reflect where the transfer left off, and also setting `TRBSTS` to '4' and `HWO` to 0. If the currently active TRB is a Link TRB, the controller sets `TRBSTS` to '4' but preserves `HWO` as '1'.
4. Issue a Get Endpoint State endpoint command for each active endpoint, and save the bits that are returned for use after coming out of hibernation. For more details, refer to "Command 3: Get Endpoint State (DEPGETSTATE)" on page 319".

   `DEPCMDPAR2` returns 32 bits of the endpoint state. In addition, software must remember if the endpoint is currently in a halted state. The endpoint is in a halted state if software has issued a "Set STALL" command and has not issued a "Clear STALL" command.
5. Set `DCTL.RunStop` to '0', `DCTL.KeepConnect` to '1', and wait for `DSTS.Halted` to be set to '1'.

   Software must acknowledge (by writing to `GEVNTCOUNTn`) any events that are generated while it is waiting for Halted to be set to '1'. Because the controller is in the process of entering hibernation, it need not process these events, but only acknowledge them.

   The controller completes any pending DMA before setting Halted to '1'. Even if the host is disconnected at this point, the controller does not generate any events, and only saves this information in the DSTS register.

   After setting `DCTL.RunStop` to '0', software can check the link state, and if it is disconnected then it must set `DCTL.KeepConnect` to '0' to force disconnect. In general, if the device is going into hibernation while disconnected, the `DCTL.KeepConnect` bit must be set to '0' to avoid reconnecting automatically.
6. Save D* and G* registers. The controller software restores most D* and G* registers as part of the re-initialization that is performed after exiting hibernation. If there are any fields that software had changed during operation (such as the `DCFG.DevAddr field`), software can retrieve their values in this step. For example:
   a. D* registers (`DCTL`, `DCFG`, `DEVTEN`)
   b. G* registers (`GSBUSCFG0/1, GCTL, GTXTHRCFG, GRXTHRCFG, GTXFIFOSIZn, GRXFIFOSIZ0, GUSB3PIPECTL0, GUSB2PHYCFG0`)

   If the software already knows their values, this step is not necessary.

Some bits within sticky registers such as GUSB2PHYCFG, GUSB3PIPECTL, GCTL, and DCTL are automatically saved and restored when power is re-applied. It is safe to write these registers again.

7. Set the DCTL.CSS bit and wait for the save state process to complete by polling for DSTS.SSS to equal '0'.
8. Communicate with the power controller to set the power state to D3. Waits until both PMUs confirm that they have entered D3.

   This switches the PHY MUXes to the PMUs. Now if the host tries to resume, the PME is asserted.
9. Remove controller well power.

6.2.3 Entering Hibernation in Device Mode While Disconnected

After the controller has been initialized and started by the device driver, there are a few scenarios where software may want to put the controller into a low power state while it is disconnected from the host:

- If the device is connected to the host and wants to perform a device-initiated disconnect.

  In this case, the PME is generated as soon as the controller goes to D3 state (as the host indicates a connect). The software can ignore the PME if it wants to be in a disconnected state and wake up only when it is ready to connect.
- If the device is connected to the host and receives a “Disconnect Detected” event indicating that the host is disconnected.
- If the device driver initializes the controller and attempts to connect to a host (by setting DCTL.RunStop to '1'), but does not achieve a connection.
- On initial start-up, if the driver wants to use the PMUs to detect the connection instead of attempting to connect to the host.

  In this case, the software initializes the controller and sets DCTL.RunStop to '1' as if it were attempting to connect to the host, followed by setting DCTL.RunStop to '0' as described as follows. These steps are required to prime the PMUs so that they are prepared to generate a PME on a wakeup event.

In the situations described in the previous bullets, the software’s save/restore functions are not used; however, the controller can be put into a low power state, Vcc removed, and the Power Controller can receive a wakeup request when a connect occurs. The software performs the following sequence of operations after “Initializing DWC_usb3 to Support Hibernation in Device Mode” on page 418.

1. Issue an End Transfer command with ForceRM=0 or 1 for any transfers that have been started.

   If ForceRM is set to ‘1’, the controller does not write back any TRBs that were in progress.
2. Before clearing the DCTL.RunStop to 0, polls the DSTS register and makes sure the DSTS[USBLnkSt] is 4 (SS_DIS).
3. Set DCTL.RunStop to '0', DCTL.KeepConnect to '0', and waits for DSTS.Halted to be set to '1'. The software must acknowledge (by writing to GEVNTCOUNTn) any events that are generated while it is

waiting for Halted to be set to '1'. Because the controller is in the process of entering hibernation, it does not need to process these events, only acknowledge them.

Setting `KeepConnect` to '0' prevents the controller from attempting to reconnect to the host when `DCTL.RunStop` is set to '0'. This flow is similar to the "Device-Initiated Disconnect" on page 339.

4. Communicate with the power controller to set the power state to D3. Waits until both PMUs confirm that they have entered D3.
5. Remove controller well power.

The PMUs are active and generate a PME if a wakeup event (such as connect) occurs. For a description of wakeup from this state, see "Exiting Hibernation in Device Mode While Disconnected" .

6.2.4 Exiting Hibernation in Device Mode While Connected

A wakeup can occur either because a PMU is requesting wakeup (through the PME interrupt), or because the software wants to start the controller to perform a remote wakeup to the host. In either case, software exits hibernation using the following sequence:

1. Enable controller well power.

   The power controller de-asserts `vcc_reset_n` to the controller. The sticky module state is also restored at this time.
2. Communicate with the power controller to set the power state to D0. Waits until both PMUs confirm that they have entered D0.

   This switches the PHY MUXes back to the controller.
3. If the power on initialization values of the `GSBUSCFG0` and `GSBUSCFG1` registers (`DWC_USB3_GSBUSCFG0_INT` and `DWC_USB3_GSBUSCFG1`) are not the same as the normal system operating value, then re-program these registers with the characteristics of the system bus.
4. Issue a "Set Scratchpad Buffer Array" device generic command and wait for completion by polling the `DGCMD.CmdAct` bit.
5. Write '1' to `DCTL.CRS` to start the restore process and wait for completion by polling the `DSTS.RSS` bit.
6. Configure the controller as described in "Device Power-On or Soft Reset" excluding the first step (Soft Reset). While issuing the `DEPCFG` commands for EP0 and EP1, writes the restore state information into `DEPCMDPAR2` and sets the Config Action to "Restore" to restore the control transfer state information. In this step, most of the D* and G* registers that may have been saved while entering hibernation are restored.

   When the software issued the `EndXfer` command with the `ForceRM` field set to '0' for the default control endpoint (physical endpoints 0 and 1) before entering hibernation, the controller may have written back an active TRB for the transfer, setting the `HWO` bit to '0'. The software must ensure that the TRB is valid and set the `HWO` bit back to '1' before re-starting the transfer in this step. The software must re-start the control transfer at the phase where it left off before hibernation. For example, if the control transfer was in the data phase before hibernation, the software issues the `StartXfer` command pointing to the data phase TRB, not a setup phase TRB.
7. Set `DCTL.RunStop=1` and `DCTL.KeepConnect=1` (while keeping other `DCTL` bits intact, that is, do a read-modify-write).

8. Poll the DSTS register until DSTS.DCNRdy is ‘0’. Read the DSTS.USBLnkSt field to see the current link state.
   a. If the value is ‘3’, or '2' (in USB 2.0 mode), the connection is still active but the link is suspended. Continue to Step 10.
   b. If the value is ‘15’, the connection is still active and the host issued a resume request. Continue to Step 10.

   Note If this is a SuperSpeed connection, it is possible that this link state is shortly followed by a USB Reset event because a USB Reset in SuperSpeed initially looks like a resume on the link.

   c. If the value is ‘14’, a USB reset was received. The software must write Resume (8) into the DCTL.ULStChngReq field immediately and then continue to Step 9.
      - Also reset the DCFG[DevAddr] field to 0 (zero).
   d. If any other value is seen, a disconnect or any other event is received. Continue to Step 9.
9. Waits for a USB Reset or Connect Done event as specified at the end of “Device Power-On or Soft Reset” on page 334, which does not follow the rest of the steps here.

   The rest of the steps assume that the controller is coming out of hibernation with the existing connection with the host.
10. Perform the steps in section “Initialization on Connect Done” on page 336.

    In this step, endpoint 0 is configured with the correct MaxPacketSize, depending on the connect speed.
11. Perform the steps in section “Initialization on SetConfiguration or SetInterface Request” on page 337.
    a. While issuing the DEPCFG commands, writes the restore state information into DEPCMDPAR2 and sets the Config Action to “Restore” to restore each endpoint's sequence number and flow control state to the value read during the save. For more details, see “Register Descriptions” on page 17.
    b. If the endpoint was in the Halted state before entering hibernation, the software must issue the Set STALL endpoint command to put the endpoint back into the Halted state.

    During this step, the software re-configures the existing endpoints and starts their transfers. When software issued the EndXfer command with the ForceRM field set to '0' before entering hibernation, the controller may have written back an active TRB for the transfer, setting the HWO bit to '0' and TRBSTS to '4'. The software must ensure that the TRB is valid and set the HWO bit back to '1' before re-starting the transfer in this step.
12. Set DCTL.ULStChngReq to '8' as described in the “DWC_usb3_block_dev Registers” on page 216.

    If the host initiated resume, this step completes the transition to U0. If the host did not initiate resume, this step causes the device to initiate resume (remote wakeup).
13. The controller is now connected to the host and continues with the transfers that are set up.

    At this point, software may process any events that occurred after Step 7, such as a new Disconnect, USB Reset, or Connect Done event. These events must be handled as specified in the“Programming DWC_usb3 in Device Mode” on page 333.

The entire process of saving and restoring must take no more than 10ms (see “Timing Dependencies” on page 426) when operating at 2.0 speeds coming out of L2.

If hibernation was initiated for an L1 entry, the process must take no more than 1.2ms (the maximum HIRD value). If LPM Errata is enabled, then the process must not take more than 10ms (the maximum BESL value).

In USB 3.0 mode, if the process takes more than 10ms, the host might give up and stop trying to wake up the link. However, in this case, the host tries again in 100ms, so the device software must wait at least 500ms before attempting to hibernate again to give the host a chance to resume the link.

After software sets `RunStop=1`, the controller may generate a Resume/Remote Wakeup Detected event. This event may be ignored because software has already determined that the controller has woken up.

6.2.5 Exiting Hibernation in Device Mode While Disconnected

If the controller is in hibernation while the host is disconnected (after following the steps in the section "Entering Hibernation in Device Mode While Disconnected" on page 420), the PMUs generate a PME if a connection occurs. If this happens, the preferred mechanism for software to resume is as follows:

Following are steps that you must perform:

1. Enable controller well power.
2. Communicate with the power controller to set the power state to D0. Waits until both PMUs confirm that they have entered D0.
3. Configure the controller as described in "Device Power-On or Soft Reset" on page 334.
4. Continue following the steps in this databook to prepare the controller to connect to the host and prepare transfers.

   A Controller Restore State request is not made because the state is not saved as part of the disconnect sequence described in "Entering Hibernation in Device Mode While Disconnected" on page 420.

   In addition, because the control transfer state is not saved, the software must reset its internal tracking of the control transfer programming model by preparing the Setup TRB as described in "Control Transfer Programming Model" on page 364. The software must do this regardless of the control transfer state before the disconnect that led to the controller entering hibernation.

Another mechanism that you can use to perform a wake up is to perform a cold reset by asserting `vcc_reset_n` and `vaux_reset_n`. However, note the following:

- This resets the controller, the PMUs, and the PHYs.
- Sticky state information stored in the PMUs (such as the `PORTSC` registers) is lost.
- The initialization required when using this method is the same as the normal "Device Power-On or Soft Reset" on page 334, including a soft reset at the beginning of initialization.

6.2.6 Device Timing Examples

The examples in "Host Timing Examples" on page 412 also apply to device mode with the following substitutions:

- `USBCMD.RunStop = DCTL.RunStop`
- `USBSTS.HCHalted = DSTS.Halted`
- `USBCMD.CSS = DCTL.CSS`
- `USBSTS.SSS = DSTS.SSS`
- `USBCMD.CRS = DCTL.CRS`
- `USBSTS.RSS = DSTS.RSS`

6.3 PHY-Initiated Hibernation Exit Event for Host and Device Modes

While the controller is in D3, the PMU may generate a PME due to events that happen on the PHY interface. The type of wakeup events depends on the PHY interface and whether the controller entered hibernation in host or device mode.

Table 6-2 PHY Wakeup Events

U3PMU		U2PMU			
Host	**Device**	**Host ULPI**	**Device ULPI**	**Host UTMI**	**Device UTMI**
connect	connect	connect	[1]	connect	connect
disconnect	disconnect	disconnect	[2]	disconnect	disconnect
overcurrent		overcurrent[3]		overcurrent	
resume	resume/ USB reset[4]	resume	resume	resume	resume
		[5]	[6]	ID change	ID change
		SRP dline		SRP dline	
		ULPI interrupt	ULPI interrupt		
			USB reset		USB reset

1. In a ULPI PHY, connect is signaled via ULPI interrupt
2. In a ULPI PHY, disconnect is signalled via ULPI interrupt
3. In a ULPI PHY, overcurrent may be signaled via ULPI interrupt or via the sideband overcurrent signal
4. In SuperSpeed, a USB reset initially looks like a resume
5. In a ULPI PHY, ID change is signaled via ULPI interrupt
6. In a ULPI PHY, ID change is signaled via ULPI interrupt

Notes:

- Connect also means “cold attach” in SuperSpeed USB.

  There are three conditions (underlined in the previous table) where the wakeup condition may be masked off based on configuration bits programmed by software. Those conditions are as follows in host mode:

- `PORTSC.WCE` controls whether the PMU asserts PME when a connection occurs.
- `PORTSC.WDE` controls whether the PMU asserts PME when a disconnect occurs.
- `PORTSC.WOE` controls whether the PMU asserts PME when an overcurrent occurs. However, in host mode with a ULPI PHY, if the overcurrent is signaled through the ULPI interrupt, the overcurrent cannot be masked off with WOE.

6.4 Timing Dependencies

The USB 2.0 Specification [USB2] and USB 3.0 Specification [USB3] place limits on how quickly a host/device must respond to activities on the bus. These requirements translate into the behavior of the power controller for how quickly it restores Vcc and provides a valid clock, and the software for how quickly it initiates a restore operation. Table 6-3 provides a summary of some of the key timings and their values.

Table 6-3 Timing Dependencies

Requirement	Reference
When device initiates resume, the host must reflect resume downstream within 1ms	[USB2] Section 7.1.7.7 “The controlling hub must rebroadcast the resume signaling within 1 ms (TURSM)”
The host must signal resume for at least 20ms	[USB2] Section 7.1.7.7 “must send the resume signaling for at least 20 ms” (TDRSMDN)
The host must send bus traffic within 3ms	[USB2] Section 7.1.7.7 “the host must begin sending bus traffic (at least the SOF token) within 3 ms of the start of the idle state”
When device initiates resume, the host must signal LFPS within 10ms	[USB3] Section 7.5.9.2 “The port shall remain in U3 when the 10-ms LFPS handshake timer times out and a successful LFPS handshake meeting the U3 wakeup handshake signaling in Section 6.9.2 is not achieved.”
When device initiates resume, if the host does not signal LFPS within 10ms, the device may try again in 100ms	[USB3] Section 7.5.9.2 “The port may initiate U3 wakeup again after a minimum of 100-ms delay.”
The host does not access the device for 10ms after resume has completed (applies to 2.0 and 3.0). However, we have noted that some hosts do not respect this requirement.	[USB2] Section 7.1.7.7 “The USB System Software must provide a 10 ms resume recovery time (TRSMRCY) during which it does not attempt to access any device connected to the affected (just-activated) bus segment.”
USB reset wakes a device from the suspend state, and the host drives reset for a minimum of 10ms.	[USB2] Section 7.1.7.5 “The reset signaling must be driven for a minimum of 10ms (TDRST).”
The host does not access the device for 10ms after reset has completed. However, we have noted that some hosts do not meet this requirement	[USB2] Section 7.1.7.5 “devices must be able to accept a SetAddress() request (refer to Section 11.24.2 and Section 9.4 respectively) after the reset recovery time 10 ms (TRSTRCY) after the reset is removed.”
After the host drives USB reset, the device must respond with a chirp within 6ms (the device must chirp for at least 1ms, and the host expects the device to finish its response within 7ms).	[USB2] Section 7.1.75 “4. The high-speed device leaves the D+ pull-up resistor connected, leaves the high-speed terminations disabled, and drives the high-speed signaling current into the D- line. This creates a Chirp K on the bus. The device chirp must last no less than 1.0 ms (TUCH) and must end no more than 7.0 ms (TUCHEND) after high-speed Reset time T0.”
When the host initiates exit from L1, the device must be prepared to receive traffic between 60us and 1.2ms, depending on the HIRD value.	[LPM-ECN] Section 4.9 Table 4-6: L1 Exit Latency (Host initiated) TL1ExitLatency1 Min: 60us Max: 1210us

The most stringent timings are related to reset during suspend, so if these timings are met, the power controller and software are implicitly meeting the timings for connect and disconnect.

Table 6-4 provides the resume timing requirements measured from the hibernation request event, including the save, hibernate, and resume process. Values with a "*" are recommended, but values with a footnote are required.

Table 6-4 Resume from L2/U3 Timing Requirements

Resume from L2/L3	Host		Device	
	USB 2.0	USB 3.0	USB 2.0	USB 3.0
Vcc Valid	100us[1]	1ms*	1ms*	1ms*
bus_clk Valid	5ms*	5ms*	5ms*	5ms*
Software Restore	23ms[2]*	10ms + 100i ms[3]	6-16ms[4] 20-30ms[5]	10ms + 100i ms[6]

1. If the device initiates wakeup, the host must reflect resume within 1ms. The sticky state restoration may take up to 900us, so Vcc should be valid within 100us
2. The host must reflect resume within 1ms, signal resume for at least 20ms, and begin sending bus traffic within 3ms. If software restore takes longer, the resume signaling will be greater than 20ms, which is still allowed.
3. In xHCI, software responds to the device-initiated resume by setting PORTSC.PLS to 0. According to the USB 3.0 spec, the host needs to start signaling LFPS within 10ms of the device initiating wakeup. If the 10ms timing is missed, the device may try again after 100ms.
4. This is a scenario where the host signals reset immediately after entering suspend. The device must respond within 6ms as described in the timing dependencies table. Software imposes a 10ms reset recovery time that is not always respected by host drivers. For this recovery, only the default control endpoint needs to be configured and DCTL.ULStChngReq must be written with Resume (8) within this time. This situation is unlikely, but allowed by the specification.
5. When the host initiates resume, it signals resume for 20ms and then software imposes an additional 10ms resume recovery time before accessing the device. The resume recovery time is not always respected by host drivers.
6. In the device programming model, software responds to the host-initiated resume by setting DCTL.ULStChngReq to 8, and the device should start signaling LFPS within 10ms of the host initiating wakeup. If software does not respond in time, the host may try to resume again after 100ms.

If device software decides to enter hibernation in the LPM-L1 state, it has much stricter resume requirements as given in Table 6-5. These requirements include the amount of time it took for software to enter hibernation (measured from when the controller issued the Hibernation Request event).

Table 6-5 Resume from L1 Timing Requirements

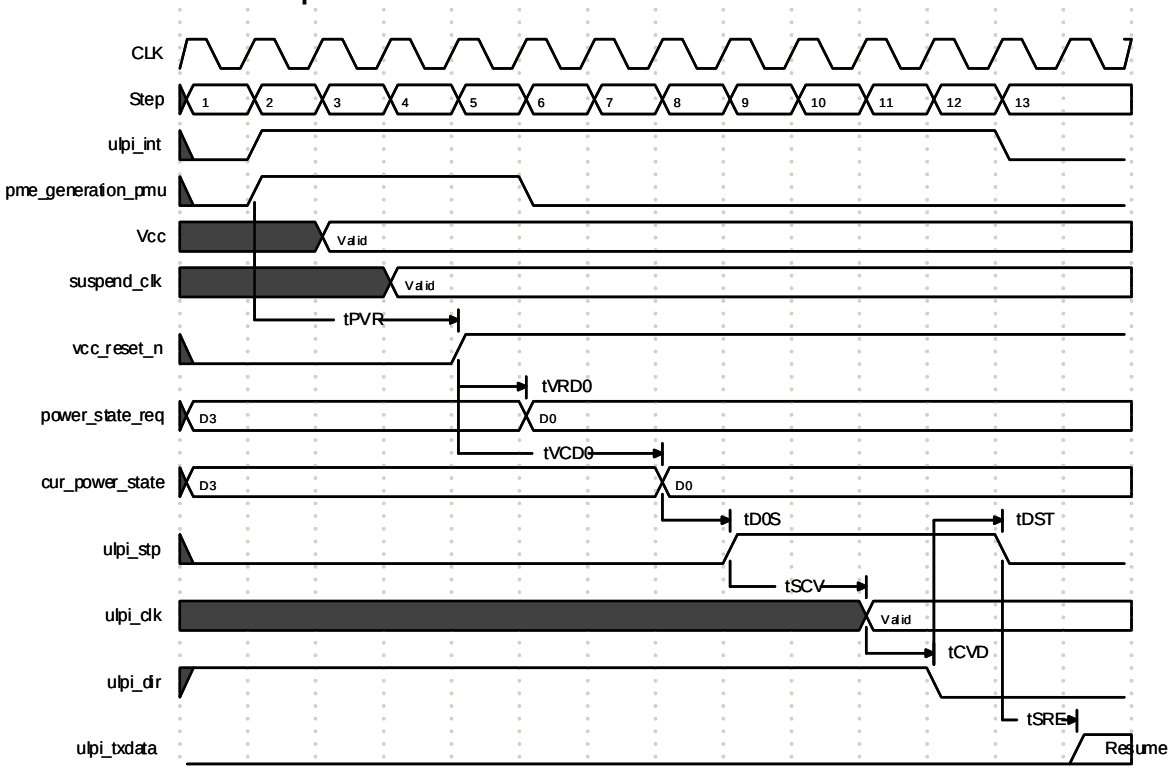
Resume from L1	Device 2.0
Vcc Valid	100us*
bus_clk Valid	100us*
Software Restore	1.2ms[1]

1. The maximum HIRD value advertised by the host is 1.2ms

6.4.1 ULPI Wakeup

Figure 6-9 illustrates the time it takes to wakeup a ULPI PHY from a hibernating state. This is applicable to hibernation with or without a connection, and is also applicable to host or device mode. If the PMU is in a low power state with a connection, the suspend_clk must be valid.

Figure 6-9 Detailed ULPI Wakeup



It takes about 25 `suspend_clk` cycles after `suspend_clk` is made valid for the controller to finish the sticky restore process, which means that in host mode the resume reflection (which is required to happen within 1ms) occurs in: `tPVR + tVCD0 + tD0S + tSCV + tCVD + tDST + tSRE`

where each parameter is described in Table 6-6.

Table 6-6 ULPI Wakeup Timing Parameters

Name	Comment	Value	Source
tPVR	PME to vcc_reset_n de-assertion		Power controller parameter
tVRD0	vcc_reset_n deassertion to request D0		Power controller parameter
tVCD0	vcc_reset_n deassertion to current D0	max(tVRD0, 25*suspend_clk_period)	Power controller/RTL parameter
tD0S	current power state D0 to STP assertion	2*suspend_clk_period	RTL parameter
tSCV	STP to ulpi_clk valid		PHY parameter
tCVD	ulpi_clk valid to ulpi_dir de-assertion		PHY parameter
tDST	ulpi_dir de-assertion to ulpi_stp de-assertion	1 * ulpi_clk_period	ULPI spec
tSRE	ulpi_stp de-assertion to resume signaling	19*ulpi_clk_period	RTL parameter

A

Internal Parameter Descriptions

Provides a description of the internal parameters that might be indirectly referenced in expressions in the Signals, Parameters, or Registers chapters. These parameters are not visible in the coreConsultant GUI and most of them are derived automatically from visible parameters. **You must not set any of these parameters directly.**

Some expressions might refer to TCL functions or procedures (sometimes identified as **function_of**) that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Table A-1 Internal Parameters

Parameter Name	Equals To
CMPLY	4'ha
D0	0
D3	3
DWC_USB3_32BIT_REGS	0
DWC_USB3_ADDR_CACHE_BCU	((DWC_USB3_ADDR_PSQ + ((DWC_USB3_MODE ==1 && DWC_USB3_EN_DBC ==0) ? 0: DWC_USB3_PSQ_FIFO_DEPTH)) << DWC_USB3_MADDR_LO)
DWC_USB3_ADDR_DFQ	(DWC_USB3_ADDR_RIQ + DWC_USB3_RIQ_FIFO_ALL_DEPTH)
DWC_USB3_ADDR_DWQ	(DWC_USB3_ADDR_DFQ + DWC_USB3_DFQ_FIFO_ALL_DEPTH)
DWC_USB3_ADDR_EVQ	(DWC_USB3_ADDR_DWQ + DWC_USB3_DWQ_FIFO_ALL_DEPTH)
DWC_USB3_ADDR_PSQ	(DWC_USB3_ADDR_EVQ + DWC_USB3_EVQ_FIFO_ALL_DEPTH)

Parameter Name	Equals To
DWC_USB3_ADDR_RCSR_GLOBAL	DWC_USB3_ALIGN_MDW(DWC_USB3_ADDR_CACHE_BCU + DWC_USB3_CACHE_BCU_DEPTH)
DWC_USB3_ADDR_RCSR_HOST	(DWC_USB3_ADDR_RCSR_GLOBAL + DWC_USB3_RCSR_GLOBAL_DEPTH)
DWC_USB3_ADDR_RIQ	(DWC_USB3_ADDR_RXQ + DWC_USB3_RXQ_FIFO_ALL_DEPTH)
DWC_USB3_ADDR_RXQ	(DWC_USB3_ADDR_TXQ + DWC_USB3_TXQ_FIFO_ALL_DEPTH)
DWC_USB3_ADDR_TXQ	0
DWC_USB3_ASPACEWIDTH	4
DWC_USB3_BMU_BUSGM_DEPTH	8
DWC_USB3_BMU_LSP_DEPTH	4
DWC_USB3_BMU_PTL_DEPTH	= ((DWC_USB3_MDWIDTH ==128) ? 4: 8)
DWC_USB3_BURSTWIDTH	8
DWC_USB3_CACHE_BCU_DEPTH	(DWC_USB3_CACHE_BCU_ENTRY_SIZE * (DWC_USB3_NUM_TXFIFO_ACTIVE + DWC_USB3_NUM_RXFIFO))
DWC_USB3_CACHE_BCU_ENTRY_SIZE	0
DWC_USB3_CACHE_TOTAL_XFER_RESOURCES	= (DWC_USB3_NUM_EPS)
DWC_USB3_CC_HS_PHYSEL	((DWC_USB3_HSPHY_INTERFACE ==2) ? 1 : (DWC_USB3_HSPHY_INTERFACE ==3) ? DWC_USB3_GUSB2PHYCFG_INIT[function] : 0)
DWC_USB3_CSR_ACCESS_TIMEOUT	17'h1ffff
DWC_USB3_DATAINFOWIDTH	4
DWC_USB3_DEVDBC_OUT_NUMP	5'h4
DWC_USB3_DFDMA_CMD_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 3: DWC_USB3_MDWIDTH ==64 ? 2: 1)
DWC_USB3_DFQ_CMDS_DEV	8
DWC_USB3_DFQ_CMDS_HST	8
DWC_USB3_DFQ_CMDS_PER_FSLS_INSTANCE	((DWC_USB3_EN_SEPARATE_DESC_QUEUES ==1 && DWC_USB3_MODE ==2) ? DWC_USB3_MAX_CB(DWC_USB3_DFQ_FIFO_DEPTH_DEV / DWC_USB3_DFDMA_CMD_DEPTH, DWC_USB3_DFQ_CMDS_PER_FSLS_INSTANCE_INT): DWC_USB3_DFQ_CMDS_PER_FSLS_INSTANCE_INT)

Parameter Name	Equals To
DWC_USB3_DFQ_CMDS_PER_FSLS_INSTANCE_INT	(DWC_USB3_DFQ_CMDS_HST / 2)
DWC_USB3_DFQ_CMDS_PER_HS_INSTANCE	(DWC_USB3_DFQ_CMDS_HST / 2)
DWC_USB3_DFQ_CMDS_PER_SS_INSTANCE	DWC_USB3_DFQ_CMDS_HST
DWC_USB3_DFQ_DEPTH_PER_FSLS_INSTANCE	(DWC_USB3_DFQ_CMDS_PER_FSLS_INSTANCE * DWC_USB3_DFDMA_CMD_DEPTH)
DWC_USB3_DFQ_DEPTH_PER_HS_INSTANCE	(DWC_USB3_DFQ_CMDS_PER_HS_INSTANCE * DWC_USB3_DFDMA_CMD_DEPTH)
DWC_USB3_DFQ_DEPTH_PER_SS_INSTANCE	(DWC_USB3_DFQ_CMDS_PER_SS_INSTANCE * DWC_USB3_DFDMA_CMD_DEPTH)
DWC_USB3_DFQ_FIFO_ALL_DEPTH	(DWC_USB3_MODE ==0 ? DWC_USB3_DFQ_FIFO_DEPTH_DEV: DWC_USB3_EN_SEPARATE_DESC_QUEUES ==0 ? DWC_USB3_DFQ_FIFO_DEPTH_HST_ONEQ: DWC_USB3_NUM_FSLS_USB_INSTANCES * DWC_USB3_DFQ_DEPTH_PER_FSLS_INSTANCE + DWC_USB3_NUM_HS_USB_INSTANCES * DWC_USB3_DFQ_DEPTH_PER_HS_INSTANCE + DWC_USB3_NUM_SS_USB_INSTANCES_ACTIVE * DWC_USB3_DFQ_DEPTH_PER_SS_INSTANCE + DWC_USB3_EN_DBC_SRCSNK * DWC_USB3_DFQ_FIFO_DEPTH_DEV)
DWC_USB3_DFQ_FIFO_DEPTH	DWC_USB3_DFQ_FIFO_DEPTH_DEV
DWC_USB3_DFQ_FIFO_DEPTH_DEV	(DWC_USB3_EN_SCALED_DESC_QUEUE_DEPTHS==0 ? 16: DWC_USB3_DFQ_CMDS_DEV * DWC_USB3_DFDMA_CMD_DEPTH)
DWC_USB3_DFQ_FIFO_DEPTH_HST_ONEQ	(DWC_USB3_EN_SCALED_DESC_QUEUE_DEPTHS==0 ? 16: DWC_USB3_DFQ_CMDS_HST * DWC_USB3_DFDMA_CMD_DEPTH)
DWC_USB3_DWDMA_DEV_CMD_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 5: DWC_USB3_MDWIDTH ==64 ? 3: 2)
DWC_USB3_DWDMA_HST_CMD_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 6: DWC_USB3_MDWIDTH ==64 ? 3: 2)
DWC_USB3_DWQ_CMDS_DEV	10
DWC_USB3_DWQ_CMDS_HST	10
DWC_USB3_DWQ_CMDS_PER_FSLS_INSTANCE	((DWC_USB3_EN_SEPARATE_DESC_QUEUES ==1 && DWC_USB3_MODE ==2) ? DWC_USB3_MAX_CB(DWC_USB3_DWQ_FIFO_DEPTH_DEV / DWC_USB3_DWDMA_HST_CMD_DEPTH, DWC_USB3_DWQ_CMDS_HST / 2): DWC_USB3_DWQ_CMDS_HST / 2)

Parameter Name	Equals To
DWC_USB3_DWQ_CMDS_PER_HS_INSTANCE	(DWC_USB3_DWQ_CMDS_HST / 2)
DWC_USB3_DWQ_CMDS_PER_SS_INSTANCE	DWC_USB3_DWQ_CMDS_HST
DWC_USB3_DWQ_DEPTH_PER_FSLS_INSTANCE	(DWC_USB3_DWQ_CMDS_PER_FSLS_INSTANCE * DWC_USB3_DWDMA_HST_CMD_DEPTH)
DWC_USB3_DWQ_DEPTH_PER_HS_INSTANCE	(DWC_USB3_DWQ_CMDS_PER_HS_INSTANCE * DWC_USB3_DWDMA_HST_CMD_DEPTH)
DWC_USB3_DWQ_DEPTH_PER_SS_INSTANCE	(DWC_USB3_DWQ_CMDS_PER_SS_INSTANCE * DWC_USB3_DWDMA_HST_CMD_DEPTH)
DWC_USB3_DWQ_FIFO_ALL_DEPTH	(DWC_USB3_MODE ==0 ? DWC_USB3_DWQ_FIFO_DEPTH_DEV: DWC_USB3_EN_SEPARATE_DESC_QUEUES ==0 ? DWC_USB3_DWQ_FIFO_DEPTH_HST_ONEQ: DWC_USB3_NUM_FSLS_USB_INSTANCES * DWC_USB3_DWQ_DEPTH_PER_FSLS_INSTANCE + DWC_USB3_NUM_HS_USB_INSTANCES * DWC_USB3_DWQ_DEPTH_PER_HS_INSTANCE + DWC_USB3_NUM_SS_USB_INSTANCES_ACTIVE * DWC_USB3_DWQ_DEPTH_PER_SS_INSTANCE + DWC_USB3_EN_DBC_SRCSNK * DWC_USB3_DWQ_FIFO_DEPTH_DEV)
DWC_USB3_DWQ_FIFO_DEPTH	DWC_USB3_DWQ_FIFO_DEPTH_DEV
DWC_USB3_DWQ_FIFO_DEPTH_DEV	(DWC_USB3_EN_SCALED_DESC_QUEUE_DEPTHS==0 ? 32: DWC_USB3_DWQ_CMDS_DEV * DWC_USB3_DWDMA_DEV_CMD_DEPTH)
DWC_USB3_DWQ_FIFO_DEPTH_HST_ONEQ	(DWC_USB3_EN_SCALED_DESC_QUEUE_DEPTHS==0 ? 32: DWC_USB3_DWQ_CMDS_HST * DWC_USB3_DWDMA_HST_CMD_DEPTH)
DWC_USB3_EN_ADP	= 0
DWC_USB3_EN_DBC_SRCSNK	((DWC_USB3_EN_DBC ==1 & (DWC_USB3_HOST_NUM_U3_ROOT_PORTS > DWC_USB3_NUM_SS_USB_INSTANCES)) ? 1: 0)
DWC_USB3_EN_DBG_PORTS	0
DWC_USB3_EN_OTG	0
DWC_USB3_EN_OTG_SS	((DWC_USB3_EN_OTG == 2) ? 1 : 0)
DWC_USB3_EN_SCALED_DESC_QUEUE_DEPTHS	= (DWC_USB3_MDWIDTH<=64 ? 0: 1)
DWC_USB3_EVDMA_CMD_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 2: 1)
DWC_USB3_EVQ_FIFO_ALL_DEPTH	(DWC_USB3_NUM_EVQ_ACTIVE * DWC_USB3_EVQ_FIFO_DEPTH)

Parameter Name	Equals To
DWC_USB3_EVQ_FIFO_DEPTH	(DWC_USB3_EN_SEPARATE_DESC_QUEUES ==0 ? 0: (DWC_USB3_NUM_FSLS_USB_INSTANCES * DWC_USB3_DWQ_CMDS_PER_FSLS_INSTANCE + DWC_USB3_NUM_HS_USB_INSTANCES * DWC_USB3_DWQ_CMDS_PER_HS_INSTANCE + DWC_USB3_NUM_SS_USB_INSTANCES * DWC_USB3_DWQ_CMDS_PER_SS_INSTANCE) * DWC_USB3_EVDMA_CMD_DEPTH)
DWC_USB3_FSPHY_INTERFACE	0
DWC_USB3_GDBGLSP_RESET_CC	('h41 + ({DWC_USB3_HC_MAXINTRS{1'b1}} << 22))
DWC_USB3_GENERATION	30
DWC_USB3_HC_AC64	((DWC_USB3_AWIDTH ==64) ? 1 : 0)
DWC_USB3_HCACHE_BI_BANDWIDTH	(((DWC_USB3_HCACHE_DEVICE_ADDRESS_TABLE + DWC_USB3_HCACHE_DEVICE_ADDRESS_TABLE_SIZE + 15) >>4) <<4)
DWC_USB3_HCACHE_BI_BANDWIDTH_SIZE	(DWC_USB3_NUM_SS_USB_INSTANCES_ACTIVE*4) + (DWC_USB3_NUM_HS_USB_INSTANCES*4) + (DWC_USB3_NUM_FSLS_USB_INSTANCES*4)
DWC_USB3_HCACHE_DEVICE_ADDRESS_TABLE	(((DWC_USB3_HCACHE_SLOT_TABLE + DWC_USB3_HCACHE_SLOT_TABLE_SIZE + 15) >>4) <<4)
DWC_USB3_HCACHE_DEVICE_ADDRESS_TABLE_SIZE	16
DWC_USB3_HCACHE_DEVICE_BASE_ADDR	(((DWC_USB3_HCACHE_HERH_DATA + DWC_USB3_HCACHE_HERH_DATA_SIZE + 15) >>4) <<4)
DWC_USB3_HCACHE_DEVICE_BASE_ADDR_SIZE	(DWC_USB3_NUM_DEVICE_SUPT + 1) * 8
DWC_USB3_HCACHE_EPINFO_BITMAP	(DWC_USB3_HCACHE_HUB_CONTEXT_ADDR + DWC_USB3_HCACHE_HUB_CONTEXT_ADDR_SIZE)
DWC_USB3_HCACHE_EPINFO_BITMAP_SIZE	((DWC_USB3_NUM_DEVICE_SUPT + 1) * DWC_USB3_HCACHE_EPINFO_BITMAP_SIZE_PER_SLOT)
DWC_USB3_HCACHE_EPINFO_BITMAP_SIZE_PER_SLOT	16
DWC_USB3_HCACHE_EPTYPE_BITMAP	(DWC_USB3_HCACHE_EPINFO_BITMAP + DWC_USB3_HCACHE_EPINFO_BITMAP_SIZE)
DWC_USB3_HCACHE_EPTYPE_BITMAP_SIZE	((DWC_USB3_NUM_DEVICE_SUPT + 1) * DWC_USB3_HCACHE_EPTYPE_BITMAP_SIZE_PER_SLOT)

Parameter Name	Equals To
DWC_USB3_HCACHE_EPTYPE_BITMAP_SIZE_PER_SLOT	8
DWC_USB3_HCACHE_HERH_DATA	(DWC_USB3_HCACHE_BI_BANDWIDTH + DWC_USB3_HCACHE_BI_BANDWIDTH_SIZE + 12)
DWC_USB3_HCACHE_HERH_DATA_SIZE	(4 + 2*16 + 2*16)*(DWC_USB3_HOST_NUM_INTERRUPTER_SUPT + DWC_USB3_EN_DBC)
DWC_USB3_HCACHE_HUB_CONTEXT_ADDR	(DWC_USB3_ADDR_RCSR_HOST + DWC_USB3_RCSR_HOST_DEPTH)
DWC_USB3_HCACHE_HUB_CONTEXT_ADDR_SIZE	8
DWC_USB3_HCACHE_PERIODIC_EPINFO_SIZE	20
DWC_USB3_HCACHE_PERIODIC_LIST	(DWC_USB3_HCACHE_EPTYPE_BITMAP + DWC_USB3_HCACHE_EPTYPE_BITMAP_SIZE)
DWC_USB3_HCACHE_PERIODIC_LIST_SIZE	(DWC_USB3_HOST_NUM_PERIODIC_EP * DWC_USB3_HCACHE_PERIODIC_EPINFO_SIZE)
DWC_USB3_HCACHE_PERIODIC_SLOTINFO_SIZE	4
DWC_USB3_HCACHE_SCRATCH_PAD	(DWC_USB3_HCACHE_SLOT_ID + DWC_USB3_HCACHE_SLOT_ID_SIZE)
DWC_USB3_HCACHE_SLOT_ID	(DWC_USB3_HCACHE_DEVICE_BASE_ADDR + DWC_USB3_HCACHE_DEVICE_BASE_ADDR_SIZE)
DWC_USB3_HCACHE_SLOT_ID_SIZE	128
DWC_USB3_HCACHE_SLOT_LIST	(DWC_USB3_HCACHE_PERIODIC_LIST + DWC_USB3_HCACHE_PERIODIC_LIST_SIZE)
DWC_USB3_HCACHE_SLOT_LIST_SIZE	((DWC_USB3_NUM_DEVICE_SUPT + 1) * DWC_USB3_HCACHE_PERIODIC_SLOTINFO_SIZE)
DWC_USB3_HCACHE_SLOT_TABLE	(((DWC_USB3_HCACHE_SLOT_LIST + DWC_USB3_HCACHE_SLOT_LIST_SIZE + 15) >>4) <<4)
DWC_USB3_HCACHE_SLOT_TABLE_SIZE	((DWC_USB3_NUM_DEVICE_SUPT/32) + 1) * 4
DWC_USB3_HC_BNC	0
DWC_USB3_HC_CAPLENGTH	DWC_USB3_HOST_CAP_REG_LEN
DWC_USB3_HC_CFC	1
DWC_USB3_HC_CIC	1
DWC_USB3_HC_CMC	1
DWC_USB3_HC_CSZ	1

Parameter Name	Equals To
DWC_USB3_HC_CTC	1
DWC_USB3_HC_DBOFF	(DWC_USB3_HC_DOORBELL_ARRAY_OFFSET)
DWC_USB3_HC_DOORBELL_ARRAY_OFFSET	(DWC_USB3_HC_RUNTIME_REGISTER_SPACE_OFFSET) + 'h20 + ('h20* (DWC_USB3_HOST_NUM_INTERRUPTER_SUPT))
DWC_USB3_HC_ERSTMAX	15
DWC_USB3_HC_EXTCAP_SUPT_PRTL_20_OFFSET	(DWC_USB3_HC_EXTCAP_USBLEGSUP_OFFSET + 16)
DWC_USB3_HC_EXTCAP_USBLEGSUP_OFFSET	(DWC_USB3_HC_DOORBELL_ARRAY_OFFSET + (256*4))
DWC_USB3_HC_FSC	1
DWC_USB3_HC_HCIVERSION	'h110
DWC_USB3_HC_HCIVERSION_10	= (DWC_USB3_HC_HCIVERSION ==0x110 || DWC_USB3_HC_HCIVERSION ==0x100)
DWC_USB3_HC_IST	'h1
DWC_USB3_HC_LEC	0
DWC_USB3_HC_LHRC	1
DWC_USB3_HC_LTC	1
DWC_USB3_HC_MAXINTRS	DWC_USB3_HOST_NUM_INTERRUPTER_SUPT
DWC_USB3_HC_MAXPSASIZE	15
DWC_USB3_HC_MAXSLOTS	DWC_USB3_NUM_DEVICE_SUPT
DWC_USB3_HC_NSS	0
DWC_USB3_HC_PAE	0
DWC_USB3_HC_PIND	0
DWC_USB3_HC_PPC	1
DWC_USB3_HC_RTSOFF	(DWC_USB3_HC_RUNTIME_REGISTER_SPACE_OFFSET)
DWC_USB3_HC_RUNTIME_REGISTER_SPACE_OFFSET	(((DWC_USB3_HOST_CAP_REG_LEN + 'h400 + ((DWC_USB3_HOST_NUM_U3_ROOT_PORTS + DWC_USB3_HOST_NUM_U2_ROOT_PORTS) * 'h10) + 'h10) >> 5) << 5)
DWC_USB3_HC_SEC	1
DWC_USB3_HC_SPC	1

Parameter Name	Equals To
DWC_USB3_HC_SPR	1
DWC_USB3_HC_U1_DEVICE_EXIT_LATENCY	'ha
DWC_USB3_HC_U2_DEVICE_EXIT_LATENCY	'h7ff
DWC_USB3_HC_U3C	1
DWC_USB3_HC_XECP	((DWC_USB3_SMI_LEGACY_SUPT_EN == 1) ? (DWC_USB3_HC_EXTCAP_USBLEGSUP_OFFSET/4) : (DWC_USB3_HC_EXTCAP_SUPT_PRTL_20_OFFSET/4))
DWC_USB3_HIBER_SCRATCHBUFS	1
DWC_USB3_HOST_CAP_REG_LEN	'h20
DWC_USB3_HOST_NUM_U2_ROOT_PORTS_PIN	DWC_USB3_HOST_NUM_U2_ROOT_PORTS
DWC_USB3_HOST_NUM_U3_ROOT_PORTS_PIN	((DWC_USB3_EN_USB2_ONLY ==1) ? 0: DWC_USB3_HOST_NUM_U3_ROOT_PORTS)
DWC_USB3_HOST_RIQ_DEPTH_PER_FSLS_INSTANCE	(DWC_USB3_MODE ==2 ? (DWC_USB3_HOST_RXQ_DEPTH_PER_FSLS_INSTANCE + (DWC_USB3_PSQ_FIFO_DEPTH/2) + 2): (DWC_USB3_HOST_RXQ_DEPTH_PER_FSLS_INSTANCE + 2))
DWC_USB3_HOST_RIQ_DEPTH_PER_HS_INSTANCE	(DWC_USB3_HOST_RXQ_DEPTH_PER_HS_INSTANCE + 2)
DWC_USB3_HOST_RIQ_DEPTH_PER_SS_INSTANCE	(DWC_USB3_HOST_RXQ_DEPTH_PER_SS_INSTANCE + 2)
DWC_USB3_HOST_RXQ_DEPTH_PER_FSLS_INSTANCE	((DWC_USB3_MODE ==2) ? DWC_USB3_MAX_CB(DWC_USB3_HOST_RXQ_DEPTH_PER_FSLS_INSTANCE_HOST, DWC_USB3_RXQ_FIFO_DEPTH): DWC_USB3_HOST_RXQ_DEPTH_PER_FSLS_INSTANCE_HOST)
DWC_USB3_HOST_RXQ_DEPTH_PER_FSLS_INSTANCE_HOST	(2 * (1 * DWC_USB3_RXDMA_CMD_DEPTH + (DWC_USB3_HOST_NUM_CACHE_TRB_PER_FSLS_EP-1) * DWC_USB3_RXDMA_CMD_MSEG_FIFO_DEPTH))
DWC_USB3_HOST_RXQ_DEPTH_PER_HS_INSTANCE	((DWC_USB3_MODE ==2) ? DWC_USB3_MAX_CB(DWC_USB3_HOST_RXQ_DEPTH_PER_HS_INSTANCE_HOST, DWC_USB3_RXQ_FIFO_DEPTH): DWC_USB3_HOST_RXQ_DEPTH_PER_HS_INSTANCE_HOST)
DWC_USB3_HOST_RXQ_DEPTH_PER_HS_INSTANCE_HOST	(2 * (3 * DWC_USB3_RXDMA_CMD_DEPTH + (DWC_USB3_HOST_NUM_CACHE_TRB_PER_HS_EP-1) * DWC_USB3_RXDMA_CMD_MSEG_FIFO_DEPTH))

Parameter Name	Equals To
DWC_USB3_HOST_RXQ_DEPTH_PER_SS_INSTANCE	(2 * (16 * DWC_USB3_RXDMA_CMD_DEPTH + (DWC_USB3_HOST_NUM_CACHE_TRB_PER_SS_EP-1) * DWC_USB3_RXDMA_CMD_MSEG_FIFO_DEPTH))
DWC_USB3_HOST_TXFIFO_DEPTH_PER_FSLS_INSTANCE	((DWC_USB3_NUM_TXF_FSLS_PKTS * (DWC_USB3_MPS / DWC_USB3_MBYTES + 1)) + 1)
DWC_USB3_HOST_TXQ_DEPTH_PER_FSLS_INSTANCE	((DWC_USB3_MODE ==2) ? DWC_USB3_MAX_CB(DWC_USB3_HOST_TXQ_DEPTH_PER_FSLS_INSTANCE_HOST, DWC_USB3_TXQ_FIFO_DEPTH): DWC_USB3_HOST_TXQ_DEPTH_PER_FSLS_INSTANCE_HOST)
DWC_USB3_HOST_TXQ_DEPTH_PER_FSLS_INSTANCE_HOST	(2 * (1 * DWC_USB3_TXDMA_CMD_DEPTH + (DWC_USB3_HOST_NUM_CACHE_TRB_PER_FSLS_EP-1) * DWC_USB3_TXDMA_CMD_MSEG_FIFO_DEPTH))
DWC_USB3_HOST_TXQ_DEPTH_PER_HS_INSTANCE	((DWC_USB3_MODE ==2) ? DWC_USB3_MAX_CB(DWC_USB3_HOST_TXQ_DEPTH_PER_HS_INSTANCE_HOST, DWC_USB3_TXQ_FIFO_DEPTH): DWC_USB3_HOST_TXQ_DEPTH_PER_HS_INSTANCE_HOST)
DWC_USB3_HOST_TXQ_DEPTH_PER_HS_INSTANCE_HOST	(2 * (3 * DWC_USB3_TXDMA_CMD_DEPTH + (DWC_USB3_HOST_NUM_CACHE_TRB_PER_HS_EP-1) * DWC_USB3_TXDMA_CMD_MSEG_FIFO_DEPTH))
DWC_USB3_HOST_TXQ_DEPTH_PER_SS_INSTANCE	((2 + DWC_USB3_EN_ISOC_SUPT) * (16 * DWC_USB3_TXDMA_CMD_DEPTH + (DWC_USB3_HOST_NUM_CACHE_TRB_PER_SS_EP-1) * DWC_USB3_TXDMA_CMD_MSEG_FIFO_DEPTH))
DWC_USB3_HSPHY_DWIDTH	2
DWC_USB3_I2C_INTERFACE	0
DWC_USB3_MAC_PHY_CLKS_SYNC	0
DWC_USB3_MAC_RAM_CLKS_SYNC	0
DWC_USB3_MADDR_LO	((DWC_USB3_MDWIDTH == 32) ? 2: (DWC_USB3_MDWIDTH == 64) ? 3: 4)
DWC_USB3_MBYTES	(DWC_USB3_MDWIDTH / 8)
DWC_USB3_MPS	1024
DWC_USB3_NUM_EVQ_ACTIVE	(DWC_USB3_EN_SEPARATE_DESC_QUEUES ==1 ? DWC_USB3_HOST_NUM_INTERRUPTER_SUPT: 0)
DWC_USB3_NUM_FSLS_USB_INSTANCES	1

Parameter Name	Equals To
DWC_USB3_NUM_RXFIFO	(DWC_USB3_MODE ==0 ? 1: DWC_USB3_NUM_TXFIFO_HOST - DWC_USB3_EN_DBC)
DWC_USB3_NUM_RXFIFO_H	(DWC_USB3_EN_DBC_SRCSNK + DWC_USB3_NUM_SS_USB_INSTANCES + DWC_USB3_NUM_HS_USB_INSTANCES + DWC_USB3_NUM_FSLS_USB_INSTANCES)
DWC_USB3_NUM_SSIC_PORTS	= 0
DWC_USB3_NUM_SS_USB_INSTANCES_ACTIVE	((DWC_USB3_EN_USB2_ONLY ==1) ? 0: DWC_USB3_NUM_SS_USB_INSTANCES)
DWC_USB3_NUM_TOTAL_HST_USB_INSTANCES	1
DWC_USB3_NUM_TXFIFO	(DWC_USB3_MODE ==0 ? DWC_USB3_NUM_IN_EPS: DWC_USB3_MODE ==1 ? DWC_USB3_NUM_TXFIFO_HOST: DWC_USB3_NUM_TXFIFO_SDRD)
DWC_USB3_NUM_TXFIFO_ACTIVE	(DWC_USB3_MODE ==0 ? DWC_USB3_NUM_IN_EPS: DWC_USB3_MODE ==1 ? DWC_USB3_NUM_TXFIFO_HOST_ACTIVE: DWC_USB3_NUM_TXFIFO_SDRD_ACTIVE)
DWC_USB3_NUM_TXFIFO_H	DWC_USB3_NUM_TXFIFO_HOST
DWC_USB3_NUM_TXFIFO_HOST	(DWC_USB3_EN_DBC + (DWC_USB3_EN_DBC & (DWC_USB3_HOST_NUM_U3_ROOT_PORTS > DWC_USB3_NUM_SS_USB_INSTANCES)) + DWC_USB3_NUM_SS_USB_INSTANCES + DWC_USB3_NUM_HS_USB_INSTANCES + DWC_USB3_NUM_FSLS_USB_INSTANCES)
DWC_USB3_NUM_TXFIFO_HOST_ACTIVE	(DWC_USB3_EN_USB2_ONLY ==1 ? DWC_USB3_NUM_TXFIFO_HOST-1: DWC_USB3_NUM_TXFIFO_HOST)
DWC_USB3_NUM_TXFIFO_SDRD	DWC_USB3_MAX_CB(DWC_USB3_NUM_TXFIFO_HOST , DWC_USB3_NUM_IN_EPS)
DWC_USB3_NUM_TXFIFO_SDRD_ACTIVE	DWC_USB3_MAX_CB(DWC_USB3_NUM_TXFIFO_HOST _ACTIVE, DWC_USB3_NUM_IN_EPS)
DWC_USB3_NUM_U2_ROOT_PORTS	= ((DWC_USB3_MODE ==1 || DWC_USB3_MODE ==2) ? DWC_USB3_HOST_NUM_U2_ROOT_PORTS : 1)
DWC_USB3_NUM_U3_ROOT_PORTS	= ((DWC_USB3_MODE ==1 || DWC_USB3_MODE ==2) ? DWC_USB3_HOST_NUM_U3_ROOT_PORTS : (DWC_USB3_MODE ==3) ? (DWC_USB3_HUB_NUM_U3_PORTS + 1) : 1)
DWC_USB3_RAM_BUS_CLKS_SYNC	0

Parameter Name	Equals To
DWC_USB3_RCSR_DBC_DEPTH	(DWC_USB3_EN_DBC ? 32 : 0)
DWC_USB3_RCSR_GLOBAL_DEPTH	(16 * DWC_USB3_DEVICE_NUM_INT)
DWC_USB3_RCSR_HOST_DEPTH	(12*4 + 32*DWC_USB3_HOST_NUM_INTERRUPTER_SUPT + 4 *DWC_USB3_NUM_DEVICE_SUPT + DWC_USB3_RCSR_DBC_DEPTH)
DWC_USB3_REQINFOWIDTH	4
DWC_USB3_RIQ_FIFO_ALL_DEPTH	(DWC_USB3_MODE ==0 ? DWC_USB3_NUM_RXFIFO * DWC_USB3_RIQ_FIFO_DEPTH: DWC_USB3_NUM_FSLS_USB_INSTANCES * DWC_USB3_HOST_RIQ_DEPTH_PER_FSLS_INSTANCE + DWC_USB3_NUM_HS_USB_INSTANCES * DWC_USB3_HOST_RIQ_DEPTH_PER_HS_INSTANCE + DWC_USB3_NUM_SS_USB_INSTANCES_ACTIVE * DWC_USB3_HOST_RIQ_DEPTH_PER_SS_INSTANCE + DWC_USB3_EN_DBC_SRCSNK * DWC_USB3_RIQ_FIFO_DEPTH)
DWC_USB3_RIQ_FIFO_DEPTH	(DWC_USB3_RXQ_FIFO_DEPTH + (DWC_USB3_PSQ_FIFO_DEPTH/2) + 2)
DWC_USB3_RM_OPT_FEATURES	= (DWC_USB3_MBUS_TYPE ==4) ? 1 : 0
DWC_USB3_RXDMA_CMD_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 2: 1)
DWC_USB3_RXDMA_CMD_MSEG_FIFO_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 2: 1)
DWC_USB3_RXQ_FIFO_ALL_DEPTH	(DWC_USB3_MODE ==0 ? DWC_USB3_NUM_RXFIFO * DWC_USB3_RXQ_FIFO_DEPTH: DWC_USB3_NUM_FSLS_USB_INSTANCES * DWC_USB3_HOST_RXQ_DEPTH_PER_FSLS_INSTANCE + DWC_USB3_NUM_HS_USB_INSTANCES * DWC_USB3_HOST_RXQ_DEPTH_PER_HS_INSTANCE+ DWC_USB3_NUM_SS_USB_INSTANCES_ACTIVE * DWC_USB3_HOST_RXQ_DEPTH_PER_SS_INSTANCE+ DWC_USB3_EN_DBC_SRCSNK * DWC_USB3_RXQ_FIFO_DEPTH)
DWC_USB3_RXQ_FIFO_DEPTH	((DWC_USB3_DEV_EN_SCATTER_PACKETS_OF_8_TO_15_TRBS ==1) ? ((DWC_USB3_MDWIDTH ==32) ? 32: 16) : ((DWC_USB3_MDWIDTH ==32) ? 16: 8))
DWC_USB3_SMI_LEGACY_SUPT_EN	1
DWC_USB3_SSIC_GEAR	= 3
DWC_USB3_SSIC_NON_SNPS_MPHY	= 0
DWC_USB3_SSIC_NUM_LANE	= 1

Parameter Name	Equals To
DWC_USB3_SSPHY_INTERFACE	= (DWC_USB3_EN_USB2_ONLY ==1) ? 0 : 1
DWC_USB3_SYNC_RST	0
DWC_USB3_TXDMA_CMD_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 3: DWC_USB3_MDWIDTH ==64 ? 2: 1)
DWC_USB3_TXDMA_CMD_MSEG_FIFO_DEPTH	(DWC_USB3_MDWIDTH ==32 ? 2: 1)
DWC_USB3_TXQ_FIFO_ALL_DEPTH	(DWC_USB3_MODE ==0 ? DWC_USB3_TXQ_FIFO_DEVICE_DEPTH: DWC_USB3_MODE ==1 ? DWC_USB3_TXQ_FIFO_HOST_DEPTH: DWC_USB3_TXQ_FIFO_DRD_DEPTH)
DWC_USB3_TXQ_FIFO_DEPTH	((DWC_USB3_DEV_EN_SCATTER_PACKETS_OF_8_TO_15_TRBS ==1) ? ((DWC_USB3_MDWIDTH ==32) ? 32: 16) : ((DWC_USB3_MDWIDTH ==32) ? 16: 8))
DWC_USB3_TXQ_FIFO_DEVICE_DEPTH	(DWC_USB3_NUM_IN_EPS * DWC_USB3_TXQ_FIFO_DEPTH)
DWC_USB3_TXQ_FIFO_DRD_DEPTH	(DWC_USB3_TXQ_FIFO_HOST_DEPTH + ((DWC_USB3_NUM_IN_EPS > DWC_USB3_NUM_TXFIFO_HOST_ACTIVE) ? (DWC_USB3_NUM_IN_EPS - DWC_USB3_NUM_TXFIFO_HOST_ACTIVE) * DWC_USB3_TXQ_FIFO_DEPTH: 0))
DWC_USB3_TXQ_FIFO_HOST_DEPTH	(DWC_USB3_NUM_FSLS_USB_INSTANCES * DWC_USB3_HOST_TXQ_DEPTH_PER_FSLS_INSTANCE + DWC_USB3_NUM_HS_USB_INSTANCES * DWC_USB3_HOST_TXQ_DEPTH_PER_HS_INSTANCE+ DWC_USB3_NUM_SS_USB_INSTANCES_ACTIVE * DWC_USB3_HOST_TXQ_DEPTH_PER_SS_INSTANCE+ DWC_USB3_EN_DBC * DWC_USB3_TXQ_FIFO_DEPTH + DWC_USB3_EN_DBC_SRCSNK * DWC_USB3_TXQ_FIFO_DEPTH)
DWC_USB3_ULPI_CARKIT	0
GFLADJ_30MHZ	0
GFLADJ_30MHZ_SDBND_SEL	7
GUSB2PHYCFG_REG_DEF_CC	(DWC_USB3_GUSB2PHYCFG_INIT | ((DWC_USB3_HSPHY_INTERFACE == 2) ? 'h10 : 'h0) | ((DWC_USB3_MODE == 1) ? 'h140 : 'h0) | (((DWC_USB3_FREECLK_USB2_EXIST == 0) || (DWC_USB3_MODE == 0)) ? 'h0 : 'h40000000))
GUSB3PIPECTL_REG_DEF_CC	(DWC_USB3_MODE == 1) ? (DWC_USB3_GUSB3PIPECTL_INIT | 32'h00020000) : DWC_USB3_GUSB3PIPECTL_INIT

Parameter Name	Equals To
HCACHE_CRC_SAVE	HCACHE_NONLSP_SAVE + HCACHE_NONLSP_SAVE_SIZE
HCACHE_HALM_SAVE	HCACHE_HCMD_SAVE + HCACHE_HCMD_SAVE_SIZE
HCACHE_HALM_SAVE_DWORDS	((DWC_USB3_SSPHY_INTERFACE ==1) ? ((4*DWC_USB3_NUM_TOTAL_HST_USB_INSTANCES) + (DWC_USB3_NUM_SS_USB_INSTANCES) + 1) : ((4*(DWC_USB3_NUM_TOTAL_HST_USB_INSTANCES - DWC_USB3_NUM_SS_USB_INSTANCES)) + 1))
HCACHE_HALM_SAVE_SIZE	HCACHE_HALM_SAVE_DWORDS*4
HCACHE_HCMD_SAVE	DWC_USB3_HCACHE_SCRATCH_PAD
HCACHE_HCMD_SAVE_SIZE	(12 + DWC_USB3_NUM_DEVICE_SUPT + 1)*4
HCACHE_NONLSP_SAVE	HCACHE_HALM_SAVE + HCACHE_HALM_SAVE_SIZE
HCACHE_NONLSP_SAVE_SIZE	NSR_U3RHBPRTSM_SAVE_SIZE + NSR_PWRMCSR_SAVE_SIZE + NSR_CSRDEV_SAVE_SIZE
HCACHE_NUM_SP_BUFS	(HCACHE_SAVE_SIZE + (PAGE_SIZE-1)) / PAGE_SIZE
HCACHE_SAVE_BEGIN	DWC_USB3_HCACHE_HUB_CONTEXT_ADDR
HCACHE_SAVE_END	HCACHE_CRC_SAVE + NSR_CRC_SAVE_SIZE
HCACHE_SAVE_SIZE	HCACHE_SAVE_END - HCACHE_SAVE_BEGIN
HRESET	4'h9
LPBK	4'hb
NSR_CRC_SAVE_SIZE	4
NSR_CSRDEV_SAVE_SIZE	(TXRX_NO_OF_FIFOS + 2 + 7) * 4
NSR_PWRMCSR_SAVE_SIZE	(DWC_USB3_NUM_U3_ROOT_PORTS + DWC_USB3_NUM_U2_ROOT_PORTS + DWC_USB3_NUM_SSIC_PORTS + 6) * 4
NSR_U3RHBPRTSM_SAVE_SIZE	DWC_USB3_NUM_U3_ROOT_PORTS*4
P0	4'b0000
P1	4'b0001
P2	4'b0010
P3	4'b0011
P3CPM	4'b0100
P4	4'b1100

Parameter Name	Equals To
PAGE_SIZE	4096
POLL	4'h7
RECOV	4'h8
reg	
RX_DET	4'h5
SS_DIS	4'h4
SS_INACT	4'h6
TXRX_NO_OF_FIFOS	DWC_USB3_NUM_TXFIFO + DWC_USB3_NUM_RXFIFO
U0	4'h0
U1	4'h1
U2	4'h2
U3	4'h3