Elitestek® PCIe Scatter-Gather Direct Memory Access (SGDMA) IP User Guide

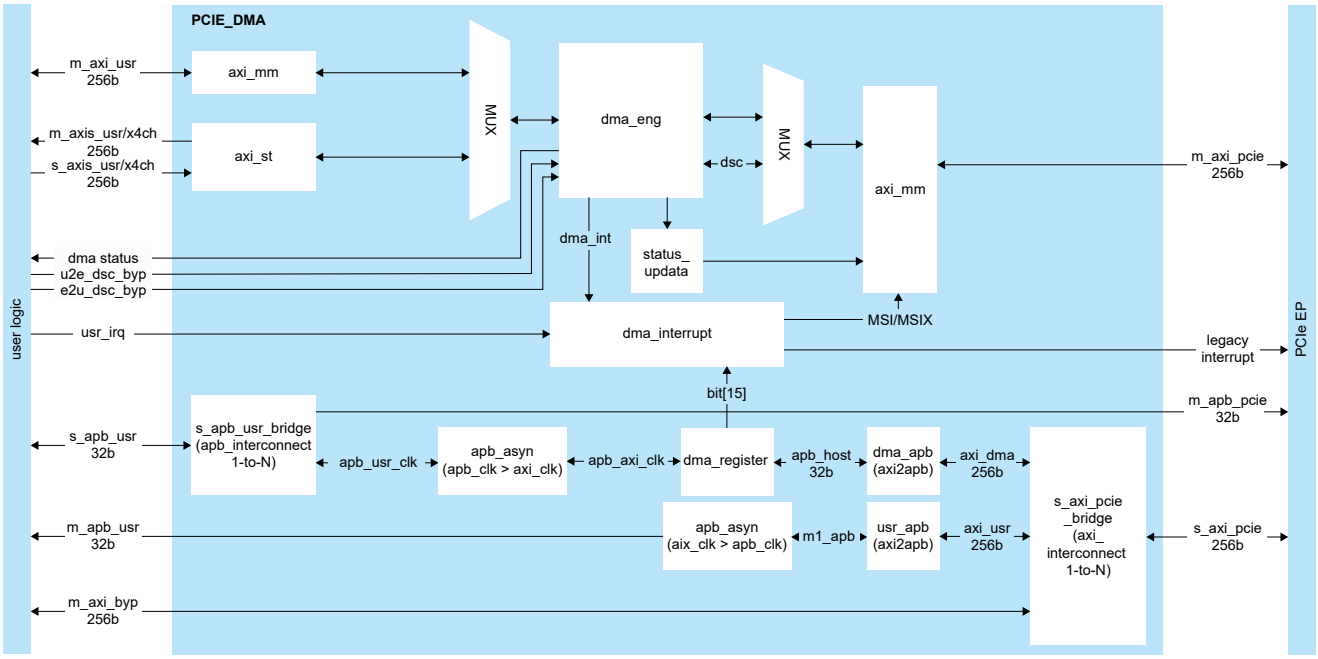
UG-TiPCIeSGDMA-v1.1
May 2025
www.elitestek.com

Contents

Introduction

This user guide describes how to implement high-performance direct memory access (DMA) using the Elitestek® PCIe Scatter-Gather Direct Memory Access (SGDMA) IP, a configurable IP core designed specifically for PCIe endpoint applications. You use this SGDMA IP with the TJ-Series or TP-Series PCIe Controller for high-speed data transfers between the FPGA and host. The basic functional block diagram is shown below.

Figure 1: SGDMA Block Diagram



The SGDMA IP enables data transfer between the host memory and FPGA user logic via the PCIe interface. It supports two transfer modes:

- Host-to-Card (HTC)
- Card-to-Host (CTH)

On the user logic side, the SGDMA provides either an AXI4 memory-mapped (AXI4-MM) master interface or an AXI4-Stream interface for integration with user logic. On the PCIe controller side, it connects via an AXI4-MM master interface. The SGDMA performs data transfers based on descriptors, which contain information such as source address, destination address, data length, and the address of the next descriptor in a chain. The SGDMA fetches and parses descriptors from host memory, and upon completing the transfer, it notifies the host of the transfer status via interrupt.

In addition, this IP integrates an SGDMA bypass mode that allows the host to access device memory directly for read/write operations without going through the SGDMA engine. This is suitable for low-latency applications involving small data volumes. Working with the SGDMA transfer engine, it provides a flexible data access solution for the system.

PCIe SGDMA Supported Features

PCIe SGDMA supports:

- Data transfer using the standard AXI4-MM protocol
- Data transfer in AXI4-Stream mode
- Master APB3 interface for host-to-card register configuration
- Slave APB3 interface for user logic-to-IP register configuration
- Legacy and MSI-X interrupts

The following table provides for a more indepth overview of the features supported by the PCIe SGDMA IP core.

Table 1: Supported Features

Feature	Available v1.0
Data width	✓ 256 bits X 64 bits (Future version) X 128 bits (Future version) X 512 bits (Future version)
Host-to-Card	✓ Single channel X Up to four channels (Future version)
Card-to-Host	✓ Single channel X Up to four channels (Future version)
DMA bypass	X Not available in v1.0 (Future version)
Descriptor	✓ Up to 256 MB data length per descriptor
Write back/polling	✓ Single channel X Up to four channels (Future version)
Legacy INTX	✓ Single channel (INTA only)
User INTR	✓ INTA ✓ MSIX X MSI (Future version)
MSIX	✓ Supported
MSI	X Not available in v1.0 (Future version)
MPS, MRRS	✓ MPS = 128, 256, 512 ✓ MRRS = 128, 256, 512, 1024, 2048, 4096 Measured in bytes
Clock frequency	✓ AXI clock = {125 to 250} MHz ✓ APB clock = {20 to 200} MHz Must use same CLK source as PCIe EP
APB interface	✓ All
AXI4-MM interface	✓ All
AXI4-Stream interface	✓ All

Resource Utilization and Performance

These resource and performance values are based on specific supported FPGAs. These values serve as a guideline only and may vary depending on device resource utilization, design congestion, and user design modifications.

Table 2: Resource Efficiency Table

FPGA Model	Logic Elements (Logic, Adders, Flip-Flops, etc.)	Memory Blocks	DSP Blocks	f_{MAX} (MHz)	Efinity® Version
TJ375N1156X	51,196	194	0	280	2024.2

Functional Description

This section outlines some of the capabilities of the PCIe SGDMA IP core.

SGDMA

The SGDMA operates in Host-to-Card (HTC) and Card-to-Host (CTH) modes. Both modes support bidirectional data transfer between the FPGA and the host. Using a chained descriptor queue mechanism, the SGDMA engine efficiently and flexibly handles data transfer tasks.

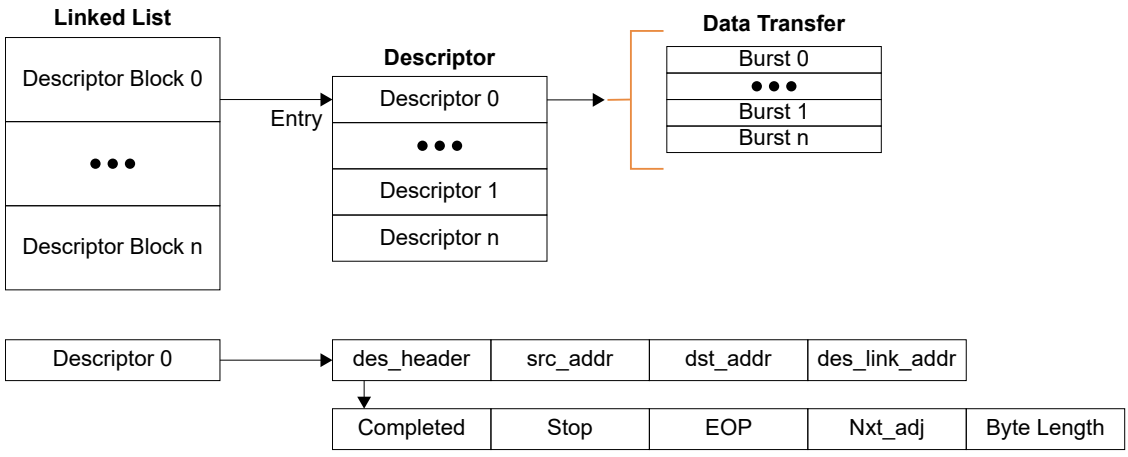
The operational flow for SGDMA is as follows:

1. Descriptor Configuration: The driver on the host side configures software descriptors and builds the transfer queue.
2. Data Transfer: The SGDMA engine actively reads the descriptor commands from the queue and performs the data transfer operations.
3. Full-Duplex Processing: The SGDMA engine supports multitasking with parallel processing. The HTC and CTH engines operate independently to handle data transfers from host to FPGA and from FPGA to host, respectively.

Descriptors

A descriptor is the basic control unit for SGDMA transfer, used to define the attributes and control information for a single data transfer. Its structure is as follows:

Figure 2: Descriptor Structure



- **Descriptor:** 32 bytes, defines the source address, destination address, and length for a single transfer. The descriptor must be 32-byte aligned in memory.
- **Descriptor Block:** Composed of multiple consecutive descriptors. The descriptors within the same block are stored consecutively in memory and cannot span a 4 KB memory boundary.
- **Descriptor List:** Composed of multiple descriptor blocks defining a complete SGDMA transfer task. The descriptor blocks are linked together using a linked list, and they do not need to be stored consecutively in memory.

Table 3: Descriptor Format Table

Offset	Field	Description
0x0	des_header_h	[4:0]: Upper 5 bits of the data transfer length. [31:5]: Reserved, default value is 0.
0x04	des_header_l	[0]: Completed bit, set to 1 to indicate that an interrupt should be generated after the data transfer is complete. [1]: Stop bit, set to 1 to indicate that the SGDMA engine should stop requesting new descriptors. [2]: EOP bit, end-of-packet flag for AXI4-Stream interface. [8:3]: nxt_adj, represents the remaining number of descriptors in the current block. [31:9]: Lower 23 bits of the data transfer length, combined with `des_header_h` [4:0] to form a 28-bit data length, in bytes.
0x08	src_addr_h	Upper 32 bits of the source address for data transfer.
0x0C	src_addr_l	Lower 32 bits of the source address for data transfer.
0x10	dst_addr_h	Upper 32 bits of the destination address for data transfer.
0x14	dst_addr_l	Lower 32 bits of the destination address for data transfer.
0x18	des_link_address_h	Upper 32 bits of the next descriptor address.
0x1C	des_link_address_l	Lower 32 bits of the next descriptor address.

The operational flow of the descriptor is as follows:

1. List Creation: The host driver creates the descriptor list and stores it in the host memory.
2. Register Configuration: The host driver configures the SGDMA registers through the BAR space and initializes the following parameters:
   - Start address of the current descriptor list (HTC: 0x300C and 0x3010; CTH: 0x400C and 0x4010).
   - Number of descriptors in the first descriptor block of the current descriptor list (HTC: 0x3004; CTH: 0x4004).
3. Transfer Start: The SGDMA engine fetches descriptors from the starting address of the current descriptor list and uses the `des_link_address` field of the descriptor to obtain the address of the next descriptor.
4. Descriptor Block Processing:
   - Descriptors within the same descriptor block have consecutive addresses and the next descriptor address can be obtained by offsetting 32 bytes from the current descriptor address.
   - When the `nxt_adj` value of a descriptor is 0, it indicates that the current descriptor is the last descriptor in the descriptor block. The next descriptor block address is obtained via the `des_link_address` field.
5. List Termination: When the `stop` bit of a descriptor is set to `1`, it indicates that the current descriptor list has ended. The SGDMA engine will stop the transfer task after completing this descriptor.

AXI4-MM Bypass Master

The AXI4-MM bypass master interface bypasses the SGDMA module, providing the host with direct access to user logic memory. The PCIe BAR4/5 address space is mapped to this interface and any transaction layer packet (TLP) targeting BAR4/5 is forwarded directly to the user logic. Additionally, the TLP address must meet the 8-byte alignment requirement.

AXI4-Stream Data Interface

In addition to the basic AXI4-MM interface, the SGDMA IP also provides a streaming AXI4-Stream user logic interface. This interface uses a non-addressed transfer mechanism, enabling

high throughput and low-latency transmission of large data streams. It is particularly suitable for continuous data transfer scenarios, such as video streams, network packet transmission, and sensor data collection.

While the CTH and HTC can be configured with different user interfaces, the two interfaces cannot be used simultaneously.

The SGDMA provides an AXI4-Stream interface for streaming transfer mode, enabling the transfer of SGDMA data between the host and the device. The HTC AXI4-Stream interface is used to transfer HTC SGDMA data to external user logic, while the CTH AXI4-Stream interface handles the transfer of CTH SGDMA data to the host. The user-side interface of the SGDMA channel supports the AXI4-Stream bus.

When data is transferred from the host to the user side, the source address is obtained from the host, but the destination address in the descriptor is not used. Stream-based data can be divided into multiple descriptors for transfer, with the end of each data packet indicated by the EOP bit in the descriptor. When transferring a descriptor containing the EOP bit, the AXI4-Stream bus sets the `tlast` signal high on the last data beat.

When selecting AXI4-Stream port mode, the IP allocates a dedicated AXI4-Stream port on the user logic side for streaming SGDMA transfers. The packet boundary is identified by the combined EOP flag in the descriptor and the `tlast` signal of the AXI4-Stream interface, ensuring accurate data packet segmentation. Note that channel interleaving is not supported in this mode. All port switching operations must be strictly aligned with the packet boundary, meaning that port switching is only allowed when EOP/`tlast` is asserted.

For CTH transfers, when the CTH engine is enabled and detects a valid descriptor, it receives the data stream from the user side through the AXI4-Stream bus. The engine transfers data byte-by-byte according to the address parameters in the descriptor until it encounters the end-of-packet (EOP) flag or completes the data transfer as specified in the current descriptor. Once the transfer is complete, the CTH engine writes the completion status information to the pre-configured write-back address on the host side. This information includes the execution result of the current descriptor and the actual transferred byte length. It should be noted that this write-back information is completely isolated from the polling-mode status registers, and follows an incremental, descriptor-based state feedback mechanism.

The write-back mechanisms of HTC and CTH are fundamentally different. HTC uses polling registers to obtain global status, while CTH uses descriptor chains to provide fine-grained task completion notifications.

APB3 Configuration Interface

The IP provides register configuration through two sets of APB3 interfaces:

- M_APB3
- S_APB3

The M_APB3 interface provides user-side register access. The host initiates read and write requests to this interface through PCIe, which is mapped to the BAR0/1 space.

The S_APB3 interface is dedicated to managing internal SGDMA registers and is restricted to internal operations within the user logic. It does not provide access to PCIe registers and does not initiate requests to the host.

Parameter Configuration

The following tables detail the registers and range of possible values used for configuration purposes.

IP Parameter Configuration

Table 4: IP Parameter Configuration

Register Name	Value	Description (New)
AXI_DATA_WIDTH	256	AXI data width on the user side.
DMA_ST_EN	0, 1	SGDMA user interface option: 0: AXI4-MM. 1: AXI4-Stream.
S_APB_USR_EN	0, 1	APB3 slave interface: 0: Disable APB3 slave interface. 1: Enable APB3 slave interface.
M_APB_USR_EN	0, 1	APB3 master interface: 0: Disable APB3 master interface. 1: Enable APB3 master interface.
DATA_PROTECT	0, 1	Data protection option: 0: Disable parity check. 1: Enable and transmit parity check.
M_AXI_BYP_EN	0, 1	BYPASS interface: 0: Disable BYPASS interface. 1: Enable BYPASS interface.
USR_INT_NUM	1-16	Number of user interrupt requests.
MSI_ENABLE	0, 1	MSI function: 0: Disable MSI. 1: Enable MSI.
MSIX_ENABLE	0, 1	MSI-X function: 0: Disable MSIX. 1: Enable MSIX.
MSIX_VEC_NUM	1-32	Enable vector, maximum number of 32. Typically, Linux uses only 1 vector for MSI. This option can be disabled.
CTH_CH_NUM	1	Single channel only.
HTC_CH_NUM	1	Single channel only.
CTH_DSC_DTH	2, 4, 8, 16, 32, 64	CTH channel descriptor queue depth: Available options range from 2 to 64.
HTC_DSC_DTH	2, 4, 8, 16, 32	HTC channel descriptor queue depth: Available options range from 2 to 32.
CTH_DSC_BYP_EN	0000-1111	Read descriptor bypass: Can be used for selected read channels, each bit corresponds to a channel. LSB corresponds to channel 0. Setting the corresponding bit to 1 enables descriptor bypass for the channel.
HTC_DSC_BYP_EN	0000-1111	Write descriptor bypass: Can be used for selected write channels, each bit corresponds to a channel. LSB corresponds to channel 0. Setting the corresponding bit to 1 enables descriptor bypass for the channel.

BAR Space Configuration Parameters

Table 5: BAR Space Configuration Parameters

Register Name	Value	Description
M0_BASE_ADDR	64'h0	BAR0 space base address, used by the host driver to configure SGDMA registers; corresponds to the AXI inbound BAR0 register configuration.
M0_ADDR_SPACE	64'h80000	BAR0 space size, default is 512 KB; corresponds to the hard IP BAR0 space configuration.
M1_BASE_ADDR	64'ha0000	BAR4 space base address, used by the host driver via the SGDMA APB master interface to configure user logic; corresponds to the AXI inbound BAR4 register configuration.
M1_ADDR_SPACE	64'h1000	BAR4 space size, default is 8 KB; corresponds to the hard IP BAR4 space configuration.
M2_BASE_ADDR	64'h90000	BAR2 space base address, used as the AXI address space for SGDMA bypass (bypassing the SGDMA engine for traffic); corresponds to the AXI inbound BAR2 register configuration.
M2_ADDR_SPACE	64'h1000	BAR2 space size, default is 16 KB; corresponds to the hard IP BAR2 space configuration.

Interface Bus

Table 6: Interface Bus

Signal Name	Direction	Description
Global Clock and Reset		
axi_clk	Input	Increase clock frequency to 250 MHz.
apb_clk	Input	Clock for APB master and slave interfaces, generated by logic. Clock frequency <200 MHz.
dma_rstn	Input	SGDMA IP asynchronous reset signal, set to 0 for reset.
m_axi4_pcie Bus		
m_axi_pcie_awready	Input	Write address interface ready signal.
m_axi_pcie_awvalid	Output	Write address interface signal: 1: Valid. 0: Invalid.
m_axi_pcie_awaddr[63:0]	Output	Address signal line, transmits address information, gives the first address of a burst transaction.
m_axi_pcie_awid[7:0]	Output	Transaction identifier, used in out-of-order transactions, identifies the write address group; default is 0.
m_axi_pcie_awlen[7:0]	Output	AXI4 extended burst length supports INCR burst types for 1 to 256 transfers.
m_axi_pcie_awsize[2:0]	Output	Number of bytes per transfer, related to WDATA width. For example, if WDATA is 32-bit, AWSIZE = log2(32/8) = 2.
m_axi_pcie_wready	Input	Write data interface ready signal.
m_axi_pcie_wvalid	Output	Write data interface signal: 1: Valid. 0: Invalid.
m_axi_pcie_wdata[255:0]	Output	Data signal line, transmits data information.
m_axi_pcie_wdata_par[63:0]	Output	Write data parity.

Signal Name	Direction	Description
m_axi_pcie_wstrb[63:0]	Output	Data bus valid byte control. For a 32-bit bus, WSTRB = 4'b0010 means data in WDATA[15:8] is valid. If all 32 bits of WDATA are valid, WSTRB should be 4'b1111.
m_axi_pcie_wlast	Output	Asserted high to indicate the last data in a burst transaction.
m_axi_pcie_bready	Output	Write response interface ready signal.
m_axi_pcie_bvalid	Input	Write response interface valid signal: 1: Valid. 0: Invalid.
m_axi_pcie_bresp[1:0]	Input	Write response status: 0: OKEY (normal access successful). 1: EXOKEY. 2: SLVERR (slave error). 3: DECERR (decode error, e.g., no slave address).
m_axi_pcie_bresp_par	Input	Write response status parity.
m_axi_pcie_bid[7:0]	Input	Transaction identifier.
m_axi_pcie_bid_par	Input	Transaction identifier parity.
m_axi_pcie_arready	Input	Read address interface ready signal.
m_axi_pcie_arvalid	Output	Read address interface valid signal: 1: Valid. 0: Invalid.
m_axi_pcie_araddr[63:0]	Output	Address signal line, transmits address information.
m_axi_pcie_arid[7:0]	Output	Read address ID.
m_axi_pcie_arlen[7:0]	Output	Burst read length, AXI4 extended burst length supports INCR burst types for 1 to 256 transfers.
m_axi_pcie_arsize[2:0]	Output	Number of bytes per transfer, related to RDATA width. For example, if RDATA is 32-bit, ARSIZE = log2(32/8) = 2.
m_axi_pcie_rready	Output	Read data ready signal.
m_axi_pcie_rvalid	Input	Read data valid signal: 1: Valid. 0: Invalid.
m_axi_pcie_rid[7:0]	Input	Transaction identifier.
m_axi_pcie_rid_par	Input	Transaction identifier parity.
m_axi_pcie_rdata[255:0]	Input	Data signal line, transmits data information.
m_axi_pcie_rdata_par[63:0]	Input	Read data parity.
m_axi_pcie_rresp[1:0]	Input	Read response, indicates the status of the read transfer.
m_axi_pcie_rresp_par	Input	Read response parity.
m_axi_pcie_rlast	Input	Asserted high when the last valid data in a read burst transfer is transmitted.
s_axi4_pcie Bus		
s_axi_pcie_awready	Output	Write data interface ready signal.
s_axi_pcie_awvalid	Input	Write data valid signal: 1: Valid. 0: Invalid.
s_axi_pcie_awaddr[63:0]	Input	Address signal line, transmits address information, gives the first address of a burst transaction.
s_axi_pcie_awid[7:0]	Input	Transaction identifier, used in out-of-order transactions, identifies the write address group.
s_axi_pcie_awlen[7:0]	Input	AXI4 extended burst length supports INCR burst types for 1 to 256 transfers.
s_axi_pcie_awsize[2:0]	Input	Number of bytes per transfer, related to WDATA width. For example, if WDATA is 32-bit, AWSIZE = log2(32/8) = 2.

Signal Name	Direction	Description
s_axi_pcie_wready	Output	Write address interface ready signal.
s_axi_pcie_wvalid	Input	Write address interface valid signal: 1: Valid. 0: Invalid.
s_axi_pcie_wdata[255:0]	Input	Data signal line, transmits data information.
s_axi_pcie_wstrb[31:0]	Input	Data bus valid byte control. For example, for a 32-bit bus, WSTRB = 4'b0010 means data in WDATA[15:8] is valid. If all 32 bits of WDATA are valid, WSTRB should be 4'b1111.
s_axi_pcie_wlast	Input	Asserted high to indicate the last data in a burst transaction.
s_axi_pcie_bready	Input	Write response interface ready signal.
s_axi_pcie_bvalid	Output	Write response interface valid signal: 1: Valid. 0: Invalid.
s_axi_pcie_bresp[1:0]	Output	Write response status: 0: OKEY (normal access successful). 1: EXOKEY. 2: SLVERR (slave error). 3: DECERR (decode error, e.g., no slave address).
s_axi_pcie_bresp_par	Output	Response parity bit.
s_axi_pcie_bid[7:0]	Output	Transaction identifier.
s_axi_pcie_bid_par	Output	Transaction identifier parity.
s_axi_pcie_arready	Output	Read address interface ready signal.
s_axi_pcie_arvalid	Input	Read address interface valid signal: 1: Valid. 0: Invalid.
s_axi_pcie_araddr[63:0]	Input	Address signal line, transmits address information.
s_axi_pcie_arid[7:0]	Input	Transaction identifier.
s_axi_pcie_arlen[7:0]	Input	Burst read length, AXI4 extended burst length supports INCR burst types for 1 to 256 transfers. Burst transfers have the following rules: • Wrapping burst length must be 2, 4, 8, or 16. • Bursts cannot cross 4KB boundaries. • Early termination of burst transfers is not supported.
s_axi_pcie_arsize[2:0]	Input	Number of bytes per transfer, related to RDATA width. For example, if RDATA is 32-bit, ARSIZE = log2(32/8) = 2.
s_axi_pcie_rready	Input	Read data ready signal.
s_axi_pcie_rvalid	Output	Read data interface valid signal: 1: Valid. 0: Invalid.
s_axi_pcie_rid[7:0]	Output	Transaction identifier.
s_axi_pcie_rid_par	Output	Transaction identifier parity.
s_axi_pcie_rdata[255:0]	Output	Data signal line, transmits data information.
s_axi_pcie_rdata_par[63:0]	Output	Read data parity.
s_axi_pcie_rresp[1:0]	Output	Read response, indicates the status of the read transfer.
s_axi_pcie_rresp_par	Output	Read response parity.
s_axi_pcie_rlast	Output	Asserted high when the last valid data in a read burst transfer is transmitted.
m_axi4_user Bus		
m_axi_usr_awready	Input	Write address interface ready signal.

Signal Name	Direction	Description
m_axi_usr_awvalid	Output	Write address interface valid signal: 1: Valid. 0: Invalid.
m_axi_usr_awaddr[63:0]	Output	Address signal line, transmits address information, gives the first address of a burst transaction.
m_axi_usr_awid[7:0]	Output	Transaction identifier, used in out-of-order transactions, identifies the write address group.
m_axi_usr_awlen[7:0]	Output	AXI4 extended burst length supports INCR burst types for 1 to 256 transfers.
m_axi_usr_awsize[2:0]	Output	Number of bytes per transfer, related to WDATA width.
m_axi_usr_wready	Input	Write data interface ready signal.
m_axi_usr_wvalid	Output	Write data valid signal: 1: Valid. 0: Invalid.
m_axi_usr_wdata [AXI_DATA_WIDTH -1:0]	Output	Data signal line, transmits data information.
m_axi_usr_wdata_par [AXI_DATA_WIDTH /8-1:0]	Output	Write data parity.
m_axi_usr_wstrb [AXI_DATA_WIDTH/8-1:0]	Output	Data bus valid byte control. For example, for a 32-bit bus, WSTRB = 4'b0010 means data in WDATA[15:8] is valid. If all 32 bits of WDATA are valid, WSTRB should be 4'b1111.
m_axi_usr_wstrb_par [AXI_DATA_WIDTH/64-1:0]	Output	Parity for m_axi_usr_wstrb.
m_axi_usr_wlast	Output	Asserted high when the last valid data in a write burst transfer is transmitted.
m_axi_usr_bready	Output	Write response interface ready signal.
m_axi_usr_bvalid	Input	Write response interface valid signal: 1: Valid. 0: Invalid.
m_axi_usr_bresp[1:0]	Input	Write response status: 0: OKEY (normal access successful). 1: EXOKEY. 2: SLVERR (slave error). 3: DECERR (decode error, e.g., no slave address).
m_axi_usr_bid[7:0]	Input	Transaction identifier.
m_axi_usr_arready	Input	Read address interface ready signal.
m_axi_usr_arvalid	Output	Read address interface valid signal: 1: Valid. 0: Invalid.
m_axi_usr_araddr[63:0]	Output	Address signal line, transmits address information.
m_axi_usr_arid[7:0]	Output	Transaction identifier.
m_axi_usr_arlen[7:0]	Output	Burst read length, AXI4 extended burst length supports INCR burst types for 1 to 256 transfers.
m_axi_usr_arsize[2:0]	Output	Number of bytes per transfer in a read burst.
m_axi_usr_rready	Output	Read data interface ready signal.
m_axi_usr_rvalid	Input	Read data valid signal: 1: Valid. 0: Invalid.
m_axi_usr_rid[7:0]	Input	Transaction identifier.
m_axi_usr_rdata [AXI_DATA_WIDTH -1:0]	Input	Data signal line, transmits data information.

Signal Name	Direction	Description
m_axi_usr_rdata_par [AXI_DATA_WIDTH /8-1:0]	Input	Read data parity.
m_axi_usr_rresp[1:0]	Input	Read response, indicates the status of the read transfer.
m_axi_usr_rlast	Input	Asserted high when the last valid data in a read burst transfer is transmitted.
m_apb_usr Bus		
m_apb_usr_pready	Input	Ready signal used by the slave to extend the APB data transfer cycle. Active high.
m_apb_usr_psel	Output	Transfer selection signal from master to slave. Active high.
m_apb_usr_pwrite	Output	Read/write indication signal. High for write, low for read.
m_apb_usr_penable	Output	Enable signal indicating the second and subsequent cycles of an APB transfer. Active high.
m_apb_usr_paddr[31:0]	Output	Address bus, up to 32 bits.
m_apb_usr_pwdata[31:0]	Output	Write data bus, up to 32 bits.
m_apb_usr_prdata[31:0]	Input	Read data bus, up to 32 bits.
m_apb_usr_pslverror	Input	Transfer failure indication signal returned by the slave. Optional. Active high.
AXI4-Stream Bus		
HTC		
s_axis_usr_tready_n	Output	Asserted high to indicate the SGDMA side is ready to receive data. When both tready and tvalid are high, data is transferred. When tvalid is high but tready is low, the user side must hold the valid data until tready goes high.
s_axis_usr_tvalid_n	Input	Asserted high by the user side to indicate that tdata is valid.
s_axis_usr_tlast_n	Input	Asserted high by the logic side to indicate the last data in a burst transfer.
s_axis_usr_tdata_n [AXI_DATA_WIDTH -1:0]	Input	Transfer data bus. Bit width is defined by the AXI_DATA_WIDTH parameter.
s_axis_usr_tdata_par [AXI_DATA_WIDTH/8 -1:0]	Input	Parity bits for the tdata bus.
s_axis_usr_tkeep_n [AXI_DATA_WIDTH/8-1:0]	Input	Byte-valid indicator for the last data word in a packet. Each bit corresponds to one byte: 1: Valid. 0: Invalid.
CTH		
m_axis_usr_tready[NUM]	Input	Asserted high to indicate the user side is ready to receive data. When both tready and tvalid are high, data is transferred. When tvalid is high but tready is low, the SGDMA side must hold the valid data until tready goes high.
m_axis_usr_tvalid_n	Output	Asserted high by the SGDMA to indicate that tdata is valid.
m_axis_usr_tlast_n	Output	Asserted high to indicate the last data in a burst transfer.
m_axis_usr_tdata_n [AXI_DATA_WIDTH -1:0]	Output	Transfer data bus. Bit width is defined by the AXI_DATA_WIDTH parameter.
m_axis_usr_tdata_par [AXI_DATA_WIDTH/8 -1:0]	Output	Parity bits for the tdata bus.
m_axis_usr_tkeep_n [AXI_DATA_WIDTH/8 -1:0]	Output	Byte-valid indicator for the last data word in a packet. Each bit corresponds to one byte: 1: Valid. 0: Invalid.
m_apb_pcie Bus		
m_apb_pcie_pready	Input	Ready signal used by the slave to extend the APB data transfer cycle. Active high.

Signal Name	Direction	Description
m_apb_pcie_psel	Output	Transfer selection signal from master to slave. Active high.
m_apb_pcie_pwrite	Output	Read/write indication signal. High for write, low for read.
m_apb_pcie_penable	Output	Enable signal indicating the second and subsequent cycles of an APB transfer. Active high.
m_apb_pcie_paddr[23:0]	Output	Address bus, up to 24 bits.
m_apb_pcie_pwdata[31:0]	Output	Write data bus, up to 32 bits.
m_apb_pcie_pwdata_par[3:0]	Output	End-to-end parity bits for m_apb_pcie_pwdata.
m_apb_pcie_pstrb[3:0]	Output	Write strobe signals for sparse data transfer on the write data bus.
m_apb_pcie_pstrb_par	Output	End-to-end parity bits for m_apb_pcie_pstrb.
m_apb_pcie_prdata[31:0]	Input	Read data bus, up to 32 bits.
m_apb_pcie_prdata _par[3:0]	Input	End-to-end parity bits for m_apb_pcie_prdata.
m_apb_pcie_pslverror	Input	Transfer failure indication signal returned by the slave. Optional. Active high.
s_apb_usr Bus		
s_apb_usr_pready	Output	Ready signal used by the slave to extend the APB data transfer cycle. Active high.
s_apb_usr_psel	Input	Transfer selection signal from master to slave. Active high.
s_apb_ usr_pwrite	Input	Read/write indication signal. High for write, low for read.
s_apb_usr_penable	Input	Enable signal indicating the second and subsequent cycles of an APB transfer. Active high.
s_apb_usr_paddr[31:0]	Input	Address bus, up to 25 valid bits.
s_apb_usr_pwdata[31:0]	Input	Write data bus, up to 32 bits.
s_apb_usr_prdata[31:0]	Output	Read data bus, up to 32 bits.
s_apb_usr_pslverror	Output	Transfer failure indication signal returned by the slave. Optional. Active high.
m_byp_axi4 Bus		
m_axi_byp_awready	Input	Write address interface ready signal.
m_axi_byp_awvalid	Output	Write address interface valid signal: 1: Valid. 0: Invalid.
m_axi_byp_awaddr [AXI_ADDR_WIDTH -1:0]	Output	Address signal line, transmits address information, gives the first address of a burst transaction.
m_axi_byp_awid [AXI_ID_WIDTH -1:0]	Output	Transaction identifier, used in out-of-order transactions, identifies the write address group.
m_axi_byp_awlen[7:0]	Output	AXI4 extended burst length supports INCR burst types for 1 to 256 transfers.
m_axi_byp_awsize[2:0]	Output	Number of bytes per transfer, related to WDATA width. For example, if WDATA is 32-bit, AWSIZE = log2(32/8) = 2.
m_axi_byp_wready	Input	Write data interface ready signal.
m_axi_byp_wvalid	Output	Write data valid signal: 1: Valid. 0: Invalid.
m_axi_byp_wdata [AXI_DATA_WIDTH -1:0]	Output	Data signal line, transmits data information.
m_axi_byp_wdata_par [AXI_DATA_WIDTH/8 -1:0]	Output	End-to-end parity bits for m_axi_byp_wdata.
m_axi_byp_wstrb [AXI_DATA_WIDTH/8 -1:0]	Output	Data bus valid byte control. For a 32-bit bus, WSTRB = 4'b0010 means data in WDATA[15:8] is valid. If all 32 bits of WDATA are valid, WSTRB should be 4'b1111.
m_axi_byp_wlast	Output	Asserted high to indicate the last data in a burst transaction.
m_axi_byp_bready	Output	Write response interface ready signal.

Signal Name	Direction	Description
m_axi_byp_bvalid	Input	Write response interface valid signal: 1: Valid. 0: Invalid.
m_axi_byp_bresp[1:0]	Input	Write response status: 0: OKEY (normal access successful). 1: EXOKEY. 2: SLVERR (slave error). 3: DECERR (decode error, e.g., no slave address).
m_axi_byp_bid [AXI_ID_WIDTH -1:0]	Input	Transaction identifier.
m_axi_byp_arready	Input	Read address interface ready signal.
m_axi_byp_arvalid	Output	Read address interface valid signal: 1: Valid. 0: Invalid.
m_axi_byp_araddr [AXI_ADDR_WIDTH -1:0]	Output	Address signal line, transmits address information.
m_axi_byp_arid [AXI_ID_WIDTH -1:0]	Output	Transaction identifier.
m_axi_byp_arlen[7:0]	Output	Burst read length, AXI4 extended burst length supports INCR burst types for 1 to 256 transfers.
m_axi_byp_arsize[2:0]	Output	Number of bytes per transfer, related to RDATA width. For example, if RDATA is 32-bit, ARSIZE = log2(32/8) = 2.
m_axi_byp_rready	Output	Read data ready signal.
m_axi_byp_rvalid	Input	Read data interface valid signal: 1: Valid. 0: Invalid.
m_axi_byp_rid [AXI_ID_WIDTH -1:0]	Input	Transaction identifier.
m_axi_byp_rdata [AXI_DATA_WIDTH -1:0]	Input	Data signal line, transmits data information.
m_axi_byp_rresp[1:0]	Input	Read response, indicates the status of the read transfer.
m_axi_byp_rlast	Input	Asserted high when the last valid data in a read burst transfer is transmitted.
Local Descriptor Configuration Bus		
e2u_n (where *n* indicates the channel number, ranging from 0 to 3)		
e2u_dsc_byp_ready_n	Output	Descriptor receive status indication signal.
e2u _dsc_byp_dst_addr_n[63:0]	Input	Destination address for data transfer.
e2u _dsc_byp_src_addr_n[63:0]	Input	Source address for data transfer.
e2u _dsc_byp_len_n[27:0]	Input	Length of data to be transferred (in bytes).
e2u _dsc_byp_ctl_n[2:0]	Input	Descriptor control bits: • Bit[2]: EOP — End of Packet for AXI-Stream interface. • Bit[1]: Stop — When set to 1, instructs the SGDMA descriptor engine to stop fetching new descriptors after this one. • Bit[0]: Complete — When set to 1, instructs the SGDMA engine to generate an interrupt upon completing this transfer.
e2u _dsc_byp_valid_n	Input	Valid signal indicating a locally configured descriptor.
u2e		

Signal Name	Direction	Description
u2e _dsc_byp_ready[NUM]	Output	Descriptor receive status indication signal: 1: Interface ready to receive descriptor. 0: Interface not ready. **Note:** When this signal transitions from 1 to 0, one additional descriptor can still be accepted.
u2e _dsc_byp_dst_addr_n[63:0]	Input	Destination address for data transfer.
u2e _dsc_byp_src_addr_n[63:0]	Input	Source address for data transfer.
u2e _dsc_byp_len_n[27:0]	Input	Length of data to be transferred (in bytes).
u2e _dsc_byp_ctl_n[2:0]	Input	Descriptor control bits: • Bit[2]: EOP — End of Packet for AXI-Stream interface. • Bit[1]: Stop — When set to 1, instructs the SGDMA descriptor engine to stop fetching new descriptors after this one. • Bit[0]: Complete — When set to 1, instructs the SGDMA engine to generate an interrupt upon completing this transfer.
u2e _dsc_byp_valid_n	Input	Valid signal indicating a locally configured descriptor.
SGDMA Engine Status Indicators and Interrupt Signals		
Status Indicators		
cfg_max_payload_size[2:0]	Input	MPS (Max Payload Size) indicator from PCIe IP. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. Others: Reserved.
cfg_max_read_req_size[2:0]	Input	MRRS (Max Read Request Size) indicator from PCIe IP. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. 3'b011: 1024 bytes. 3'b100: 2048 bytes. 3'b101: 4096 bytes. Others: Reserved.
HTC_STS[8*CHN_NUM-1:0]	Output	Working status of the HTC engine. Bit[7:4]: Reserved Bit[3]: Run Bit[2]: Interrupt Bit[1]: DSC done Bit[0]: Busy
CTH_STS[8*CHN_NUM-1:0]	Output	Working status of the CTH engine. Bit[7:4]: Reserved Bit[3]: Run Bit[2]: Interrupt Bit[1]: DSC done Bit[0]: Busy
Interrupt Signals		
cfg_msi_enable[3:0]	Input	MSI interrupt enable signal. 0: Disabled. Other values: Enabled.
cfg_msix_enable[3:0]	Input	MSI-X interrupt enable signal. 0: Disabled. Other values: Enabled.
legacy_irq	Output	Legacy interrupt request signal. Held high until acknowledged and cleared.
legacy_irq_ack	Input	Legacy interrupt acknowledge signal.

Signal Name	Direction	Description
usr_irq [NUM_USR_IRQ-1:0]	Input	User interrupt request signals. Held high until acknowledged and cleared.
usr_irq_ack [NUM_USR_IRQ-1:0]	Output	User interrupt acknowledge signal.

Register List

Table 7: Register List

Target	Channel	Dword Offset	Name	R/W	Description
0		0x00	cfg_identifier	RO	
		0x04	reg_user_max_payload	RO	User-defined maximum payload register
		0x08	reg_user_max_read_req	RO	User-defined maximum read request register
		0x0C	reg_write_timeout	RO	Write timeout register
		0x18	msi_enable&msix_enable	RO	MSI/MSI-X interrupt enable
		0x1C	max_payload	RO	Maximum payload register
		0x20	max_read_req	RO	Maximum read request register
		0x24	clk_pciew	RO	PCIe width: (0, 1, 2, 3) represent (64, 128, 256, 512)
		0x28	reg_pcie_ctl	RO	PCIe control register
		0x4C	reg_rq_meter_multi, reg_tag_count, reg_axi_desc, reg_axi_writeback, reg_rq_meter_multi_upd	RO	
1		0x00	dsc_identifier	R	
		0x04	dsc_crd_en	RW	Enable signal for HTC/CTH channel credit
		0x0C	dsc_stop		Descriptor fetch stop signal for the corresponding channel
		0x18	dsc_crd_en	RW	Enable signal for CTH channel credit
2		0x00	irq_identifier	R	
		0x04	user_irq	R	User interrupt request indicator
		0x08	dma_irq	R	SGDMA interrupt request indicator
		0x0C	user_irq_i	R	User interrupt pending indicator
		0x10	dma_irq_i	R	SGDMA interrupt pending indicator
		0x14	user_irq_lut	RW	User interrupt vector lookup
		0x18	user_irq_lut	RW	User interrupt vector lookup
		0x1C	user_irq_lut	RW	User interrupt vector lookup
		0x20	user_irq_lut	RW	User interrupt vector lookup
		0x24	dma_irq_lut	RW	SGDMA interrupt vector lookup
		0x28	dma_irq_lut	RW	SGDMA interrupt vector lookup
		0x2C	user_en	RW	User interrupt enable register
		0x38	dma_en	RW	SGDMA interrupt enable register
3	0/1/2/3	0x04	htc_dsc_adj	RW	Number of adjacent descriptors for HTC

Target	Channel	Dword Offset	Name	R/W	Description
		0x08	htc_dsc_crd	RW	HTC channel credit value
		0x0C	htc_dsc_addr_l	RW	Lower 32 bits of HTC descriptor base address
		0x10	htc_dsc_addr_h	RW	Upper 32 bits of HTC descriptor base address
		0x14	htc _dma_wb_addr_l	RW	Lower 32 bits of the host memory address where the HTC engine writes back the number of completed descriptors
		0x18	htc _dma_wb_addr_h	RW	Upper 32 bits of the host memory address where the HTC engine writes back the number of completed descriptors
		0x1C	htc _dma_ctl	RW	HTC engine control register
		0x28	htc _dma_status	RO	HTC engine status register
		0x30	htc _dma_dsc_compl_cnt	RO	HTC engine SGDMA completed descriptor count register
		0x38	htc _dma_intr_mask	RW	HTC engine SGDMA interrupt enable register
4	0/1/2/3	0x04	cth_dsc_adj	RW	Number of adjacent descriptors for CTH
		0x08	cth_dsc_crd	RW	CTH channel credit value
		0x0C	cth_dsc_addr_l	RW	Lower 32 bits of the CTH descriptor base address
		0x10	cth_dsc_addr_h	RW	Upper 32 bits of the CTH descriptor base address
		0x14	cth _dma_wb_addr_l	RW	Lower 32 bits of the host memory address where the CTH engine writes back the number of completed descriptors
		0x18	cth _dma_wb_addr_h	RW	Upper 32 bits of the host memory address where the CTH engine writes back the number of completed descriptors
		0x1C	cth _dma_ctl	RW	CTH engine control register
		0x28	cth _dma_status	RW	CTH engine status register
		0x30	cth _dma_dsc_compl_cnt	RW	CTH engine SGDMA completed descriptor count register
		0x38	cth _dma_intr_mask	RW	CTH engine SGDMA interrupt enable register

cfg_identifier (0x0000)

Table 8: cfg_identifier (0x0000)

Bit(s)	Default	Access	Description
31:24	8'h0	RO	Reserved
23:12	12'h100	RO	Identifier
12:5	8'h0	RO	Reserved
4:0	4'h3	RO	Configuration identifier

reg_user_max_payload (0x0004)

Table 9: reg_user_max_payload (0x0004)

Bit(s)	Default	Access	Description
6:4	3'h5	RW	Programmed maximum payload size issued to the user application. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. 3'b011: 1024 bytes. 3'b100: 2048 bytes. 3'b101: 4096 bytes.
3	–	–	Reserved
2:0	3'h5	RO	Actual maximum payload size issued to the user application. This value may be lower than user_prg_payload due to IP configuration or data path width. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. 3'b011: 1024 bytes. 3'b100: 2048 bytes. 3'b101: 4096 bytes.

reg_user_max_read_req (0x0008)

Table 10: reg_user_max_read_req (0x0008)

Bit(s)	Default	Access	Description
6:4	3'h5	RW	Maximum read request size issued to the user application. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. 3'b011: 1024 bytes. 3'b100: 2048 bytes. 3'b101: 4096 bytes.
3	–	–	Reserved
2:0	3'h5	RO	Actual maximum read request size issued to the user application. This value may be lower than user_max_read due to PCIe configuration or data path width. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. 3'b011: 1024 bytes. 3'b100: 2048 bytes. 3'b101: 4096 bytes.

reg_write_timeout (0x000C)

Table 11: reg_write_timeout (0x000C)

Bit(s)	Default	Access	Description
4:0	5'h0	RW	Write flush timeout. Applies to the AXI4-Stream CTH channel. This register specifies the number of clock cycles the channel waits for data before flushing received write data from PCIe. This operation closes the descriptor and triggers a writeback. A value of 0 disables the timeout. Timeout period = 2^{value} clock cycles.

msi_enable and msix_enable (0x0018)

Table 12: msi_enable & msix_enable (0x0018)

Bit(s)	Default	Access	Description
1	–	RO	MSI interrupt enable.
0	–	RO	MSI-X interrupt enable.

max_payload (0x001C)

Table 13: max_payload (0x001C)

Bit(s)	Default	Access	Description
2:0	–	RO	Maximum write payload size. This value is the minimum of the PCIe IP Maximum Payload Size (MPS) and the SGDMA/Bridge Subsystem for PCIe parameter. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. 3'b011: 1024 bytes. 3'b100: 2048 bytes. 3'b101: 4096 bytes.

max_read_req (0x0020)

Table 14: max_read_req (0x0020)

Bit(s)	Default	Access	Description
2:0	–	RO	Maximum read request size. This value is the minimum of the PCIe IP MPS and the SGDMA/Bridge Subsystem for PCIe parameter. 3'b000: 128 bytes. 3'b001: 256 bytes. 3'b010: 512 bytes. 3'b011: 1024 bytes. 3'b100: 2048 bytes. 3'b101: 4096 bytes.

clk_pciew (0x0024)

Table 15: clk_pciew (0x0024)

Bit(s)	Default	Access	Description
2:0	–	RO	PCIe AXI4-Stream interface width. 0: 64-bit 1: 128-bit 2: 256-bit 3: 512-bit

dsc_crd_en (0x1004)

Table 16: dsc_crd_en (0x1004)

Bit(s)	Default	Access	Description
31:20	–	–	Reserved
19:16	4'h0	RW	HTC engine descriptor credit enable. One bit per HTC engine; active high.
15:4	–	–	Reserved
3:0	4'h0	RW	CTH engine descriptor credit enable. One bit per CTH engine; active high.

dsc_stop (0x100C)

Table 17: dsc_stop (0x100C)

Bit(s)	Default	Access	Description
31:20	–	–	Reserved
19:16	4'h0	RW	Pause control bits for the descriptor engine to stop fetching descriptors from CTH engines. One bit per CTH engine; setting a bit high pauses fetching, setting it low resumes operation.
15:4	–	–	Reserved
3:0	1'h0	RW	Pause control bits for the descriptor engine to stop fetching descriptors from HTC engines. One bit per HTC engine; setting a bit high pauses fetching, setting it low resumes operation.

irq_identifier (0x2000)

Table 18: irq_identifier (0x2000)

Bit(s)	Default	Access	Description
31:24	8'h0	RO	Reserved
23:12	12'h100	RO	SGDMA internal register group identifier.
12:5	8'h0	RO	Reserved
4:0	4'h2	RO	IRQ interrupt register group type ID.

usr_irq (0x2004)

Table 19: usr_irq (0x2004)

Bit(s)	Name	Access	Description
[USER_INTERRUPT_NUM:0]	usr_irq	RO	User interrupt request. This register reflects the interrupt status only when both the interrupt source and the corresponding enable mask register are asserted.

dma_irq (0x2008)

Table 20: dma_irq (0x2008)

Bit(s)	Default	Access	Description
[31:NUM_ENGINE]	—	—	Reserved
[NUM_ENGINE-1:0]	'h0	RO	SGDMA interrupt request status for HTC and CTH engines. NUM_ENGINE represents the total number of HTC and CTH engines. Each bit corresponds to the interrupt request of one engine. A bit is asserted only when both the interrupt source and its corresponding interrupt enable bit are active. The mapping of each bit to the respective engine is illustrated in the reference diagram. HTC engines are mapped starting from bit [0], followed by CTH engines.

user_irq_in (0x200C)

Table 21: user_irq_in (0x200C)

Bit(s)	Name	Access	Description
[USER_INTERRUPT_NUM:0]	usr_irq_in	RO	User interrupt pending. This register indicates the presence of a pending event. The pending event can be cleared by removing the event cause condition on the source component.

dma_irq_i (0x2010)

Table 22: dma_irq_i (0x2010)

Bit(s)	Default	Access	Description
[31:NUM_ENGINE]	–	–	Reserved
[NUM_ENGINE-1:0]	'h0	RO	Interrupt pending status for each engine. Each bit corresponds to a read or write engine. This register indicates that a pending interrupt condition exists. The pending status can be cleared by removing the cause of the interrupt condition at the source component. The bits for HTC (Host to Card) engines always start from bit 0. The bits for CTH (Card to Host) engines follow the last HTC bit.

user_irq_lut (0x2014)

Table 23: user_irq_lut (0x2014)

Bit(s)	Name	Access	Description
28:24	clk_usr_irq_lut [{2'h0, 2'h3}]	RW	default: 5'h0 Vector 3 Vector ID used when user IRQ usr_irq[3] triggers an interrupt.
20:16	clk_usr_irq_lut [{2'h0, 2'h2}]	RW	default: 5'h0 Vector 2 Vector ID used when user IRQ usr_irq[2] triggers an interrupt.
12:8	clk_usr_irq_lut [{2'h0, 2'h1}]	RW	default: 5'h0 Vector 1 Vector ID used when user IRQ usr_irq[1] triggers an interrupt.
4:0	clk_usr_irq_lut [{2'h0, 2'h0}]	RW	default: 5'h0 Vector 0 Vector ID used when user IRQ usr_irq[0] triggers an interrupt.

user_irq_lut (0x2018)

Table 24: user_irq_lut (0x2018)

Bit(s)	Name	Access	Description
28:24	clk_usr_irq_lut [{2'h1, 2'h3}]	RW	default: 5'h0 Vector 7 Vector ID used when user IRQ usr_irq[7] triggers an interrupt.
20:16	clk_usr_irq_lut [{2'h1, 2'h2}]	RW	default: 5'h0 Vector 6 Vector ID used when user IRQ usr_irq[6] triggers an interrupt.
12:8	clk_usr_irq_lut [{2'h1, 2'h1}]	RW	default: 5'h0 Vector 5 Vector ID used when user IRQ usr_irq[5] triggers an interrupt.

Bit(s)	Name	Access	Description
4:0	clk_usr_irq_lut [{2'h1, 2'h0}]	RW	default: 5'h0 Vector 4 Vector ID used when user IRQ usr_irq[4] triggers an interrupt.

user_irq_lut (0x201C)

Table 25: user_irq_lut (0x201C)

Bit(s)	Name	Access	Description
28:24	clk_usr_irq_lut [{2'h2, 2'h3}]	RW	default: 5'h0 Vector 11 Vector ID used when user IRQ usr_irq[11] triggers an interrupt.
20:16	clk_usr_irq_lut [{2'h2, 2'h2}]	RW	default: 5'h0 Vector 10 Vector ID used when user IRQ usr_irq[10] triggers an interrupt.
12:8	clk_usr_irq_lut [{2'h2, 2'h1}]	RW	default: 5'h0 Vector 9 Vector ID used when user IRQ usr_irq[9] triggers an interrupt.
4:0	clk_usr_irq_lut [{2'h2, 2'h0}]	RW	default: 5'h0 Vector 8 Vector ID used when user IRQ usr_irq[8] triggers an interrupt.

user_irq_lut (0x2020)

Table 26: user_irq_lut (0x2020)

Bit(s)	Name	Access	Description
28:24	clk_usr_irq_lut [{2'h3, 2'h3}]	RW	default: 5'h0 Vector 15 Vector ID used when user IRQ usr_irq[15] triggers an interrupt.
20:16	clk_usr_irq_lut [{2'h3, 2'h2}]	RW	default: 5'h0 Vector 14 Vector ID used when user IRQ usr_irq[14] triggers an interrupt.
12:8	clk_usr_irq_lut [{2'h3, 2'h1}]	RW	default: 5'h0 Vector 13 Vector ID used when user IRQ usr_irq[13] triggers an interrupt.
4:0	clk_usr_irq_lut [{2'h3, 2'h0}]	RW	default: 5'h0 Vector 12 Vector ID used when user IRQ usr_irq[12] triggers an interrupt.

dma_irq_lut (0x2024)

Table 27: dma_irq_lut (0x2024)

Bit(s)	Name	Access	Description
28:24	clk_dma_irq_lut [{2'h0, 2'h3}]	RW	default: 5'h0 Vector 3 Vector ID used when SGDMA channel 3 triggers an interrupt.
20:16	clk_dma_irq_lut [{2'h0, 2'h2}]	RW	default: 5'h0 Vector 2 Vector ID used when SGDMA channel 2 triggers an interrupt.
12:8	clk_dma_irq_lut [{2'h0, 2'h1}]	RW	default: 5'h0 Vector 1 Vector ID used when SGDMA channel 1 triggers an interrupt.
4:0	clk_dma_irq_lut [{2'h0, 2'h0}]	RW	default: 5'h0 Vector 0 Vector ID used when SGDMA channel 0 triggers an interrupt.

dma_irq_lut (0x2028)

Table 28: dma_irq_lut (0x2028)

Bit(s)	Name	Access	Description
28:24	clk_dma_irq_lut [{2'h1, 2'h3}]	RW	default: 5'h0 Vector 7 Vector ID used when SGDMA channel 7 triggers an interrupt.
20:16	clk_dma_irq_lut [{2'h1, 2'h2}]	RW	default: 5'h0 Vector 6 Vector ID used when SGDMA channel 6 triggers an interrupt.
12:8	clk_dma_irq_lut [{2'h1, 2'h1}]	RW	default: 5'h0 Vector 5 Vector ID used when SGDMA channel 5 triggers an interrupt.
4:0	clk_dma_irq_lut [{2'h1, 2'h0}]	RW	default: 5'h0 Vector 4 Vector ID used when SGDMA channel 4 triggers an interrupt.

user_en (0x202C)

Table 29: user_en (0x202C)

Bit(s)	Name	Access	Description
[USER_INTERRUPT_NUM:0]	usr_ena	RW	default: 'h0 user_int_enmask User interrupt enable mask 0: Interrupt generation is blocked when the user interrupt source is asserted. 1: An interrupt is generated on the rising edge of the user interrupt source. If both the "Enable Mask" and the source are set, a user interrupt is generated.

dma_en (0x2038)

Table 30: dma_en (0x2038)

Bit(s)	Name	Access	Description
[DMA_INTERRUPT_NUM:0]	dma_ena	RW	channel_int_enmask SGDMA interrupt enable mask. Each read/write engine is mapped to one bit. 0: Interrupt generation is blocked when the interrupt source is asserted. HTC bits always start from bit 0. CTH bits are placed after the last HTC bit, depending on the `HTC_CH_NUM` parameter. 1: An interrupt is generated on the rising edge of the interrupt source. If both the enmask bit and the source are asserted, an interrupt is generated.

htc_dsc_adj (0x3004)

Table 31: htc_dsc_adj (0x3004)

Bit(s)	Default	Access	Description
[31:6]	–	–	Reserved, fixed to 0.
[5:0]	6'h0	RW	Number of descriptors following the starting descriptor in the first descriptor block of the HTC engine.

htc_dsc_crd (0x3008)

Table 32: htc_dsc_crd (0x3008)

Bit(s)	Default	Access	Description
[31:10]	–	–	Reserved, fixed to 0.
[9:0]	10'h0	RW	Descriptor credit value to be added for the HTC engine. This register is valid only when enabled per channel in the Descriptor Credit Mode register.

htc_dma_ctl (0x301C)

Table 33: htc _dma_ctl (0x301C)

Bit(s)	Default	Access	Description
31:28	–	–	Reserved
27:23	5'h0	RW	Enables descriptor error logging in the Status register (0x18) when set to all 1s (0x1F). An error stops the engine; if the interrupt enable bit is asserted, an interrupt is generated simultaneously.
22:18	5'h0	RW	Enables write error logging in the Status register when set to all 1s (0x1F). An error stops the engine; if the interrupt enable bit is asserted, an interrupt is generated simultaneously.
17:13	5'h0	RW	Enables read error logging in the Status register when set to all 1s (0x1F). An error stops the engine; if the interrupt enable bit is asserted, an interrupt is generated simultaneously.
12	–	–	Reserved
11	1'b0	RW	In AXI4-Stream mode, disables HTC write-back and enables default write-back when set to 1.
10	0x0	RW	Polling mode enable. When set to 1, the HTC engine automatically writes back the completion count to a predefined memory address after processing descriptors with the Completed flag, for host polling.
9	1'b0	RW	Non-incrementing address mode control. When set to 1, HTC user interface uses non-incrementing addresses. This is only supported for AXI4-MM read interface.
8:7	–	–	Reserved
6	1'b0	RW	Enables logging of descriptor completion flags in the Status register. If the corresponding interrupt enable bit is asserted, an interrupt is triggered simultaneously.
5	1'b0	RW	Enables logging of descriptor stop flags in the Status register. If the corresponding interrupt enable bit is asserted, an interrupt is triggered simultaneously.
4	1'b0	RW	Enables logging of descriptor flag mismatch in the Status register. If the corresponding interrupt enable bit is asserted, an interrupt is triggered simultaneously.
3	1'b0	RW	Enables logging of address alignment errors in the Status register. If the corresponding interrupt enable bit is asserted, an interrupt is triggered simultaneously.
2	1'b0	RW	Enables logging of idle stop in the Status register. If the corresponding interrupt enable bit is asserted, an interrupt is triggered simultaneously.
1	1'b0	RW	Enables logging of data length not being a multiple of AXI4-Stream interface width in the Status register. If the corresponding interrupt enable bit is asserted, an interrupt is triggered simultaneously.
0	1'b0	RW	Run bit for the HTC engine. When set to 1, the engine starts data transfer. Clearing this bit stops the engine after completing the current descriptor if busy.

htc_dma_status (0x3028)

Table 34: htc _dma_status (0x3028)

Bit(s)	Default	Access	Description
31:24	–	–	Reserved

Bit(s)	Default	Access	Description
23:19	5'h0	RW	Write error status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. Bit descriptions: • bit[23:21]: Reserved. • bit[20]: Slave interface error. • bit[19]: Decoder error.
18:14	5'h0	RW	Read error status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. Bit descriptions: • bit[18]: Unexpected completion. • bit[17]: Header EP. • bit[16]: Parity error. • bit[15]: Completer abort. • bit[14]: Unsupported request.
13:9	5'h0	RW	Descriptor error status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. Bit descriptions: • bit[13]: Unexpected completion. • bit[12]: Header EP. • bit[11]: Parity error. • bit[10]: Completer abort. • bit[9]: Unsupported request.
8:7	–	–	Reserved
6	1'b0	RW	Descriptor Completed flag status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high by the HTC engine after completing the data transfer of a descriptor with the Completed flag.
5	1'b0	RW	Descriptor Stop flag status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high by the HTC engine after completing the data transfer of a descriptor with the Stop flag.
4	1'b0	RW	Idle transition status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high by the HTC engine after it transitions to the idle state following the clearing of the run bit.
3	1'b0	RW	Data length not a multiple of AXI4-Stream width status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high when the HTC engine detects that the data transfer length is not an integer multiple of the AXI4-Stream interface width.
2	1'b0	RW	Descriptor flag mismatch status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high when the HTC engine detects a mismatch in descriptor flags.

Bit(s)	Default	Access	Description
1	1'b0	RW	Address boundary exception status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high when the source or destination address does not meet boundary requirements.
0	1'b0	RO	HTC engine Busy status. This bit is set to 1 when the HTC engine is performing data transfer, and cleared to 0 when the engine is idle.

htc_dma_dsc_compl_cnt (0x3030)

Table 35: htc_dma_dsc_compl_cnt (0x3030)

Bit(s)	Default	Access	Description
[31:0]	32'h0	RO	Indicates the number of descriptors that have completed data transfer. This register is cleared when the run bit [0] of the control register transitions from 0 to 1.

htc_dma_intr_mask (0x3038)

Table 36: htc_dma_intr_mask (0x3038)

Bit(s)	Default	Access	Description
31:24	–	–	Reserved
23:19	5'h0	RW	Corresponds to the write error status fields in the Status register. • Bit[23:21]: Reserved. • Bit[20]: Interrupt enable bit for slave interface error. • Bit[19]: Interrupt enable bit for decoder error. When set to 1, an interrupt is generated simultaneously as the corresponding error is logged in the Status register.
18:14	5'h0	RW	Corresponds to the read error status fields in the Status register. • Bit[18]: Interrupt enable bit for unexpected completion. • Bit[17]: Interrupt enable bit for header EP. • Bit[16]: Interrupt enable bit for parity error. • Bit[15]: Interrupt enable bit for completer abort. • Bit[14]: Interrupt enable bit for unsupported request. When set to 1, an interrupt is generated simultaneously as the corresponding error is logged in the Status register.
13:9	5'h0	RW	Corresponds to the descriptor error status fields in the Status register. Shares the same bit definitions as bits[18:14]. When set to 1, an interrupt is generated simultaneously as the corresponding error is logged in the Status register.
8:7	–	–	Reserved
6	1'b0	RW	Interrupt enable bit corresponding to the status field for descriptors with the Completed flag. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
5	1'b0	RW	Interrupt enable bit corresponding to the status field for descriptors with the Stop flag. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
4	1'b0	RW	Interrupt enable bit corresponding to the idle status field of the HTC engine. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.

Bit(s)	Default	Access	Description
3	1'b0	RW	Interrupt enable bit corresponding to the status field for data transfer lengths not being an integer multiple of the AXI4-Stream interface width. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
2	1'b0	RW	Interrupt enable bit corresponding to the status field for descriptor flag mismatches. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
1	1'b0	RW	Interrupt enable bit corresponding to the status field for boundary violations of source or destination addresses. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
0	–	–	Reserved

cth_dsc_adj (0x4004)

Table 37: cth_dsc_adj (0x4004)

Bit(s)	Default	Access	Description
[31:6]	–	–	Reserved, fixed to 0.
[5:0]	6'h0	RW	Number of descriptors following the initial descriptor in the first descriptor block of the CTH engine in host memory.

cth_dsc_crd (0x4008)

Table 38: cth_dsc_crd (0x4008)

Bit(s)	Default	Access	Description
[31:10]	–	–	Reserved, fixed to 0.
[9:0]	10'h0	RW	Descriptor credit value to be added for the CTH engine. This register is valid only when enabled per channel in the Descriptor Credit Mode register.

cth_dma_ctl (0x401C)

Table 39: cth _dma_ctl (0x401C)

Bit(s)	Default	Access	Description
31:28	–	–	Reserved
27:23	5'h0	RW	When all bits in this field are set to 1 (i.e., 0x1F), the descriptor error logging in the Status register (0x18) is enabled. Upon detecting a corresponding error, the engine halts. If the associated interrupt enable bit is set to 1, an interrupt is generated.
22:18	5'h0	RW	When all bits in this field are set to 1 (i.e., 0x1F), the write error logging in the Status register is enabled. Upon detecting a corresponding error, the engine halts. If the associated interrupt enable bit is set to 1, an interrupt is generated.
17:13	5'h0	RW	When all bits in this field are set to 1 (i.e., 0x1F), the read error logging in the Status register is enabled. Upon detecting a corresponding error, the engine halts. If the associated interrupt enable bit is set to 1, an interrupt is generated.
12	–	–	Reserved

Bit(s)	Default	Access	Description
11	1'b0	RW	When the CTH engine's user-side interface is set to AXI4-Stream mode, setting this bit to 1 disables CTH writeback information and enables the default writeback behavior.
10	0x0	RW	Writeback enable bit for polling mode. When set to 1, after the CTH engine completes data transfer for descriptors with the Completed flag, the writeback engine writes the number of completed descriptors to a preconfigured memory address for the host to check transfer completion.
9	1'b0	RW	Non-incremental address mode control bit. When set to 1, address auto-increment is disabled during data transfer from the user interface side of the CTH engine. This function only applies to the AXI4-MM read interface.
8:7	–	–	Reserved
6	1'b0	RW	When set to 1, enables logging of descriptors with the Completed flag in the Status register. If the corresponding interrupt enable bit is 1, an interrupt is generated.
5	1'b0	RW	When set to 1, enables logging of descriptors with the Stop flag in the Status register. If the corresponding interrupt enable bit is 1, an interrupt is generated.
4	1'b0	RW	When set to 1, enables logging of descriptor ID mismatch in the Status register. If the corresponding interrupt enable bit is 1, an interrupt is generated.
3	1'b0	RW	When set to 1, enables logging of address misalignment in the Status register. If the corresponding interrupt enable bit is 1, an interrupt is generated.
2	1'b0	RW	When set to 1, enables logging of idle halt in the Status register. If the corresponding interrupt enable bit is 1, an interrupt is generated.
1	1'b0	RW	When set to 1, enables logging of data length not being an integer multiple of the AXI4-Stream interface data width in the Status register. If the corresponding interrupt enable bit is 1, an interrupt is generated.
0	1'b0	RW	CTH engine run enable bit. When set to 1, the CTH engine starts data transfer. When cleared to 0, the transfer is stopped. If the engine is busy when cleared, it finishes the current descriptor before stopping.

cth_dma_status (0x4028)

Table 40: cth_dma_status (0x4028)

Bit(s)	Default	Access	Description
31:24	–	–	Reserved
23:19	5'h0	RW	Write error status field. This field is cleared when bit[0] (Run) of the control register transitions from 0 to 1. Each bit has the following meaning: • bit[23:21]: Reserved. • bit[20]: Slave interface error. • bit[19]: Decoder error.
18:14	5'h0	RW	Read error status field. This field is cleared when bit[0] (Run) of the control register transitions from 0 to 1. Each bit has the following meaning: • bit[18]: Unexpected completion. • bit[17]: Header EP. • bit[16]: Parity error. • bit[15]: Completer aborted. • bit[14]: Unsupported request.

Bit(s)	Default	Access	Description
13:9	5'h0	RW	Descriptor error status field. This field is cleared when bit[0] (Run) of the control register transitions from 0 to 1. Each bit has the following meaning: • bit[13]: Unexpected completion. • bit[12]: Header EP. • bit[11]: Parity error. • bit[10]: Completer aborted. • bit[9]: Unsupported request.
8:7	–	–	Reserved
6	1'b0	RW	Status field for descriptors with the Completed flag. This field is cleared when bit[0] (Run) of the control register transitions from 0 to 1. When the corresponding status logging in the control register is enabled, this bit is asserted high after the CTH engine completes data transfer for descriptors with the Completed flag.
5	1'b0	RW	Status field for descriptors with the Stop flag. Same behavior as bit[6], but for descriptors with the Stop flag.
4	1'b0	RW	Idle transition status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high by the CTH engine after it transitions to the idle state following the clearing of the run bit.
3	1'b0	RW	Data length not a multiple of AXI4-Stream width status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high when the CTH engine detects that the data transfer length is not an integer multiple of the AXI4-Stream interface width.
2	1'b0	RW	Descriptor flag mismatch status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high when the CTH engine detects a mismatch in descriptor flags.
1	1'b0	RW	Address boundary exception status field. This field is cleared when the run bit [0] of the control register transitions from 0 to 1. When the corresponding logging enable bit in the control register is set to 1, this bit is asserted high when the source or destination address does not meet boundary requirements.
0	1'b0	RO	CTH engine Busy status. This bit is set to 1 when the CTH engine is performing data transfer, and cleared to 0 when the engine is idle.

cth_dma_dsc_compl_cnt (0x4030)

Table 41: cth_dma_dsc_compl_cnt (0x4030)

Bit(s)	Default	Access	Description
[31:0]	32'h0	RO	Indicates the number of descriptors that have completed data transfer. This register is cleared when the run bit [0] of the control register transitions from 0 to 1.

cth_dma_intr_mask (0x4038)

Table 42: cth_dma_intr_mask (0x4038)

Bit(s)	Default	Access	Description
31:24	–	–	Reserved
23:19	5'h0	RW	Corresponds to the write error status fields in the Status register. • Bit [23:21]: Reserved. • Bit [20]: Interrupt enable bit for slave interface error. • Bit [19]: Interrupt enable bit for decoder error. When set to 1, an interrupt is generated simultaneously as the corresponding error is logged in the Status register.
18:14	5'h0	RW	Corresponds to the read error status fields in the Status register. • Bit[18]: Interrupt enable bit for unexpected completion. • Bit[17]: Interrupt enable bit for header EP. • Bit[16]: Interrupt enable bit for parity error. • Bit[15]: Interrupt enable bit for completer abort. • Bit[14]: Interrupt enable bit for unsupported request. When set to 1, an interrupt is generated simultaneously as the corresponding error is logged in the Status register.
13:9	5'h0	RW	Corresponds to the descriptor error status fields in the Status register. Shares the same bit definitions as bits[18:14]. When set to 1, an interrupt is generated simultaneously as the corresponding error is logged in the Status register.
8:7	–	–	Reserved
6	1'b0	RW	Interrupt enable bit corresponding to the status field for descriptors with the Completed flag. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
5	1'b0	RW	Interrupt enable bit corresponding to the status field for descriptors with the Stop flag. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
4	1'b0	RW	Interrupt enable bit corresponding to the idle status field of the CTH engine. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
3	1'b0	RW	Interrupt enable bit corresponding to the status field for data transfer lengths not being an integer multiple of the AXI4-Stream interface width. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
2	1'b0	RW	Interrupt enable bit corresponding to the status field for descriptor flag mismatches. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
1	1'b0	RW	Interrupt enable bit corresponding to the status field for boundary violations of source or destination addresses. When set to 1, an interrupt is generated simultaneously as the status is logged in the Status register.
0	–	–	Reserved

MSI-X Interrupt Registers (0x8000)

The SGDMA IP supports MSI-X interrupts. The interrupt vector table supports up to 32 interrupt vectors. The table starts at address `0x8000`, and the Pending Bit Array (PBA) is located at address `0x8FE0`. Details are as follows.

Table 43: Interrupt Vector Table

Byte Offset	Bits	Default	Access	Description
0x00	[31:0]	32'h0	RW	Lower 32 bits of the address for MSI-X interrupt vector 0.
0x04	[31:0]	32'h0	RW	Upper 32 bits of the address for MSI-X interrupt vector 0.
0x08	[31:0]	32'h0	RW	Data field for MSI-X interrupt vector 0.
0x0C	[31:0]	32'hFFFFFFFF	RW	Control field for MSI-X interrupt vector 0. • Bits [31:1]: Reserved. • Bit [0]: Mask bit for the MSI-X interrupt vector. When set to 1, the interrupt message is suppressed; when cleared to 0, the interrupt message is allowed.
...	–	–	–	–
0x1F0	[31:0]	32'h0	RW	Lower 32 bits of the address for MSI-X interrupt vector 31.
0x1F4	[31:0]	32'h0	RW	Upper 32 bits of the address for MSI-X interrupt vector 31.
0x1F8	[31:0]	32'h0	RW	Data field for MSI-X interrupt vector 31.
0x1FC	[31:0]	32'hFFFFFFFF	RW	Control field for MSI-X interrupt vector 31. • Bits [31:1]: Reserved. • Bit [0]: Mask bit for the MSI-X interrupt vector. When set to 1, the interrupt message is suppressed; when cleared to 0, the interrupt message is allowed.
0xFE0	[31:0]	32'h0	RW	Pending Bit Array (PBA). Each bit corresponds to an interrupt vector.

Software Driver

Driver Build System Requirements

The current driver supports Linux operating systems. Reference environments are listed below:

- Kernel version: 4.12 to 6.x
- **Ubuntu 22.04 Desktop**
- **Ubuntu 22.04 Server**

Environment Dependencies

To install the required dependencies, follow the steps below.

1. Open the terminal.
2. Enter the following command:

```
sudo apt-get install make g++ linux-headers-generic -y
```

   - You will be prompted to enter your password.
   - An internet connection is required for this step.

Driver Installation

To install the drivers, follow the steps below.

1. Extract the driver package.
2. Navigate to the **driver** directory.
3. Run the following command to compile the driver:

```
#make
```

The kernel module **pcie_dma.ko** is generated.

Loading Drivers

Run the following command to load the drivers:

```
sudo insmod pcie_dma.ko intr=3
```

The table below describes the parameters.

Table 44: Parameter Descriptions

Parameter Name	Value	Description
intr	0: Auto 1: Reserved 2: Legacy mode 3: MSI-X mode	Interrupt mode. Suitable for low CPU usage scenarios.
poll	1: Enable polling mode	Polling mode for maximum performance. Provides the highest bandwidth at the cost of additional CPU usage.

Operational Demonstration (Summary)

Table 45: Demo Operation Summary

Category	Step	Command	Description
Install	Environment setup	`sudo apt-get install make g++ linux-headers-generic -y`	Run once during the first installation.
	Driver compilation	`cd driver` `make`	
Use	Load driver	`cd driver` `sudo insmod pcie_dma.ko intr=3`	Execute once after system startup.

Example Design Description

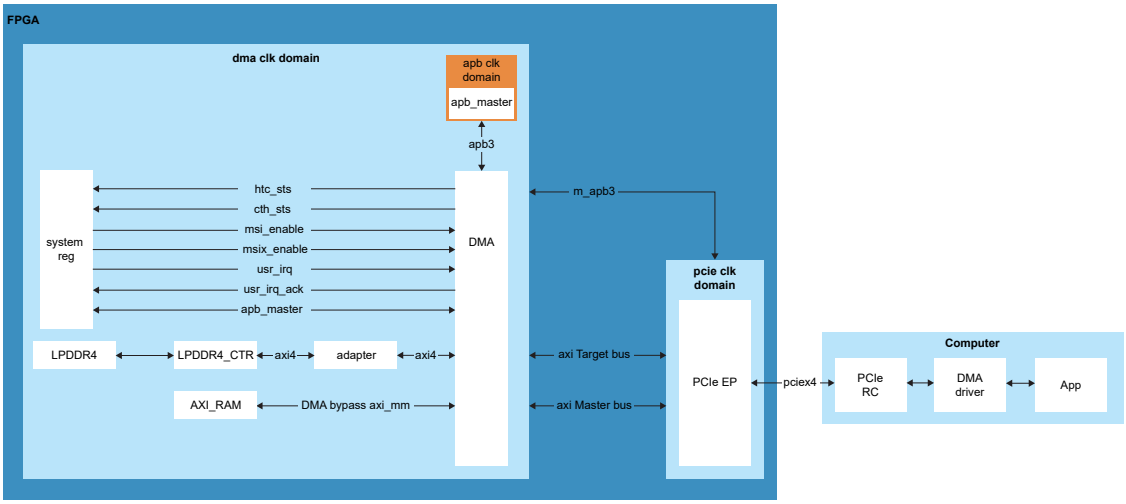
The following sections provide an overview of the SGDMA IP configuration.

Overall Block Diagram

As shown in the figure below, the framework is divided into three clock domains:

- the APB (hardcore configuration) clock domain,
- the PCIe hardcore clock domain, and
- the SGDMA clock domain.

Figure 3: DMA_AXI_MM Example Overall Block Diagram



Functional verification is categorized based on the three data interfaces in AXI MM mode:

- **SGDMA User AXI MM Interface**: This interface handles data from PCIe traffic passed through SGDMA. The data flow can be verified using descriptors sent by the host driver, testing SGDMA's data transfer functionality (CTH, HTC).
- **AXI Bypass Interface**: This data interface allows PCIe traffic to bypass SGDMA and directly access data. It can be mapped to the PCIe BAR space for use by the host system.
- **APB Master Interface**: This data interface enables PCIe to bypass SGDMA to access the APB master interface. It can also be mapped to the PCIe BAR space, allowing the host system to configure user logic.

Inbound Configuration

The example design configures three BARs, which are mapped to the AXI target address space through register settings. The size of the hardcore BAR address space corresponds to the **ADDR_SPACE** parameters in the table below, measured in bytes.

- The **SGDMA m_apb3** bus is connected to **BAR0**.
- The **system_reg** module is connected to **BAR4**.
- The **axi_ram** module for SGDMA bypass testing is connected to **BAR2**.

Figure 4: Inbound Configuration

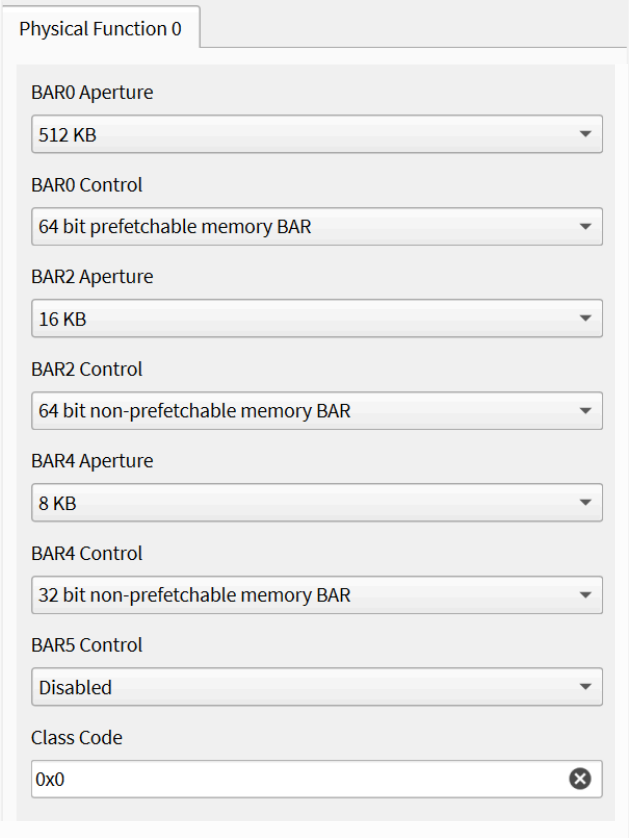


Table 46: Inbound BAR Register Mapping Table

Register Name	Value	Description (new)
M0_BASE_ADDR	64'h0	Base address of BAR0, used by the host driver to configure SGDMA registers. Corresponds to AXI inbound BAR0 register config.
M0_ADDR_SPACE	64'h80000	BAR0 address space size, default is 512 KB. Corresponds to hardcore BAR0 config.
M1_BASE_ADDR	64'h90000	Base address of BAR4, used by the host driver to configure the APB master interface. Corresponds to AXI inbound BAR4 register config.
M1_ADDR_SPACE	64'h2000	BAR4 address space size, default is 8 KB. Corresponds to hardcore BAR4 config.
M2_BASE_ADDR	64'ha0000	Base address of BAR2, used for SGDMA bypass AXI address space (traffic bypassing SGDMA). Corresponds to AXI inbound BAR2 register config.
M2_ADDR_SPACE	64'h4000	BAR2 address space size, default is 16 KB. Corresponds to hardcore BAR2 config.

Table 47: Inbound Configuration Table

Address [Region0]	Address	Value
bar0_addr0	0x400840	0x00000000 (default)
bar0_addr1	0x400844	0x00000000 (default)
bar2_addr0	0x400850	0x000a0000
bar2_addr1	0x400854	0x00000000 (default)
bar4_addr0	0x400860	0x00090000
bar4_addr1	0x400864	0x00000000 (default)

Outbound Configuration

The Region0 configuration for PCIe outbound is shown below. It maps access from the SGDMA's `m_pcie_axi4` bus to the host's 48-bit address space.

Table 48: Outbound Configuration Table

Address [Region0]	Address	Value
ob_addr0	0x400000	0x0000002f
ob_addr1	0x400004	0x00000000 (default)
desc0	0x400008	0x00000002
desc1	0x40000c	0x00000000 (default)
desc2	0x400010	0x00000000 (default)
desc3	0x400014	0x00000000 (default)
axi_addr0	0x400018	0x0000002f
axi_addr1	0x40001c	0x00000000 (default)

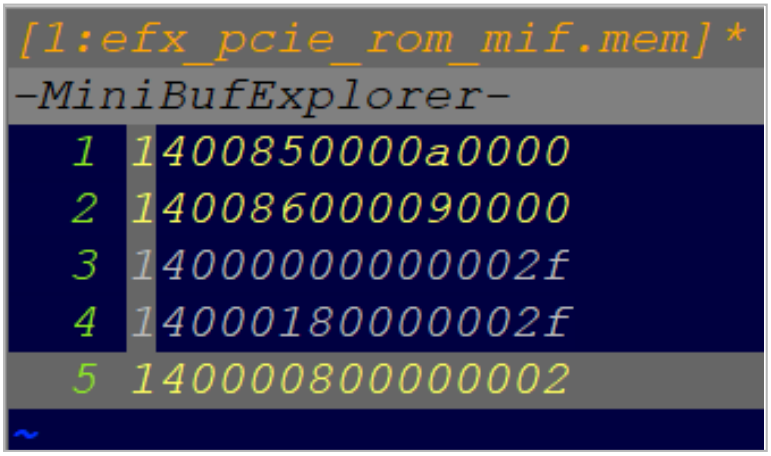
Register Configuration

In the example design project, register configuration is performed at system reboot by reading a user-defined ROM table (`efx_pcie_rom_mif.mem`). In this table, a high bit value of `1` indicates a write operation, while `0` indicates a read operation. For example, `400850` represents the register address to be configured, and `000a0000` is the value to be written.

Fill in the table based on your specific needs and the PCIe register configuration completes itself automatically.

The following example illustrates the register configuration corresponding to the two tables discussed earlier. Registers marked with default values in the table do not need to be configured separately.

Figure 5: Register Configuration



Example Design

This example design demonstrates the capabilities of PCIe DMA to transfer data from a Linux workstation to onboard memory through either the AXI4-MM or AXI4-Stream interfaces. This section will guide you through the hardware setup, installing the DMA driver, and using the test application to perform random payload read and write onto the device memory.

Requirements

This example design will require the following:

- A Linux-based computer (this example design is tested on Linux Kernel version 6.8.0.59-generic Ubuntu 22.04.1)
- Efinity® software v2024.2.294.1 or higher
- TJ-Series TJ375 N1156X Development Kit
- USB-C cable

What's in the Package

The package includes the following files and directories:

Table 49: Package Files and Directories

File or Directory	Description
README.md	Readme file for the example design
/apps/	Test program directory
/apps/dma_test.cpp	Test program source code
/apps/Makefile	Makefile for test program
/apps/bin/	
/apps/bin/dma_test.bin	Compiled test program
/driver/	DMA driver source code
/driver/cdev_ioctl.h	
/driver/common.h	

File or Directory	Description
/driver/device.c	
/driver/dma.c	
/driver/dma.h	
/driver/dma_regs.h	
/driver/file_io.c	
/driver/file_io.h	
/driver/interrupt.c	
/driver/Makefile	Makefile for DMA Driver
/driver/misc.c	
/driver/misc.h	
/driver/poll.c	
/driver/poll.h	
/efinity_project/	Efinity project for example design
/efinity_project/<efinity_project>	

Hardware Setup

Follow these steps to setup your hardware for the demonstration design:

1. Power on the TJ375 N1156X Development Kit (TJ375N1156X-DK) and connect the TJ375N1156X-DK to your computer via the USB-C cable.
2. Launch the Efinity Programmer on your computer.
3. Set the programming mode to **SPI Active using JTAG Bridge (new)**, as shown in the following figure.

Figure 6: Programming Mode

4. Program the device.
   a. Program AXI4-MM mode using the **efx_pcie_edma_example_design_MM.hex** bitstream file located in the **/efinity_project/bitstream/** directory.
   b. Program AXI4-Stream mode using the **efx_pcie_edma_example_design_ST.hex** bitstream file located in **/efinity_project/bitstream/** directory.
5. Connect the TJ375N1156X-DK to the test device (motherboard) as described in the .
6. Verify that the Elitestek® PCIe SGDMA IP can be detected using the following command:

```
sudo lspci -vv -d 1f7a:0100
```

Installing the Linux Driver

Refer to **Driver Build System Requirements** on page 35 for advice on building the Linux driver.

1. Build the Linux DMA driver located in **driver/** directory.
2. Run this command to load the driver in MSI-X interrupt mode.

```
sudo insmod pcie_dma.ko intr=3
```

3. Verify the installation using this command:

```
sudo dmesg
```

4. The expected output from the command is shown below:

```
...
```

```
[ 6183.221787] [PROBE] EDMA driver create cdev interfaces!
[ 6183.221791] [DBG] ecdev 0x000000008d395da4, 236:1, (null), type 0x1.
[ 6183.221865] [DBG] ecdev 0x00000000c3f59d9d, 236:32, (null), type 0x3.
[ 6183.221881] [DBG] ecdev 0x00000000c7c99469, 236:36, (null), type 0x4.
[ 6183.221894] [DBG] ecdev 0x000000004c6da4f5, 236:10, (null), type 0x2.
[ 6183.221908] [DBG] ecdev 0x000000001ef83cfa, 236:11, (null), type 0x2.
[ 6183.221926] [DBG] ecdev 0x00000000bbce0086, 236:12, (null), type 0x2.
[ 6183.221939] [DBG] ecdev 0x000000001cba867e, 236:13, (null), type 0x2.
[ 6183.221952] [DBG] ecdev 0x00000000b8274753, 236:14, (null), type 0x2.
[ 6183.221964] [DBG] ecdev 0x0000000034582fc0, 236:15, (null), type 0x2.
[ 6183.221978] [DBG] ecdev 0x00000000e02c9ff8, 236:16, (null), type 0x2.
[ 6183.221990] [DBG] ecdev 0x0000000085cb250a, 236:17, (null), type 0x2.
[ 6183.222004] [DBG] ecdev 0x0000000036162313, 236:18, (null), type 0x2.
[ 6183.222016] [DBG] ecdev 0x000000007dff87a0, 236:19, (null), type 0x2.
[ 6183.222028] [DBG] ecdev 0x0000000033c3214b, 236:20, (null), type 0x2.
[ 6183.222055] [DBG] ecdev 0x00000000f4e659e9, 236:21, (null), type 0x2.
[ 6183.222076] [DBG] ecdev 0x00000000d2e6c8a0, 236:22, (null), type 0x2.
[ 6183.222092] [DBG] ecdev 0x0000000057bf16c1, 236:23, (null), type 0x2.
[ 6183.222120] [DBG] ecdev 0x000000005a320ebf, 236:24, (null), type 0x2.
[ 6183.222134] [DBG] ecdev 0x000000002f811708, 236:25, (null), type 0x2.
[ 6183.222147] [DBG] ecdev 0x000000003f9899bf, 236:0, (null), type 0x0.
[ 6183.222164] [DBG] ecdev 0x000000000892811e, 236:64, (null), type 0x5.
[ 6183.222189] [DBG] ecdev 0x0000000024442409, 236:68, (null), type 0x6.
[ 6183.222207] [DBG] ecdev 0x000000009e1bfd00, 236:100, (null), type 0x7.
[ 6183.222221] [PROBE] Pcie dma driver probe Efinix device Done!
```

Running the User Driver Application

A test application is included to perform read-write testing at a random address with a random-length payload. Use the following command to run the user driver test application:

```
apps/bin/dma_test.bin
```

Important: The command to run the user driver test application must be run with root permission.

The following shows the expected output from running this command:

```
/dev/pcie_dma0_htc_0: 3
/dev/pcie_dma0_cth_0: 4

****************************************
TEST : Random size write at random address
****************************************
Transfer & Readback 314 B at 0x3e23f363 ... completed in 173 us
Transfer & Readback 65 B at 0x32b1fb36 ... completed in 489 us
Transfer & Readback 224 B at 0x0284df69 ... completed in 71 us
Transfer & Readback 133 B at 0x3d588efc ... completed in 70 us
Transfer & Readback 655 B at 0x07a65c3a ... completed in 70 us
Transfer & Readback 448 B at 0x5df7ee79 ... completed in 68 us
…
```

Simulation Environment Testbench

The aim of this simulation is to transfer two sets of descriptors in opposite directions: two in HTC and two in CTH.

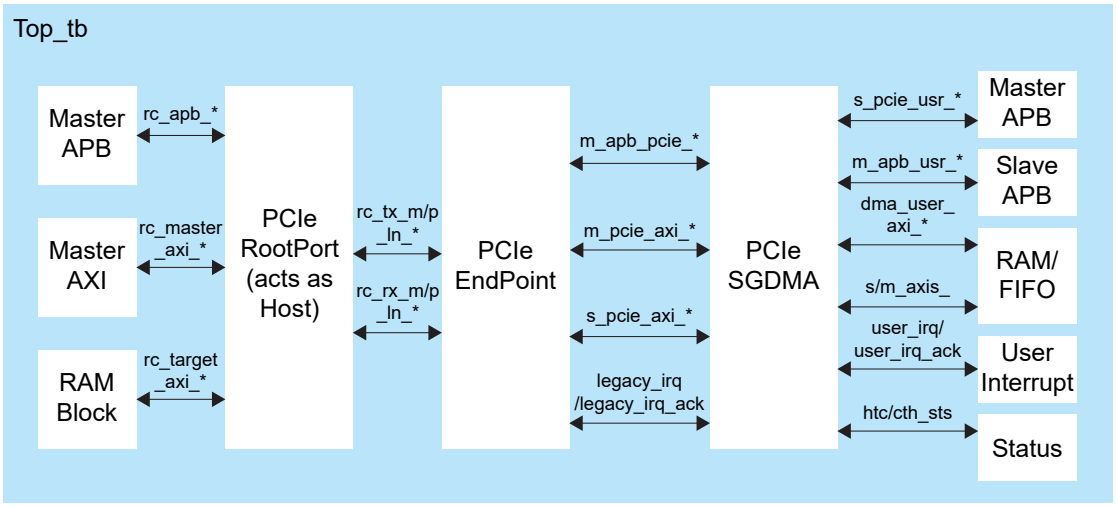
Each descriptor comprises 128 bytes. Data is sent over either AXI4-MM or AXI-Stream in the HTC direction before being sent in the CTH direction.

- HTC: The SGDMA transfers data from the host to the DMA before storing it in user logic (RAM/FIFO)
- CTH: Data is read from the user logic (RAM/FIFO) before being sent from the SGDMA to the host RAM block

This data is read from user RAM in AXI4-MM. Once the transfer is complete, the received data is validated by comparing against the original data at its source. In the case of the AXI-Stream, the data is compared at different locations within host RAM after HTC and CTH data trasnfers are complete.

The following block diagram illustrates the simulation testbench.

Figure 7: Testbench Block Diagram



To run the simulation, a Makefile is included in the delivery (see following table). Use the following command to run the simulation:

```
make sim=<sim_tool> vif=[0 | 1]
```

Arguments:

sim_tool : Simulation tool selection

- mti = Siemens Questasim
- mds = Siemens Modelsim
- vcs = Synopsys VCS
- aldec = Aldec Riviera Pro

vif : DMA interface selection

- 0 = AXI_MM mode
- 1 = AXI_ST mode

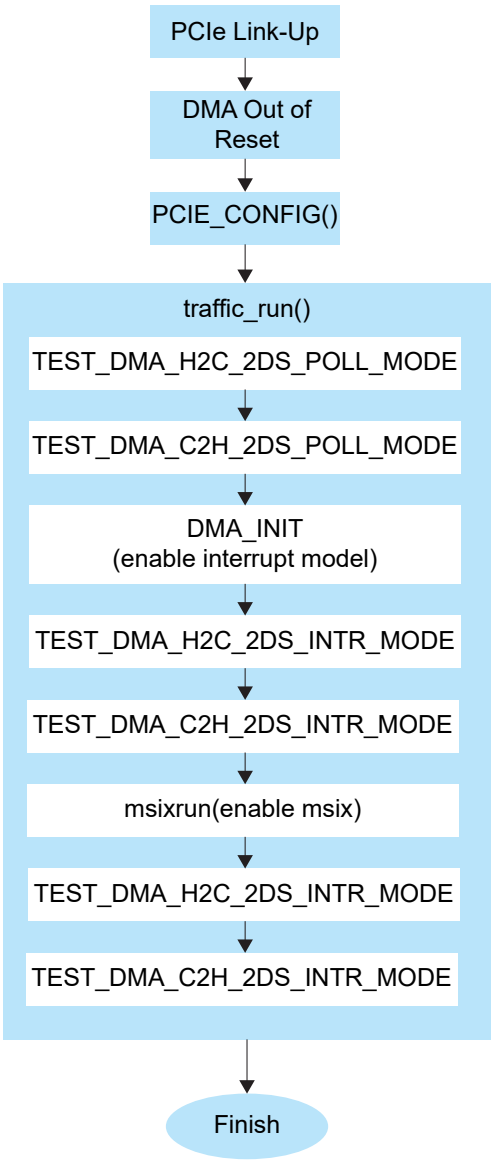
The simulation output will be generated in **dma_traffic/sim_<*timestamp*>**

Table 50: Testbench Files and Directories

File or Directory	Description
/model/efx_serdes_2q_top.<simulator>.v	Titanium transceiver block simulation model
/model/efx_serdes_2q_top_sim.v	Simulation model wrapper
/sim/Makefile	Makefile
/sim/run	Simulation run commands
/sim/filelist.f	Simulator command arguments file
/sim/tb/top_tb.sv	Testbench top module
/sim/tb/dma_port.sv	Port declaration for the test bench
/sim/tb/dma_task.sv	Testbench design file
/sim/tb/rc_test.sv	Testbench design file
/sim/tb/efx_apb_slave_tb.sv	Testbench design file
/sim/tb/ep_pcr_write_pattern_merged.v	Efinity generated
/sim/tb/rc_pcr_write_pattern_merged.v	Efinity generated
/rtl	Design RTL directory

The simulation test flows in the order as outlined below.

Figure 8: Test Flowchart



Revision History

Table 51: Revision History

Date	Version	Description
May 2025	1.1	Added **Simulation Environment Testbench** on page 43.
May 2025	1.0	Initial release.