DMA Controller Core User Guide

UG-CORE-DMA-v2.0
January 2024
www.efinixinc.com

Contents

Introduction

The direct memory access (DMA) controller allows a hardware subsystem to access the main memory with minimal monitoring from the central processing unit (CPU). This reduces CPU workload when the system is required to transfer large amount of memory multiple times. The DMA Controller core provides a mechanism to transfer data between memory and AXI4-stream interface using direct mode or scatter-gather mode.

Use the IP Manager to select IP, customize it, and generate files. The DMA Controller core has an interactive wizard to help you set parameters. The wizard also has options to create a testbench and/or example design targeting an Efinix® development board.

Important: Open-source Java 64-bit runtime environment is required for generating the DMA Controller core in the IP Manager. You can get the installer and instructions from:
- https://www.java.com/en/download/manual.jsp (Java 8)
- https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)
- http://jdk.java.net/16/ (OpenJDK 16)

Features

- Memory interface in AXI4 full-duplex or AXI3 half-duplex
- Programmable memory interface width
- Enhance bandwidth with hardware read and write queue
- Priority-weighted round robin scheduling
- Control and status register with APB3 interface
- Up to 8 channels of AXI4-stream interface
- Programmable stream interface width
- Supports direct and scatter-gather (SG) modes
- Includes example design targeting the Trion® T120 BGA576 Development Board

Device Support

Table 1: DMA Controller Core Device Support

FPGA Family	Supported Device
Trion	All
Titanium	All

Resource Utilization and Performance

Note: The resources and performance values provided are based on some of the supported FPGAs. These values are just guidance and can change depending on the device resource utilization, design congestion, and user design.

Table 2: Titanium Resource Utilization and Performance

FPGA	Logic and Adders	Flip-flops	Memory Blocks	DSP48 Blocks	f_{MAX} (MHz)[1]	Efinity® Version[2]
Ti60 F225 C4	2,628	2,307	10	0	223	2021.2

Table 3: Trion® Resource Utilization and Performance

FPGA	Logic Utilization (LUTs)	Registers	Memory Blocks	Multipliers	f_{MAX} (MHz)[1]	Efinity® Version[2]
T120 BGA576 C4	3,038	2,424	14	0	108	2021.2

Release Notes

You can refer to the IP Core Release Notes for more information about the IP core changes. The IP Core Release Notes is available in the Efinity Downloads page under each Efinity software release version.

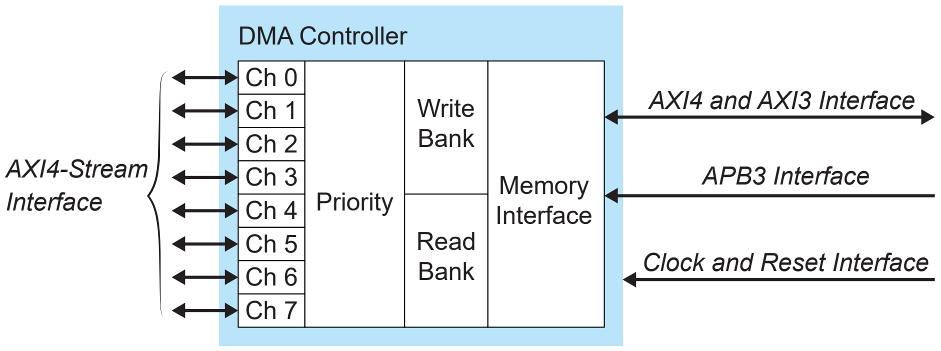
Note: You must be logged in to the Support Portal to view the IP Core Release Notes.

[1] Using default parameter settings.
[2] Using Verilog HDL.

Functional Description

The direct memory access memory controller has a two central buffer banks that stores incoming data and outgoing data from each channel. The controller's memory interface width can be down-sized internally to improve f_{MAX} and reduce required area. The priority scheduling scheme in the DMA Controller core manages the AXI4-stream interface. It determines which interface receives higher priority to serve and allows the stream's data to go to memory interface.

Figure 1: DMA Controller Core Block Diagram



Ports

Table 4: Clock and Reset Interface

n = Channel number.

Port	Direction	Description
clk	Input	DMA core operating clock.
reset	Input	Active high DMA core asynchronous reset.
ctrl_clk	Input	APB3 clock when APB3 interface operates in asynchronous mode.
ctrl_reset	Input	Active high APB3 reset when APB3 interface operates in asynchronous mode.
dat[*n*]_i_clk	Input	AXI4-stream input port clock when interface operates in asynchronous mode.
dat[*n*]_i_reset	Input	Active high AXI4-stream input port reset when interface operates in asynchronous mode.
dat[*n*]_o_clk	Input	AXI4-stream output port clock when interface operates in asynchronous mode.
dat[*n*]_o_reset	Input	Active high AXI4-stream output port reset when interface operates in asynchronous mode.

Table 5: APB3 Interface

The APB interface is synchronous by default. Set the IP manager parameter to change to asynchronous.

Port	Direction	Description
ctrl_PADDR [13:0]	Input	APB3 Address.
ctrl_PENABLE	Input	Enable port. This signal indicates the second and subsequent cycles of APB transfers.
ctrl_PSEL	Input	Select port. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required.
ctrl_PWRITE	Input	Operation port. 0: APB read access 1: APB write access
ctrl_PWDATA [31:0]	Input	Write Data. This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is HIGH.
ctrl_PREADY	Output	Ready port. The slave uses this signal to extend an APB transfer.
ctrl_PRDATA [31:0]	Output	Read Data port. The selected slave drives this bus during read cycles when PWRITE is low.
ctrl_PSLVERROR	Output	This signal indicates a transfer failure but it is unused in DMA core.

Table 6: AXI4-Stream Interface

n = Channel number.

Port	Direction	Description
dat[n]_i_tvalid	Input	Indicates the master is driving a valid transfer. A transfer takes place when both dat0_i_tvalid and dat0_i_tready are asserted.
dat[n]_i_tready	Output	Indicates that slave port can accept a transfer.
dat[n]_i_tdata [m-1:0]	Input	Payload from master port. m = Memory Interface Width
dat[n]_i_tkeep [m-1:0]	Input	Byte qualifier that indicates whether content of the associated byte of tdata is processed as part of the data stream. m = Memory Interface Width
dat[n]_i_tdest [3:0]	Input	Routing information for the data stream.
dat[n]_i_tlast	Input	Indicates the boundary of a packet.
dat[n]_o_tvalid	Output	Indicates the master is driving a valid transfer. A transfer takes place when both tvalid and dat0_i_tready are asserted.
dat[n]_o_tready	Input	Indicates that a slave port can accept a transfer.
dat[n]_o_tdata [m-1:0]	Output	Payload to slave port. m = Memory Interface Width
dat[n]_o_tkeep [m-1:0]	Output	Byte qualifier that indicates whether content of the associated byte of data is processed as part of the data stream. m = Memory Interface Width
dat[n]_o_tdest [3:0]	Output	Provides routing information for the data stream.
dat[n]_o_tlast	Output	Indicates the boundary of a packet.

Table 7: AXI4 Interface

Ports	Direction	Description
write_awvalid	Input	Indicates that the channel is signalling a valid write address and control information.
write_awready	Output	Indicates that the controller is ready to accept an address and associated control signals.
write_awaddr [31:0]	Input	The write address gives the address of the first transfer in a write/read burst transaction.
write_awlen [7:0]	Input	Burst length. Indicates the exact number of transfers in a burst. Determines the number of data transfers associated with the address. Effective burst length = io_arw_payload_len + 1
write_awsize [2:0]	Input	Burst size. Indicates the size of each transfer in the burst. 3b000: 1 bytes 3b001: 2 bytes 3b010: 4 bytes 3b011: 8 bytes 3b100: 16 bytes 3b101: 32 bytes 3b110: 64 bytes 3b111: 128 bytes
write_awburst [1:0]	Input	Burst type. The burst type and the size information, determines how the address for each transfer within the burst is calculated. 2b00: fixed burst 2b01: linear burst 2b10: wrap burst
write_awlock	Input	Reserved.
write_awcache [3:0]	Input	Memory type. This signal indicates how transactions are required to progress through a system.
write_awprot [2:0]	Input	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access.
write_awqos [3:0]	Input	The quality-of-service identifier.
write_region [3:0]	Input	Permits a single physical interface on a slave to be used for multiple logical interfaces.
write_wvalid	Input	Write valid. Indicates that valid write data and strobes are available.
write_wready	Output	Write ready. Indicates that the slave can accept the write data.
write_wdata [n-1:0]	Input	Write data. n = Memory Interface Width
write_wstrb [n-1:0]	Input	Write strobes. Indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus. n = Memory Interface Width /8
write_wlast	Input	Write last. Indicates the last transfer in a write burst.
write_bvalid	Input	Write response valid. Indicates that the channel is signalling a valid write response.
write_bready	Output	Response ready. Indicates that the master can accept a write response.

Ports	Direction	Description
read_arvalid	Input	Indicates that the channel is signalling a valid read address and control information.
read_arready	Output	Indicates that the controller is ready to accept an address and associated control signals.
read_araddr [31:0]	Input	The read address gives the address of the first transfer in a write/read burst transaction.
read_arlen [7:0]	Input	Burst length. Indicates the exact number of transfers in a burst. Determines the number of data transfers associated with the address. Effective burst length = io_arw_payload_len + 1
read_arsize [2:0]	Input	Burst size. Indicates the size of each transfer in the burst. 3b000: 1 bytes 3b001: 2 bytes 3b010: 4 bytes 3b011: 8 bytes 3b100: 16 bytes 3b101: 32 bytes 3b110: 64 bytes 3b111: 128 bytes
read_arburst [1:0]	Input	Burst type. The burst type and the size information, determines how the address for each transfer within the burst is calculated. 2b00: fixed burst 2b01: linear burst 2b10: wrap burst
read_arlock	Input	Reserved.
read_awcache [3:0]	Input	Memory type. This signal indicates how transactions are required to progress through a system.
read_awprot [2:0]	Input	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access.
read_awqos [3:0]	Input	The quality-of-service identifier.
read_region [3:0]	Input	Permits a single physical interface on a slave to be used for multiple logical interfaces.
read_rvalid	Output	Read address valid. Indicates that the channel is signalling valid read address and control information.
read_rready	Input	Read ready. Indicates that the master can accept the read data and response information.
read_rdata [n-1:0]	Output	Read data. n = Memory Interface Width
read_rresp [1:0]	Output	Read response. Indicates the status of the read transfer. This controller only responds 2b00 or OKAY.
read_rlast	Output	Read last. Indicates the last transfer in a read burst.

Table 8: AXI3 Interface

Port	Direction	Description
axi_arwvalid	Input	Indicates that the channel is signalling a valid write/read address and control information.
axi_arwready	Output	Indicates that the controller is ready to accept an address and associated control signals.
axi_arwaddr [31:0]	Input	The write address gives the address of the first transfer in a write/read burst transaction.
axi_arwlen [7:0]	Input	Burst length. Indicates the exact number of transfers in a burst. Determines the number of data transfers associated with the address. Effective burst length = io_arw_payload_len + 1
axi_arwsize [2:0]	Input	Burst size. Indicates the size of each transfer in the burst. 3b000: 1 bytes 3b001: 2 bytes 3b010: 4 bytes 3b011: 8 bytes 3b100: 16 bytes 3b101: 32 bytes 3b110: 64 bytes 3b111: 128 bytes
axi_arwburst [1:0]	Input	Burst type. The burst type and the size information, determines how the address for each transfer within the burst is calculated. 2b00: Fixed burst 2b01: Linear burst 2b10: Wrap burst
axi_arwlock	Input	Reserved.
axi_arwcache [3:0]	Input	Memory type. This signal indicates how transactions are required to progress through a system.
axi_arwprot [2:0]	Input	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access.
axi_arwqos [3:0]	Input	The quality-of-service identifier.
axi_arwregion [3:0]	Input	Permits a single physical interface on a slave to be used for multiple logical interfaces.
axi_arwwrite	Input	Indicates the channel is accepting a write or read transfer. 0: Read 1: Write
axi_wvalid	Input	Write valid. Indicates that valid write data and strobes are available.
axi_wready	Output	Write ready. Indicates that the slave can accept the write data.
axi_wdata [n-1:0]	Input	Write data. n = Memory Interface Width
axi_wstrb [n-1:0]	Input	Write strobes. Indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus. n = Memory Interface Width /8
axi_wlast	Input	Write last. Indicates the last transfer in a write burst.

Port	Direction	Description
axi_bvalid	Input	Write response valid. Indicates that the channel is signalling a valid write response.
axi_bready	Output	Response ready. Indicates that the master can accept a write response.
axi_rvalid	Output	Read address valid. Indicates that the channel is signalling valid read address and control information.
axi_rready	Input	Read ready. Indicates that the master can accept the read data and response information.
axi_rdata	Output	Read data. n = Memory Interface Width
axi_rresp [1:0]	Output	Read response. Indicates the status of the read transfer. This controller only responds 2b00 or OKAY.
axi_rlast	Output	Read last. Indicates the last transfer in a read burst

Table 9: Conduit Interface

Port	Direction	Description
io_[*n*]_descriptorUpdate	Output	Outputs a pulse to indicate that the descriptor transfer is complete. This port is enabled only when parameter SG mode is enabled. *n* is the channel number.

DMA Operation

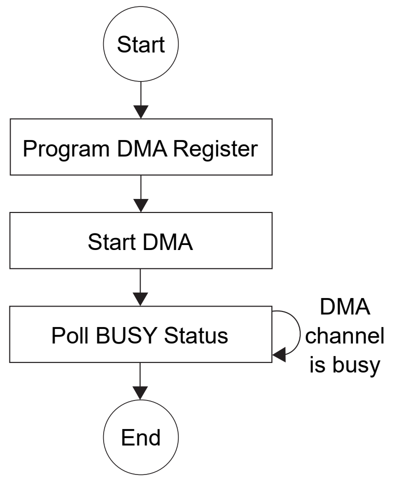
The DMA Controller core operates in two modes, which are direct mode and scatter-gather mode. You can choose the operation mode based on your application scenario. The scatter-gather mode is disabled by default. You need to enable it before using this mode.

Transfer information like source address, destination address, channel selection, port selection, and byte of transfer are programmed through the APB3 interface. Status information, such as the BUSY state or progress of a transfer, can be retrieved through the same interface. Although each channel can have an input port or an output port, both ports cannot work in parallel due to the channel operation only allows it either in a read or a write operation. When programmed as a read operation, the DMA controller retrieves data from the main memory and outputs to AXI4-stream output interface. The AXI4-stream input interface is idle during the read operation.

Direct Mode

You can transfer a chunk of big-sized data by using the direct mode. This mode requires you to program the DMA Controller core by providing the source or destination address, number of bytes to transfer over, which channel you would like to use, and which AXI4-stream ports you would like to use. After that, start the DMA Controller core to perform the transfer.

Figure 2: Direct Mode Flow Diagram

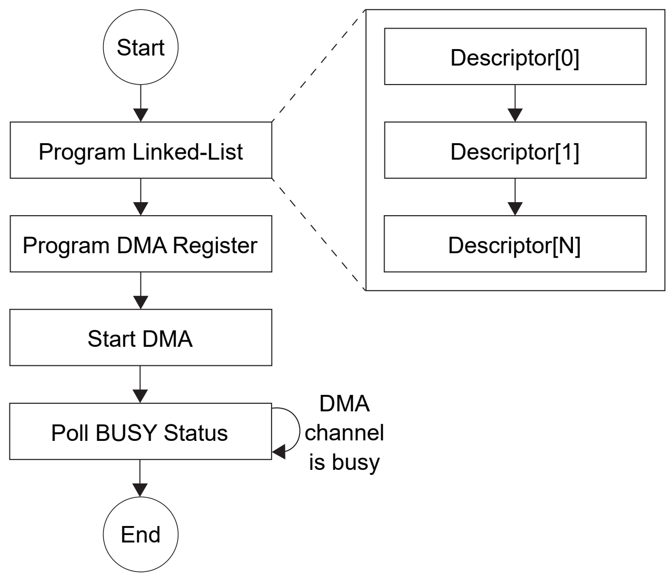


You can continuously poll the channel status register to see if the channel is still performing the transfer. The `BUSY` status deassert from high to low when the DMA Controller core is done transferring the data.

Scatter-Gather

The scatter-gather is used when an application is required to do multiple transfers in one-time programming. Before starting the DMA Controller core, you must prepare a descriptor. A descriptor contains information such as source or destination address, number of bytes to transfer over, which channel you would like to use, and which AXI4-stream port you would like to use. Multiple descriptors linked together and formed a linked-list. Each descriptor is required to mention the address of the next descriptor so that the DMA Controller core can look for the next descriptor when the current descriptor is served. The linked-list is usually stored in the main memory.

Figure 3: Scatter-Gather Mode Flow Diagram



Unlike direct mode transfer, the DMA Controller core is out of the BUSY state once it runs through the linked-list and complete all transfer of the descriptors.

Descriptor Registers

Table 10: Descriptor Register Definition

All descriptor registers are 32-bit write operation and any unlisted bits are reserved bits.

Address Offset	Bit	Register Description
0x00	Status	
	31	DMASG_DESCRIPTOR_STATUS_COMPLETED Indicates that the descriptor as complete and the controller stops linked-list execution if this flag is set.
	30	For AXI4-Stream to memory transfers, this bit indicates if the descriptor marks the end of a received packet. Can be used when the bit 13 of DMASG_CHANNEL_INPUT_CONFIG_STREAM is set.
	[26:0]	DMASG_DESCRIPTOR_STATUS_BYTES
0x01	Control	
	30	For memory to AXI4-Stream transfers, indicates if an end-of-packet should be sent at the end of the transfer.
	[25:0]	DMASG_DESCRIPTOR_CONTROL_BYTES Number of bytes (minus one) reserved at the descriptor FROM/TO addresses. For example, if you want to transfer 10 bytes, this field should be set to 9.
0x02	From (LSB)	
	[31:0]	For memory to either memory or AXI4-Stream transfers, this is the memory address of the input data.
0x03	From (MSB)	
	[63:32]	For memory to either memory or AXI4-Stream transfers, this is the memory address of the input data.
0x04	To (LSB)	
	[31:0]	For memory or AXI4-Stream to memory transfers, this is the memory address of the output data.
0x05	To (MSB)	
	[63:32]	For memory or AXI4-Stream to memory transfers, this is the memory address of the output data.
0x06	Next (LSB)	
	[31:0]	Memory address of the next descriptor for execution.
0x07	Next (MSB)	
	[63:32]	Memory address of the next descriptor for execution.

Descriptor Examples

The following code shows the descriptor structure example in C language.

```
struct dmasg_descriptor {
// See all DMASG_DESCRIPTOR_STATUS_* defines
// Updated by the DMA at the end of each descriptor and when a S -> M packet is completely
 transferred into memory
u32 status;
// See all DMASG_DESCRIPTOR_CONTROL_* defines
u32 control;
// For M -> ? transfers, memory address of the input data
u64 from;
// For ? -> M transfers, memory address of the output data
u64 to;
// Memory address of the next descriptor
u64 next;
}
```

You need to define the descriptor to either write to memory or read from memory. Depending on whether it is a read or write operation:

- Memory to AXI4-Stream—The `from` register stores the target address of the memory, and the `to` should be `0`.
- AXI4-Stream to Memory—The `to` register stores the target address of the memory, and the `from` should be `0`.

The `control` register stores the number of bytes to be accessed.

The following descriptor examples show the linked-list operations:

Note: The `dmasg_input_memory`, `dmasg_output_memory`, `dmasg_input_stream`, are `dmasg_linked_list_start` functions that are defined in the driver (**dmasg.h**) file of the SoC SDK.

Memory to AXI4-Stream Descriptor Example

```
volatile struct dmasg_descriptor descriptors1[FRAME_RATE+1]  __attribute__ ((aligned (64)));
   for (int j=0; j<FRAME_RATE+1; j=j+1 ) {
      if(j == FRAME_RATE){
         descriptors1[j].status  = DMASG_DESCRIPTOR_STATUS_COMPLETED;
      } else {
         descriptors1[j].control = (u32)((BUFFER_SIZE)-1)  | 1 << 30;;
         descriptors1[j].from    = (u32)(sb32 + (j *(BUFFER_SIZE)));
         descriptors1[j].to      = 0;
         descriptors1[j].next    = (u32) (descriptors1 + (j+1));
         descriptors1[j].status  = 0;
         }
      }
   dmasg_input_memory (DMASG_BASE, ST32_OUT,  sb32, 16);
   dmasg_output_stream(DMASG_BASE, ST32_OUT, 0, 0, 0, 1);
   dmasg_linked_list_start(DMASG_BASE, ST32_OUT, (u32) descriptors1);
```

AXI4-Stream to Memory Descriptor Example

```
volatile struct dmasg_descriptor descriptors0[FRAME_RATE+1]  __attribute__ ((aligned (64)));
   for (int j=0; j<FRAME_RATE+1; j=j+1 ) {
      if(j == FRAME_RATE){
         descriptors0[j].status  = DMASG_DESCRIPTOR_STATUS_COMPLETED;
      } else {
         descriptors0[j].control = (u32)((BUFFER_SIZE)-1)  | 1 << 30;;
         descriptors0[j].from    = 0;
         descriptors0[j].to      = (u32)(db32 + (j *(BUFFER_SIZE)) );
         descriptors0[j].next    = (u32) (descriptors0 + (j+1));
         descriptors0[j].status  = 0;
      }
   }
   dmasg_output_memory (DMASG_BASE, ST32_IN,  db32, 16);
   dmasg_input_stream(DMASG_BASE, ST32_IN, 0, 1, 0);
   dmasg_linked_list_start(DMASG_BASE, ST32_IN, (u32) descriptors0);
```

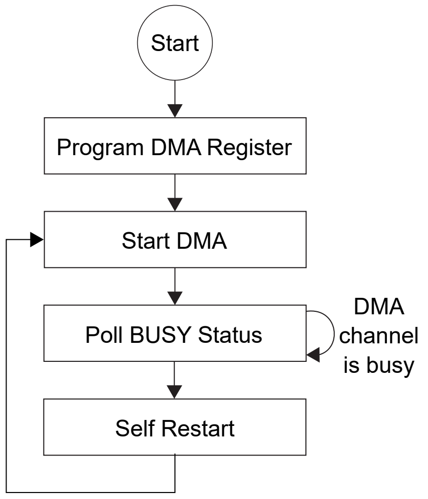
Memory to Memory Descriptor Example

```
volatile struct dmasg_descriptor descriptors2[FRAME_RATE+1]  __attribute__ ((aligned (64)));
   for (int j=0; j<FRAME_RATE+1; j=j+1 ) {
      if(j == FRAME_RATE) {
         descriptors2[j].status  = DMASG_DESCRIPTOR_STATUS_COMPLETED;
      } else {
         descriptors2[j].control = (u32)((BUFFER_SIZE)-1)  | 1 << 30;;
         descriptors2[j].from    = (u32)(sb32 + (j *(BUFFER_SIZE)));
         descriptors2[j].to      = (u32)(db32 + (j *(BUFFER_SIZE)) );
         descriptors2[j].next    = (u32) (descriptors2 + (j+1));
         descriptors2[j].status  = 0;
      }
   }
   dmasg_input_memory  ( DMASG_BASE, ST32_M2M,  sb32, 16);
   dmasg_output_memory (DMASG_BASE, ST32_M2M,  db32, 16);
   dmasg_linked_list_start(DMASG_BASE, ST32_M2M, (u32) descriptors2);
```

Channel Circular Buffer Mode

Circular buffer mode is used when an application is required to transfer exact amount of data on a fixed start address. The DMA Controller core self-restarts when a transfer has completed and start executing the transfer again with information provided from the previous transfer. This mode is applicable in direct mode transfer.

Figure 4: Channel Circular Buffer Mode Flow Diagram



Priority-weighted Round-robin Scheduler

Multiple channels compete to get access to the main memory transfer. Therefore, you can program the priority register to ensure that data stream with higher priority is served accordingly. This scheduler is disabled by default. You can enable it in the IP manager parameter to get the multiple data streams managed by the scheduler. Set the priority-weighted round-robin scheduler in the `DMASG_CHANNEL_PRIORITY` register.

The scheduler prioritizes channel with a higher priority value. Additionally, the weight value guarantees the higher priority data stream is served uninterruptedly correlated to the `byte_per_burst`. For example, if the channel's `byte_per_burst` is 256 bytes, and the programmed weight is 3, the channel can transfer 1024 ((weight + 1) * `byte_per_burst`) bytes without interruption from another channel.

Interrupt Control

Each channel is capable to turn on interrupt signal upon data transfer completion. This can further reduce CPU workload to keep monitoring on BUSY state of DMA Controller core. The interrupt signal alerts the CPU and the CPU can react accordingly.

DMA Controller Registers

Table 11: Address Mapping

All registers are 32-bit and any unlisted bits are reserved bits.

Address Offset	Bit	Register Description	Operation
0x00	DMASG_CHANNEL_INPUT_ADDRESS		
	[31:0]	Write source address.	Write
0x08	DMASG_CHANNEL_INPUT_STREAM		
	[7:0]	Write inputs ports to identify physical input port to use.	Write
0x0C	DMASG_CHANNEL_INPUT_CONFIG		
	0	Use to transfer to AXI4-stream port. The bit is always set to 0.	Write
	12	0: Transfer from AXI4-stream port 1: Transfer from memory port	
	13	Completion on packet. Limits the descriptor to only contain one packet and force its completion when it fully transferred into memory.	
	14	Wait on Packet. Ensures the channel waits for the beginning of a packet before capturing the data (avoid desync).	
0x10	DMASG_CHANNEL_OUTPUT_ADDRESS		
	[31:0]	Write the destination address.	Write
0x18	DMASG_CHANNEL_OUTPUT_STREAM		
	[7:0]	Write the outputs ports to identify the physical output port to use.	Write
	[15:8]	Write the source port ID.	
	[23:16]	Write the sink port ID.	
0x1C	DMASG_CHANNEL_OUTPUT_CONFIG		
	0	Transfer to AXI4-stream port. The bit is always set to 0.	Write
	12	0: Transfer to AXI4-stream port 1: transfer to memory port	
	13	Last bit. Specify end of packet to be sent at the end of a transfer.	
0x20	DMASG_CHANNEL_DIRECT_BYTES		
	[31:0]	Write the number of bytes to be transferred in direct mode.	Write
0x2C	DMASG_CHANNEL_STATUS		
	0	Set to 1 to start DMA in direct mode (also as a busy bit).	Write
	1	Set to 1 to start self-restart.	

Address Offset	Bit	Register Description	Operation
	2	Set to 1 to set a channel to stop itself.	
	4	Set to 1 to start a channel using a linked-list. Ensure linked-list support is enabled by the channel.	
	0	1: Channel busy 0: Channel ready	Read
0x44	DMASG_CHANNEL_PRIORITY		
	[2:0]	Set the priority of channel 0 to 7 (7 has the highest priority).	Write
	[10:8]	Set the weight of channel 0 to 7 (7 has the highest priority).	
0x50	DMASG_CHANNEL_INTERRUPT_ENABLE		
	0	Enable interrupt at the end of each descriptor.	Write
	2	Enable interrupt when the channel is out of busy.	
	3	Enable interrupt when the status of descriptor is updated.	
	4	Enable interrupt when the channel done transferring a packet (AXI4-stream to memory).	
0x54	DMASG_CHANNEL_INTERRUPT_PENDING		
	0	Interrupt mask status at the end of each descriptor.	Read
	2	Interrupt mask status when the channel is out of busy.	
	3	Interrupt mask status when the status of descriptor is updated.	
	4	Interrupt mask status when done transferring a packet (AXI4-stream to memory)	
	0	Clear interrupt mask at the end of each descriptor.	Write
	2	Clear interrupt mask when the channel is out of busy.	
	3	Clear interrupt mask when the status of descriptor is updated.	
	4	Clear interrupt mask when the channel done transferring a packet (AXI4-stream to memory).	
0x60	DMASG_CHANNEL_PROGRESS_BYTES		
	[31:0]	Read the numbers of bytes transferred for the current descriptor. This register monitors transfer progress in SG mode only. Refer to DMASG_CHANNEL_STATUS (bit 0) for Direct mode.	Read
0x70	DMASG_CHANNEL_LINKED_LIST_HEAD		
	[31:0]	Write the address of first descriptor in a linked list.	Write

Table 12: Channel base Address

Channel	Offset
0	0x000
1	0x080
2	0x100
3	0x180
4	0x200
5	0x280
6	0x300
7	0x380

Direct Mode Register Control Examples

Figure 5: Single-Channel in Direct Mode

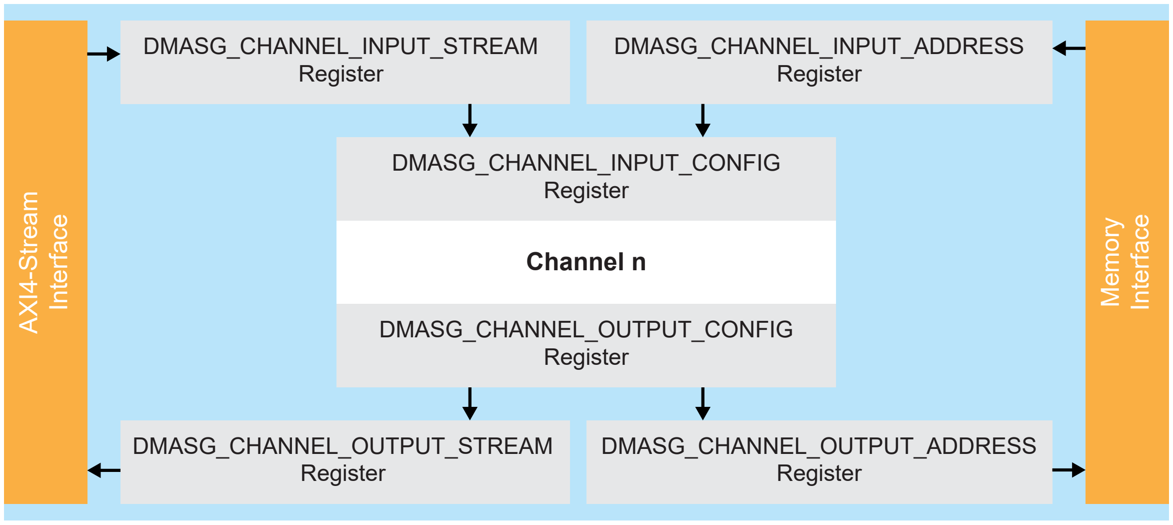

AXI4-Stream Interface
DMASG_CHANNEL_INPUT_STREAM Register
DMASG_CHANNEL_INPUT_ADDRESS Register
DMASG_CHANNEL_INPUT_CONFIG Register
Channel n
DMASG_CHANNEL_OUTPUT_CONFIG Register
DMASG_CHANNEL_OUTPUT_STREAM Register
DMASG_CHANNEL_OUTPUT_ADDRESS Register
Memory Interface

Table 13: Single-Channel Memory to AXI4-Stream Port in Direct Mode

Step	Description	Related Register
1	Set the channel input source from memory port.	DMASG_CHANNEL_INPUT_CONFIG Set bit 12 to 1.
2	Set the start memory address to read.	DMASG_CHANNEL_INPUT_ADDRESS
3	Set the output of channel to AXI4-Stream output port.	DMASG_CHANNEL_OUTPUT_CONFIG Set bit 12 to 0.
4	Configure the AXI4-Stream output port.	DMASG_CHANNEL_OUTPUT_STREAM For a single-channel controller, port ID is 0.
5	Set the number of bytes to transfer.	DMASG_CHANNEL_DIRECT_BYTES
6	Start the DMA in direct mode.	DMASG_CHANNEL_STATUS Set bit 0 to 1. If you use circular buffer mode, set bit 1 to 1.

Table 14: Single-Channel AXI4-Stream to Memory Port in Direct Mode

Step	Description	Related Register
1	Set the input of channel from AXI4-Stream port.	DMASG_CHANNEL_INPUT_CONFIG Set bit 12 to 0.
2	Configure the AXI4-Stream input port.	DMASG_CHANNEL_INPUT_STREAM For a single-channel controller, port ID is 0.
3	Set the output of channel to memory port.	DMASG_CHANNEL_OUTPUT_STREAM Set bit 12 to 1.
4	Set the start of memory address to write to.	DMASG_CHANNEL_OUTPUT_ADDRESS
5	Set the number of bytes to transfer.	DMASG_CHANNEL_DIRECT_BYTES
6	Start the DMA in direct mode.	DMASG_CHANNEL_STATUS Set bit 0 to 1. If you use circular buffer mode, set bit 1 to 1.

Table 15: Single-Channel Memory to Memory Port in Direct Mode

Step	Description	Related Register
1	Set the channel input source from memory port.	DMASG_CHANNEL_INPUT_CONFIG Set bit 12 to 1.
2	Set the start memory address to read.	DMASG_CHANNEL_INPUT_ADDRESS
3	Set the output of channel to memory port.	DMASG_CHANNEL_OUTPUT_CONFIG Set bit 12 to 1.
4	Set the start of memory address to write to.	DMASG_CHANNEL_OUTPUT_ADDRESS
5	Set the number of bytes to transfer.	DMASG_CHANNEL_DIRECT_BYTES
6	Start the DMA in direct mode.	DMASG_CHANNEL_STATUS Set bit 0 to 1. If you use circular buffer mode, set bit 1 to 1.

SG Mode Register Control Examples

Figure 6: Single-Channel in SG Mode

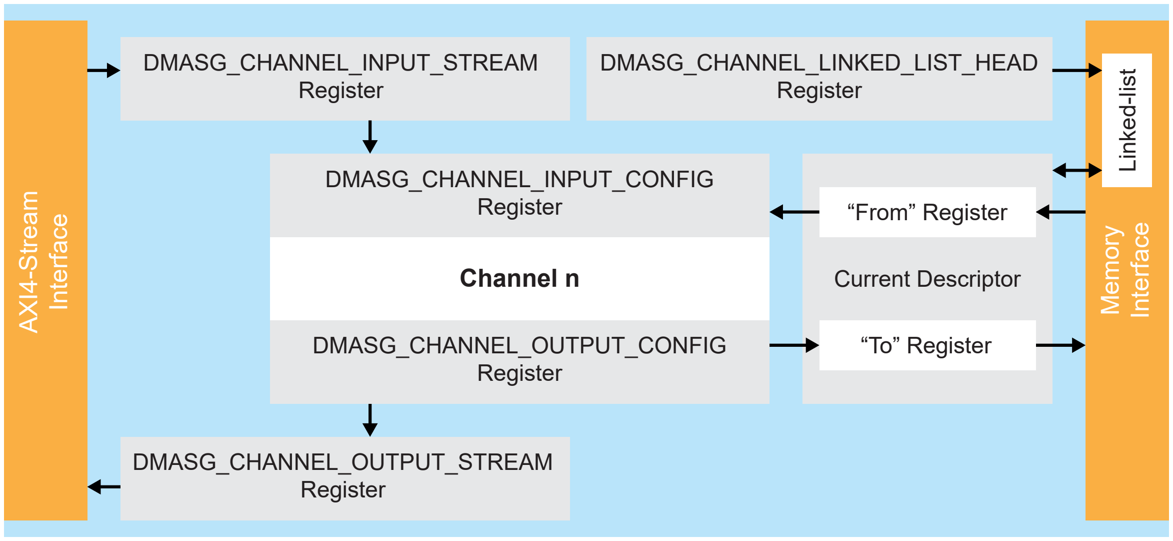

AXI4-Stream Interface
DMASG_CHANNEL_INPUT_STREAM Register
DMASG_CHANNEL_LINKED_LIST_HEAD Register
Linked-list
DMASG_CHANNEL_INPUT_CONFIG Register
"From" Register
Channel n
Current Descriptor
DMASG_CHANNEL_OUTPUT_CONFIG Register
"To" Register
Memory Interface
DMASG_CHANNEL_OUTPUT_STREAM Register

Table 16: Single-Channel Memory to AXI4-Stream Port in SG Mode

Step	Description	Related Register
1	Define the linked-list of descriptors in the memory.	See **Memory to AXI4-Stream Descriptor Example** on page 14.
2	Set the channel input source from memory port.	DMASG_CHANNEL_INPUT_CONFIG Set bit 12 to 1.
3	Set the output of channel to AXI4-Stream port.	DMASG_CHANNEL_OUTPUT_CONFIG Set bit 12 to 0.
4	Configure the AXI4-Stream output port.	DMASG_CHANNEL_OUTPUT_STREAM For a single-channel controller, port ID is 0.
5	Set the start address of the first descriptor.	DMASG_CHANNEL_LINKED_LIST_HEAD
6	Start the DMA in SG mode.	DMASG_CHANNEL_STATUS Set bit 4 to 1.

Table 17: Single-Channel AXI4-Stream to Memory Port in SG Mode

Step	Description	Related Register
1	Define the linked-list of descriptors in the memory.	See **AXI4-Stream to Memory Descriptor Example** on page 14.
2	Set the input of channel from AXI4-Stream port.	DMAG_CHANNEL_INPUT_CONFIG set bit 12 to 0
3	Configure the AXI4-Stream input port.	DMASG_CHANNEL_INPUT_STREAM For a single-channel controller, port ID is 0.
4	Set the output of channel to memory port.	DMASG_CHANNEL_OUTPUT_CONFIG Set bit 12 to 1.
5	Set the start address of the first descriptor	DMASG_CHANNEL_LINKED_LIST_HEAD
6	Start the DMA in SG mode.	DMASG_CHANNEL_STATUS Set bit 4 to 1.

Table 18: Single-Channel Memory to Memory Port in SG Mode

Step	Description	Related Register
1	Define the linked-list of descriptors in the memory.	See **Memory to Memory Descriptor Example** on page 15.
2	Set the channel input source from memory port.	DMASG_CHANNEL_INPUT_CONFIG Set bit 12 to 1.
3	Set the output of channel to memory port.	DMASG_CHANNEL_OUTPUT_CONFIG Set bit 12 to 1
4	Set the start address of the first descriptor	DMASG_CHANNEL_LINKED_LIST_HEAD
5	Start the DMA in SG mode.	DMASG_CHANNEL_STATUS Set bit 4 to 1.

IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinix® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinix development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.

Note: Not all Efinix IP cores include an example design or a testbench.

Generating the DMA Controller Core with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose **Bridge and Adaptors** > **DMA Controller** core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.

   Note: You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the *Customizing the DMA Controller* section.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinix® development board and/or testbench. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.

   Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>_tmpl.v**—Verilog HDL instantiation template.
- **<module name>_tmpl.vhd**—VHDL instantiation template.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains generated RTL and testbench files.

Customizing the DMA Controller

The core has parameters so you can customize its function. You set the parameters in the General tab of the core's IP Configuration window.

Table 19: DMA Controller Core Parameters (General Tab)

Parameter	Options	Description
Efinix AXI3 Interface Wrapper	Disable, Enable	Disable: Use AXI-4 full duplex memory interface. (Default) Enable: Use AXI-3 half duplex memory interface.
Memory Interface External Width	8, 16, 32, 64, 128, 256, 512	Memory interface width that connects externally. Default: 128
Buffer Bank Words	128, 256, 512, 1024, 2048, 4096	Buffer depth. Default: 1024
Buffer Bank Width	32, 64, 128	Buffer data width in bits. Default: 32
Memory Write Queue	Enable, Disable	Enable hardware queue to improve overall write throughput. Default: Disable
Memory Read Queue	Enable, Disable	Enable hardware queue to improve overall read throughput. Default: Disable
APB3 Interface in Asynchronous	Enable, Disable	Enabling asynchronous makes the APB3 signals and interrupt to be on a separate clock domain. Default: Disable
Priority-Weighted Round Robin Scheduler	Enable, Disable	Enable priority-weighted, round-robin scheduler. Default: Disable

Table 20: DMA Controller Core Parameters (Channel Tab)

Parameter	Options	Description
Channel *n* Enable	Enable, Disable	Instantiate channel on the DMA controller. Default: Disable for all except Channel 0
Channel *n* Asynchronous Mode	Enable, Disable	Run channel with asynchronous clock. Default: Disable
Channel *n* SG Mode	Enable, Disable	Enable scatter-gather mode. Default: Disable
Channel *n* Output Port	Enable, Disable	Enable AXI4-stream input interface. Default: Enable
Channel *n* Input Port	Enable, Disable	Enable AXI4-stream output interface. Default: Enable
Channel *n* Data width	8, 16, 32, 64, 128, 256, 512	AXI4-stream interface width. Default: 32
Channel *n* Buffer Size	128, 256, 512, 1024, 2048, 4096, 8192	Set the buffer size for channel 0. Any other enabled channel follows channel 0 buffer size. See Buffer Size Settings on page 24 for more information. Default: 1024
Channel *n* Max Burst Size	16, 32, 64, 128, 256, 512, 1024	Maximum number of bytes within a burst transfer. Default: 64
Channel *n* Circular Buffer Mode	Enable, Disable	Enable circular buffer/self-restart mode. Default: Disable

Buffer Size Settings

You need to set the **Channel Buffer Size** corresponding to the Buffer Bank Size. The equations are as follows:

Total Channel Buffer Size = Channels Buffer Size x No. of enabled channel
Buffer Bank Size = Buffer Bank Words x Buffer Bank Width (in byte) x Bank Count

Total Channel Buffer Size ≤ Buffer Bank Size

Note:
- Total Channel Buffer Size = Total Buffer Bank Size is recommended for resource optimization. However, the Total Channel Buffer Size can be lower than the Total Buffer Bank Size.
- Bank Count is fixed to 2.

Example:

- Buffer Bank Words = 128
- Buffer Bank Width = 128 bit (16 byte)

Buffer Bank Size = 128 x 16 x 2 = 4096 bytes

Depending on the number of enabled channels, you can set the **Channels Buffer Size** to:
- One channel: Channels Buffer Size ≤ 4096
- Two channels: Channels Buffer Size ≤ 2048
- Three channels: Channels Buffer Size ≤ 1024

DMA Controller Example Design

You can choose to generate the example design when generating the core in the IP Manager Configuration window. Compile the example design project and download the **.hex** or **.bit** file to your board.

Note: The Efinity IP Manager will prompt a message asking you to update the Sapphire SoC. For this example design, do not update the Sapphire SoC.

The example design uses a fixed set of parameter settings as shown in the following table.

Table 21: Example Design Parameter Settings

IP Manager Parameter	Setting
Efinix AXI3 Interface Wrapper	Disable
Memory Interface External Width	64
Buffer Bank Words	512
Buffer Bank Width	32
Memory Write Queue	Disable
Memory Read Queue	Disable
APB3 Interface in Asynchronous	Disable
Priority-Weighted Round Robin Scheduler	Disable
Channel 0 and 1 Enable	Enable
Channel 0 and 1 Asynchronous Mode	Enable
Channel 0 and 1 SG Mode	Enable
Channel 0 and 1 Output Port	Channel 0: Enable Channel 1: Disable
Channel 0 and 1 Input Port	Channel 0: Disable Channel 1: Enable
Channel 0 and 1 Data width	32
Channel 0 and 1 Buffer Size	1024
Channel 0 and 1 Max Burst Size	256
Channel 0 and 1 Circular Buffer Mode	Disable

The example designs the target the Trion® T120 BGA576 Development Board. The example design consists of a RISC-V SoC and the DMA Controller core. The instantiated DMA Controller core has two channels, channel `0` with AXI4-stream output port and channel 1 with AXI4-stream input port. Each port is connected with a loopback tester which is a FIFO buffer, to fetch the data from the output of channel `0` to the input of channel 1.

Figure 7: Example Design Block Diagram

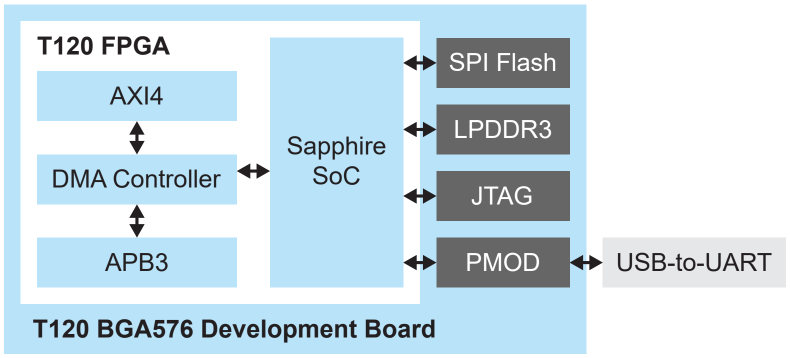


Table 22: Example Design Implementation

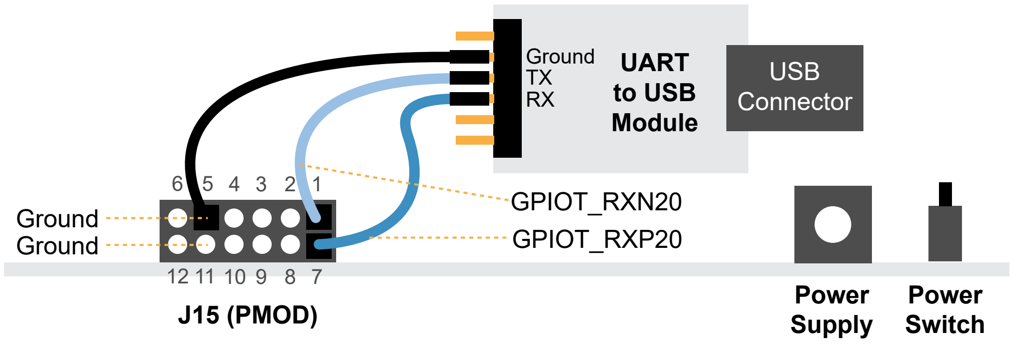
FPGA	LUTs	Registers	Memory Blocks	Multipliers	f_{MAX} (MHz)[3]		Efinity® Version[4]
					System Clock	SD Host	
T120 BGA576 C4	9,341	8,348	85	4	66	108	2021.2

(3) Using default parameter settings.
(4) Using Verilog HDL.

Set Up a USB-to-UART Module

The Trion® T120 BGA576 Development Board does not have a USB-to-UART converter, therefore, you need to use a separate USB-to-UART converter module. A number of modules are available from various vendors; any USB-to-UART module should work.

Figure 8: Connect the UART Module to PMOD Header J15



1. Connect the UART module to the PMOD port J15
   - *RX*—GPIOT_RXP20, which is pin 1 on PMOD J15
   - *TX*—GPIOT_RXN20, which is pin 7 on PMOD J15
   - *Ground*—Use ground pin 5 or 11 on PMOD J15.
2. Plug the UART module into a USB port on your computer. The driver should install automatically if needed.

Finding the COM Port (Windows)

1. Type Device Manager in the Windows search box.
2. Expand **Ports (COM & LPT)** to find out which COM port Windows assigned to the UART module; it is listed as USB Serial Port (COM*n*) where *n* is the assigned port number. Note the COM number.

Finding the COM Port (Linux)

In a terminal, type the command:

```
dmesg | grep ttyUSB
```

The terminal displays a series of messages about the attached devices.

```
usb <number>: <adapter> now attached to ttyUSB<number>
```

There are many USB-to-UART converter modules on the market. Some use an FTDI chip which displays a message similar to:

```
usb 3-3: FTDI USB Serial Device converter now attached to ttyUSB0
```

However, the Trion® T120 BGA576 Development Board also has an FTDI chip and gives the same message. So if you have both the UART module and the board attached at the same time, you may receive three messages similar to:

```
usb 3-3: FTDI USB Serial Device converter now attached to ttyUSB0
usb 3-2: FTDI USB Serial Device converter now attached to ttyUSB1
usb 3-2: FTDI USB Serial Device converter now attached to ttyUSB2
```

In this case the second 2 lines (marked by `usb 3-2`) are the development board and the first line (`usb 3-3`) is the UART module.

Using this Example with the RISC-V SDK

Before working with the software included with this example design, you should already be familiar with using the Sapphire SoC and RISC-V SDK. Specifically, you should know how to:

- Launch Efinity RISC-V Embedded Software IDE
- Import the DMA design example software project (FreeRTOS is not required)

Note: Ensure that the **dmasg** option is available in the **Import Sample Project Wizard** when importing the project.

- Build the software project.
- Launch the debug script and debug the example using the OpenOCD debugger.
- Open a UART terminal.

Learn more: Refer to the related chapter of the Sapphire RISC-V SoC Hardware and Software User Guide for detailed instructions on how to perform these tasks.

Running the Example Design

After you debug the example software, the design transfers 512 KB of data with the direct mode and followed by the scatter-gather mode.

The UART terminal prints the following message if the test is successful:

```
DMA+SOC TEST
Running DMA Direct Mode Loopback Test...
DMA Direct Mode Loopback Test Passed!
Running DMA SG Mode Loopback Test...
DMA SG Mode Loopback Test Passed!
```

DMA Controller Testbench

You can choose to generate the testbench when generating the core in the IP Manager Configuration window.

Note: You must include all **.v** files generated in the **/testbench** directory in your simulation.

Efinix provides a simulation script for you to run the testbench quickly using the Modelsim software. To run the Modelsim testbench script, run `vsim -do modelsim.do` in a terminal application. You must have Modelsim installed on your computer to use this script.

The example design includes a simulation testbench, **tb_soc.v**, which simulates the example design. After running the simulation, the test prints the following message:

```
       0 -------------------------------------------------------
       0 [EFX_INFO]: Start executing DMA TEST
       0 [EFX_INFO]: Memory controller and memory model in this simulation do not show real
                     performance or hardware and thus may impact the behaviour of DMA
       0 -------------------------------------------------------
 3577550 [EFX_INFO]: Transferring in Direct Mode detected
 3577550 [EFX_INFO]: Transferring in Direct Mode detected
10241630 [EFX_INFO]: Transferring in SG Mode detected
11241630 [EFX_INFO]: TEST PASSED!
```

Revision History

Table 23: Revision History

Date	Version	Description
January 2024	2.0	Added Buffer Size Settings section. (DOC-1570 and DOC-1629)
November 2023	1.9	Corrected DMASG_CHANNEL_INPUT_CONFIG Address Mapping. (DOC-1534)
September 2023	1.8	Updated DMASG_CHANNEL_PROGRESS_BYTES, DMASG_CHANNEL_INPUT_CONFIG and DMASG_CHANNEL_OUTPUT_CONFIG description. (DOC-1442) Updated example design configuration.
June 2023	1.7	Added Device Support and release notes sections. (DOC-1234) Added 512 support for Memory Interface External Width.
March 2023	1.6	Improved register address mapping and register control examples. (DOC-1167) Updated IP Manager parameters. Added information about setting up USB-to-UART module and RISC-V required prerequisites. Updated testbench output message. Added SG mode linked-list descriptor examples.
February 2023	1.5	Added note about the resource and performance values in the resource and utilization table are for guidance only.
December 2022	1.4	Added New in Version topic.
January 2022	1.3	Updated resource utilization table. (DOC-700)
December 2021	1.2	Added io_[*n*]_descriptorUpdate port. Updated the DMA Controller core IP Manager parameters.
October 2021	1.1	Added note about open-source Java 64-bit runtime environment is required for generating the DMA Controller core in the IP Manager. Added note to state that the f_{MAX} in Resource Utilization and Performance, and Example Design Implementation tables were based on default parameter settings. Updated design example target board to production Titanium Ti60 F225 Development Board and updated Resource Utilization and Performance, and Example Design Implementation tables. (DOC-553)
June 2021	1.0	Initial release.