ARC HS Cluster DMA Controller Option for ARC HS Processors

Highlights

- Tightly coupled to an ARC HS cluster for low latency
- Efficient DMA transfers
- Up to 16 independent programmable DMA channels
- User programmable channel prioritization
- Concurrent operation with CPUs in the cluster
- Can be used with single core or multi-core clusters
- Polling, event or interrupt based synchronization
- Supports unaligned data transfers

High-Performance DMA Controller for ARC HS Processors

The ARC® HS Cluster DMA controller option efficiently moves data around an SoC without the involvement of the CPU. The Cluster DMA is fast and optimized for DesignWare® ARC HS processors enabling fast DMA transfers with minimum gate-count and power consumption. The Cluster DMA can be used with a single HS core or with a multicore HS cluster to efficiently move data.

The cluster DMA controller uses a client-server architecture. Multiple DMA clients, one client for each core, share a single DMA server (Figure 1).

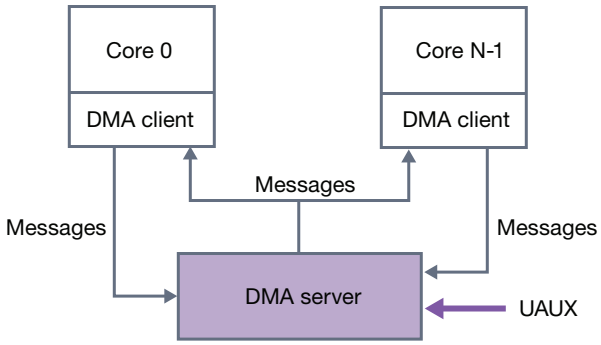

Figure 1: Cluster DMA Client Server Architecture

Communication between the client and the server for setting up descriptors and handling responses is mostly through asynchronous (non-blocking) messages, so that the core is not stalled. DMA clients and the DMA server include functionality to:

- Set up messages in DMA client auxiliary registers
- Trigger message transfer from client to server
- Receive messages from the server
- Track the status of pending messages

The DMA client inside the processor cores includes support for polling-based, interrupt-based and event based synchronization. A DMA client has a single interrupt at core level.

DMA Controller

The DMA controller uses descriptors, handles, channels, and bus transactions.

A descriptor defines an entire DMA transfer, copying data from one physical address range to another physical address range. A DMA descriptor includes at least four fields:

- The source physical byte address, from which data needs to be copied.
- The destination physical byte address, to which data needs to be copied.
- The length of the DMA transfer. The length is in the range [1B:128KB-1B] bytes and supports arbitrary byte alignments.
- Attributes of the descriptor. For example, a field to indicate if an interrupt needs to be raised when the DMA server has finished processing the descriptor.

A descriptor is assigned to a channel in the DMA server. Each channel includes a FIFO for storing the descriptors associated with that channel. A channel processes its descriptors in order. Descriptors in different channels may be processed out of order. A channel splits up large transfers into bursts for the system bus (AXI) or the internal buses. The burst length does not exceed the maximum burst length configured; these bursts are referred to as bus transactions.

A new DMA descriptor is pushed into a channel by copying the descriptor from the client into one of the DMA server channels. Upon successful pushing, the DMA server returns a handle to the DMA client, the handle being a pointer to the associated descriptor in the DMA server. The handle can be used to poll the status of the DMA transfer.

Channels with equal priority are arbitrated in a round-robin fashion. All channels have equal priorities. The DMA server back-end takes care of issuing and tracking the source bus read transactions and the associated destination bus write transactions.

A memory is used for storing all descriptors that have been pushed into a DMA channel. The memory is embedded in the DMA server; it is not accessible using processor load/store operations and is not memory mapped for the processor cores.

Major Cluster DMA Blocks

The cluster DMA consists of four major blocks:

- DMA client: logic for issuing and tracking DMA transfers for each core.
- DMA server front-end: logic shared by all cores with channel logic for each channel.
- Descriptor memory access logic: a wrapper around the single-port descriptor memory to run the memory at a fraction of the clock and arbitrate between multiple clients; logically the memory is part of the DMA frontend.
- DMA server backend: logic performing bus transactions.

If a cluster includes a DMA server, each core includes a DMA client that has local auxiliary registers and an interface for configuring the DMA server and polling status. DMA clients send atomic messages to the DMA-server channel logic. Descriptors are stored in the descriptor memory. The arbiter selects the highest-priority channel that is allowed to packetize one bus transaction. The bus-command packetizer reads the head of the channel-descriptor FIFO and generates a single bus transaction for the descriptor, updating and saving the descriptor in descriptor memory. The command-issue module allocates a region in the data buffer and allocates an entry in the pending transaction table while issuing the bus-read command. When read data is returned it is buffered and forwarded to one of the destination write interfaces. The pending transaction table tracks all pending transactions. The response handler issues interrupts and updates channel state upon completion of the DMA transfer.

DMA Source and Destination

The source and destination for a DMA transfer can be any combination of the following, as illustrated in Figure 2:

- LBU (AXI), if the cluster has a peripheral bus (LBU) interface.
- AXI I/O Coherency port: allows L1 coherent access to the Cluster Shared Memory (CSM) OBU, and CBU interfaces.
- Closely Coupled Memories (CCMs) in the cores.

The DMA IOC and LBU AXI4 interfaces support full out-of-order read data and write responses. All pending DMA read bus transactions are issued a unique ID so that the network-on-chip and DDR can reorder all transactions.

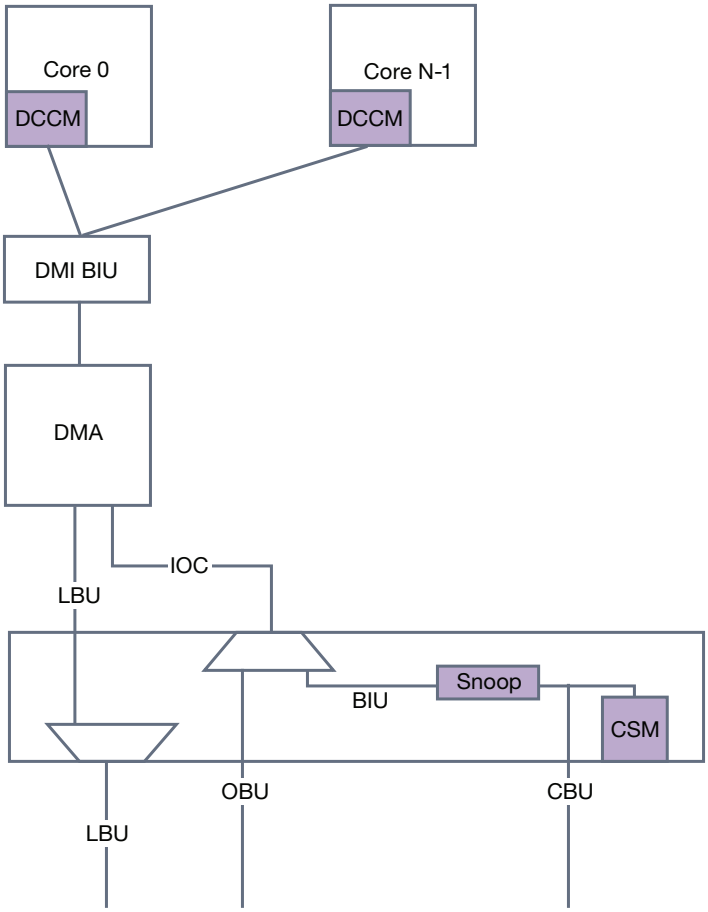

Figure 2: DMA Server Memory Interfaces

DMA Memory Map

The DMA controller memory map can be configured through cluster-level input pins such that the memory map matches the system-level memory map. All non-mapped apertures are mapped to the default IOC port of the DMA or to the LBU port, depending on a bit in the DMA server control register.

The cluster-level LBU and CBU include a round-robin arbiter to select between CPU core traffic and DMA traffic. These arbiters may need a specific policy to avoid starvation of the cores and DMA.

For cluster configurations without coherency support, the BIU connects the IOC port from the DMA to the CBU output, optionally adding an address-based splitter to access the CSM.

Configuration Support

The DMA controller is automatically reconfigured according to the system memory configuration.

If the peripheral port 0 is selected, the DMA is automatically connected to it using the DMA's dedicated LBU interface. Alternatively, if peripheral port 0 is not available, the dedicated DMA LBU interface is removed. When the DCCM DMI interfaces are configured, the dedicated DMA DMI IBP interface is connected to all applicable core interfaces. If no DMI interfaces exist, the dedicated DMA DMI IBP interface is removed.

The DMA IOC interface is present whenever there is a route out to an external memory or if the Cluster Shared Memory (CSM) is present. Accesses that do not go to the CSM, that is, external memory accesses, go to either the OBU or CBU/DBU interfaces.

The cluster DMA cannot be built without a path to an external memory.

The OBU and CBU/DBU interfaces can coexist. Alternatively, the OBU or CBU/DBU can represent one path out to an external memory. This path can be without coherency support (no SCU), or with support for both coherent and non-coherent memory transfers.

DMA Interrupt Modes

Each core has a single interrupt for the DMA client. The example issues two DMA descriptors into the same channel. The channel processes the DMA descriptors in-order, so only the completion of the last DMA raises an interrupt, which triggers a callback function.

For bare-metal applications with a fixed channel allocation more simple synchronization can be implemented, for example, based on spinlocks and sleep instructions, and avoiding mutexes and callbacks.

Unaligned Data Transfers

The Cluster DMA supports data transfer to unaligned addresses. That is, the DMA source and destination addresses can be byte-aligned and the transfer can be of any size.

Documentation

The following documentation is available for the ARC Cluster DMA Option for ARC HS processors:

- ARCv2 ISA Programmers Reference Manual
- ARC HS Databook
- DesignWare ARC EM Integration Guide

Testing, Compliance and Quality

Verification of the ARC Cluster DMA follows a bottom-up verification methodology from block-level through system-level. The module follows a functional coverage-driven test plan, which includes testing for ARCv2 ISA compliance as well as state- and control-specific coverage points that have been exercised using constrained pseudo-random environments and a random instruction sequence generator.

ARC HS Processors

The DesignWare ARC HS family of 32-bit processors is based on the scalable ARCv2 Instruction Set Architecture (ISA) and is optimized to deliver maximum performance efficiency (DMIPS/mW and DMIPS/mm2) making it ideally suited for embedded applications with high-speed data and signal processing requirements. All HS processors are available in single-, dual- and quad-core configurations.

The ARC HS3x cores includes the multicore-capable HS34, HS36 and HS38 processors. The HS34 is a high-performance cacheless processor, while the HS36 includes up to 64KB of instruction and data caches. The HS38, optimized for applications running Linux, has a full-featured memory management unit (MMU) supporting a 40-bit physical address space and page sizes up to 16 megabytes, giving designers the ability to directly address a terabyte of memory with faster data access and higher system performance.

The ARC HS4x/HS4xD cores, which includes the HS44, HS46, HS48, HS45D and HS47D processors, implements a dual-issue superscalar architecture that delivers up to 6000 DMIPS per core (16ff typical conditions). The HS46 and HS48 offer instruction and data caches (up to 64 KBs of each) and support for full Level 1 (L1) cache coherency. The HS48 also incorporates up to eight megabytes of Level 2 (L2) cache and a full-featured memory management unit (MMU) supporting symmetric multiprocessing (SMP) Linux. The HS45D and HS47D support more than 150 DSP-optimized instructions, delivering a unique combination of high-performance control and high-efficiency digital signal processing.

About MIPS:

MIPS by GlobalFoundries delivers software to silicon with RISC-V for building physical AI platforms. MIPS delivers software-hardware co-design, optimized AI, and custom ASSP design and manufacturing. Together with ARC, MIPS delivers the open, standards-based processor IP portfolio for embedded applications. Physical AI is built on MIPS.

For more information, visit **www.mips.com/arc.**