Aurora VLSI

Unleashing the Power of the Internet

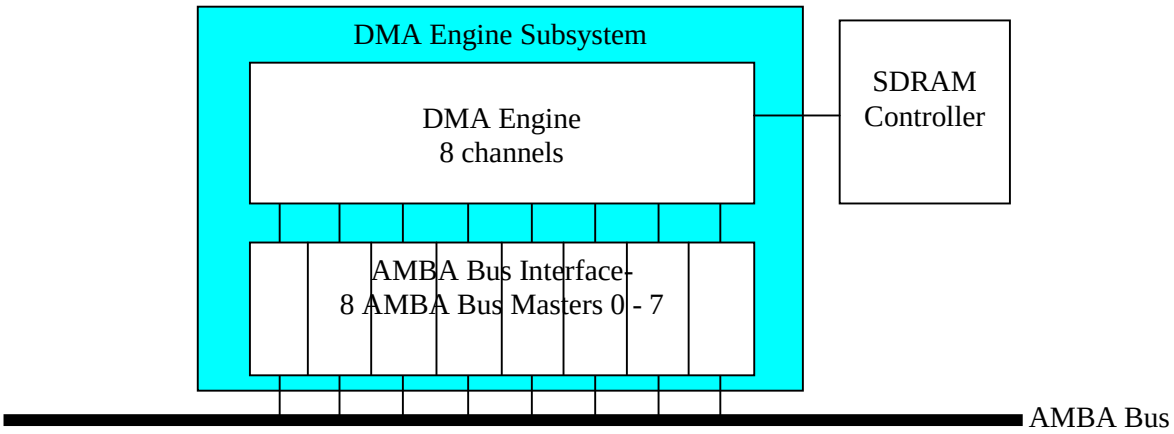
AU-SB3000: DMA Engine AMBA Subsystem Core

AMBA AHB Bus DMA Engine

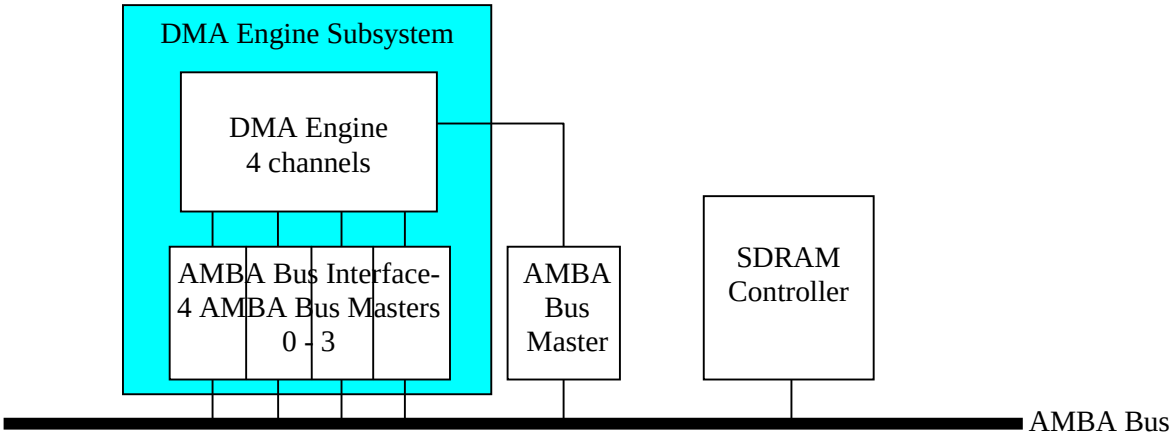
The AU-SB3000 DMA Engine AMBA Subsystem provides Direct Memory Access (DMA) functionality for AMBA based SOCs. It moves blocks of data between main memory and AMBA AHB Bus peripheral devices, and between areas in main memory. The main memory interface is generic, and will typically connect to a block such as an SDRAM controller. Up to eight independent DMA channels are supported. The DMA Engine AMBA Subsystem Core is available as a synthesizable Verilog model. Contact CustomerService@auroravlsi.com.

Each DMA channel is an AMBA Bus master. The DMA Engine Subsystem may be used in a variety of system configurations. It is extremely versatile to fit almost any AMBA based SOC that can benefit from DMA, as shown in the figures below. Some of the possible alternatives are:

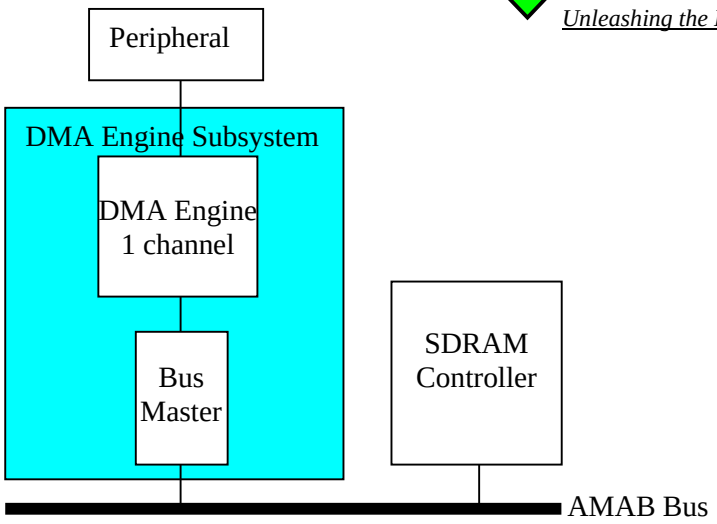
- Bus based RAM controller or bus independent RAM controller
- 1 to 8 DMA channels
- Multiple RAM requesters or RAM dedicated to the DMA Engine



Eight Channels, Isolated RAM Controller



Four Channels, Bus Based RAM Controller



Bus Based Peripheral DMA Channel

DMA Engine AMBA Subsystem features are summarized:

DMA Engine

- 1 to 8 channels- user configurable
- DMA between
  - bus devices and RAM memory
  - two memory areas in RAM memory
- Physical DMA
- Programmable source starting address
- Programmable destination starting address
- Programmable transfer count- up to 64 Kbytes
- Programmable bus interface transaction size- 8 to 32 Kbytes
- Programmable bus data transfer size- 1, 2, 4, or 8 bytes
- Memory interface transaction size optimized for RAM burst operations
- Locked DMA operation optional (software programmable)
- Direct software writes or information extracted from descriptors in memory, to program DMA control information
- Scatter/gather DMA using a chained descriptor list in memory as the DMA control information source
- Host processor initiates the DMA operation
- Interrupts signal the end of DMA operations
- Several error types end DMA operations, are recognized and logged
- Dedicated AMBA Bus master interface for each DMA channel
- Shared memory interface
- Round robin arbitration for the shared memory interface
- Bus slave device can optionally determine transfer count and start of the DMA operation
- Request/acknowledge handshake with bus slaves for most efficient bus usage

AMBA Bus Interface

- 32 bit or 64 bit AMBA AHB Bus- user configurable
- Fully pipelined for highest throughput
- Supports all required AMBA AHB Bus features
- AMBA Bus read error returned to the user with the read data

The core is delivered as a synthesizeable RTL Verilog model. Deliverables include:

- RTL Verilog source code model of the core
- Verilog testbench and test cases
- Synthesis scripts examples
- Complete detailed documentation and training class notes

DMA Engine

DMA operations include:

- memory to memory block moves
- memory to bus device moves
- bus device to memory moves

Block moves of up to 64K bytes are supported. The exact transfer count of each DMA operation is set by software. DMA operations are done as a series of main memory and AMBA Bus transactions to move the data block. Main memory accesses are sized to take advantage of higher performance burst accesses to memories such as SDRAMs. The length of each AMBA Bus transaction is software programmable so that it can be optimized according to system characteristics. Each individual AMBA Bus transaction is from 8 bytes to 32K bytes according to a value programmed into a DMA channel's control register. Additionally, the data size of each data transfer on the AMBA Bus is software programmable to be one, two, four, or eight bytes.

The series of accesses that make up a complete DMA operation may be locked together so that no other device gets the memory and AMBA Bus until the DMA operation is finished. This is under software control.

The source and destination starting addresses are set by software. These addresses are incremented by the DMA Engine to form the request addresses at the memory and AMBA Bus interfaces of the DMA Engine Subsystem. All addresses are physical addresses.

The DMA control information that is set by software- starting address, transfer count, bus transaction size, data transfer size, and lock flag, can be set by direct software writes to DMA Engine registers that hold this DMA control information. Alternatively, this DMA control information can be set from descriptors in memory that hold the DMA control information. Scatter/gather DMA is done using a chained descriptor list as the DMA control information source. When using descriptors to set the DMA control information, the descriptors are initialized by software. To support DMA configuration from descriptors, the DMA Engine contains logic to read the descriptors from memory, and load the appropriate DMA Engine registers with the DMA control information.

A DMA operation begins when software enables a DMA channel, after setting the source and destination starting addresses, transfer count, bus transaction size, and lock feature. The DMA Engine moves the data block, and the DMA operation ends naturally when the number of bytes specified by the transfer count have been moved. A DMA operation may also end early due to an error from the AMBA Bus that is passed to it by the AMBA Bus Interface.

When a DMA operation ends, an interrupt informs the host processor. Each DMA channel has a register that identifies the event that caused the interrupt- a natural DMA operation end or any one of several types of errors.

For highest performance and versatility, each DMA channel has a dedicated AMBA Bus master interface. The DMA Engine has a single interface to memory that is shared by all DMA channels. The DMA Engine uses round robin arbitration to choose the DMA channel that gets each available time slot on the memory interface.

Although the host processor sets the transfer count and initiates the DMA operation, a bus slave device can effectively do this also. A bus slave determines the transfer count by ending a DMA operation early with its error signal. To control the DMA request initiation from the bus slave, once the host has enabled the DMA channel, the bus slave can withhold the data transfer until it wants to start the DMA operation. For this to be possible each channel should have its own bus master so that if one bus master is waiting for a bus slave to respond, other channels may still conduct DMA operations through their dedicated bus masters. Additionally, to attain most efficient bus usage, a bus slave DMA request followed by a DMA channel acknowledge handshake is provided so that each bus burst begins upon the bus slave DMA request that is made when the bus slave can transfer the entire burst without wait states.

AMBA Bus Interface

The AMBA Bus Interface consists of a dedicated AMBA Bus Master block for each DMA channel. The AMBA Bus Interface can connect to either a 32 bit or 64 bit AMBA AHB Bus. A Verilog parameter indicates the AMBA Bus width.

The AMBA Masters of the AMBA Bus Interface, are fully pipelined for highest throughput, and therefore highest system performance. Each AMBA Master can issue AMBA Bus requests, drive AMBA Bus data, and capture data from the AMBA Bus, each AMBA Bus clock cycle.

Each AMBA Master supports all required AMBA AHB Bus features including all AMBA burst and wrapping types, AMBA sizes up to the AMBA Bus width, and the AMBA Bus lock feature. Request lengths of up to 32 Kbytes from the user are supported and translated into one or more AMBA Bus transactions. When an AMBA slave responds with the RETRY or SPLIT response, the AMBA Master responds accordingly to eventually complete the transaction. This is transparent to the user (accept for the additional delay).

Each DMA channel of the DMA Engine block issues AMBA Bus requests to a dedicated AMBA Master. After accepting a request, the AMBA Master initiates and completes an AMBA Bus transaction for the accepted request. The AMBA Master interfaces directly onto the AMBA bus and takes care of all AMBA Bus protocol requirements for the transaction. If the transaction is a read transaction, the AMBA Master assembles the read data from the AMBA Bus and returns it to the requesting DMA channel.

The AMBA Master returns read errors from the AMBA Bus to the DMA Engine. The read error from a slave's ERROR response, is returned to the DMA Engine along with the read data that came with the slave's ERROR response. The user interprets the read data that came with the read error according to the slave's definition of read data with the ERROR response. Slave ERROR responses upon AMBA Bus writes are not returned to the user. Writes fail silently.