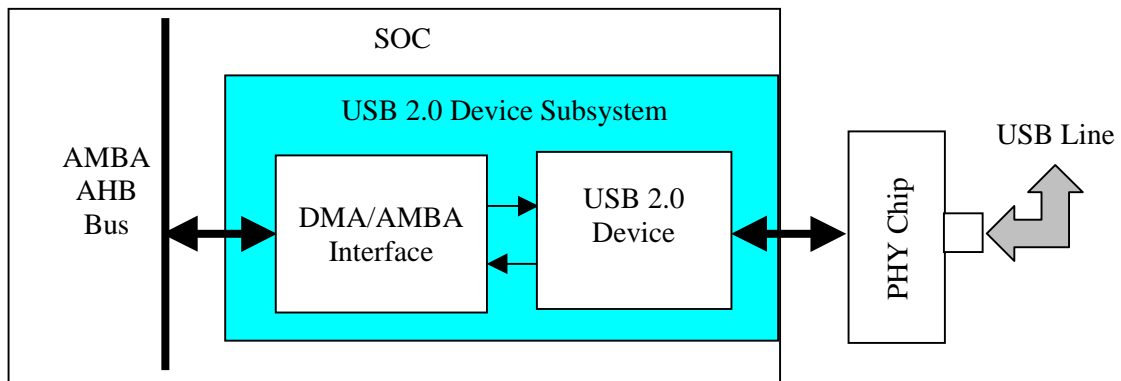


## **AU-UB7220: USB 2.0 Device AMBA Subsystem Core** **(Mini Version)**

### **AMBA AHB Bus USB 2.0 Device with DMA**

The AU-UB7220 USB 2.0 Device AMBA Subsystem provides a USB 2.0 Device peripheral subsystem for AMBA based SOCs. It contains a USB 2.0 Device that connects seamlessly to the AMBA AHB Bus. It minimizes gate count by including only the required USB endpoints plus two bulk/iso transfer endpoints. A DMA Engine is included to move bulk and isochronous USB data. The figure below shows its use within an SOC. The USB 2.0 Device AMBA Subsystem Core is available as a synthesizable Verilog model from Aurora VLSI, Inc. Contact [CustomerService@auroravlsi.com](mailto:CustomerService@auroravlsi.com).



The application uses two bulk/iso pipes for data transfer. Endpoint 2 (EP2) sends data over the bulk/iso IN pipe. Endpoint 3 (EP3) receives data from the bulk/iso OUT pipe. Data typically resides in memory at the two bulk/iso endpoints. Direct Memory Access- DMA, between the DMA/AMBA Interface block of the USB 2.0 Device Subsystem, and AMBA Bus targets, is used to transfer the EP2 and EP3 data to/from the USB 2.0 Device block. When doing DMA, the USB 2.0 Device Subsystem is an AMBA Bus master.

The interrupt pipe is used for event notification- interrupts for events such as media change, media no longer ready, etc. The application posts interrupts in the Interrupt Registers, and the USB Host reads the interrupt registers with IN transactions to EP1. All the USB, class specific, and vendor specific commands are decoded and executed as register transfers through Control Endpoint- EP0. Configuration, interface, endpoint status registers, etc. are also accessed through EP0. The USB 2.0 Device Subsystem is an AMBA Bus slave for interrupt, command, configuration, and status register accesses. All such register accesses originate in an AMBA Bus master outside of the USB 2.0 Device Subsystem, such as an embedded host processor.

USB 2.0 Device AMBA Subsystem features are summarized:

USB 2.0 Device

- Four endpoints:
  - EP0- control endpoint, accepts SETUP, IN, and OUT control transactions
  - EP1- interrupt endpoint, accepts IN and OUT interrupt transactions
  - EP2- IN endpoint; accepts IN bulk and isochronous transactions
  - EP3- OUT endpoint; accepts OUT bulk and isochronous transactions
- Includes control pipe registers and USB Descriptor RAMs at EP0
- USB command execution in EP0
- 8 or 16 bit UTMI interface
- Accepts stalls from the application logic
- Low gate count

DMA/AMBA Interface

- AMBA AHB Bus interface
- 4 channel DMA Engine
  - bulk/iso IN data from EP1 and EP2 to the USB 2.0 Device block
  - bulk/iso OUT data from the USB 2.0 Device block to EP1 and EP3
- Physical DMA addresses
- Programmable DMA starting address
- Programmable DMA transfer count- up to 64 Kbytes
- Programmable DMA AMBA Bus interface transaction size- 8 to 1024 bytes
- Programmable DMA AMBA Bus data transfer size- 4 or 8 bytes
- Locked DMA operation optional (software programmable)
- Direct software writes or information extracted from descriptors in memory, to program DMA control information
- 2 AMBA Bus master interfaces- one for IN data, one for OUT data
- AMBA Bus slave interface for register reads and writes
- Interrupts:
  - DMA completed
  - IN transaction data sent
  - OUT transaction data received

The core is delivered as a synthesizable 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

## **USB 2.0 Device**

The USB 2.0 Device Subsystem includes the Stargate SSU7220 USB 2.0 Device Core. Additional logic at the application interface of the USB 2.0 Device provides a DMA Engine, AMBA Bus interface, and host processor interrupts for the USB 2.0 Device Subsystem.

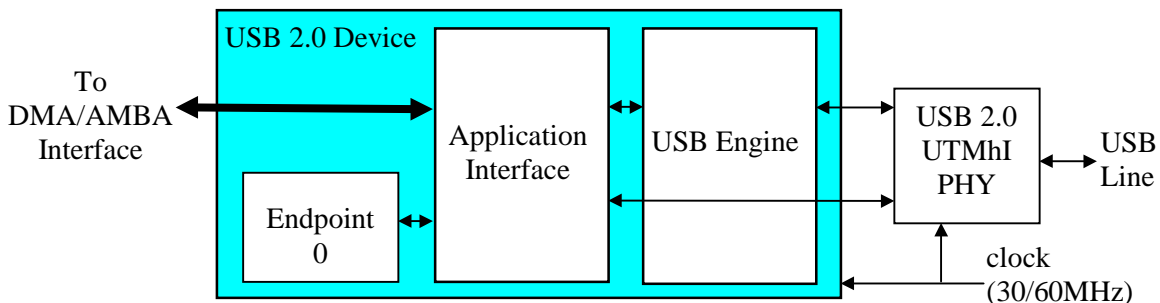
A block diagram of the USB 2.0 Device is shown below.

There are three major blocks in USB 2.0 Device, the USB Engine, the Application Interface, and Endpoint 0.

The USB Engine keeps track of a transaction on TXVALID, from SOP to EOP. On SOP, it checks the validity of the address and endpoint, and initiates the appropriate data transaction based on the status of the endpoint FIFOs. It handles the data retry mechanism using data toggling and generates appropriate handshakes.

The Application Interface provides a simple mechanism to interface to the DMA/AMBA Interface block. All interrupt, bulk, and isochronous endpoint FIFOs, registers, and any other memory elements are in the DMA/AMBA Interface. Control endpoint (EPO) registers and USB Descriptor RAMs are in the Endpoint 0 block. The Application Interface controls USB accesses to all these endpoint FIFOs, registers, and other memory elements. Each endpoint is controlled independently. This allows simultaneous access to any number of endpoints. Setting and clearing of stall conditions are also controlled through the Application Interface.

Endpoint 0 contains the registers and USB Descriptor RAMs that are referenced by Endpoint 0 control pipe commands that arrive over the USB line from the USB Host. These commands are decoded in the Endpoint 0 block, and then executed as register or USB Descriptor RAM reads and writes.



## **DMA/AMBA Interface**

The DMA/AMBA Interface includes a two channel DMA Engine. One channel is used to transfer bulk/iso IN data from the data's source, over the AMBA Bus, to the USB 2.0 Device block. Bulk/iso OUT data is sent from the USB 2.0 Device block, over the AMBA Bus, to the data's destination, by the second DMA channel.

Block moves of up to 64K bytes are supported by the DMA Engine. The exact transfer count of each DMA operation is the size of the data block that is being transferred, and is set by software. DMA operations are done as a series of USB 2.0 Device Subsystem FIFO accesses, and bus transactions to move the data block. The length of each bus transaction is software programmable so that it can be optimized according to system characteristics. Each individual bus transaction is from 8 bytes to 1024 bytes according to a value programmed into a DMA channel's control register. Additionally, the data size of each data transfer on the bus is software programmable to be 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 AMBA Bus until the DMA operation is finished. This is under software control.

The DMA starting address is set by software. This is the data source starting address in memory for bulk/iso IN data, and the data destination starting address in memory for bulk/iso OUT data. These starting addresses are incremented by the DMA Engine to form the AMBA Bus transaction addresses as the DMA operation progresses. 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 USB 2.0 Device Subsystem 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/AMBA Interface contains logic to read the descriptors from memory, and load the appropriate DMA/AMBA Interface registers with the DMA control information.

A DMA operation begins when the DMA channel is enabled, after the starting address, transfer count, bus transaction size, data transfer size, and lock flag are configured. The DMA channel moves the data block, and the DMA operation ends when the entire data block has been moved.

The DMA/AMBA Interface generates interrupts to notify the embedded host processor of events that are important to driver software. These interrupts include:

- Interrupt upon a completed DMA operation
- Interrupt upon completely sending IN data of a transaction
- Interrupt upon completely receiving OUT data of a transaction