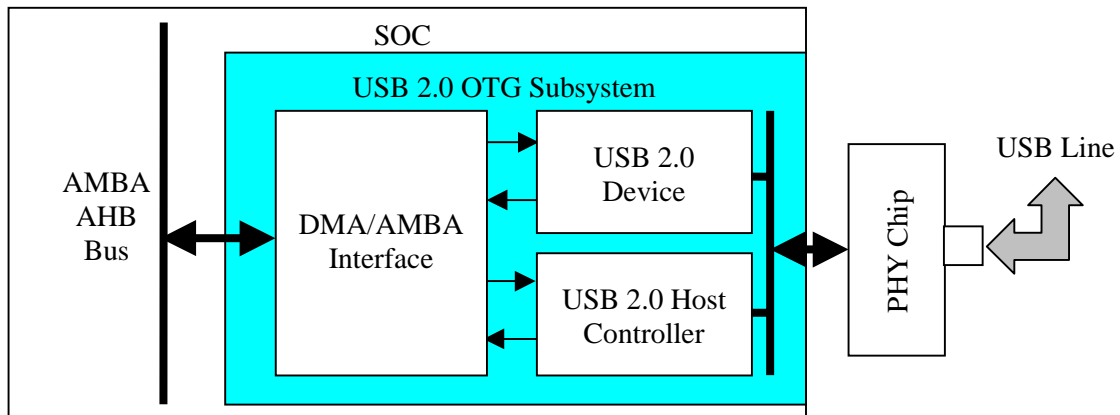


## **AU-UB9520: USB 2.0 OnTheGo AMBA Subsystem Core** **AMBA AHB Bus USB 2.0 OTG with DMA**

The AU-UB9520 USB 2.0 OnTheGo (OTG) AMBA Subsystem provides a USB 2.0 OnTheGo (OTG) peripheral subsystem for AMBA based SOCs. It contains a USB 2.0 Device and Host Controller that connects seamlessly to the AMBA AHB Bus. As a dual mode USB 2.0 Controller, its intended use is:

- in USB hosts and devices
- as a host with several (up to eight) device ports
- in device mode as a device or hub
- where more fully featured USB drivers (software) are required

It is a type A dual mode USB 2.0 Controller and therefore, begins each session as the USB host that then, when requested, will relinquish its host status to become a USB device/hub. When operating as a host, it supports up to eight downstream ports. The USB 2.0 OTG AMBA Subsystem includes the required USB endpoints plus eight bulk/iso transfer endpoints for use when in device mode. A DMA Engine is included to move the USB data to and from memory. When in host mode, the DMA Engine supports the Universal Host Controller Interface (UHCI) and Extended Host Controller Interface (EHCI). The figure below shows its use within an SOC. The USB 2.0 OTG AMBA Subsystem Core is available as a synthesizable Verilog model from Aurora VLSI, Inc. Contact [CustomerService@auroravlsi.com](mailto:CustomerService@auroravlsi.com).



In USB 2.0 device mode, the application uses eight bulk/iso pipes for data transfer. Endpoint 2, 4, 6, and 8 (EP2, EP4, EP6, and EP8) send data over the bulk/iso IN pipes. Endpoint 3, 5, 7, and 9 (EP3, EP5, EP7, and EP9) receive data from the bulk/iso OUT pipes. Data typically resides in memory at the eight bulk/iso endpoints. 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, standard, class specific, and vendor specific commands are decoded and executed as register transfers through Control Endpoint- EP0.

In USB 2.0 host mode, the application schedules USB transactions and provides, in memory, a list of the scheduled transactions along with their data/data buffers and other transactions characteristics. The USB 2.0 Host Controller works from this list in memory and issues the scheduled transactions. It also tracks the frames and issues SOF transactions.

Direct Memory Access- DMA, between the DMA/AMBA Interface block of the USB 2.0 OTG Subsystem, and AMBA Bus targets, is used to transfer endpoint data to/from the USB 2.0 Device and Host Controller blocks. For DMA, the USB 2.0 OTG Subsystem is an AMBA Bus master.

The USB 2.0 OTG 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 OTG Subsystem, such as an embedded host processor.

USB 2.0 OTG AMBA Subsystem features are summarized:

- Type A dual mode USB 2.0 Controller
- USB high speed and full speed operation
- Session Request Protocol (SRP) supported in both device and host modes
- Host Negotiation Protocol (HNP) supported in both device and host modes
- 4 or 8 bit ULPI (low pin count UTMI+) interface

#### USB 2.0 Device

- Ten endpoints:
  - EP0- control endpoint, accepts SETUP, IN, and OUT control transactions
  - EP1- interrupt endpoint, accepts IN and OUT interrupt transactions
  - EP2, EP4, EP6, EP8- IN endpoints, IN bulk and isochronous transactions
  - EP3, EP5, EP7, EP9- OUT endpoints, OUT bulk and isochronous transactions
- Includes control pipe registers and USB Descriptor RAMs at EP0
- USB command execution in EP0
- Can function as a hub that enables peer to peer USB communication among dual mode devices attached to its ports

#### USB 2.0 Host Controller

- Initiates all required USB high speed and full speed transaction types
- Up to eight downstream ports
- UHCI (full speed operation) and EHCI (high speed operation) support
- UHCI and EHCI list and transaction descriptor processing in conjunction with DMA

#### DMA/AMBA Interface

- AMBA AHB Bus interface
- 12 channel DMA Engine
  - USB data from/to each endpoint to/from the USB 2.0 Device block
  - USB data from/to host data buffers to/from the USB 2.0 Host Controller
- Physical DMA addresses
- Programmable DMA starting address, transfer count, transaction size, transfer size
- Locked DMA operation optional (software programmable)
- Direct software writes or information extracted from descriptors in memory, to program DMA control information
- 3 AMBA Bus master interfaces- DMA transmit data, DMA receive data, non-DMA
- AMBA Bus slave interface for register reads and writes
- Interrupts- device mode and host mode

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

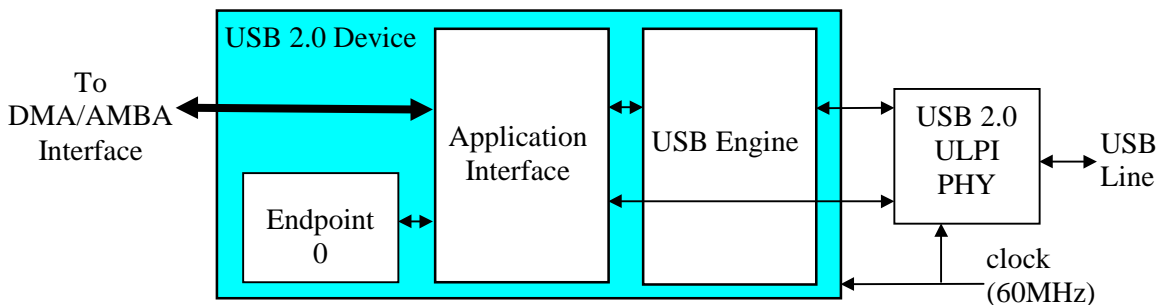
## **USB 2.0 Device**

The USB 2.0 OTG Subsystem includes the Stargate SSU7320 USB 2.0 Device Core. 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 (EP0) 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.



## **USB 2.0 Host Controller**

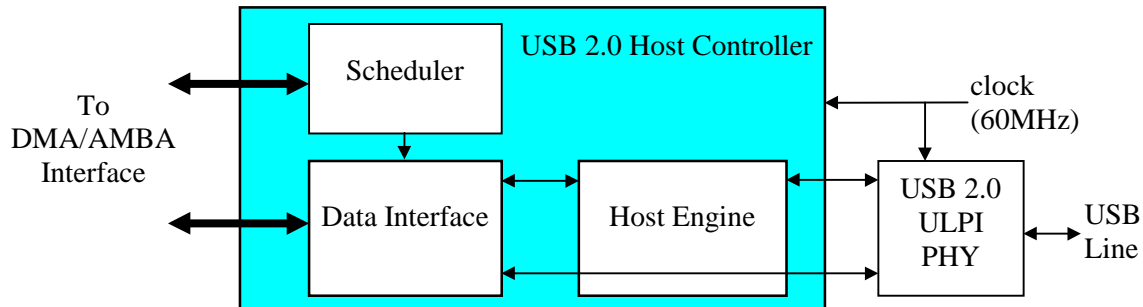
The USB 2.0 OTG Subsystem includes a Host Controller Core. A block diagram of the USB 2.0 Host Controller is shown below. There are three major blocks in USB 2.0 Host Controller, the Scheduler, the Data Interface, and the Host Engine.

The Scheduler generates the USB frame number. For each USB frame, using its frame number, the Scheduler traverses the USB transaction lists that have been set up by software for that frame. The Scheduler extracts all USB transaction information- PID, address, endpoint, etc., for each transaction and passes this information to the Data Interface. The Scheduler also identifies the location and size of the data buffer in memory for each transaction, and uses this information to set up a DMA channel that moves the data to/from the data buffer in memory.

The Data Interface provides USB side control to FIFOs in the DMA/AMBA Interface of the USB 2.0 OTG Subsystem. It generates the USB side FIFO addresses and control signals to move the

USB data between the USB line and these FIFOs. The Data Interface also combines the USB transaction information from the Scheduler with the data flow to initiate USB transactions in the Host Engine.

The Host Engine accepts USB transaction requests from the Data Interface. After accepting a transaction request, the Host Engine translates this request into a series of USB packets that it sends to the PHY and receives from the PHY. The Host Engine tracks each transaction from start to finish, including retries and error detection.



### **DMA/AMBA Interface**

The DMA/AMBA Interface includes an twelve channel DMA Engine. One channel for each bulk/iso IN endpoint (EP1, EP2, EP4, EP6, and EP8) 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 for each bulk/iso OUT endpoint (EP1, EP3, EP5, EP7, and EP9) is sent from the USB 2.0 Device block, over the AMBA Bus, to the data's destination, by a DMA channel dedicated to that OUT endpoint. OUT data from a host OUT data buffer in memory is moved to the Host Controller block by another dedicated DMA channel. The last DMA channel moves IN data from the Host Controller block to a host IN data buffer in memory.

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 or the Scheduler. DMA operations are done as a series of USB 2.0 OTG 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 or the Scheduler. This is the data source starting address in memory for bulk/iso transmit data, and the data destination starting address in memory for bulk/iso received 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 OTG 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 (device mode)
- Interrupt upon completely receiving OUT data of a transaction (device mode)
- Interrupt upon completely receiving IN data of a transaction (host mode)
- Interrupt upon completely sending OUT data of a transaction (host mode)
- Error interrupts
- NAK interrupts
- Stall interrupts
- USB line condition interrupts
- Other UHCI and EHCI interrupts