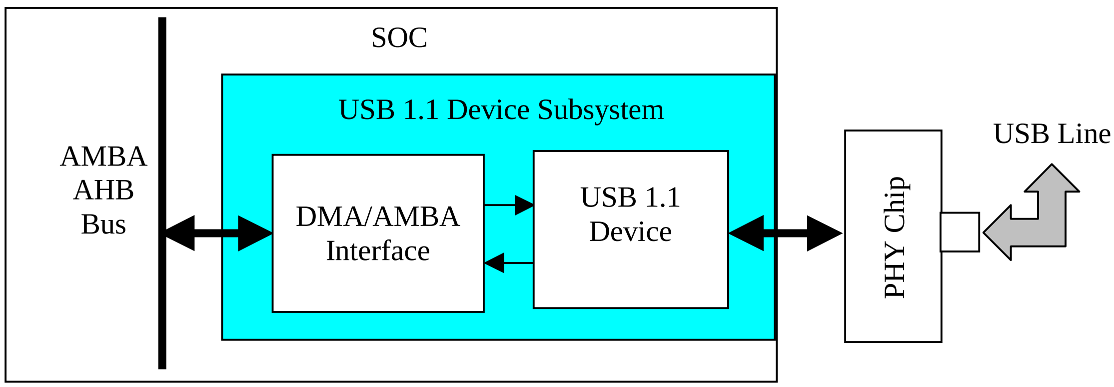
Aurora VLSI
Unleashing the Power of the Internet

AU-UB7311: USB 1.1 Device AMBA Subsystem Core

AMBA AHB Bus USB 1.1 Device with DMA

The AU-UB7311 USB 1.1 Device AMBA Subsystem provides a USB 1.1 Device peripheral subsystem for AMBA based SOCs. It contains a USB 1.1 Device that connects seamlessly to the AMBA AHB Bus. It includes the required USB endpoints plus eight 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 1.1 Device AMBA Subsystem Core is available as a synthesizable Verilog model from Aurora VLSI, Inc. Contact CustomerService@auroravlsi.com.



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. Direct Memory Access- DMA, between the DMA/AMBA Interface block of the USB 1.1 Device Subsystem, and AMBA Bus targets, is used to transfer the EP2, EP3, EP4, EP5, EP6, EP7, EP8, and EP9 data to/from the USB 1.1 Device block. When doing DMA, the USB 1.1 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 1.1 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 1.1 Device Subsystem, such as an embedded host processor.

USB 1.1 Device AMBA Subsystem features are summarized:

USB 1.1 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; accept IN bulk and isochronous transactions
  - EP3, EP5, EP7, EP9- OUT endpoints; accept OUT bulk and isochronous transactions
- Serial interface to PHY transceiver chip
- Accepts stalls from the application logic

DMA/AMBA Interface

- AMBA AHB Bus interface
- 8 channel DMA Engine
  - bulk/iso IN data from each bulk/iso IN endpoint to the USB 1.1 Device block
  - bulk/iso OUT data from the USB 1.1 Device block to each bulk/iso OUT endpoint
- 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
- Dedicated AMBA Bus master interface for each DMA channel
- 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 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

USB 1.1 Device

The USB 1.1 Device Subsystem includes the Stargate SSU7311 USB 1.1 Device Core. Additional logic at the application interface of the USB 1.1 Device provides a DMA Engine, AMBA Bus interface, and host processor interrupts for the USB 1.1 Device Subsystem.

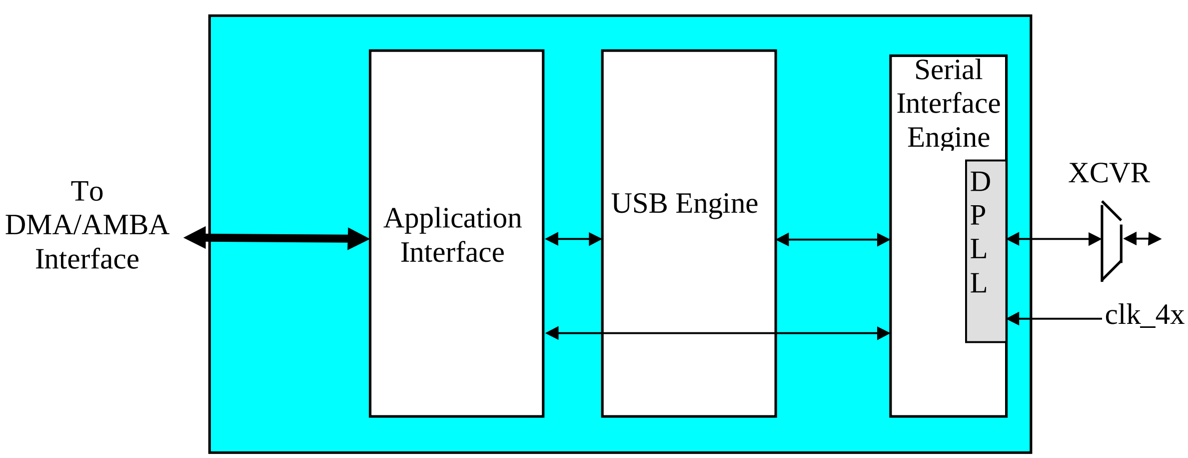
A block diagram of the USB 1.1 Device is shown below.

There are three major blocks in USB 1.1 Device, the Serial Interface Engine, USB Engine, and the Application Interface.

The Serial Interface Engine interfaces with a standard USB PHY transceiver on the line side. A DPLL is used to extract the clock from the received data stream. The Serial Interface Engine converts the received serial data stream into parallel bytes and delivers them to the USB Engine. In transmit mode, it converts the parallel bytes from the USB Engine into a serial data stream, and sends this serial data to the PHY transceiver. As part of the USB protocol, it performs SYNC detection, bit stuffing/unstuffing, PID decoding, NRZI encoding/decoding, and CRC checking/generation for token and data packets. It also monitors the line to detect reset, end of packet (EOP), start of packet (SOP), and idle conditions. In transmit mode, it generates SOP, EOP, and resume signaling. The Serial Interface Engine keeps track of the byte boundaries.

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 endpoint FIFOs, registers, and any other memory elements are in the DMA/AMBA Interface. The Application Interface allows direct access to all these endpoint FIFOs, registers, and other memory elements. It also controls the addressing and control signals for the USB 1.1 Device side of these 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.



DMA/AMBA Interface

The DMA/AMBA Interface includes an eight channel DMA Engine. One channel for each bulk/iso IN endpoint (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 1.1 Device block. Bulk/iso OUT data for each bulk/iso OUT endpoint (EP3, EP5, EP7, and EP9) is sent from the USB 1.1 Device block, over the AMBA Bus, to the data's destination, by a DMA channel dedicated to that OUT endpoint.

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 1.1 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 1.1 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