69 GRUSBDC - USB Device controller

69.1 Overview

The Universal Serial Bus Device Controller provides a USB 2.0 function interface accessible from an AMBA-AHB bus interface. The core must be connected to the USB through an external PHY (shown in figure 235) compliant to either UTMI, UTMI+ or ULPI. Both full-speed and high-speed mode are supported.

Endpoints are controlled through a set of registers accessed through an AHB slave interface. Each of the up to 16 IN and 16 OUT endpoints can be individually configured to any of the four USB transfer types.

USB data cargo is moved to the core's internal buffers using a master or a slave data interface. The data slave interface allows access directly to the internal buffers using AHB transactions and therefore does not need external memory. This makes it suitable for slow and simple functions. The data master interface requires an additional AHB master interface through which data is transferred autonomously using descriptor based DMA. This is suitable for functions requiring large bandwidth.

These two interfaces are mutually exclusive and cannot be present in the same implementation of the core.

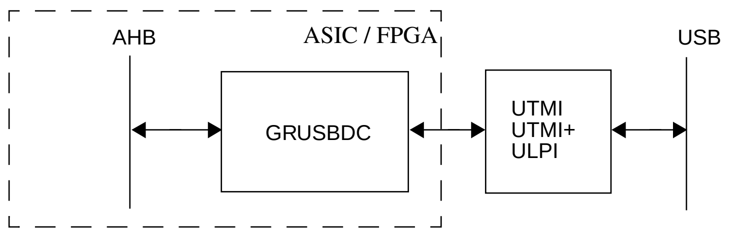

Figure 235. GRUSBDC connected to an external PHY device.

69.2 Operation

69.2.1 System overview

Figure 236 shows the internal structure of the core. This section briefly describes the function of the different blocks.

The Speed Negotiation Engine (SNE) detects connection by monitoring the VBUS signal on the USB connector. When a steady 5 V voltage is detected the SNE waits for a reset and then starts the High-speed negotiation. When the Speed negotiation and reset procedure is finished the selected speed mode (full-speed or high-speed) is notified to the Serial Interface Engine (SIE) which now can start operation. The SNE also detects and handles suspend and resume operations.

The SIE is enabled when the SNE notifies that the reset procedure has finished. It then waits for packets to arrive and processes them according to the USB 2.0 specification. The data cargo is stored to an internal buffer belonging to the recipient endpoint.

The AHB Interface Engine AIE is responsible for transferring USB data cargo from the endpoint's internal buffers to the AHB bus using descriptor based DMA through an AHB master interface when configured in master mode or by direct accesses to the AHB slave interface when configured in slave mode.

For received data it is then up to the external (to the device controller) function to continue processing of the USB data cargo after it has been transferred on the AHB bus. The function is the application specific core which determines the functionality of the complete USB device. It sets up endpoints in the device controller and notifies their existence through the appropriate USB descriptors. When the function wants to transmit a packet it either uses the slave interface to write to the endpoint buffers or establishes a DMA transfer.

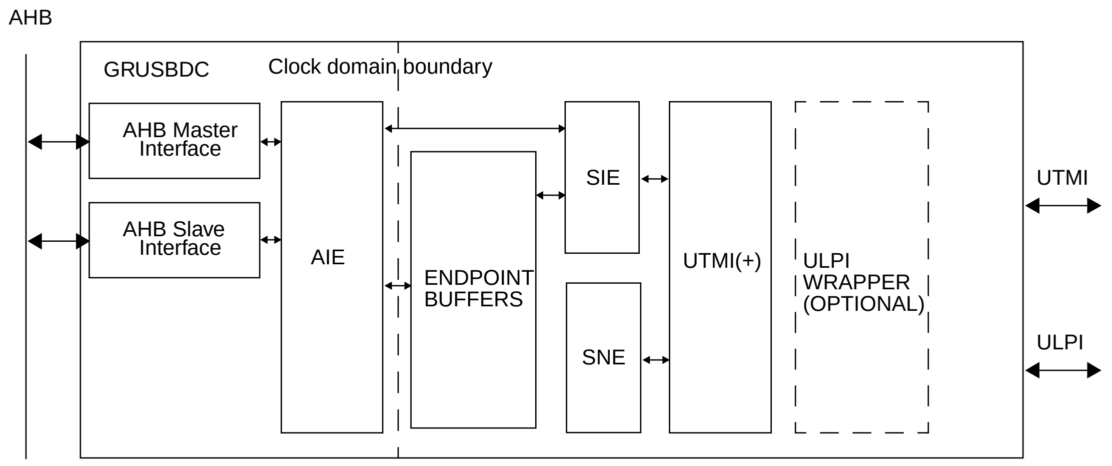

Figure 236. Block diagram of the internal structure of the core.

69.2.2 PHY interface

The core supports three different interfaces to the external PHY which is used to connect to the USB bus. The supported interfaces are UTMI, UTMI+ and ULPI. UTMI+ is an extension of the UTMI specification with optional additional support for host controllers and on-the-go devices while UTMI only supports devices. There are different so called levels in the UTMI+ specification, each with an added degree of support for hosts and on-the-go. The lowest level and common denominator for all the levels is identical to UTMI and uses the exact same signals. The core only supports devices and thus the support for UTMI+ refers to level 0. The data path to the UTMI/UTMI+ cores can be 8-bits or 16-bits wide and also uni- or bi-directional. All combinations are supported by the core.

The UTMI+ Low Pin Interface (ULPI) specifies a generic reduced pin interface and how it can be used to wrap a UTMI+ interface. The core has UTMI/UTMI+ as the main interface (they are identical) and when ULPI is used an extra conversion layer is added. When ULPI is enabled, the UTMI layer is always in 8-bit mode since this is what is required by the ULPI specification.

69.2.3 Speed Negotiation Engine (SNE)

The SNE detects attach, handles reset, high-speed handshake and suspend/resume operation. It also contains support for the various test-modes, which all USB device have to support.

The attached state is entered when a valid VBUS signal is detected. After this the core waits for a USB reset and then starts the high-speed handshake which determines whether full-speed or high-speed mode should be entered. No bus traffic will be accepted by the core until a valid reset has been detected.

The core supports soft connect/disconnect which means that the pull-up on the D+ line can be controlled from a user accessible register. The pull-up is disabled after reset and thus the function implementation has full control over when the device will be visible to the host.

The SNE also continuously monitors for the suspend condition (3 ms of idle on the USB bus) when the suspend state will be entered. The suspend state is left either through an USB reset or resume signaling. The resume signaling can come from either a downstream facing port (hub or host controller) or the device itself (Remote wakeup). The device controller core can generate remote wakeup signaling which is activated through an user accessible register. If this feature is used the function should indicate this in the descriptor returned to a device GetStatus request.

Transactions are only handled by the SIE if the SNE is in full-speed or high-speed mode. When in suspend, notattached, attached or during the reset process all transactions will be dropped.

The current status of the SNE such as VBUS valid, suspend active, USB reset received and current speed mode can be accessed through a status register. Each of these status bits have a corresponding interrupt enable bit which can be used to generate interrupts when a change occurs in the status bits.

If full-speed only mode is desired the core can be set to not perform the high-speed handshake through a register.

The different test-modes required by the USB standard are also enabled through user accessible registers. When enabled they can only be left by power-cycling the complete core or resetting the device controller (by using the rst signal to the core, not an USB reset).

The test modes are Test_SE0_NAK, Test_J, Test_K and Test_Packet.

In Test_SE0_NAK mode the high-speed receiver is enabled and only valid IN transactions (CRC correct, device and endpoint addresses match, PID is not corrupt) are responded to with a NAK.

Test_J continuously drives a high-speed J state.

Test_K continuously drives a high-speed K state.

Test_Packet repetitively sends a test packet. Please refer to the USB 2.0 standard for the packet contents. Minimum interpacket delay when device is sending two or more consecutive packets seems not to be specified in the standard. The core uses 192 bit times as its minimum delay which is the maximum value of the various minimum delays in the standard for any packet sequence and should therefore be compliant.

69.2.4 Serial Interface Engine (SIE)

The SIE handles transmission and reception of USB packets. The core will not respond to any transactions until a reset has been received and either full-speed or high-speed mode has been successfully entered.

The SIE always begins with waiting for a token packet. Depending on the type of token, data is either transferred from the core to the host or in the opposite direction. Special tokens are handled without any data transfers. The special tokens are PING and SOF which cause only a handshake to be sent or the frame number to be stored respectively. IN tokens initiate transfers to the host while SETUP and OUT tokens initiate transfers to the device. Packets received in the token stage with other PIDs than those mentioned in this paragraph are discarded.

A data packet is transmitted in the next stage if the token determined that data should be transferred to the host. When the data transmission is finished the core waits for a handshake before returning to the token stage.

If data was determined to be transferred to the device the core waits for and receives a data packet and then sends a handshake in return before entering the token stage again.

More detailed descriptions of the SIE and how it interacts with the core function are found in section 69.5.

69.2.5 Endpoint buffers

Each endpoint has two buffers to which packets are stored. The core automatically alternates between them when a packet has been received/transmitted so that data from one of the buffers can be transferred on the AHB bus while a new packet is being received/transmitted on the USB to/from the second buffer.

The state of the buffers affects the handshake sent to the host at the end of a transaction. In high-speed mode BULK OUT endpoints and CONTROL OUT endpoints which are not in the SETUP stage support the PING protocol. This means that at the end of an OUT transaction to one of these endpoints the device should return ACK if it could accept the current data and has space for another packet. For the USB device controller this is done if the second buffer for the endpoint is empty when the transaction is ready.

If the second buffer is non-empty a NYET is sent instead. If the current data could not be accepted (both buffers non-empty when the packet arrives) a NAK is returned.

For other endpoint types in high-speed mode and all endpoint types in full-speed mode an ACK is always returned if the data is accepted and a NAK if it could not be accepted.

An endpoint buffer can be configured to be larger than the maximum payload for that endpoint. For IN endpoints the writing of data larger than the maximum payload size to a buffer will result in a number of maximum sized packets being transferred ending with a packet smaller than or equal to the maximum size.

In the OUT direction larger buffers are only used for high-bandwidth endpoints where more than one transaction per microframe can occur for that endpoint. In that case the data from all packets during one microframe is stored in the order it arrives to a single buffer and is then handed over to the AHB interface. All non-high-bandwidth endpoints always store one packet data cargo to a buffer.

The endpoint buffers do not use separate physical RAM blocks in hardware instead they reside consecutively in the same memory space to avoid wasting memory.

69.2.6 AMBA Interface Engine (AIE)

The AIE can either be configured in slave mode or master mode. This is selected in synthesis process with a VHDL generic. Both cannot be present at the same time. The two interfaces will be described separately in this section.

Master interface

In master mode an AHB master interface is included in the core and handles all data transfers to and from the cores internal buffers using DMA operations. The DMA operation is described in detail in section 69.3. There is a separate DMA engine in the IN direction and the OUT direction respectively. They are multiplexed on the single master interface available for the core on the AHB bus. This scheme is used to limit the load on the AHB bus.

If both engines request the bus at the same time the owner will always be switched. That is, if the OUT direction DMA engine currently was allowed to make an access and when finished it still requests the bus for a new transaction and at the same time the IN direction engine also requests the bus, the IN direction will be granted access. If the situation is the same after the next access ownership will be switched back to the OUT engine etc.

The IN engine only reads data (note that this only applies to DATA, descriptor status is written) from the bus and always performs word transfers. Any byte alignment and length can still be used since this will only cause the core to skip the appropriate amount of leading and trailing bytes from the first and last words read.

The OUT engine writes data to the bus and performs both word and byte transfers. If the start transfer for an access is not word-aligned byte writes will be performed until a word boundary is reached. From then and onwards word writes are performed in burst mode until less than 4 bytes are left. If remaining number of bytes is not zero byte writes are performed for the last accesses. The byte accesses are always done as single accesses.

The bursts are of type incremental burst of unspecified length (refer to AMBA specification for more details). The core can only operate in big-endian mode that is the byte at the lowest address in a word is the most significant byte. This corresponds to bits 31 downto 24 in the GRLIB implementation. The first byte received on the USB will be stored to the msb location. In a single byte the lowest bit index corresponds to the first bit transmitted on the USB.

Slave interface

The slave interface is used for accessing registers and also for data transfers when the core is configured in slave mode. Byte, half-word and word accesses are supported. At least one waitstate is always inserted due to the pipelined nature of the interface but the upper limit is not fixed due to registers being accessed across clock domains. The maximum number of waitstates will thus depend on the difference in clock frequency between the USB and AHB clock domains. An upper bound can be calculated when the clock frequencies have been determined.

69.2.7 Synchronization

There are two clock domains in the core: the AHB clock domain and the USB clock domain. The AHB clock domain runs on the same clock as the AHB bus while the USB domain runs on the UTMI or ULPI clock. The boundary is between then AIE, SIE and Endpoint buffers. All signals between the two domains are synchronized and should be declared as false paths during synthesis.

69.2.8 Reset generation

The main reset (AMBA reset) resets AHB domain registers, synchronization registers between the USB and AHB clock domains, the USB PHY and USB SIE registers. Endpoint specific registers related to the state of the USB protocol in the SIE are reset when an UBS reset is received.

69.2.9 Synthesis

All number of endpoints with up to maximum size payloads cannot be supported due to limitations in the RAM block generator size in GRLIB. The maximum size also varies with technology. Note that large buffers can also have a large timing impact at least on FPGA since the a large RAM buffer will consist of several separate physical block RAMs located at different places causing large routing delays.

As mentioned in section 69.2.7, signals between the clock domains are synchronized and should be declared as false paths.

The complete AHB domain runs at the same frequency as the AHB bus and will be completely constrained by the bus frequency requirement.

The USB domain runs on different frequencies depending on the data path width. In 8-bit mode the frequency is 60 MHz and in 16-bit mode it is 30 MHz. Input and output constraints also need to be applied to the signals to and from the PHY. Please refer to the PHY documentation and/or UTMI/ULPI specification for the exact values of the I/O constraints.

69.3 DMA operation

DMA operation is used when the core is configured in AHB master mode. Each IN and each OUT endpoint has a dedicated DMA channel which transfers data to and from the endpoint's internal buffers using descriptor based autonomous DMA. Each direction (IN and OUT) has its own DMA engine which requests the AHB master interface in contention with the other direction. Also each endpoint in a direction contends for the usage of the DMA engine with the other endpoints in the same direction. The arbitration is done in a round-robin fashion for all endpoints which are enabled and have data to send or receive.

The operation is nearly identical in both directions and the common properties will be explained here while the differences are outlined in the two following sub-sections.

The DMA operation is based on a linked list of descriptors located in memory. Each endpoint has its own linked list. The first word in a descriptor is the control word which contains an enable bit that determines whether the descriptor is active or not and other control bits. The following word is a pointer to a memory buffer where data should be written to or read from for this descriptor. The last word is a pointer to the location of the next descriptor. A bit in the control word determines if the next descriptor pointer is valid or not. If not valid the descriptor fetching stops after the current descriptor is processed and the DMA channel is disabled.

The DMA operation is started by first setting up a list with descriptors in memory and then writing a pointer to the first descriptor to the endpoint's descriptor pointer register in the core and setting the descriptor available bit. The pointer register is updated as the list is traversed and can be read through the AHB slave interface. When the list is ended with a descriptor that has its next descriptor available bit disabled the list must not be touched until the core has finished processing the list and the channel is disabled. Otherwise a deadlock situation might occur and behavior is undefined.

Another way to use the linked list is to always set the next descriptor available bit and instead make sure that the last descriptor is disabled. This way new descriptors can be added and enabled on the fly to the end of the list as long as the descriptor available bit is always set after the new descriptors have been written to memory. This ensures that no dead lock will occur and that no descriptors are missed. Figure 237 shows the structure of the descriptor linked list.

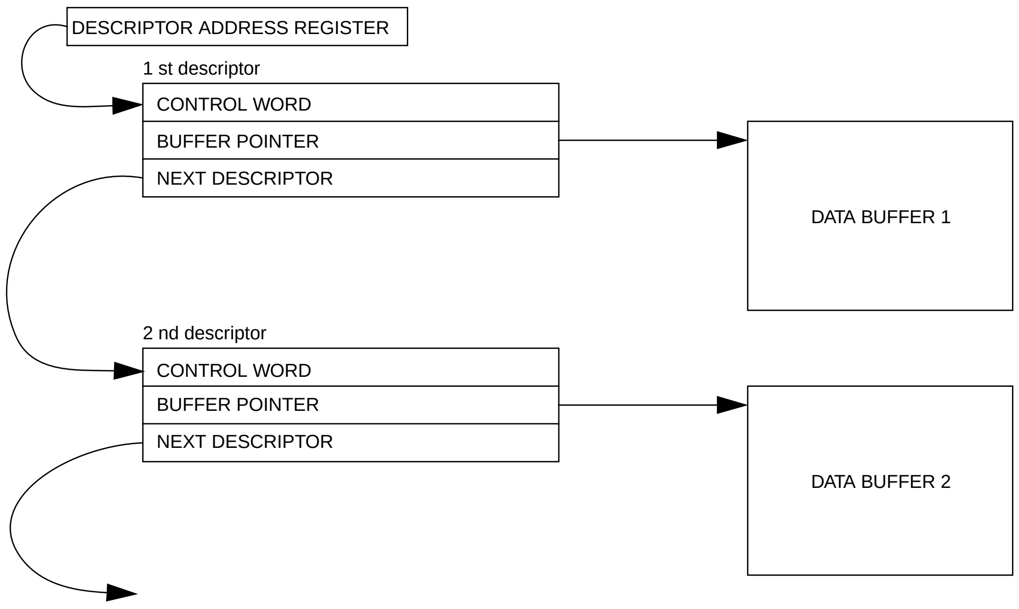

Figure 237. Example of the structure of a DMA descriptor linked list in memory.

69.3.1 OUT endpoints

The DMA operation for OUT endpoints conforms to the general description in the previous subsection. There are small differences in individual bits and the meaning of the length field. The contents of the different descriptor words can be found in the tables below.

When a descriptor has been enabled it will be fetched by the core when the descriptor available bit is set and as soon as a buffer for the corresponding endpoint contains data received from the USB it will be written to memory starting from the address specified in the buffer pointer word of the descriptor. The contents of a single internal memory buffer is always written to a single descriptor buffer. This always corresponds to a single USB packet except for high-bandwidth isochronous and interrupt endpoints. The number of bytes written is stored in the length field when writing is finished which is indicated by the enable bit being cleared. Then the SETUP status bit will also be valid. When the enable bit is cleared the memory location can be used again.

Interrupts are generated if requested as soon as the writing to memory is finished. The endpoint can also be configured to generate an interrupt immediately when a packet has been received to the internal buffers. This can not be enabled per packet since the core cannot associate a received packet with a specific descriptor in advance. This interrupt is enabled from the endpoint's control register.

When the data has been fetched from the internal buffer it is cleared and can be used by the SIE again for receiving a new packet.

Table 576. OUT descriptor word 0 (address offset 0x0) ctrl word

31 18	17	16	15	14	13	12 0
RESERVED	SE	RE	IE	NX	EN	LENGTH

31: 18	RESERVED
17	Setup packet (SE) - The data was received from a SETUP packet instead of an OUT.
16	RESERVED
15	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been read to the internal buffers and handed over to the SIE. This does not mean that packet has also been transmitted.
14	Next descriptor available (NX) - The next descriptor field is valid and points to the next descriptor.
13	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
12: 0	LENGTH - The number of bytes received. Valid when the EN bit has been cleared by the core.

Table 577. OUT descriptor word 1 (address offset 0x4) Buffer pointer

31 0
ADDRESS

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

Table 578. OUT descriptor word 2 (address offset 0x8) Next descriptor pointer

31 2	1 0
NDP	RES

31: 2	Next descriptor pointer (NDP) - Pointer to the next descriptor.
1: 0	RESERVED

69.3.2 IN endpoints

The DMA operation for IN endpoints conforms to the general DMA description. There are small differences in individual bits and the meaning of the length field. The contents of the different descriptor words can be found in the tables below.

When a descriptor has been enabled and the descriptor available bit is set the core will start processing the descriptor and fetch the number of bytes indicated in the length field to an internal buffer belonging to the endpoint as soon as one is available. An interrupt will be generated if requested when data has been written to the internal buffer and status has been written back to the descriptor. The packet might not have been transmitted on the USB yet.

A separate interrupt is available which is generated when the packet has actually been transmitted. It needs to be enabled from the endpoint's control register and also in the descriptor (using the PI bit) for each packet that should generate the interrupt.

A descriptor with length zero will result in a packet with length zero being transmitted while a length larger than the maximum payload for the endpoint will result in two or more packets with all but the last being of maximum payload in length. The last transaction can be less than or equal to the maximum payload. If the length field is larger than the internal buffer size the data will not written to the internal buffer and status will be immediately written to the descriptor with an error bit set.

When the more bit is set the data from the current descriptor is written to the internal buffer and it then continues to the next descriptor without enabling the buffer for transmission. The next descriptor's data is also read to the same buffer and this continues until a descriptor is encountered which does not have more set.

If the total byte count becomes larger than the internal buffer size the packet is not sent (the data from the internal buffer is dropped) and the ML bit is set for the last descriptor. Then the descriptor fetching starts over again.

If the next bit is not set when the more bit is, the core will wait for a descriptor to be enabled without letting other endpoints access the AHB bus in between.

Table 579. IN descriptor word 0 (address offset 0x0) ctrl word

31 19	18	17	16	15	14	13	12 0
RESERVED	MO	PI	ML	IE	NX	EN	LENGTH

31: 19	RESERVED
18	More (MO) - The data from the next descriptor should be read to the same buffer.
17	Packet sent interrupt (PI) - Generate an interrupt when packet has been transmitted on the USB.
16	Maximum length violation (ML) - Attempted to transmit a data cargo amount larger than the buffer.
15	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been read to the internal buffers and handed over to the SIE. This does not mean that packet has also been transmitted.
14	Next descriptor available (NX) - The next descriptor field is valid and points to the next descriptor.
13	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
12: 0	LENGTH - The number of bytes to be transmitted.

Table 580. IN descriptor word 1 (address offset 0x4) Buffer pointer

31 0
ADDRESS

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

Table 581. IN descriptor word 2 (address offset 0x8) Next descriptor pointer

31 2	1 0
NDP	RES

31: 2	Next descriptor pointer (NDP) - Pointer to the next descriptor.
1: 0	RESERVED

69.4 Slave data transfer interface operation

The AHB slave interface is used for data transfers instead of the DMA interface when the core is configured in AHB slave mode. This mode is selected by setting the aiface VHDL generic to 0. In this mode the core's internal buffers containing data to/from USB packets are accessed directly using AHB read and write transfers. This is usually much slower than DMA but much simpler and does not require any external memory, thus suitable for slow devices which need to be small and simple.

As for the DMA mode each endpoint is operated separately using four registers at the same addresses. Two of them, the control and status registers, are the same while the DMA control and the descriptor address registers have been replaced with the slave control and slave write/read data registers. The slave interface does not use descriptors so these four registers provides the complete control of the endpoint.

The details for data transfers in the two different endpoint directions will be explained in separate sections.

69.4.1 OUT slave endpoint

As stated earlier, in slave mode the core’s buffers are accessed directly from the AHB bus through the slave interface. For OUT endpoints it has to be checked that data is available in the selected buffer and then reserve it. This is done using the slave control register by writing a one to the CB bit. The CB bit is always automatically cleared when the write access is finished and then the BS, DA and BUFCNT read-only bits/fields contain valid information. The BS bit is set to 0 if buffer 0 is currently selected and to 1 if buffer 1 is selected. DA is set to 1 if data is available in the buffer and in that case BUFCNT contains the number of bytes.

If data is available (DA is 1) it can be read from the slave data read register. One byte at a time is read using byte accesses, two bytes using half-word accesses and four bytes using word accesses. No other widths are supported. In each case data is available from bit 31 and downwards regardless of the value of the two least significant address bits. This is summarized in table 582. The BUFCNT field is continuously updated when reading the buffer so that it can be monitored how many bytes are left.

*Table 582.*AHB slave interface data transfer sizes

Size (byte)	AHB transfer size (HSIZE)	Data alignment (HRDATA)
1	byte (000)	31:24
2	half-word (001)	31:16
4	word (010)	31:0

When all the data has been read, a new buffer can be acquired by writing a one to CB. In this case when a buffer is currently reserved and the DA bit is set it will be released when CB is written. If a new buffer was available it will be reserved and DA is set to 1 again. If no new buffer is available DA will be 0 and the process has to be repeated. The current buffer (if one is selected) will be released regardless of whether a new one is available or not. Also, if all data has not been read yet when a buffer change request is issued the rest of the data will be lost.

The core does not have to be polled to determine whether a packet is available. A packet received interrupt is available which can be enabled from the control register and when set an interrupt will be generated each time a packet is stored to the internal buffers. The status of the buffers can also be read through the endpoint’s status register without actually reserving the buffer.

One buffer consists of the data payload from one single packet except for high-bandwidth interrupt and isochronous endpoints for which up to three packet data payloads can reside in a single buffer.

A buffer does not have to be read consecutively. Buffers for several endpoints can be acquired simultaneously and read interleaved with each other.

69.4.2 IN slave endpoint

The slave operation of IN endpoints is mostly identical to that for OUT. For IN endpoints it has to be checked that a buffer is free and then reserve it before writing data to it. This is done using the slave control register by writing a one to the EB bit. The EB bit is always automatically cleared when the write access is finished and then the BS, BA and BUFCNT fields are updated. The BS bit is set to 0 if buffer 0 is currently selected and to 1 if buffer 1 is selected. BA is set to 1 if a buffer is available to write data to. BUFCNT contains the number of bytes currently written to the selected buffer. It is cleared to zero when a new buffer is acquired.

If a buffer is available (BA is 1) data can be written through the slave data write register. One byte at a time is written using byte accesses, two bytes using half-word accesses and four bytes using word accesses. No other widths are supported. In each case data should be placed from bit 31 and downwards regardless of the value of the two least significant address bits. This is summarized in table 583.

*Table 583.*AHB slave interface data transfer sizes

Size (byte)	AHB transfer size (HSIZE)	Data alignment (HWDATA)
1	byte (000)	31:24
2	half-word (001)	31:16
4	word (010)	31:0

When all the data has been written, the buffer is enabled for transmission by writing a one to EB. If a new buffer was available it will be reserved and BA is set to 1 again. If no new buffer is available BA will be 0 and the process has to be repeated. The current buffer (if one is reserved) will be enabled for transmission regardless of whether a new one is available or not.

The core does not have to be polled to determine whether a buffer is available. A packet transmitted interrupt is available which can be enabled from the endpoint control register. Then when enabling a packet for transmission the PI bit in the slave control register can be set which will be cause an interrupt to be generated when this packet has been transmitted and cleared from the internal buffers. The status of the buffers can also be read through the endpoint's status register without actually reserving the buffer.

Maximum payload size packets will be generated from the buffer until the last packet which will contain the remaining bytes.

A buffer does not have to be written consecutively. Buffers for several endpoints can be acquired simultaneously and written interleaved with each other. This will however cause additional waitstates to be inserted.

69.5 Endpoints

An endpoint needs to have both its AHB and USB function configured before usage. The AHB configuration comprises the DMA operation described in the previous section. The USB configuration comprises enabling the endpoint for USB transactions, setting up transfer type (control, bulk, isochronous, interrupt), payload size, high-bandwidth among others. The configuration options are accessed through a register available from the AHB slave interface. See section 69.8 for the complete set of options.

When setting the configuration options the endpoint valid bit should be set. This will enable transfers to this endpoint as soon as a USB reset has been received. If the endpoint is reconfigured the valid bit must first be set to zero before enabling it again. Otherwise the endpoint will not be correctly initialized. When the endpoint is enabled the toggle scheme will be reset and data buffers cleared and buffer selectors set to buffer zero. The maximum payload, number of additional transactions and transfer type fields may only be changed when endpoint valid is zero or when setting endpoint valid to one again after being disabled. Other bits in the endpoint control register can be changed at any time.

No configuration options should be changed when the endpoint is enabled except the halt, control halt and disable bits.

An endpoint can also be halted by setting the halt bit of the endpoint. This will cause all transactions to receive a STALL handshake. When clearing the halt condition the toggle scheme will also be reset as required by the USB standard.

When the endpoint is setup, data transfers from and to the endpoint can take place. There is no difference in how data is transferred on the AHB bus depending on the selected transfer type. This only affects the transfers on the USB. For control endpoints some extra handling is required by the core user during error conditions which will be explained in the control endpoint section.

Packets that are received to an endpoint (independent of endpoint type) with a larger payload than the configured maximum value for the endpoint will receive a STALL handshake and cause the endpoint to enter halt mode.

When a USB reset is detected the CS, ED and EH bits of the endpoint control register will be cleared for all endpoints. The EV bit will also be cleared except for control endpoint 0.

The CB bit in the endpoint control register is for clearing the internal buffers of an endpoint. When set the data will be discarded from the buffers, the data available bits for the corresponding buffers will be cleared and the CB bit is cleared when it is done. This will however not work if a transaction is currently active to the same endpoint so this feature must be used with caution.

69.5.1 Control endpoints

Endpoint 0 must always be a control endpoint according to the USB standard and be accessible as soon as a USB reset is received. The core does not accept any transactions until a USB reset has been received so this endpoint can be enabled directly after power up. More control endpoints can be enabled as needed with the same constraints as the default control endpoint except that they should not be accessible until after configuration. If the function controlling the core is slow during startup it might not be able complete configuration of endpoint 0 before a USB reset is received. This problem is avoided in the core because its pull-up on D+ is disabled after reset which gives the function full control of when the device will be visible on the bus.

A control endpoint is a message pipe and therefore transfers data both in the IN and OUT direction. Thus control endpoints must use the endpoint in both directions with the same number in the device controller. This requires both to be configured in the same mode (same transfer type, payload ...). Otherwise device behavior is undefined.

A control transfer is always started with a SETUP transaction which will be received to the OUT endpoint. If the control transfer is a write the subsequent data phase will be in the OUT direction and this data will also be received to the OUT endpoint. The function should read both the setup data and the other data cargo and respond correspondingly. If the request was valid the function should enable a zero length packet for the endpoint in the IN direction which will lead to a valid status stage. If an error is detected it should instead halt the endpoint. There are two alternatives for this: A non-clearing halt which will last even after the next SETUP transaction or a clearing halt which will be removed when the next SETUP is received. The latter is the recommended behavior in the USB standard since the other will require the complete core to be reset to continue operation if the permanent halt appears on the default control endpoint.

The core can detect errors in single transactions which cause the endpoint to enter halt mode automatically. In this case the clearing halt feature will be used for control endpoints.

If a SETUP transaction indicates a control read the data phase will be in the IN direction. In that case the core user should enable data for the endpoint in the IN direction if the request was accepted otherwise the halt feature should be set. The transfer is finished when the host sends a zero length packet to the OUT endpoint.

Note that when entering halt for a control endpoint both the IN and OUT endpoints halt bits should be set.

Each time a control endpoint receives a setup token the buffers in the IN direction are emptied. This is done to prevent inconsistencies if the data and status stage were missing or corrupted and thus the data never fetched. The old data would still be in the buffer and the next setup transaction would receive erroneous data. The USB standard states that this can happen during error conditions and a new SETUP is transmitted before the previous transfer finished. The core user can also clear the buffer through the IN endpoint control register and is encouraged to do this when it detects a new SETUP before finishing the previous transfer. This must be done since the user might have enabled buffers after the core cleared them when receiving the new SETUP.

Whether data received to a descriptor for an OUT endpoint was from a SETUP transaction or an OUT transaction is indicated in a descriptor status bit.

69.5.2 Bulk endpoints

Bulk endpoints are stream pipes and therefore only use a single endpoint in either the IN or OUT direction. The endpoint with the same number in the other direction can be used independently. Data is accessed normally through the AHB interface and no special consideration need to be taken apart from the general endpoint guidelines.

69.5.3 Interrupt endpoints

Interrupt data is handled in the same manner as for bulk endpoints on the AHB interface. The differences only appear on the USB. These endpoints are also of stream type.

Interrupt endpoints support a high-bandwidth state which means that more than one transaction per microframe is performed. This necessitates buffers larger than the maximum payload size. The endpoint should be configured with a buffer larger or equal to the maximum payload times the number of transactions. All transactions will be received to/transmitted from the same buffer. The endpoint is configured as a high-bandwidth endpoint by setting the number of additional transactions to non-zero in the endpoint control register.

69.5.4 Isochronous endpoints

Isochronous endpoints are of stream type are identical to other endpoints regarding the handling on the AHB bus.

A big difference between isochronous endpoints and the other types is that they do not use handshakes. If no data is available when an IN token arrives to an Isochronous endpoint a data packet with length 0 is transmitted. This will indicate to the host that no error occurred but data was not ready. If no packet is sent the host will not know whether the packet was corrupted or not.

When not in high-bandwidth mode only one transaction in the OUT direction will be stored to a single buffer. In high-bandwidth mode all transactions during a microframe are stored to the same buffer.

In the IN direction data is always transferred from the same buffer until it is out of data. For high-bandwidth endpoints the buffer should be configured to be the maximum payload times the number of transactions in size.

Isochronous high-bandwidth endpoints use PID sequencing. When an error is detected in the PID sequence in the OUT direction no data is handed over to AHB domain for the complete microframe.

69.6 Device implementation example in master mode

This section will shortly describe how the USB device controller can be used in master mode.

A function controlling the device controller and implementing the actual application specific device will be needed. It can be either hardware, software or a combination. The only requirement is that it can control the device controller through the AHB bus.

The first thing needed for successful operation is a correctly configured PHY. This is automatically done by the device controller.

After this the device controller waits for attachment to the USB bus indicated by the VBUS becoming valid. This can be notified to the function either by polling or an interrupt. The time of attachment can be controlled by the function through the pull-up enable/disable bit in the core control register. When disabled the USB host will not notice the device even when it is plugged in.

After attachment a USB reset needs to be received before transactions are allowed to be accepted. This can also notified by polling or an interrupt.

Only control endpoint 0 should be accessible after reset. The function is responsible for enabling and configuring at the right time. It can wait until a USB reset has been received but it is easier to enable it immediately after power-up. This can be done since the device controller will not accept any transactions until USB reset has been received. When enabling the endpoint descriptors should also be enabled for both the IN and OUT direction and also the descriptor available bits should be set.

Then the endpoint is ready to accept packets and the function should wait for SETUP packets arriving. It can be notified of packets arriving either through polling or interrupts. The core should process the requests and return descriptors as requested. When a Set address request is received the function should write the new address to the device controller's global control register. It will take effect after the next successful IN transaction for the control endpoint. This should correspond to the status stage of the Set address transfer.

When a Set configuration request is received the function should enable the appropriate interface and endpoints according to the selected configuration. This is done by writing to the various endpoint control registers. The function is responsible for advertising the configurations and interfaces through the descriptors requested by SETUP transactions.

When the endpoints for the selected configuration are enabled the function should also setup the DMA operation. Then it is ready to transmit and receive data through the application specific endpoints. Interrupts can be used to notify that new packets have transferred and then polling will determine which endpoint had a status change.

69.7 Device implementation example in slave mode

This section will shortly describe how the USB device controller can be used in slave mode.

A function controlling the device controller and implementing the actual application specific device will be needed. It can be either hardware, software or a combination. The only requirement is that it can control the device controller through the AHB bus.

The first thing needed for successful operation is a correctly configured PHY. This is automatically done by the device controller.

After this the device controller waits for attachment to the USB bus indicated by the VBUS becoming valid. This can be notified to the function either by polling or an interrupt. The time of attachment can be controlled by the function through the pull-up enable/disable bit in the core control register. When disabled the USB host will not notice the device even when it is plugged in.

After attachment an USB reset needs to be received before transactions are allowed to be accepted. This can also notified by polling or an interrupt.

Only control endpoint 0 should be accessible after reset. The function is responsible for enabling and configuring at the right time. It can wait until a USB reset has been received but it is easier to enable it immediately after power-up. This can be done since the device controller will not accept any transactions until USB reset has been received. When enabling the endpoint packet interrupts should be enabled or the function should start polling the buffer status so that it will notice when packets arrive.

Then the endpoint is ready to accept packets and the function should wait for SETUP packets arriving. When a packet arrives the core should process the requests and return USB descriptors as requested. When a Set address request is received the function should write the new address to the device controller's global control register. It will take effect immediately. It should not be written until the status stage has been finished for the Set address request. It can be determined that the request has finished if a packet transmitted interrupt is enabled for the handshake packet of the request and an interrupt is received.

When a Set configuration request is received the function should enable the appropriate interface and endpoints according to the selected configuration. This is done by writing to the various endpoint control registers. The function is responsible for advertising the configurations and interfaces through the descriptors requested by SETUP transactions.

When the endpoints for the selected configuration are enabled the function should also enable interrupts or start polling these endpoints. Then it is ready to transmit and receive data through the application specific endpoints. Interrupts can be used to notify that new packets have been transferred and then status reads will determine which endpoint had a status change.

69.8 Registers

The core is programmed through registers mapped into AHB address space.

*Table 584.*GRUSBDC registers

AHB address offset	**Register**
0x00	OUT Endpoint 0 control register
0x04	OUT Endpoint 0 slave ctrl / DMA ctrl register
0x08	OUT Endpoint 0 slave data / DMA descriptor address register
0x0C	OUT Endpoint 0 status register
0x10-0x1C	OUT Endpoint 1
...	
0xF0-0xFC	OUT Endpoint 15
0x100-0x1FC	IN Endpoints 0-15
0x200	Global Ctrl register
0x204	Global Status register

Table 585. GRUSBDC OUT endpoint control register

31 21	20	19	18	17 7	6 5	4 3	2	1	0
BUFSZ	PI	CB	CS	MAXPL	NT	TT	EH	ED	EV

31: 21	Buffer size (BUFSZ) - Size/8 in bytes of one hardware buffer slot for this endpoint. Two slots are available for each endpoint.
20	Packet received interrupt (PI) - Generate an interrupt for each packet that is received on the USB for this endpoint (packet has been stored in the internal buffers). Reset value: ‘0’.
19	Clear buffers (CB) - Clears any buffers for the endpoint that contain data if the buffer is not currently active.
18	Control Stall (CS) - Return stall for data and status stages in a control transfer. Automatically cleared when the next setup token is received. Only used when the endpoint is configured as a control endpoint.
17: 7	Maximum payload (MAXPL) - Sets the maximum USB payload (maximum size of a single packet sent to/from the endpoint) size for the endpoint. All bits of the field are not always used. The maximum value for the maximum payload is determined with a generic for each endpoint. Not Reset.
6: 5	Number of transactions (NT) - Sets the number of additional transactions per microframe for high-speed endpoints and per frame for full-speed endpoints. Only valid for isochronous endpoints. Not Reset.
4: 3	Transfer type (TT) - Sets the transfer type for the endpoint. “00”=CTRL, “01” =ISOCH, “10”=BULK, “11”=INTERRUPT. Only OUT endpoints should be set to the CTRL type and then the IN endpoint with the same number will be automatically used. It is important not to use OUT endpoints that do not have a corresponding IN endpoint as a CTRL endpoint. Not Reset.
2	Endpoint halted (EH) - Halt the endpoint. If set, all transfers to this endpoint will receive a STALL handshake. Reset value: ‘0’.
1	Endpoint disabled (ED) - Disables the endpoint. If set, all transfers to this endpoint will receive a NAK handshake. Reset value: ‘0’.
0	Endpoint valid (EV) - Enables the endpoint. If not enabled, all transfers to this endpoint will be ignored and no handshake is sent. Reset value; ‘0’.

Table 586. GRUSBDC OUT slave control register.

31 17	16	15 3	2	1	0
RESERVED	SE	BUFCNT	DA	BS	CB

31: 17	RESERVED
16	Setup packet (SE) - The data was received from a SETUP packet instead of an OUT.
15: 3	Buffer counter (BUFCNT) - The number bytes available(OUT)
2	Data available (DA) - Set to one if a valid packet was acquired when requested using the CB. If no valid packet was available it is set to zero. Reset value: ‘0’.
1	Buffer select (BS) - Current buffer selected. Read only.
0	Change or acquire buffer (CB) - If no buffer is currently active try to acquire a new one. If one is already acquired, free it and try to acquire a new one.

Table 587. GRUSBDC OUT slave buffer read register.

31 0
DATA

31: 0	Data (DATA) - In AHB slave mode, data is fetched directly from the internal buffer by reading from this register. Data always starts from bit 31. For word accesses bits 31-0 are valid, for half-word bits 31-16 and for byte accesses bits 31-24.

Table 588. GRUSBDC OUT DMA control register.

31 11	10	9 4	3	2	1	0
RESERVED	AE	RESERVED	AD	AI	IE	DA

31: 11	RESERVED
10	AHB error (AE) - An AHB error has occurred for this endpoint.
9: 4	RESERVED
3	Abort DMA (AD) - Disable descriptor processing (set DA to 0) and abort the current DMA transfer if one is active. Reset value: ‘0’.
2	AHB error interrupt (AI) - Generate interrupt when an AHB error occurs for this endpoint.
1	Interrupt enable (IE) - Enable DMA interrupts. Each time data has been received or transmitted to/ from a descriptor with its interrupt enable bit set an interrupt will be generated when this bit is set.
0	Descriptors available (DA) - Set to indicate to the GRUSBDC that one or more descriptors have been enabled.

Table 589. GRUSBDC OUT descriptor address register.

31 2	1 0
DESCADDR	RES

31: 2	Descriptor table address (DESCADDR) - Address to the next descriptor.Not Reset.
1: 0	RESERVED

Table 590. GRUSBDC OUT endpoint status register

31 30	29	28 16	15 3	2	1	0
RES	PR	B1CNT	B0CNT	B1	B0	BS

31: 30	RESERVED.

Table 590. GRUSBDC OUT endpoint status register

29	Packet received (PR) - Set each time a packet has been received (OUT) and stored in the internal buffers. Cleared when written with a ‘1’.
28: 16	Buffer 1 byte count (B1CNT) - Number of bytes in buffer one.
15: 3	Buffer 0 byte count (B0CNT) - Number of bytes in buffer zero.
2	Buffer 1 data valid (B1) - Set when buffer one contains valid data.
1	Buffer 0 data valid (B0) - Set when buffer zero contains valid data.
0	Buffer select (BS) - The currently selected buffer.

Table 591. GRUSBDC IN endpoint control register

31 21	20	19	18	17 7	6 5	4 3	2	1	0
BUFSZ	PI	CB	CS	MAXPL	NT	TT	EH	ED	EV

31: 21	Buffer size (BUFSZ) - Size/8 in bytes of one hardware buffer slot for this endpoint. Two slots are available for each endpoint.
20	Packet transmitted interrupt (PI) - Generate an interrupt each time a packet has been transmitted on the USB and the internal buffer is cleared. Reset value: ‘0’.
19	Clear buffers (CB) - Clears any buffers for the endpoint that contain data if the buffer is not currently active.
18	Control Stall (CS) - Return stall for data and status stages in a control transfer. Automatically cleared when the next setup token is received. Only used when the endpoint is configured as a control endpoint.
17: 7	Maximum payload (MAXPL) - Sets the maximum USB payload (maximum size of a single packet sent to/from the endpoint) size for the endpoint. All bits of the field are not always used. The maximum value for the maximum payload is determined with a generic for each endpoint. Not Reset.
6: 5	Number of transactions (NT) - Sets the number of additional transactions per microframe for high-speed endpoints and per frame for full-speed endpoints. Only valid for isochronous endpoints. Not Reset.
4: 3	Transfer type (TT) - Sets the transfer type for the endpoint. “00”=CTRL, “01” =ISOCH, “10”=BULK, “11”=INTERRUPT. Only OUT endpoints should be set to the CTRL type and then the IN endpoint with the same number will be automatically used. It is important not to use OUT endpoints that do not have a corresponding IN endpoint as a CTRL endpoint. Not Reset.
2	Endpoint halted (EH) - Halt the endpoint. If set, all transfers to this endpoint will receive a STALL handshake. Reset value: ‘0’.
1	Endpoint disabled (ED) - Disables the endpoint. If set, all transfers to this endpoint will receive a NAK handshake. Reset value: ‘0’.
0	Endpoint valid (EV) - Enables the endpoint. If not enabled, all transfers to this endpoint will be ignored and no handshake is sent. Reset value; ‘0’.

Table 592. GRUSBDC IN slave control register.

31 17	16 4	3	2	1	0
RESERVED	BUFCNT	PI	BA	BS	EB

31: 17	RESERVED
16: 4	Buffer counter (BUFCNT) - The number of bytes written in the current buffer.
3	Packet interrupt enable (PI) - Generate interrupt when the activated packet has been transmitted. Should be set together with EB when enabling a packet for transmission. Reset value: ‘0’.
2	Buffer active (BA) - A free buffer was acquired and is available for use.
1	Buffer select (BS) - Current buffer selected. Read only.
0	Enable (EB) - Enable current buffer for transmission if one has been acquired and try to acquire a new buffer. If no data has been written to the buffer a zero length packet will be transmitted.

Table 593. GRUSBDC IN slave buffer read/write register.

31	0
DATA	

31: 0	Data (DATA) - Data written to this register is placed into the current buffer for transmission. Byte, Halfword and word sizes are allowed but only a word aligned address should be used. This means that data is always placed on 31-0 for word, 31-16 for half-word and 31-24 for byte.

Table 594. GRUSBDC IN DMA control register.

31 11	10	9 4	3	2	1	0
RESERVED	AE	RESERVED	AD	AI	IE	DA

31: 11	RESERVED
10	AHB error (AE) - An AHB has occurred for this endpoint.
9: 4	RESERVED
3	Abort DMA (AD) - Disable descriptor processing (set DA to 0) and abort the current DMA transfer if one is active. Reset value: ‘0’.
2	AHB error interrupt (AI) - Generate interrupt when an AHB error occurs for this endpoint.
1	Interrupt enable (IE) - Enable DMA interrupts. Each time data has been received or transmitted to/from a descriptor with its interrupt enable bit set an interrupt will be generated when this bit is set.
0	Descriptors available (DA) - Set to indicate to the GRUSBDC that one or more descriptors have been enabled.

Table 595. GRUSBDC IN descriptor address register.

31 2	1 0
DESCADDR	RES

31: 2	Descriptor table address (DESCADDR) - Address to the next descriptor.Not Reset.
1: 0	RESERVED

Table 596. GRUSBDC IN endpoint status register

31 30	29	28 16	15 3	2	1	0
RES	PT	B1CNT	B0CNT	B1	B0	BS

31: 30	RESERVED.
29	Packet transmitted (PT) - Packet has been transmitted and cleared from the internal buffers. Cleared when written with a ‘1’.
28: 16	Buffer 1 byte count (B1CNT) - Number of bytes in buffer one.
15: 3	Buffer 0 byte count (B0CNT) - Number of bytes in buffer zero.
2	Buffer 1 data valid (B1) - Set when buffer one contains valid data.
1	Buffer 0 data valid (B0) - Set when buffer zero contains valid data.
0	Buffer select (BS) - The currently selected buffer.

Table 597. GRUSBDC ctrl register

31	30	29	28	27	26	14	13	12	11 9	8	7 1	0
SI	UI	VI	SP	FI	RESERVED	EP	DH	RW	TS	TM	UA	SU

31:	Suspend interrupt (SI) - Generate interrupt when suspend status changes. Reset value: ‘0’.
30	USB reset (UI) - Generate interrupt when USB reset is detected. Reset value: ‘0’.
29	VBUS valid interrupt (VI) - Generate interrupt when VBUS status changes. Reset value: ‘0’.
28	Speed mode interrupt (SP) - Generate interrupt when Speed mode changes. Reset value: ‘0’.
27	Frame number received interrupt (FI) - Generate interrupt when a new Start of frame (SOF) token is received. Reset value: ‘0’.
26: 15	RESERVED
14	Enable pull-up (EP) - Enable pull-up on the D+ line signaling a connect to the host. Reset value: ‘0’.
13	Disable High-speed (DH) - Disable high-speed handshake to make the core full-speed only.
12	Remote wakeup (RW) - Start remote wakeup signaling. It is self clearing and will be cleared when it has finished transmitting remote wakeup if it was currently in suspend mode. If not in suspend mode when set it will self clear immediately. Writes to this bit when it is already asserted are ignored. Reset value: ‘0’.
11: 9	Testmode selector (TS) - Select which testmode to enter. “001”= Test_J, “010”= Test_K, “011”= Test_SE0_NAK, “100”= Test_Packet.
8	Enable test mode (TM) - Set to one to enable test mode. Note that the testmode cannot be left without resetting or power-cycling the core and cannot be entered if hsdis is set to ‘1’. Reset value: ‘0’.
7: 1	USB address (UA) - The address assigned to the device on the USB bus.
0	Set USB address (SU) - Write with a one to set the usb address stored in the USB address field.

Table 598. GRUSBDC status register

31 28	27 24	23	22 18	17	16	15	14	13 11	10 0
NEPI	NEPO	DM	RESERVED	SU	UR	VB	SP	AF	FN

31: 28	Number of implemented IN endpoints (NEPI) - The number of configurable IN endpoints available in the core (including endpoint 0) minus one.
27: 24	Number of implemented OUT endpoints (NEPO) - The number of configurable OUT endpoints available in the core (including endpoint 0) minus one.
23	Data mode (DM) - 0 = core uses slave mode for data transfers, 1 = core uses master mode (DMA) for data transfers.
22: 18	RESERVED
17	Suspended (SU) - Set to ‘0’ when the device is suspended and ‘1’ when not suspended.
16	USB reset (UR) - Set each time an USB reset has been detected. Cleared when written with a ‘1’.
15	Vbus valid (VB) - Set to one when a valid voltage has been detected on the USB vbus line.
14	Speed (SP) - The current speed mode of the USB bus. ‘0’ = high-speed, ‘1’ = full-speed.
13: 11	Additional frames (AF) - Number of additional frames received with the current frame number.
10: 0	Frame number (FN) - The value of the last SOF token received.

69.9 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x021. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

69.10 Configuration options

Table 599 shows the configuration options of the core (VHDL generics). Buffer sizes are given in bytes and determines the sizes of one hardware buffer for the endpoint. Two buffers are available for each endpoint. The buffer size must be equal to or larger than the desired maximum payload size. In the case of high-bandwidth endpoints the buffer size must be equal to or larger than the maximum payload times the number of transactions per (micro) frame.

*Table 599.*Configuration options

Generic	Function	Allowed range	Default
hsindex	AHB slave index.	0 - NAHBSLV-1	0
hirq	AHB interrupt number	0 - NAHBIRQ-1	0
haddr	AHB slave address	-	0
hmask	AHB slave address mask	-	16#FFF#
hmindex	AHB master index	0 - NAHBMST-1	
aiface	0 selects the AHB slave interface for data transfer while 1 selects the AHB master interface.	0 - 1	0
memtech	Memory technology used for blockrams (endpoint buffers).	0 - NTECH	0
uiface	0 selects the UTMI interface while 1 selects ULPI.	0 - 1	0
dwidth	Selects the data path width for UTMI.	8 - 16	8
ninep	Number of IN endpoints	1 - 16	1
noutep	Number of OUT endpoints	1 - 16	1
i0	Buffer size for IN endpoint 0.	8, 16, 24, ... , 3072	1024
i1	Buffer size for IN endpoint 1.	8, 16, 24, ... , 3072	1024
i2	Buffer size for IN endpoint 2.	8, 16, 24, ... , 3072	1024
i3	Buffer size for IN endpoint 3.	8, 16, 24, ... , 3072	1024
i4	Buffer size for IN endpoint 4.	8, 16, 24, ... , 3072	1024
i5	Buffer size for IN endpoint 5.	8, 16, 24, ... , 3072	1024
i6	Buffer size for IN endpoint 6.	8, 16, 24, ... , 3072	1024
i7	Buffer size for IN endpoint 7.	8, 16, 24, ... , 3072	1024
i8	Buffer size for IN endpoint 8.	8, 16, 24, ... , 3072	1024
i9	Buffer size for IN endpoint 9.	8, 16, 24, ... , 3072	1024
i10	Buffer size for IN endpoint 10.	8, 16, 24, ... , 3072	1024
i11	Buffer size for IN endpoint 11.	8, 16, 24, ... , 3072	1024
i12	Buffer size for IN endpoint 12.	8, 16, 24, ... , 3072	1024
i13	Buffer size for IN endpoint 13.	8, 16, 24, ... , 3072	1024
i14	Buffer size for IN endpoint 14.	8, 16, 24, ... , 3072	1024
i15	Buffer size for IN endpoint 15.	8, 16, 24, ... , 3072	1024
o0	Buffer size for OUT endpoint 0.	8, 16, 24, ... , 3072	1024
o1	Buffer size for OUT endpoint 1.	8, 16, 24, ... , 3072	1024
o2	Buffer size for OUT endpoint 2.	8, 16, 24, ... , 3072	1024
o3	Buffer size for OUT endpoint 3.	8, 16, 24, ... , 3072	1024
o4	Buffer size for OUT endpoint 4.	8, 16, 24, ... , 3072	1024
o5	Buffer size for OUT endpoint 5.	8, 16, 24, ... , 3072	1024
o6	Buffer size for OUT endpoint 6.	8, 16, 24, ... , 3072	1024
o7	Buffer size for OUT endpoint 7.	8, 16, 24, ... , 3072	1024
o8	Buffer size for OUT endpoint 8.	8, 16, 24, ... , 3072	1024
o9	Buffer size for OUT endpoint 9.	8, 16, 24, ... , 3072	1024
o10	Buffer size for OUT endpoint 10.	8, 16, 24, ... , 3072	1024
o11	Buffer size for OUT endpoint 11.	8, 16, 24, ... , 3072	1024
o12	Buffer size for OUT endpoint 12.	8, 16, 24, ... , 3072	1024

*Table 599.*Configuration options

Generic	Function	Allowed range	Default
o13	Buffer size for OUT endpoint 13.	8, 16, 24, ... , 3072	1024
o14	Buffer size for OUT endpoint 14.	8, 16, 24, ... , 3072	1024
o15	Buffer size for OUT endpoint 15.	8, 16, 24, ... , 3072	1024
oepol	Select polarity of output enable signal. 1 selects active high and 0 selects active low.	0 - 1	0
syncprst	Use synchronously deasserted rst to PHY. This requires that the PHY generates a clock during reset.	0 - 1	0
prsttime	Reset time in microseconds needed for the PHY. Set to zero to disable this feature and use the reset time enforced by the reset to the GRUSBDC core.	>= 0	0
sysfreq	System frequency (HCLK input) in kHz. This is used together with prsttime to calculate how many clock cycles are needed for the PHY rst.	>= 0	50000
keepclk	This generic determines wheter or not the USB transceiver will be suspended and have its clock turned off during USB suspend. Set this generic to 1 if the clock should not be turned off. This might be needed for some technologies that can’t handle that the USB clock is turned off for long periods of time.	0 - 1	0
sepirq*	Set this generic to 1 if three seperate interrupt lines should be used, one for status related interrupts, one for IN endpoint related interrupts, and one for OUT endpoint related interrupts. The irq number for the three different interrupts are set with the hirq (status), irqi (IN), and irqo (OUT) generics. If sepirq = 0 then only interrupts with irq number hirq will be generated.	0 - 1	0
irqi*	Sets the irq number for IN endpoint related interrupts. Only used if sepirq generic is set to 1.	0 - NAHBIRQ-1	1
irqo*	Sets the irq number for OUT endpoint related interrupts. Only used if sepirq generic is set to 1.	0 - NAHBIRQ-1	2

* The values of these generics are stored in the first User-Defined word of the core’s AHB plug-n-play area as follows: bit 0 = sepirq, bits 7:4 = irqi, bits 11:8 = irqo. Please see the AHBCTRL section of GRLIB IP Core User’s Manual.

69.11 Signal descriptions

Table 600 shows the interface signals of the core (VHDL ports).

*Table 600.*Signal descriptions

Signal name	Field	Type	Function	Active
UCLK	N/A	Input	USB UTMI/ULPI Clock	-
HCLK		Input	AMBA Clock	-
HRST		Input	AMBA Reset	Low
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
USBI	datain[15:0]	Input	UTMI/UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI/UTMI+ mode.	-
	rxactive	Input	UTMI/UTMI+	High
	rxvalid	Input	UTMI/UTMI+	High
	rxvalidh	Input	UTMI/UTMI+ 16-bit	High

*Table 600.*Signal descriptions

Signal name	Field	Type	Function	Active
	rxerror	Input	UTMI/UTMI+	High
	txready	Input	UTMI/UTMI+	High
	linestate[1:0]	Input	UTMI/UTMI+	-
	nxt	Input	ULPI	High
	dir	Input	ULPI	High
	vbusvalid	Input	UTMI+	High
USBO	dataout[15:0]	Output	UTMI/UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI/UTMI+ mode.	-
	txvalid	Output	UTMI+	High
	txvalidh	Output	UTMI+ 16-bit	High
	opmode[1:0]	Output	UTMI+	-
	xcvrselect[1:0]	Output	UTMI/UTMI+. Bit 1 is constant low.	-
	termselect	Output	UTMI/UTMI+	-
	suspendm	Output	UTMI/UTMI+	Low
	reset	Output	Transceiver reset signal. Asserted asynchronously and deasserted synchrnously to the USB clock.	**
	stp	Output	ULPI	High
	oen	Output	Data bus direction control for ULPI and bi-directional UTMI/UTMI+ interfaces.	***
	databus16_8	Output	UTMI+. Constant high for 16-bit interface, constant low for 8-bit interface.	-
	dppulldown	Output	UTMI+. Constant low.	High
	dmpulldown	Output	UTMI+. Constant low.	High
	idpullup	Output	UTMI+. Constant low.	High
	drvvbus	Output	UTMI+. Constant low.	High
	dischrgvbus	Output	UTMI+. Constant low.	High
	chrgvbus	Output	UTMI+. Constant low.	High
	txbitstuffenable	Output	UTMI+. Constant low.	High
	txbitstuffenableh	Output	UTMI+. Constant low.	High
	fslsserialmode	Output	UTMI+. Constant low.	High
	tx_enable_n	Output	UTMI+. Constant high.	Low
	tx_dat	Output	UTMI+. Constant low.	High
	tx_se0	Output	UTMI+. Constant low.	High

* See GRLIB IP Library User’s Manual.

** Depends on transceiver interface. Active high for UTMI/UTMI+ and active low for ULPI.

*** Implementation dependent.

69.12 Library dependencies

Table 601 shows libraries used when instantiating the core (VHDL libraries).

*Table 601.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSB	Signals, component	GRUSBDC component declarations, USB signals

69.13 Instantiation

This example shows how the core can be instantiated.

```
usbdc0: GRUSBDC
    generic map(
      hsindex       => 4,
      hirq          => 0,
      haddr         => 16#001#,
      hmask         => 16#FFF#,
      hmindex       => 14,
      aiface        => 1,
      memtech       => memtech,
      uiface        => 0,
      dwidth        => 8,
      nepi          => 16,
      nepo          => 16)
    port map(
      uclk          => uclk,
      usbi          => usbi,
      usbo          => usbo,
      hclk          => clkm,
      hrst          => rstn,
      ahbmi         => ahbmi,
      ahbmo         => ahbmo(14),
      ahbsi         => ahbsi,
      ahbso         => ahbso(4)
     );

usb_d_pads: for i in 0 to 15 generate
  usb_d_pad: iopad generic map(tech => padtech, slew => 1)
    port map (usb_d(i), usbo.dataout(i), usbo.oen, usbi.datain(i));
end generate;

usb_h_pad:iopad generic map(tech => padtech, slew => 1)
  port map (usb_validh, usbo.txvalidh, usbo.oen, usbi.rxvalidh);

usb_i0_pad : inpad generic map (tech => padtech) port map (usb_txready,usbi.txready);
usb_i1_pad : inpad generic map (tech => padtech) port map (usb_rxvalid,usbi.rxvalid);
usb_i2_pad : inpad generic map (tech => padtech) port map (usb_rxerror,usbi.rxerror);
usb_i3_pad : inpad generic map (tech => padtech) port map (usb_rxactive,usbi.rxactive);
usb_i4_pad : inpad generic map (tech => padtech) port map
(usb_linestate(0),usbi.linestate(0));
usb_i5_pad : inpad generic map (tech => padtech) port map
(usb_linestate(1),usbi.linestate(1));
usb_i6_pad : inpad generic map (tech => padtech) port map (usb_vbus, usbi.vbusvalid);

usb_o0_pad : outpad generic map (tech => padtech, slew => 1) port map (usb_reset,usbo.reset);
usb_o1_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_suspend,usbo.suspendm);
usb_o2_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_termsel,usbo.termselect);
usb_o3_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_xcvrsel,usbo.xcvrselect(0));
usb_o4_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_opmode(0),usbo.opmode(0));
usb_o5_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_opmode(1),usbo.opmode(1));
usb_o6_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_txvalid,usbo.txvalid);

usb_clk_pad : clkpad generic map (tech => padtech, arch => 2) port map (usb_clkout, uclk);
```