WISHBONE DMA/Bridge IP Core

Author: Rudolf Usselmann
rudi@asics.ws

Rev. 1.2
July 27, 2001

Revision History

Rev.	Date	Author	Description
0.1	23/1/01	Rudolf Usselmann	First Draft Internal release
0.2	28/1/01	RU	First public release
0.3	13/3/01	RU	- Removed buffers and all references to buffered transfers - Updated WISHBONE interface signals - Updated Pass-Through mode section - Added Linked List Buffers - Updated registers - Added Bandwidth Allocation Section
0.4	16/3/01	RU	- Added DMA Request and Acknowledge Section - Added Forcing Next Descriptor Section - Added NDn_I signals - Moved the USE_ED bit from COR to Channel CSR register - Added SZ_WB bit to channel CSR register
1.0	00/3/01	RU	- Added Appendix B: Core File Structure - Modified Introduction - Removed “Preliminary Draft” notice
1.01	8/4/01	RU	- Corrected syntax and grammar - Fixed some descriptions - Clarified the linked lists
1.2	6/6/01	RU	- Changed Register Order, major reorganization. - Added Circular Buffer Support (Address Mask Registers). - Added FIFO support in memory (Software pointer Register). - Modified to support up to 31 channels. - Modified to support 2,4 and 8 priority levels. - Filled in Appendix A, Core HW Configuration. - Added Circular Buffers Section. - Added FIFO Buffers Section.

1 Introduction

This core provides DMA transfers between two WISHBONE interfaces. Transfers can also be performed on the same WISHBONE interface. It can also act as a bridge, allowing masters on each WISHBONE interface to directly access slaves on the other interface.

This implementation is designed to work with two WISHBONE interfaces running at the same clock.

The WISBONE specification and additional information about WISHBONE SoC can be found at:

http://www.opencores.org/wishbone/

The Main features of the DMA/Bridge are:

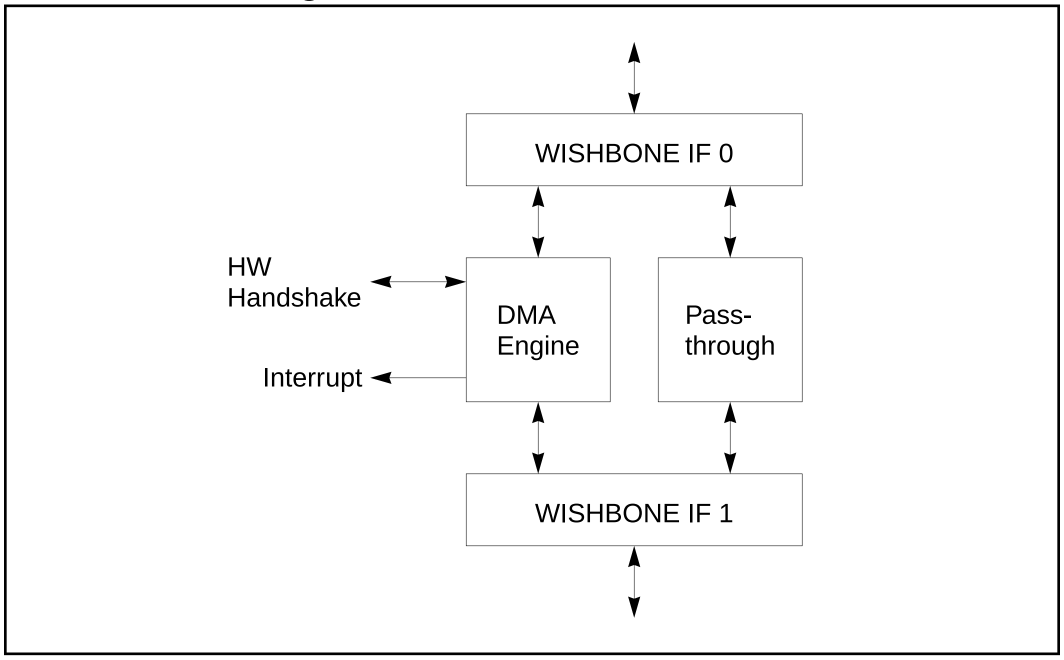
- Up to 31 DMA Channels
- 2, 4 or 8 priority levels
- Linked List Descriptors Support
- Circular Buffer Support
- FIFO buffer support
- Hardware handshake support

(This page intentionally left blank)

2 Architecture

Below figure illustrates the overall architecture of the core.

Figure 1: Core Architecture Overview



It consists of 3 main building blocks: Two WISHBONE interfaces, a DMA engine and pass through logic.

2.1. WISHBONE Interface

The DMA/Bridge core has two master and slave capable WISHBONE interfaces. Both interfaces are WISHBONE SoC bus specification Rev. B compliant.

This implementation implements a 32 bit bus width and does not support other bus widths.

WISHBONE COMPATIBLE

2.2. DMA Engine

The DMA engine is a up to 31 channel DMA engine that supports transfers between the two interfaces as well as transfers on the same interface (block copy). Each channel can be programmed to have a different priority. Channels with the same priority are serviced in a round robin fashion.

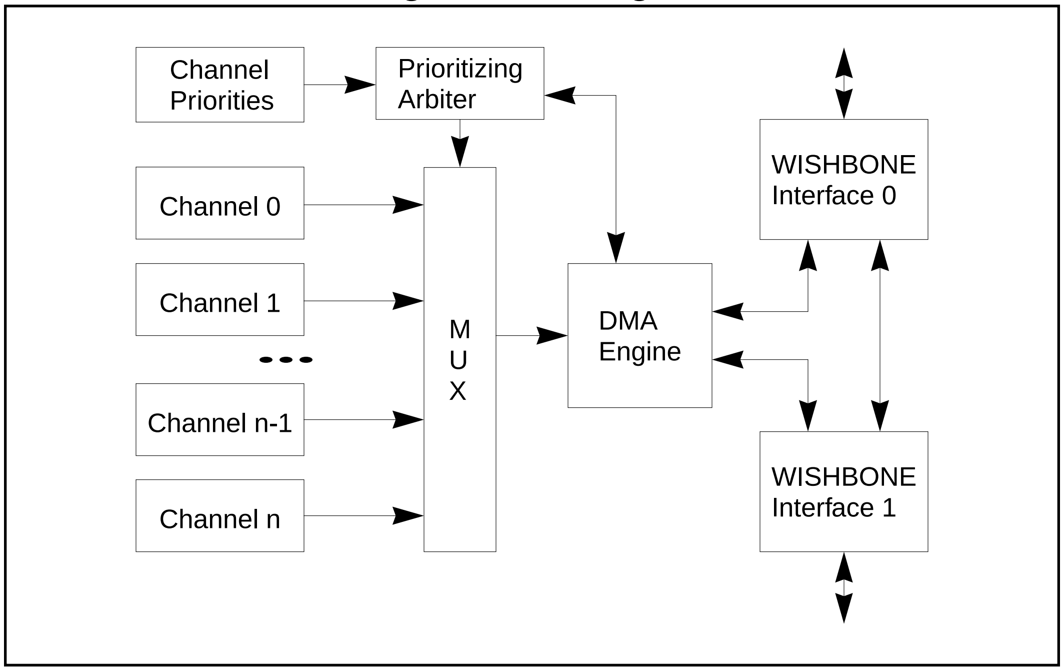
2.3. Pass Through

This block performs the bridging operation between the two WISHBONE interfaces. It includes a two entry deep write buffer in each direction. The write buffer can be disabled if desired.

3 Operation

The WISHBONE DMA/Bridge consists of up to 31 DMA channels, the actual DMA engine, and a channel prioritizing arbiter (see “Figure 2: DMA Engine”).

Figure 2: DMA Engine



3.1. Prioritizing Arbiter

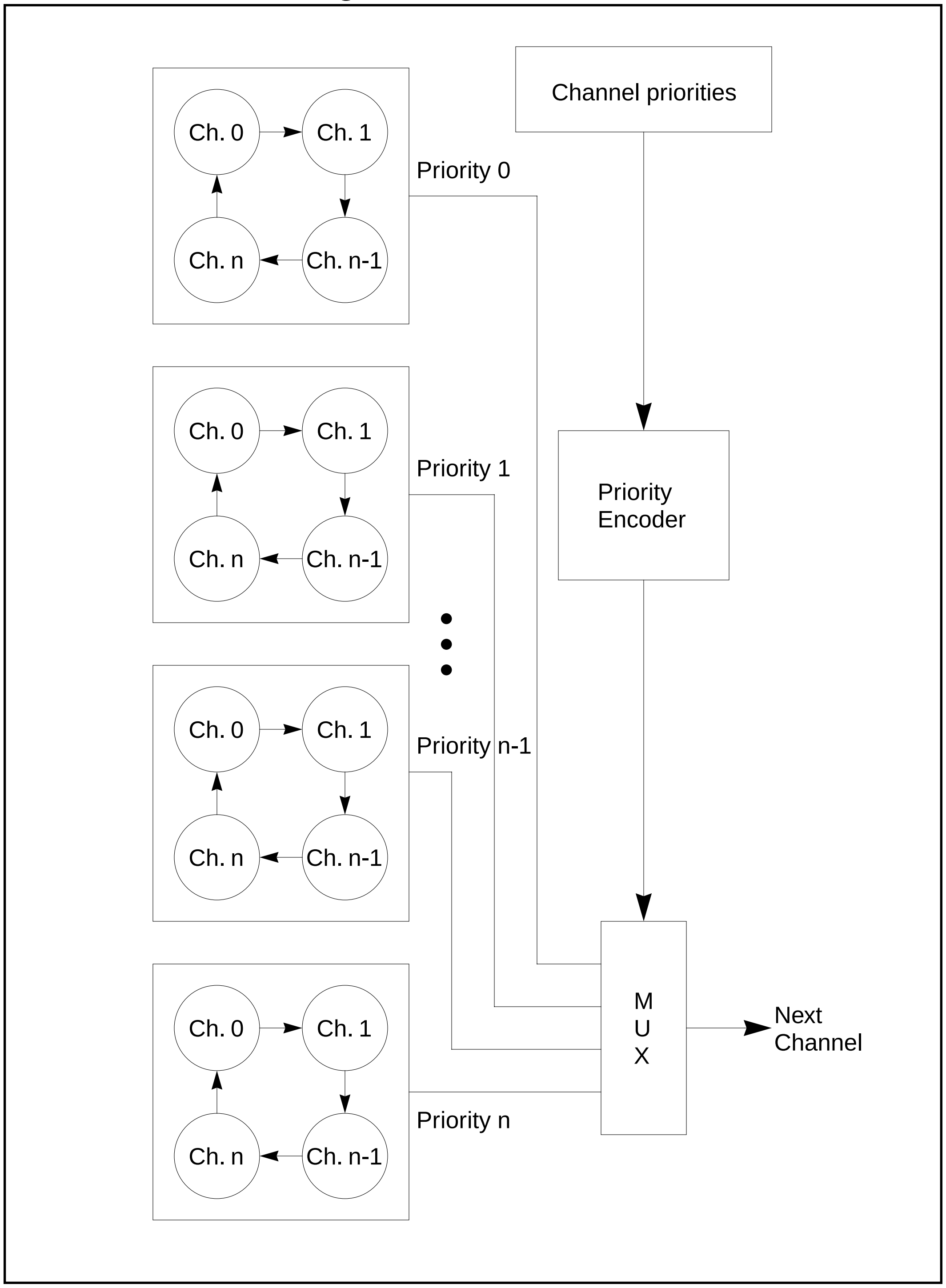
The prioritizing arbiter will select the next channel to process, based first on priority, and secondarily, if all priorities are equal, in a round robin way. Each channel has a 3[1] bit priority value associated with it. A value of 0 identifies a channel with very low priority, a value of 7 identifies a channel with very high priority.

1. Implementation Dependent. This core supports 2, 4 and 8 priority levels. Please see Appendix A “Core HW Configuration” on page 29 for more information.

Channels with the same priority are processed in a round robin way, as long as there are no channels with a higher priority.

"Figure 3: Channel Arbiter" on page 6 illustrates the internal operation of the channel arbiter.

Figure 3: Channel Arbiter



Care should be taken when using priorities, as channels with lower priorities may be locked out and never serviced, if channels with higher priority are being continuously serviced.

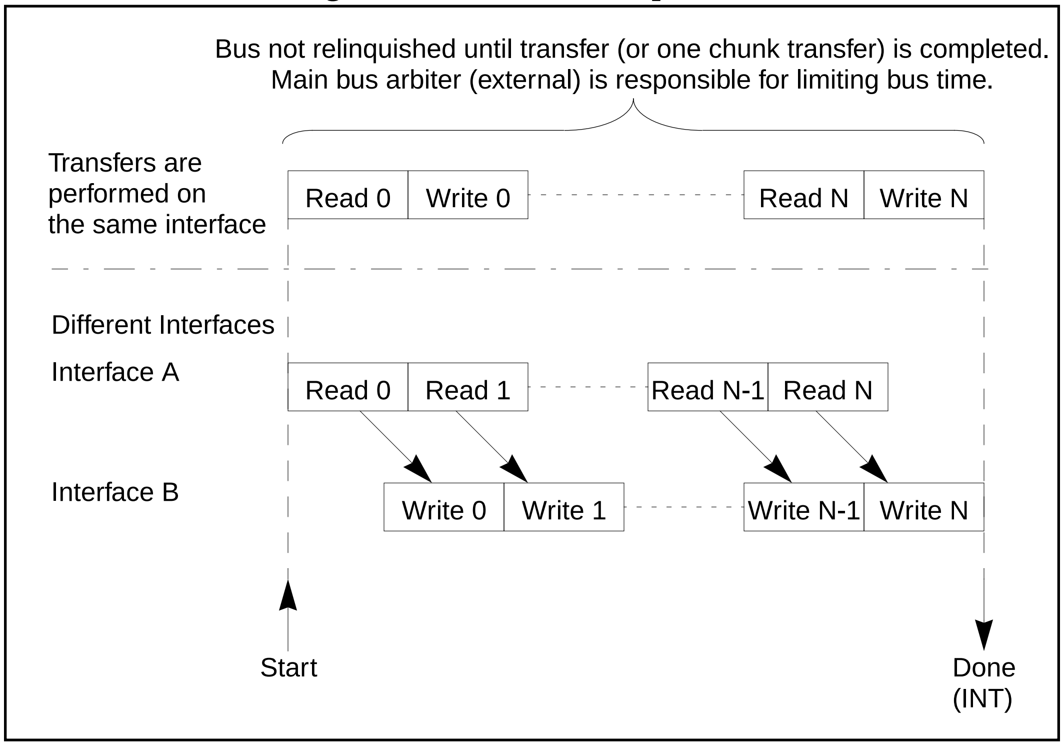
3.2. DMA Engine

The DMA engine can be programmed to perform various transfer operations. This section will illustrate several transfer options and their operation.

3.2.1. Normal (Software) DMA Operation

This is a simple DMA operation performing a block copy. “Figure 4: Normal DMA Operation” illustrates the operation.

Figure 4: Normal DMA Operation


Bus not relinquished until transfer (or one chunk transfer) is completed.
Main bus arbiter (external) is responsible for limiting bus time.
Transfers are performed on the same interface
Read 0
Write 0
Read N
Write N
Different Interfaces
Interface A
Read 0
Read 1
Read N-1
Read N
Interface B
Write 0
Write 1
Write N-1
Write N
Start
Done (INT)

In this example the DMA engine performs a block copy from one location to another, either on the same interface or on a different interface. The DMA engine will leave the CYC_O[1] signal asserted until it has completed the transfer. The transfer begins when either the local controller/CPU writes to the channel CSR register. When the transfer is completed, the DMA engine will assert an interrupt (if enabled) or go to the idle state. If the auto restart bit (ARS) is set, it immediately restarts the operation. When the ARS bit is set, the DMA engine will continue restarting until the ARS bit is cleared in the channel CSR register. The software can also force the channel to stop by writing a one to the STOP bit in the channel

1. CYC_O is a WISHBONE interface signal. See Section 5 “Core IOs” on page 27 for more information.

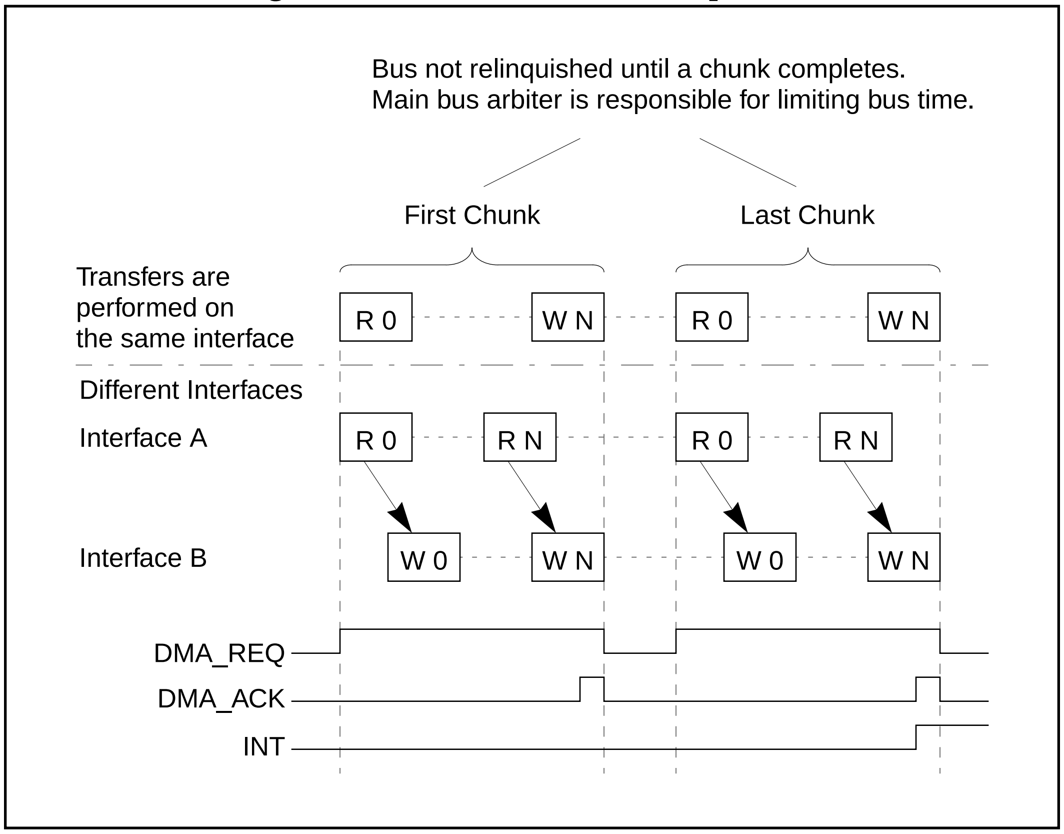
CSR register. In this case the DMA channel will immediately stop and indicate an error condition by setting the ERR bit in the channel CSR register and asserting an error interrupt (if enabled).

If CHK_SZ is not zero, the channel has to re-arbitrate for the interfaces after each CHK_SZ of words has been transferred. This is particularly useful when setting up all channels with the same priority and requiring "fair" bus usage distribution and low latency.

3.2.2. HW Handshake Mode

Below figure illustrates HW handshake DMA operations, where one full DMA transfer requires more than one external trigger.

Figure 5: HW Handshake DMA Operation


Bus not relinquished until a chunk completes.
Main bus arbiter is responsible for limiting bus time.
First Chunk
Last Chunk
Transfers are performed on the same interface
R 0
W N
R 0
W N
Different Interfaces
Interface A
R 0
R N
R 0
R N
Interface B
W 0
W N
W 0
W N
DMA_REQ
DMA_ACK
INT

In this mode the DMA engine will wait for the external trigger (DMA_REQ) to be asserted before starting the DMA transfer. Each time the trigger is asserted it will transfer CHK_SZ number of words (one chunk). After each chunk transfer it will assert DMA_ACK to acknowledge the transfer. After TOT_SZ number of words have been transferred, an interrupt is asserted (if enabled).

After each chunk transfer the DMA channel has to re-arbitrate internally for the usage of the WISHBONE interfaces.

If the ARS bit is set, the DMA channel will reload the values programmed into the channel registers and restart the operation. This loop will continue until the

ARS bit is cleared or the STOP bit is set. When the STOP bit is set, the DMA engine will immediately stop the transfer, set the ERR bit, and assert an error interrupt (if enabled).

3.3. Linked List Descriptors

In this mode the DMA engine will fetch the channel descriptors from memory attached to interface 0. The descriptors are similar to the channel registers, except that after completion a new descriptor may be loaded. The descriptors are provided in a linked list format.

Figure 6: Linked List Descriptor'

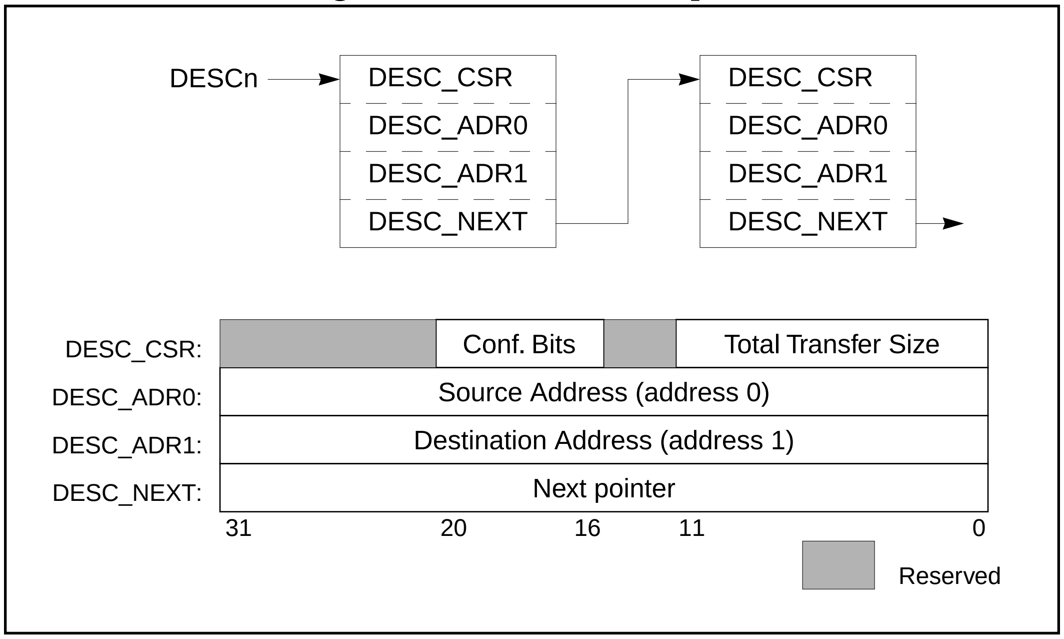


Table 1: Definition of bits in the DESC_CSR word

Bit #	Description
20	EOL: If set, indicates that this is the last descriptor in the list
19	Increment Source Address (same as INC_SRC in CSR)
18	Increment Destination Address (same as INC_DSR in CSR)
17	Source Select (same as SRC_SEL in SCR)
16	Destination Select (same as DST_SEL in CSR)
11-0	Total Transfer Size (same as TOT_SZ in SZ register)

To use external descriptors, the Linked List Descriptor Pointer register for the appropriate channel must be programmed with the address of a valid descriptor. The chunk size must also be set to the desired value in the channel SZ register. Then the USE_ED bit in the channel CSR register must be set to enable external descriptors. After that, the channel enable bit (CH_EN) must be set in the CSR of the channel. Now the DMA engine will start processing descriptors from memory. Normal (Software) and hardware handshake modes are supported with external descriptors. The ARS bit in the channel CSR register has no meaning when using external descriptors and is ignored.

When the DMA engine finishes processing a descriptor, it will attempt to load the next descriptor, pointed to by the DESC_NEXT entry in the descriptor. If the EOL bit in the current DESC_CSR entry is set, the DMA engine will stop, set the DONE bit in the channel CSR register, and assert an interrupt, if enabled.

Note:

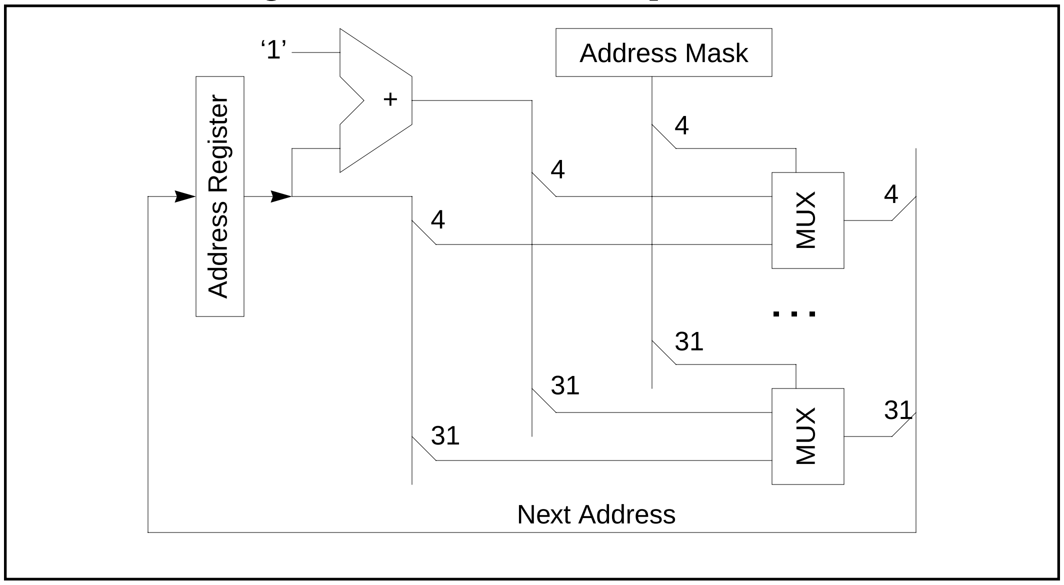
Bits 19-16 in the DESC_CSR register are copied to the channel CSR register bits 4-1.
Bits 11-0 in the DESC_CSR are copied to the channel SZ register bits 11-0.
DESC_ADR0 is copied to channels address 0 register.
DESC_ADR1 is copied to channels address 1 register.
DESC_NEXT is copied to the channels DESC register.

3.4. Circular Buffers

Circular buffers are buffers that will never go beyond the allocated memory space. These buffers will “wrap-around” and start at the beginning f the buffer when they have reached the last entry in the buffer. They are implemented by providing a Mask register for both the source and destination address. This mask is applied to the address when it is incremented. Only bits that are set to ‘1’ in the mask will be incremented.

The lower four bits of the Address Mask are ignored, making the circular buffer at least 4 entries (16 bytes) deep.

Figure 7: Circular Buffers Implementation



3.5. FIFO Buffer Implementation

The DMA engine supports implementing FIFO style buffers in main memory. This is accomplished using circular buffers and using a Software Pointer Register to determine the last location software has read or written.

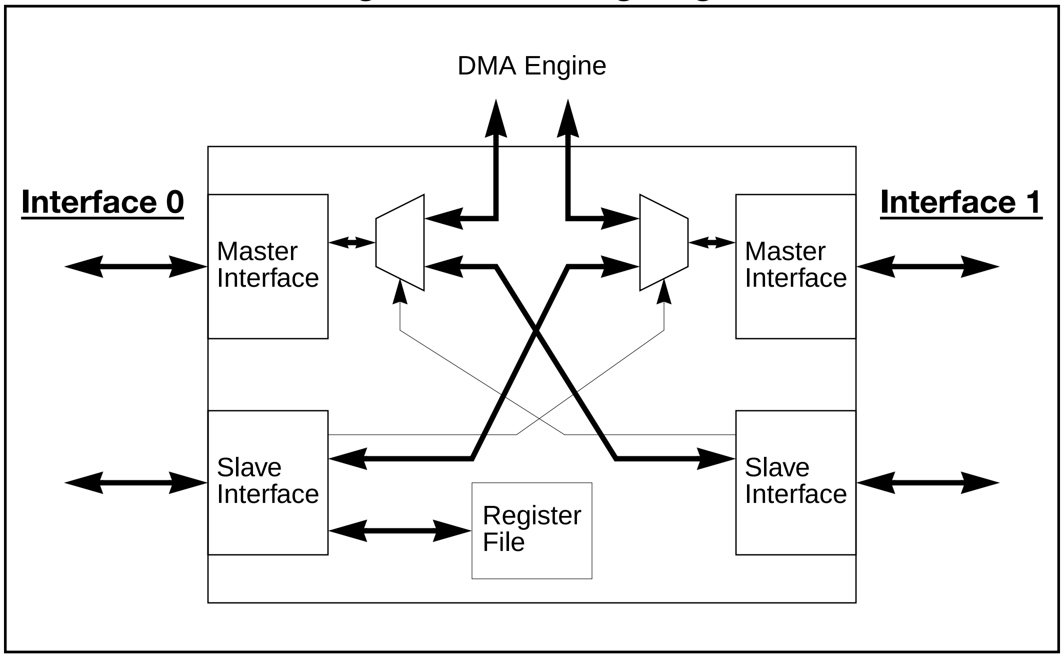
The Software Pointer Register is compared to the current DMA Address, and if they are equal, the DMA will stop processing the channel until the Software pointer is updated. The software is responsible for properly updating the Software Pointer Register.

Software Pointer is always compared to the DMA address that will be placed on the WISHBONE Interface 0.

3.6. Pass Through Operation

In pass through mode, this core acts as a bridge. It does not add any functionality to pass-through traffic. The pass-through logic is combinatorial only (e.g. in pass-through mode signals are not latched). Below figure illustrates the pass-though logic.

Figure 8: Pass Through Logic



3.7. Bandwidth Allocation

The CHK_SZ field can also be used to distribute bandwidth between channels. This is done by setting up all channels with equal priority values. Then the bandwidth for each channel can be calculated as follows:

```
// Calculate the total bandwidth available (100%)
TOT_BW = CH0_CHK_SZ + CH1_CHK_SZ + CH2_CHK_SZ + CH3_CHK_SZ
```

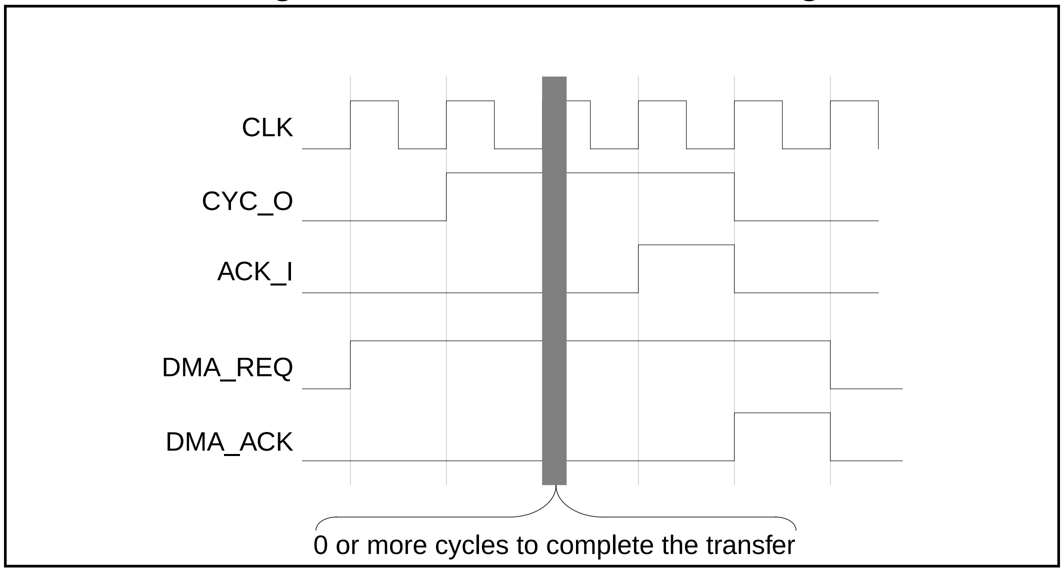
```
CH0_BW = CH0_CHK_SZ/TOT_BW*100 // Channel 0 bandwidth (percent)
CH1_BW = CH1_CHK_SZ/TOT_BW*100 // Channel 1 bandwidth (percent)
CH2_BW = CH2_CHK_SZ/TOT_BW*100 // Channel 2 bandwidth (percent)
CH3_BW = CH3_CHK_SZ/TOT_BW*100 // Channel 3 bandwidth (percent)

Example:
CH0_CHK_SZ = 8
CH1_CHK_SZ = 4
CH2_CHK_SZ = 4
CH3_CHK_SZ = 1
TOT_BW = 8+4+4+1 = 17 (100%)
CH0_BW = 8/17*100 = 47%
CH1_BW = 4/17*100 = 23.5%
CH2_BW = 4/17*100 = 23.5%
CH3_BW = 1/17*100 = 5.8%
```

3.8. DMA Request and Acknowledge (HW Handshake)

In Hardware Handshake mode external request and acknowledge signals are used to start a transfer of a chunk and indicate when the transfer has completed. If CHK_SZ is zero, TOT_SZ number of words will be transferred.

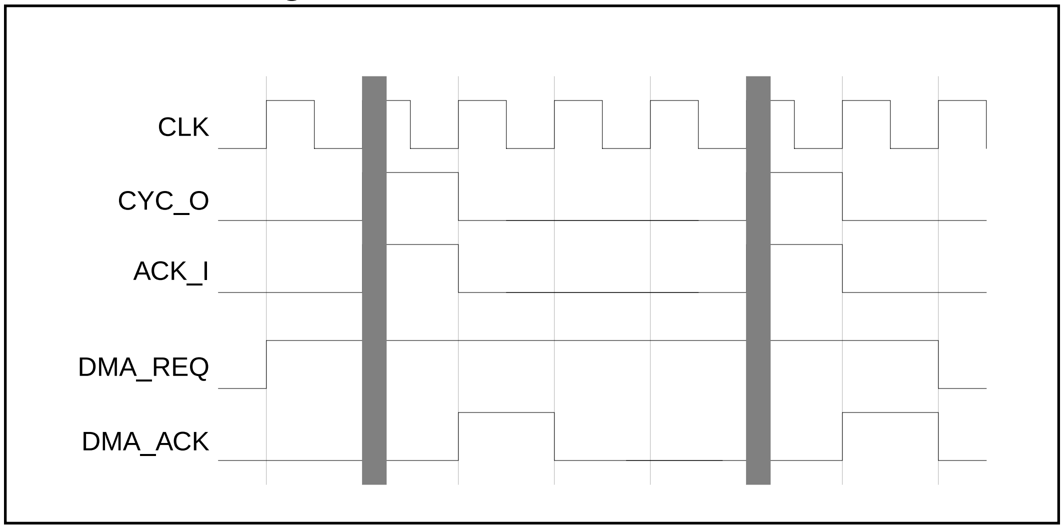
Figure 9: DMA_REQ/DMA_ACK Timing



The DMA_ACK signal will be asserted one cycle after a chunk has been transferred. The chunk size may also be set to one, in which case only one WISHBONE[1] transfer will occur. If DMA_REQ is not de-asserted after DMA_ACK is asserted, another transfer will be initiated, after the channel has re-arbitrated for.

1. For simplicity reasons only a partial WISHBONE signal list is shown.

Figure 10: Back to Back DMA Transfers



3.9. Forcing Next Descriptor

The DMA core provides a special feature that allows a device to force the DMA engine to advance to the next descriptor in a Linked List. This feature is particularly useful to devices that wish to keep a dedicated descriptor for a certain piece of data (e.g one packet payload) but do not know the exact size of the data.

This feature only works with external descriptors in linked lists and hardware handshake mode.

There are two ways to force the next descriptor:

The first way is to assert the NDn_I signal at least two cycles before the DMA_REQ_n signal. In this case the descriptor for the channel will be invalidated and marked as “serviced” and when the DMA_REQ_n is asserted the next descriptor will be fetched from the address pointed to by the current descriptor. If the current descriptor’s EOL bit is set, the DMA channel will stop and clear the enable bit in the channel’s CSR. To start DMA operation on this channel again, software has to reset the DESCn register and the channel CSR register.

The second way is to assert NDn_I together with DMA_REQ_n. It must stay asserted until DMA_ACK_n is asserted by the DMA, at which point NDn_I must be de-asserted. In this case, the DMA will first finish transferring the current chunk size and than invalidate the current descriptor by marking it “serviced”. If the SZ_WB bit is set in the channel CSR register, the DMA will write the total number of remaining bytes to be transferred back to the DESC_CSR in memory. This will allow the software to easily track the actual number of bytes transferred.

(This page intentionally left blank)

4 Core Registers

This section describes all control and status register inside the WISHBONE DMA/Bridge core. The *Address* field indicates a relative address in hexadecimal. *Width* specifies the number of bits in the register, and *Access* specifies the valid access types to that register. RW stands for read and write access, RO for read only access. A 'C' appended to RW or RO indicates that some or all of the bits are cleared after a read.

All RESERVED bits should always be written with zero. Reading RESERVED bits will return undefined values. Software should follow this model to be compatible to future releases of this core.

Table 2: Control/Status Registers

Name	Addr.	Width	Access	Description
CSR	0	32	RW	Main Configuration & Status Register
INT_MSK_A	4	32	RW	Interrupt Mask for INTA_O output
INT_MSK_B	8	32	RW	Interrupt Mask for INTB_O output
INT_SRC_A	c	32	RO	Interrupt Source for INTA_O output
INT_SRC_B	10	32	RO	Interrupt Source for INTB_O output
Channel 0 Registers				
CH0_CSR	20	32	RW	Control Status Register
CH0_SZ	24	32	RW	Transfer Size
CH0_A0	28	32	RW	Address 0
CH0_AM0	2c	32	RW	Address Mask 0
CH0_A1	30	32	RW	Address 1
CH0_AM1	34	32	RW	Address Mask1
CH0_DESC	38	32	RW	Linked List Descriptor Pointer
CH0_SWPTR	3c	32	RW	Software Pointer
Channel 1 Registers				

Table 2: Control/Status Registers

Name	Addr.	Width	Access	Description
CH1_CSR	40	32	RW	Control Status Register
CH1_SZ	44	32	RW	Transfer Size
CH1_A0	48	32	RW	Address 0
CH1_AM0	4c	32	RW	Address Mask 0
CH1_A1	50	32	RW	Address 1
CH1_AM1	54	32	RW	Address Mask1
CH1_DESC	58	32	RW	Linked List Descriptor Pointer
CH1_SWPTR	5c	32	RW	Software Pointer
Channel 2 Registers				
CH2_CSR	60	32	RW	Control Status Register
CH2_SZ	64	32	RW	Transfer Size
CH2_A0	68	32	RW	Address 0
CH2_AM0	6c	32	RW	Address Mask 0
CH2_A1	70	32	RW	Address 1
CH2_AM1	74	32	RW	Address Mask1
CH2_DESC	78	32	RW	Linked List Descriptor Pointer
CH2_SWPTR	7c	32	RW	Software Pointer
Channel 3 Registers				
CH3_CSR	80	32	RW	Control Status Register
CH3_SZ	84	32	RW	Transfer Size
CH3_A0	88	32	RW	Address 0
CH3_AM0	8c	32	RW	Address Mask 0
CH3_A1	90	32	RW	Address 1
CH3_AM1	94	32	RW	Address Mask1
CH3_DESC	98	32	RW	Linked List Descriptor Pointer
CH3_SWPTR	9c	32	RW	Software Pointer
Starting Addr.	a0	**Channel 4 Registers**		
Starting Addr.	c0	**Channel 5 Registers**		
Starting Addr.	e0	**Channel 6 Registers**		
Starting Addr.	100	**Channel 7 Registers**		

Table 2: Control/Status Registers

Name	Addr.	Width	Access	Description
Starting Addr.	120			**Channel 8 Registers**
Starting Addr.	140			**Channel 9 Registers**
Starting Addr.	160			**Channel 10 Registers**
Starting Addr.	180			**Channel 11 Registers**
Starting Addr.	1a0			**Channel 12 Registers**
Starting Addr.	1c0			**Channel 13 Registers**
Starting Addr.	1e0			**Channel 14 Registers**
Starting Addr.	200			**Channel 15 Registers**
Starting Addr.	220			**Channel 16 Registers**
Starting Addr.	240			**Channel 17 Registers**
Starting Addr.	260			**Channel 18 Registers**
Starting Addr.	280			**Channel 19 Registers**
Starting Addr.	2a0			**Channel 20 Registers**
Starting Addr.	2c0			**Channel 21 Registers**
Starting Addr.	2e0			**Channel 22 Registers**
Starting Addr.	300			**Channel 23 Registers**
Starting Addr.	320			**Channel 24 Registers**
Starting Addr.	340			**Channel 25 Registers**
Starting Addr.	360			**Channel 26 Registers**
Starting Addr.	380			**Channel 27 Registers**
Starting Addr.	3a0			**Channel 28 Registers**
Starting Addr.	3c0			**Channel 29 Registers**
Starting Addr.	3e0			**Channel 30 Registers**

4.1. Main Configuration Status Register (CSR)

This is the main configuration register of the DMA/Bridge core.

Table 3: CSR Register

Bit #	Access	Description
31:1	RO	RESERVED

Table 3: CSR Register

Bit #	Access	Description
0	RW	PAUSE Writing a 1 to this register will pause the DMA engine (all channels). Writing a 0 will enable/resume all operations. Reading this bit will return the status of the DMA engine: 1-Paused; 0-Normal Operation. The DMA engine will only pause after it has completed the current transfer.

Value after reset:
COR: 00 h

4.2. Interrupt Mask Register (INT_MSK_n)

The interrupt mask registers define the functionality of the INTA_O and INTB_O outputs. A bit set to a logical one enables the generation of the interrupt for that source, a zero disables the generation of an interrupt. The interrupt mask register INT_MSK_A specifies the behavior for the INTA_O output, the INT_MASK_B register for the INTB_O output.

Table 4: Interrupt Mask Register

Bit #	Access	Description
31	RO	RESERVED
30	RW	Interrupt Enable: Enable DMA Channel 30 Interrupts
29	RW	Interrupt Enable: Enable DMA Channel 29 Interrupts
28	RW	Interrupt Enable: Enable DMA Channel 28 Interrupts
27	RW	Interrupt Enable: Enable DMA Channel 27 Interrupts
26	RW	Interrupt Enable: Enable DMA Channel 26 Interrupts
25	RW	Interrupt Enable: Enable DMA Channel 25 Interrupts
24	RW	Interrupt Enable: Enable DMA Channel 24 Interrupts
23	RW	Interrupt Enable: Enable DMA Channel 23 Interrupts
22	RW	Interrupt Enable: Enable DMA Channel 22 Interrupts
21	RW	Interrupt Enable: Enable DMA Channel 21 Interrupts
20	RW	Interrupt Enable: Enable DMA Channel 20 Interrupts
19	RW	Interrupt Enable: Enable DMA Channel 19 Interrupts
18	RW	Interrupt Enable: Enable DMA Channel 18 Interrupts
17	RW	Interrupt Enable: Enable DMA Channel 17 Interrupts

Table 4: Interrupt Mask Register

Bit #	Access	Description
16	RW	Interrupt Enable: Enable DMA Channel 16 Interrupts
15	RW	Interrupt Enable: Enable DMA Channel 15 Interrupts
14	RW	Interrupt Enable: Enable DMA Channel 14 Interrupts
13	RW	Interrupt Enable: Enable DMA Channel 13 Interrupts
12	RW	Interrupt Enable: Enable DMA Channel 12 Interrupts
11	RW	Interrupt Enable: Enable DMA Channel 11 Interrupts
10	RW	Interrupt Enable: Enable DMA Channel 10 Interrupts
9	RW	Interrupt Enable: Enable DMA Channel 9 Interrupts
8	RW	Interrupt Enable: Enable DMA Channel 8 Interrupts
7	RW	Interrupt Enable: Enable DMA Channel 7 Interrupts
6	RW	Interrupt Enable: Enable DMA Channel 6 Interrupts
5	RW	Interrupt Enable: Enable DMA Channel 5 Interrupts
4	RW	Interrupt Enable: Enable DMA Channel 4 Interrupts
3	RW	Interrupt Enable: Enable DMA Channel 3 Interrupts
2	RW	Interrupt Enable: Enable DMA Channel 2 Interrupts
1	RW	Interrupt Enable: Enable DMA Channel 1 Interrupts
0	RW	Interrupt Enable: Enable DMA Channel 0 Interrupts

Value after reset:

INT_MSK: 0000h

4.3. Interrupt Source Register (INT_SRCn)

This register identifies the source of an interrupt. INT_SRC_A register indicates the source for INTA_O output, INT_SRC_B register indicates the source for INTB_O output. Whenever the function controller receives an interrupt, the interrupt handler must read this register to determine the source and cause of the interrupt. Some of the bits in this register will be cleared after a read. The software interrupt handler must make sure it keeps whatever information is required to handle the interrupt.

Table 5: Interrupt Source Register

Bit #	Access	Description
31	RO	RESERVED

Table 5: Interrupt Source Register

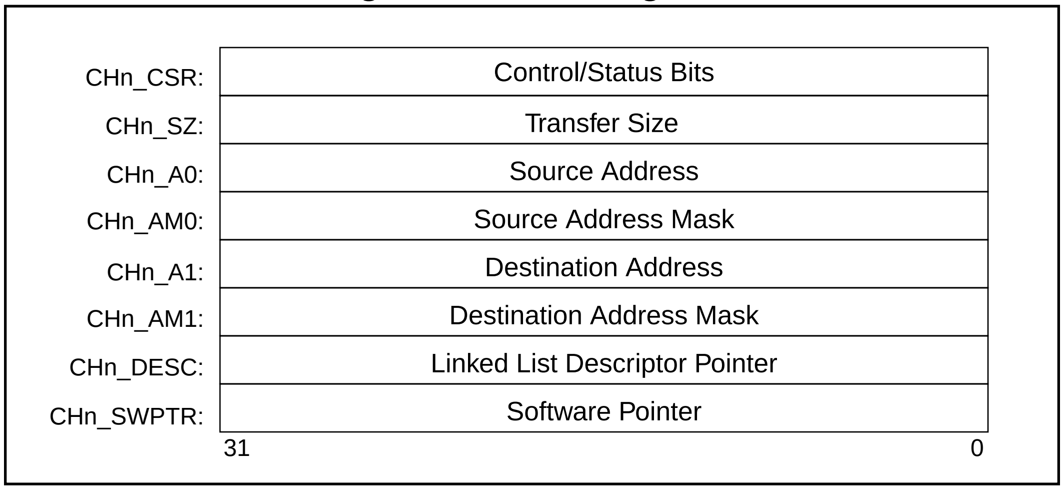
Bit #	Access	Description
30	RW	Interrupt Source: DMA Channel 30
29	RW	Interrupt Source: DMA Channel 29
28	RW	Interrupt Source: DMA Channel 28
27	RW	Interrupt Source: DMA Channel 27
26	RW	Interrupt Source: DMA Channel 26
25	RW	Interrupt Source: DMA Channel 25
24	RW	Interrupt Source: DMA Channel 24
23	RW	Interrupt Source: DMA Channel 23
22	RW	Interrupt Source: DMA Channel 22
21	RW	Interrupt Source: DMA Channel 21
20	RW	Interrupt Source: DMA Channel 20
19	RW	Interrupt Source: DMA Channel 19
18	RW	Interrupt Source: DMA Channel 18
17	RW	Interrupt Source: DMA Channel 17
16	RW	Interrupt Source: DMA Channel 16
15	RW	Interrupt Source: DMA Channel 15
14	RW	Interrupt Source: DMA Channel 14
13	RW	Interrupt Source: DMA Channel 13
12	RW	Interrupt Source: DMA Channel 12
11	RW	Interrupt Source: DMA Channel 11
10	RW	Interrupt Source: DMA Channel 10
9	RW	Interrupt Source: DMA Channel 9
8	RW	Interrupt Source: DMA Channel 8
7	RW	Interrupt Source: DMA Channel 7
6	RW	Interrupt Source: DMA Channel 6
5	RW	Interrupt Source: DMA Channel 5
4	RW	Interrupt Source: DMA Channel 4
3	RW	Interrupt Source: DMA Channel 3
2	RW	Interrupt Source: DMA Channel 2
1	RW	Interrupt Source: DMA Channel 1
0	RW	Interrupt Source: DMA Channel 0

Value after reset:
INT_SRC: 0000h

4.4. Channel Registers

Each channel has 4 registers associated with it. These registers have exactly the same definition for each channel.

Figure 11: Channel Registers



4.4.1. Channel CSR Register (CHn_CSR)

The configuration and status bits specify the operation mode of the channel, as well as reporting any specific channel status.

Table 6: Channel CSR Register

Bit #	Access	Description
31:23	RO	RESERVED
22	ROC	Interrupt Source: Channel transferred CHK_SZ
21	ROC	Interrupt Source: Channel Done
20	ROC	Interrupt Source: Channel Error
19	RO	RESERVED
18	RW	INE_CHK_DONE Enable I Channel Interrupt after each CHK_SZ has been transferred
17	RW	INE_DONE Enable I Channel Interrupt when Channel is Done
16	RW	INE_ERR Enable I Channel Interrupt on Errors

Table 6: Channel CSR Register

Bit #	Access	Description
15:13	RW	Channel Priority (0 Indicating the lowest priority)
12	ROC	ERR DMA channel stopped due to error
11	RO	DONE DMA channel done (This bit will not be set unless the ARS bit is cleared.)
10	RO	BUSY DMA channel busy
9	WO	STOP Writing a one to this bit will cause the DMA to stop its current transfer and set the ERR bit.
8	RW	SZ_WB Enables the writing back of the remaining size to the DESC_CSR when USE_ED is set and NDn_I was asserted with DMA_REQ. See 3.9. “Forcing Next Descriptor” on page 13 for more information.
7	RW	USE_ED Use External Descriptor Linked List
6	RW	ARS Automatically restart the channel when transfer completes 0: Auto restart disabled 1: Automatically restarts the DMA channel after TOT_SZ of bytes have been transferred. The original values programmed into the channel registers are reloaded and the transfer starts al over again.
5	RW	MODE 0: Normal Mode 1: HW Handshake Mode
4	RW	INC_SRC 0: Do not increment source address (Address 0) 1: Increment source address (Address 0)
3	RW	INC_DST 0: Do not increment destination address (Address 1) 1: Increment destination address (Address 1)
2	RW	SRC_SEL 0: Interface 0 is the source 1: Interface 1 is the source
1	RW	DST_SEL 0: Interface 0 is the destination 1: Interface 1 is the destination
0	RW	CH_EN Channel Enabled

Value after reset:

CHn_CSR: 0000h

4.4.2. Channel Size Register (CHn_SZ)

The transfer size register specifies the total and “chunk” transfer sizes for each channel.

Table 7: Channel Size Register

Bit #	Access	Description
31:25	RO	RESERVED
24:16	RW	CHK_SZ Chunk transfer size. Specifies the number of words (4 byte entities) to be transferred at one given time (not implying they are buffered, but that they will be transferred for each start event in one bus request cycle). If chunk size is zero, the DMA engine will always perform TOT_SZ transfers. Maximum chunk size is 2K bytes.
15:12	RO	RESERVED
11:0	RW	TOT_SZ Total Transfer Size. Specifies the number of words (4 byte entities) to be transferred. Maximum total transfer size is16K bytes.

Value after reset:

CHn_SZ: UNDEFINED

4.4.3. Channel Address Registers (CHn_Am)

The Address Registers specify the source and destination address. Address register zero is the source address, address register one is the destination address. Both registers are 30 bits wide.

Table 8: Address Register

Bit #	Access	Description
31:2	RW	Address
1:0	RO	RESERVED

Value after reset:

CHn_Am: UNDEFINED

4.4.4. Channel Address Mask Registers (CHn_AMm)

The Address Mask registers specify the increment mask for the source and destination address. Address Mask Register zero is applied to the source address, Address Mask Register one to the destination address. Both registers are 28 bits wide.

Table 9: Address Mask Register

Bit #	Access	Description
31:4	RW	Address Mask
3:0	RO	RESERVED

Value after reset:

CHn_AMm: FFFFFFFCh

4.4.5. Linked List Descriptor Pointer (CHn_DESC)

The Linked List Descriptor Pointer register specifies the location of the Linked List Descriptor. The value of this register will be overwritten with the Next pointer in the Descriptor, after the descriptor has been fetched.

Table 10: Linked List Descriptor Pointer

Bit #	Access	Description
31:2	RW	Address Mask
1:0	RO	RESERVED

Value after reset:

CHn_DESC: UNDEFINED

4.4.6. Software Pointer (CHn_SWPTR)

The Software Pointer is a register that is written by software and indicates the last location in a circular buffer that has been read/written. The DMA engine will not cross the address pointed to by the Software pointer and stall the channel until the software pointer has been updated. This feature enables the implementation of FIFO buffers in memory.

Table 11: Software Pointer Register

Bit #	Access	Description
31	RW	SWPTR_EN 1 - Enable Software Pointer 0 - Disable Software Pointer
30:2	RW	Software pointer
1:0	RO	RESERVED

Value after reset:

CHn_SWPTR: 0000h

(This page intentionally left blank)

5 Core IOs

5.1. Interface IOs

Both interfaces are WISHBONE Rev. B compliant. The DMA/Bridge core can be a slave or master on either interface. Actual interface 0 signals are prefixed with “WB0_”, interface 1 signals with “WB1_”. Both interfaces comprise of the following signals.

Table 12: Host Interface (WISHBONE)

Name	Width	Direction	Description
ADDR_I	32	I	Address Input (for Slave)
ADDR_O	32	O	Address Output (from Master)
MDATA_I	32	I	Master Interface Data Input
MDATA_O	32	O	Master Interface Data Output
SDATA_I	32	I	Slave Interface Data Input
SDATA_O	32	O	Slave Interface Data Output
SEL_I	4	I	Input for Slave. Indicates which bytes are valid on the data bus. Whenever this signal is not 1111b during a valid access, the ERR_O is asserted.
SEL_O	4	O	Output from Master. Indicates which bytes are valid on the data bus. Whenever this signal is not 1111b during a valid access, the ERR_O is asserted.
WE_I	1	I	Input for Slave. Indicates a Write Cycle when asserted high.
WE_O	1	O	Output from Master. Indicates a Write Cycle when asserted high.
CYC_I	1	I	Input for Slave. Encapsulates a valid transfer cycle.
CYC_O	1	O	Output from Master. Encapsulates a valid transfer cycle.
STB_I	1	I	Input for Slave. Indicates a valid transfer.
STB_O	1	O	Output from Master. Indicates a valid transfer.

Table 12: Host Interface (WISHBONE)

Name	Width	Direction	Description
ACK_O	1	O	Output from Slave. Acknowledgment Output. Indicates a normal Cycle termination.
ACK_I	1	I	Input for Master. Acknowledgment Output. Indicates a normal Cycle termination.
ERR_O	1	O	Output from Slave. Error Acknowledgment Output. Indicates an abnormal cycle termination.
ERR_I	1	I	Input for Master. Error Acknowledgment Output. Indicates an abnormal cycle termination.
RTY_O	1	O	Output from Slave. Retry Output. Indicates that the interface is not ready, and the master should retry this operation.
RTY_I	1	I	Input for Master. Retry Output. Indicates that the interface is not ready, and the master should retry this operation.

5.2. Additional Control IOs

This section describes additional control signals. Except for the clock and reset signals all other signals are special extensions and directly a part of the WISHBONE specification.

Table 13: Additional IOs

Name	Width	Direction	Description
CLK_I	1	I	Clock input
RST_I	1	I	Reset Input
DMA_REQ_I	31	I	DMA Request (trigger input)
DMA_ACK_O	31	O	DMA Acknowledge (Asserted when the DMA is done with the transfer)
ND_I	31	I	Force Next Descriptor advancing
INTA_O	1	O	Interrupt Output A
INTB_O	1	O	Interrupt Output B

Appendix A

Core HW Configuration

This Appendix describes the configuration of the core. This step is performed before final Synthesis and tape-out of the core.

The source file "wb_dma_primitives.v" contains primitive functions that one might want to optimize or replace for a specific technology or implementation.

The source file "wb_dma_defines.v" contains the configuration items described in the following sections.

A.1. Channel Select

This section defines how many channels a given implementation supports. The supported channels must be in sequence. Channel 0 is always present and can not be removed. Any channel that should not be implemented must be commented out.

```
`define     HAVE_CH1    1
`define     HAVE_CH2    1
`define     HAVE_CH3    1
`define     HAVE_CH4    1
//`define   HAVE_CH5    1
//`define   HAVE_CH6    1
//`define   HAVE_CH7    1
//`define   HAVE_CH8    1
//`define   HAVE_CH9    1
//`define   HAVE_CH10   1
//`define   HAVE_CH11   1
//`define   HAVE_CH12   1
//`define   HAVE_CH13   1
//`define   HAVE_CH14   1
//`define   HAVE_CH15   1
//`define   HAVE_CH16   1
//`define   HAVE_CH17   1
//`define   HAVE_CH18   1
//`define   HAVE_CH19   1
//`define   HAVE_CH20   1
//`define   HAVE_CH21   1
//`define   HAVE_CH22   1
//`define   HAVE_CH23   1
//`define   HAVE_CH24   1
//`define   HAVE_CH25   1
//`define   HAVE_CH26   1
```

```
//`define   HAVE_CH27   1
//`define   HAVE_CH28   1
//`define   HAVE_CH29   1
//`define   HAVE_CH30   1
```

In this example only channels 0 through 4 are implemented.

5.3. Auto Reload Support

This section defines which channels support the ARS (auto reload) feature. For each channel that does support the ARS feature, the HAVE_ARSn value must be set to ‘1’. For channels that do not support the ARS feature to ‘0’. Channels that do not support the ARS feature will ignore the ARS bit in the channel CSR register.

```
`define     HAVE_ARS0   1
`define     HAVE_ARS1   1
`define     HAVE_ARS2   0
`define     HAVE_ARS3   0
`define     HAVE_ARS4   1
`define     HAVE_ARS5   1
`define     HAVE_ARS6   1
`define     HAVE_ARS7   1
`define     HAVE_ARS8   1
`define     HAVE_ARS9   1
`define     HAVE_ARS10  1
`define     HAVE_ARS11  1
`define     HAVE_ARS12  1
`define     HAVE_ARS13  1
`define     HAVE_ARS14  1
`define     HAVE_ARS15  1
`define     HAVE_ARS16  1
`define     HAVE_ARS17  1
`define     HAVE_ARS18  1
`define     HAVE_ARS19  1
`define     HAVE_ARS20  1
`define     HAVE_ARS21  1
`define     HAVE_ARS22  1
`define     HAVE_ARS23  1
`define     HAVE_ARS24  1
`define     HAVE_ARS25  1
`define     HAVE_ARS26  1
`define     HAVE_ARS27  1
`define     HAVE_ARS28  1
`define     HAVE_ARS29  1
`define     HAVE_ARS30  1
```

In this example Channel 2 and 3 do not support the ARS feature.

5.4. Linked List Descriptors Support

This section defines which channels support the Linked List Descriptors. For each channel that does support the Linked List Descriptors feature, the HAVE_EDn value must be set to ‘1’. For channels that do not support the Linked List Descriptors feature to ‘0’. Channels that do not support Linked List Descriptors will ignore the USE_ED bit in the channel CSR register and will not have the Linked List Descriptor Pointer register.

```
`define     HAVE_ED0    1
`define     HAVE_ED1    1
`define     HAVE_ED2    1
`define     HAVE_ED3    1
`define     HAVE_ED4    0
`define     HAVE_ED5    0
`define     HAVE_ED6    1
`define     HAVE_ED7    1
`define     HAVE_ED8    1
`define     HAVE_ED9    1
`define     HAVE_ED10   1
`define     HAVE_ED11   1
`define     HAVE_ED12   1
`define     HAVE_ED13   1
`define     HAVE_ED14   1
`define     HAVE_ED15   1
`define     HAVE_ED16   1
`define     HAVE_ED17   1
`define     HAVE_ED18   1
`define     HAVE_ED19   1
`define     HAVE_ED20   1
`define     HAVE_ED21   1
`define     HAVE_ED22   1
`define     HAVE_ED23   1
`define     HAVE_ED24   1
`define     HAVE_ED25   1
`define     HAVE_ED26   1
`define     HAVE_ED27   1
`define     HAVE_ED28   1
`define     HAVE_ED29   1
`define     HAVE_ED30   1
```

In this example Channel 4 and 5 do not support the Linked List Descriptors feature.

A.2. Circular Buffer Support

This section defines which channels support Circular Buffers. For each channel that does supports Circular Buffers, the HAVE_CBUFn value must be set to ‘1’. For channels that do not support Circular Buffers to ‘0’. Channels that do not support circular buffers will not have the Address Mask Registers (they will be forced to all ‘1’ internally) and will also not have the Software pointer register.

```
`define     HAVE_CBUF0  1
`define     HAVE_CBUF1  0
`define     HAVE_CBUF2  0
`define     HAVE_CBUF3  1
`define     HAVE_CBUF4  1
`define     HAVE_CBUF5  1
`define     HAVE_CBUF6  1
`define     HAVE_CBUF7  1
`define     HAVE_CBUF8  1
```

```
`define     HAVE_CBUF9  1
`define     HAVE_CBUF10 1
`define     HAVE_CBUF11 1
`define     HAVE_CBUF12 1
`define     HAVE_CBUF13 1
`define     HAVE_CBUF14 1
`define     HAVE_CBUF15 1
`define     HAVE_CBUF16 1
`define     HAVE_CBUF17 1
`define     HAVE_CBUF18 1
`define     HAVE_CBUF19 1
`define     HAVE_CBUF20 1
`define     HAVE_CBUF21 1
`define     HAVE_CBUF22 1
`define     HAVE_CBUF23 1
`define     HAVE_CBUF24 1
`define     HAVE_CBUF25 1
`define     HAVE_CBUF26 1
`define     HAVE_CBUF27 1
`define     HAVE_CBUF28 1
`define     HAVE_CBUF29 1
`define     HAVE_CBUF30 1
```

In this example Channel 1 and 2 do not support Circular Buffers.

A.3. Priority Levels Select

The two define statements below select the number of priorities that the DMA engine supports.

If PRI_8 is defined, 8 levels of priorities are supported. If PRI_4 is defined then 4 levels of priorities are supported. If neither is defined then two levels of priorities are supported. PRI_4 and PRI_8 should never be both defined at the same time.

```
//`define   PRI_8       1
`define     PRI_4       1
```

A.4. Register File Base Address Select

This define selects how the slave interface determines if the internal register file or pass through mode are selected.

This should be a simple address decoder. “wb_addr_i” is the WISHBONE address bus (32 bits wide).

```
`define     REG_SEL     (wb_addr_i[31:24] == 8'hff)
```

Note:

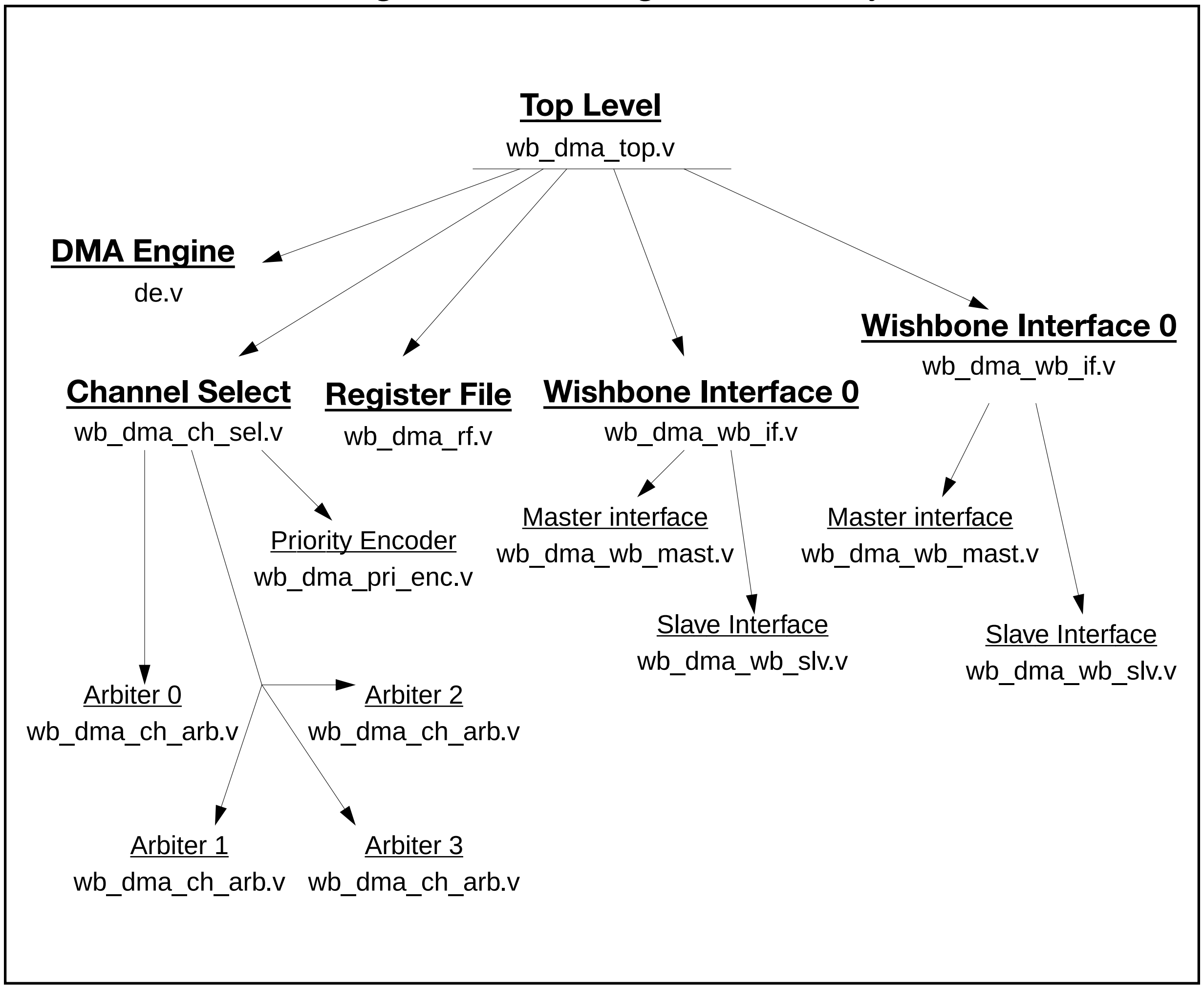
The entire pass-through mode is implemented in combinatorial logic only. So the more address lines we look at and compare here the higher will be the initial delay when pass-through mode is selected. Here we look at the top 8 address bit. If they are all 1, the register file is selected. Use this with caution!!!

Appendix B

File Structure

This section outlines the hierarchy structure of the WISHBONE DMA/Bridge core Verilog Source files.

Figure 12: DMA/Bridge Core Hierarchy Structure


Top Level
wb_dma_top.v
DMA Engine
de.v
Channel Select
wb_dma_ch_sel.v
Register File
wb_dma_rf.v
Wishbone Interface 0
wb_dma_wb_if.v
Wishbone Interface 0
wb_dma_wb_if.v
Priority Encoder
wb_dma_pri_enc.v
Master interface
wb_dma_wb_mast.v
Master interface
wb_dma_wb_mast.v
Slave Interface
wb_dma_wb_slv.v
Slave Interface
wb_dma_wb_slv.v
Arbiter 0
wb_dma_ch_arb.v
Arbiter 2
wb_dma_ch_arb.v
Arbiter 1
wb_dma_ch_arb.v
Arbiter 3
wb_dma_ch_arb.v