Specification of Simple DMA Controller with AMBA AXI

Version 0 Revision 0
July 12, 2015 (July 12, 2015)

Dynalith Systems
(http://www.dynalith.com)

335 Gwahang-No (373-1 Kusong-Dong), 3rd Fl. CHiPS B/D, KAIST,
Yusong-Gu, Daejeon 305-701, Korea

Copyright notice

1 Introduction

A simple DMA (Direct Memory Access) controller (dma_axi_simple) provides a means of data movement between two memory spaces without intervention of processing core. The dma_axi_simple has two AMBA AXI bus ports as shown in Figure 1; one is AXI slave and the other is AXI master port. DMA internal resources, i.e., CSR (Control and Status Register) are accessed through the AXI slave port and data movement is carried out through AXI master port.

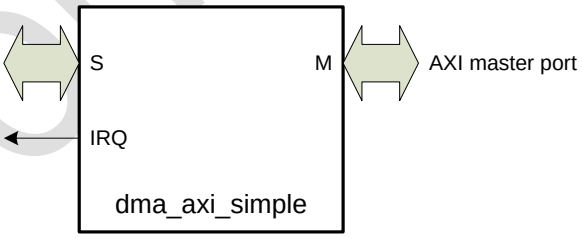

Figure 1: dma_ahb_simple block

There are some highlights as follows.

- AMBA AXI 3 and AXI 4 are supported
- Interrupt signal when data movement completes
- Up to 2^{16}-1 bytes can be moved

There are some limitations as follows.

- Starting addresses of source and destination should be the same in terms of modulus (4 x burst length)

2 IO ports

The ahb_dma supports AMBA AHB 2.0 compliant interface. The AHB slave port is used to access DMA internal resources. The AHB master port A and B are used for DMA data movement.

group	port	width	direction	description
common	HRESETn	1	Input	Asynchronous reset
	HCLK	1	Input	clock
	IRQ	1	Output	Active-high interrupt
AHB slave port	S_HSEL	1	Input	
	S_HADDR	32	Input	
	S_HTRANS	2	Input	
	S_HWRITE	1	Input	
	S_HSIZE	3	Input	
	S_HBURST	3	Input	
	S_HWDATA	32	Input	
	S_HRDATA	32	Output	
	S_HRESP	2	Output	
	S_HREADYin	1	Input	
	S_HREADYout	1	Output	
AHB master port	M_HBUSREQ	1	Output	Arbitration request
	M_HGRANT	1	Input	Arbitration grant
	M_HADDR	32	Output	
	M_HTRANS	2	Output	
	M_HPROT	4	Output	
	M_HWRITE	1	Output	
	M_HSIZE	3	Output	
	M_HBURST	3	Output	
	M_HWDATA	32	Output	
	M_HRDATA	32	Input	
	M_HRESP	2	Input	
	M_HREADY	1	Input	

3 Control and Status Registers

Name	Address offset	Bit#		description
NAME0	+000h		RO	DMA
NAME1	+004h		RO	AXI
NAME2	+008h		RO	
NAME3	+00Ch		RO	
COMP0	+010h		RO	DYNA
COMP1	+014h		RO	LITH
COMP2	+018h		RO	
COMP3	+01Ch		RO	
VERSION	+020h		RO	Version (0x2015_0712)

RESERVED	+024h			Reserved
	+028h			Reserved
	+02Ch			Reserved
CONTROL	+030h		RW	CONTROL register (default: 0x0000_0000)
			31	EN: enable
			30:2	Reserved
			1	IP: 1 when interrupt is pending
			0	IE: interrupt is enabled when 1
	+038h			Reserved
			31:0	
	+03Ch			Reserved
			31:0	
NUM	+040h			NUM register (default: 0x0001_0000)
			31	GO: start DMA when 1 and return 0 when completed
			30	BUSY (read-only)
			29	DONE (read-only)
			28:24	Reserved
			23:16	CHUNK: num of bytes for a chunk - It should be a multiple of data-bus width. - Data-bus width is used when it is 0.
			15:0	BYTES : num of bytes to move
SOURCE	+044h			SRC register (default: 0x0000_0000)
			31:0	source address
DESTINATION	+048h			DST register (default: 0x0000_0000)
			31:0	destination address

4 Operation

When ‘EN’ bit of ‘Control’ register and ‘GO’ bit of the ‘Num’ register are set to ‘1’, the DMA starts data movement according to other register value. On completion of data movement, ‘IP’ bit of ‘Control’ register is set to ‘1’ if ‘IE’ bit of ‘Control’ register is ‘1’.

Partial accesses are used when starting address is misaligned, i.e., 1, 2, 3 in terms of modulus 4.

5 API

All API returns ‘0’ when completes successfully. Otherwise, it returns non-zero value.

```
int dma_ahb_control( uint32_t *cnt );
```

- It returns the contents of control registerl

```
int dma_ahb_enable ( int en, int ie );
```

- en: 1 to enable DMA or 0 to disable DMA
- ie: 1 to enable interrupt or 0 to disable interrupt

```
int dma_ahb_clear  ( void );
```

- interrupt clear

```
int dma_ahb_go     ( uint32_t dst
                   , uint32_t src
                   , uint32_t bnum
                   , uint32_t chunk
                   , int time_out // 0 for blocking
                   );
```

- dst: destination address
- src: source address
- bnum: number of bytes to move from source to destination
- burst: burst length (e.g., 1, 4, 8, 16)
- time_out: 0 for blocking until a whole DMA completes

5.1 Typical usage

Following code shows a typical usage of API.

```
#include “dma_ahb_api.h”

int main() {
    dma_ahb_enable(1, 0);
    dma_ahb_go( ADDR_DSP_IF_START+0x80 //uint32_t dst
              , ADDR_DSP_IF_START      //uint32_t src
              , 0x80 // uint32_t bnum
              , 8 // uint32_t chunk
              , 0 // int time_out // 0 for blocking
              );
}
```

Wish list

- Independent source and destination address

References

[1] AMBA Specification, Rev. 3.0, ARM, www.arm.com.

Revision history

- 2015.07.12: Version 0 Revision 0 is prepared by Ando Ki.