AMBA AXI Memory-Mapped to Stream with Direct Memory Access

Version 0 Revision 0

April 4, 2019 (April 4, 2019)

Future Design Systems, Inc.
www.future-ds.com / contact@future-ds.com

Abstract

This document describes specifications AMBA AXI Memory-Mapped to Stream Controller, which utilizes Direct Memory Access.

Table of Contents

1 Overview

'axi_mem2stream' block reads data through AMBA AXI protocol and then pushes the data through AMBA AXI Stream protocol.

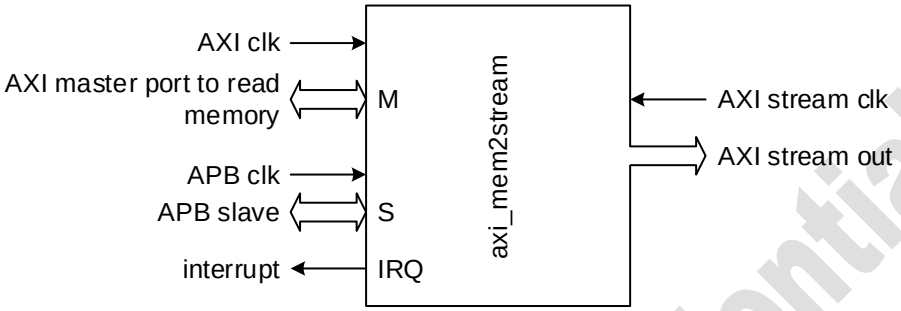

Figure 1: Overview

This controller (axi_mem2stream) provides a means of data movement from memory-mapped memory to stream without intervention of processing core. This controller has following ports as shown in Figure 1.

- AXI master port reads data,
- APB slave port refers to internal registers, and
- AXI stream port pushes data.

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
- Single and continuous modes of movement

There are some limitations as follows.

- Starting addresses of source should be a multiple of stream data width

2 Macros, Parameters and Control-Status Registers

2.1 Parameters

Parameter	Meaning	Default
AXI_MST_ID	Master ID	1
AXI_WIDTH_CID	Bit-width of AXI channel ID It carries AXI_MST_ID	4
AXI_WIDTH_ID	Bit-width of AXI transaction ID	4
AIX_WIDTH_AD	Bit-width of AXI address (AxADDR)	32
AXI_WIDTH_DA	Bit-width of AXI data (xDATA)	32

AXI_WIDTH_DS	Bit-width of AXI data strobe	4
AXI_WIDTH_SID	Bit-width of AXI ID for slave It carries channel ID and transaction ID	8
AXIS_WIDTH_DATA	Bit-width of stream data (AXIS_TDATA)	32
AXIS_WIDTH_DS	Bit-width of stream data strobe (AXIS_TSTRB)	4
APB_AW	Bit-width of APB address (PADDR)	32
APB_DW	Bit-width of APB data (PxDATA)	32
APB_DS	Bit-width of APB data strobe (PSTRB)	4

2.2 CSR

Name	Address offset		Bit#	description
VERSION	+00h		RO	Version (0x2019_0405)
RESERVED	+04h			Reserved
	+08h			Reserved
	+0Ch			Reserved
CONTROL	+10h		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
	+14h			Reserved
	+18h			Reserved
	+1Ch			Reserved
START0	+20h		RW	
			31:0	Staring address [31:0] inclusive
START1	+24h		RW	
			31:0	Starting address [63:32] inclusive
END0	+28h		RW	
			31:0	Ending address [31:0] exclusive
END1	+2Ch		RW	
			31:0	Ending address [63:32] exclusive
NUM	+30h			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	CONT - continuous mode when 1 * move data from STAT to END repeatedly. * It can be stop by setting ‘GO’ 0 and it stops at the boundary of packet. - single mode when 0 * only move data from START to END

				once
			27:24	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. - It should be a multiple of bit-width of stream data bus.
			15:0	BYTES : num of bytes to move -The number of bytes of a packet. -It should be a multiple of bit-width of stream data bus. -It should be a multiple of chunk size.
	+34h			
	+38h			
	+3Ch			
COUNT	+40h		RW	
			31:0	COUNT: the number of movements for continuous mode. ● 0 means infinite

3 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 values. On completion of data movement, ‘IP’ bit of ‘Control’ register is set to ‘1’ if ‘IE’ bit of ‘Control’ register is ‘1’.

As shown in Figure 2, a region of memory is divided into packets and each packet is further divided into chunks, where a chunk corresponds to an AXI burst. It should be noted that ‘packet’ and ‘chunk’ are in byte number. Each packet is transferred over AXI stream consisting of a series of transfers shares the same TID and TDEST and ends with TLAST.

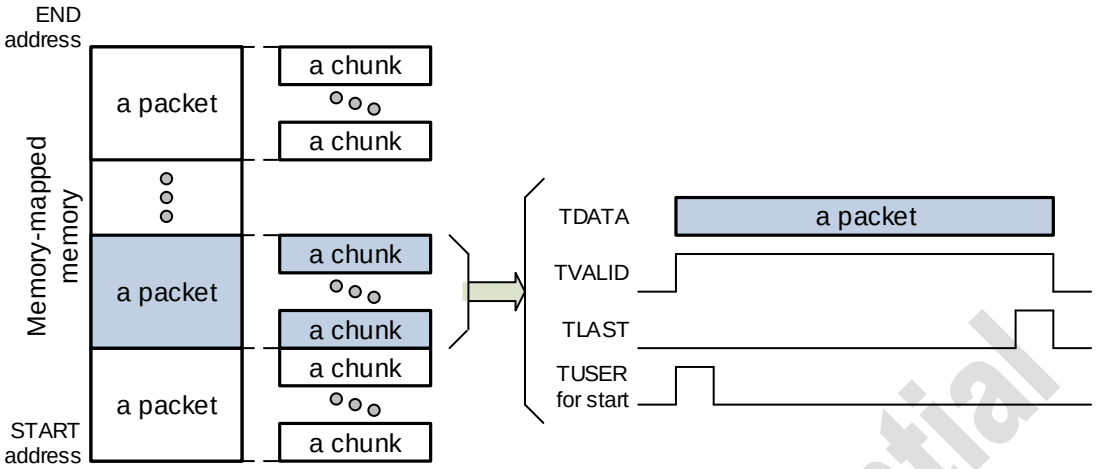

Figure 2: Data movement concept

AXI stream signals:

- AXIS_TREADY & AXIS_TVALID: dual-ready handshake signals
- AXIS_TDATA[…]: byte-wise data
- AXIS_TSTRB[…]: optional byte-wise strobe
- AXIS_TLAST: driven at the last cycle of packet
- AXIS_TSTART: Optionally driven at the first cycle of packet.

4 API

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

```
int axi_mem2stream_set( uint32_t start
                      , uint32_t frame
                      , uint16_t packet
                      , uint8_t  chunk
                      , uint32_t cnum
                      , int cont
                      , int go
                      , int time_out)
{
   volatile uint32_t end, value;
   end = start + frame;
   REGWR(CSRA_M2S_START0, start);
   REGWR(CSRA_M2S_END0  , end  );
   REGWR(CSRA_M2S_CNT   , cnum );
   value = 0;
   if (go  ) value |= M2S_num_go_MSK;
   if (cont) value |= M2S_num_cont_MSK;
   value |= (chunk<<M2S_num_chunk)&M2S_num_chunk_MSK;
   value |= (packet<<M2S_num_bnum)&M2S_num_bnum_MSK;
   REGWR(CSRA_M2S_NUM, value);
```

```
    if (cnum==0) return 0;
    int num=0;
    do { REGRD(CSRA_M2S_NUM, value);
         if (!(value&M2S_num_go_MSK)) break;
         num++;
    } while ((time_out==0)||(num<time_out));
    if ((time_out!=0)&&(num>=time_out)) return -1;
    return 0;
}
```

- start: starting address of frame
- frame: number of bytes of frame
- packet: number of bytes of packet (each packet starts with AXIS_TSTART and ends with AXIS_TLAST)
- chunk: number of bytes as a burst length (note that burst length is not byte number)
- cnum: number of movement for continuous mode
  - ✧ 0 mean infinite (continuously move all data of frame repeatedly)
  - ✧ 1 means a single transfer (i.e., move all data of frame once)
  - ✧ >1 means the number of movement
- cont: mode of transfer
  - ✧ 0: single
  - ✧ 1: continuous (related to ‘cnum’)
- go: 1 for start and 0 for nothing
- time_out: 0 for blocking to complete all transfers

4.1 Typical usage

Following code shows a typical usage of API to move ‘frame’ bytes once.

```
#include “axi_mem2stream_api.h”

int main() {
    … …
    // fill ‘frame’ bytes into the memory starting ‘start’
    axi_mem2stream_set(start,
                       frame, // frame
                       frame/2, // packet
                      4*16, // chunk
                     1, // cnum
                     0, // cont
                     1, // go
                     0); // time out
    … …

}
```

Following code shows a typical usage of API to move ‘frame’ bytes continuously.

```
#include “axi_mem2stream_api.h”
```

```
int main() {
    … …
    // fill ‘frame’ bytes into the memory starting ‘start’
    axi_mem2stream_set(start,
                                frame, // frame
                                frame/2, // packet
                               4*16, // chunk
                               0, // cnum
                               1, // cont
                               1, // go
                               1); // time out
    … …

}
```

References

[1] AMBA Specification, Rev. 3.0, ARM.
[2] AMBA 4 AXI4-Stream Protocol Version: 1.0 Specification, IHI 0051A (ID030510), ARM Limited, 2010.
[3] Future Design Systems, AMBA AXI Stream to Memory-Mapped with Direct Memory Access, FDS-TD-2019-04-003, 2019.

Revision history

☐ 2019.04.04: Prepared by Ando Ki.

– End of document –