Direct Memory Access (DMA) Controller

Design Document V1.0

Course material in ECE4278, SKKU

Author: Jungrae Kim <dale40@skku.edu)

1 Overview

This document specifies the design and implementation of a Direct Memory Access Controller (DMAC) as a part of System-on-a-Chip (SoC). The main purpose of this DMAC design is to integrate into SoC for exchange a large volume of data between memory and peripherals at high speed. The proposed DMAC works on ARM's Advanced Microcontroller Bus Architecture (AMBA) specification. The DMAC provides an AMBA APB interface to configure the IP, and an AMBA AXI interface to transfer data.

2 Architecture Specification

2.1 General Description

Some applications require transferring a volume of data between memory and peripherals without any modification on data. In software, it is commonly served by executing the `memcpy` library function in C, C++ or other languages. In C, the function has the following interface and copies `len` bytes from the object pointed by `src` to the object pointed by `dst`: `void* memcpy(void* dst, const void* src, size_t len)`.

While a pure software-based implementation of `memcpy` transfers data using CPU instructions, DMA does not use expensive CPU cycles but uses a hardware engine (DMAC) for the transfer. This can significantly speed up data transfers and allows using CPU for other jobs.

2.2 Usage Constraints

Below describe constraints in utilizing DMAC v1.

- The `src` and `dst` addresses are physical addresses.

- The `src` and `dst` addresses must be a multiple of 4.
- The `len` must be a multiple of 4.
- The maximum `len` is 0xFFFF
- Source and destination ranges must not overlap.

2.3 Programming Model

Software can use the following sequence to transfer data using DMAC.

1. Write the source address to DMA_SRC register
2. Write the destination address to DMA_DST register
3. Write length to DMA_LEN register
4. Write 1 to bit[0] of DMA_CMD register
5. Wait until DMA_STATUS register has bit[0] as 1.

2.4 Register Map

In order to control DMAC, software can configure the following registers.

Offset	Reg Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	5	4	3	2	1	0
0x00	DMA_VER	version																																
0x04 ~0xFC		Reserved																																
0x100	DMA_SRC	start_addr																																
0x104	DMA_DST	start_addr																																
0x108	DMA_LEN																	byte_len																
0x10C	DMA_CMD																																	start
0x110	DMA_STATUS																																	done
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	5	4	3	2	1	0

2.4.1 DMA VERSION

Field name	Bit range	R/W	Reset value	Description

version	[31:0]	R	0x0001_2025	The version of this DMA controller. The upper 16 bits represent the major version. The middle 8 bits represent the minor version. The lowest 8 bits represents the micro version. This document describes behaviors of major version 1.

2.4.2 DMA_SRC

Field name	Bit range	R/W	Reset value	Description
start_addr	[31:0]	R/W	0x0000_0000	start address of the source range.

2.4.3 DMA_DST

Field name	Bit range	R/W	Reset value	Description
start_addr	[31:0]	R/W	0x0000_0000	start address of the destination range.

2.4.4 DMA_LEN

Field name	Bit range	R/W	Reset value	Description
byte_len	[15:0]	R/W	0x0000	Number of bytes to be transferred from the source to the destination.

2.4.5 DMA_CMD

Field name	Bit range	R/W	Reset value	Description
start	[0]	W	N/A	Writing 1 to this field will initiate a DMA transfer based on DMA_SRC, DMA_DST, and DMA_LEN registers. Software must not write 1 when there's an on-going transfer. Writing 0 to this field does not affect operation

2.4.6 DMA_STATUS

Field name	Bit range	R/W	Reset value	Description
done	[0]	R	1	This field is 1 when there's no on-going DMA transfer. Software must wait this field to be 1 for a completion of a transfer. Software must not initiate a DMA transfer when this field is 0.

3 Micro-architecture v1.1 Specification

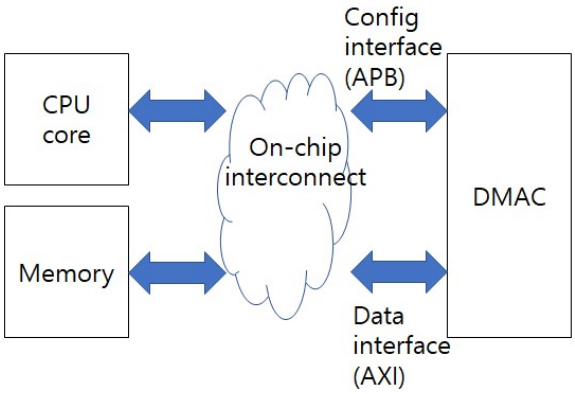
This section describes microarchitecture of a simple DMAC. It reads data from memory, buffers the data, and write the data into memory. It repeats this procedure until it completes transferring the specified number of bytes.

For simplicity, it read/writes one-cycle data (4 bytes) at a time (in other words, burst-1 transfers). For simplicity, this microarchitecture does not consider write responses from the AXI interface. Later versions will support burst transfers and write responses.

3.1 External Interface

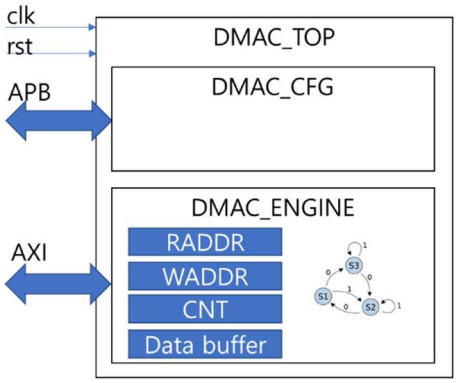
DMAC v1.1 has the following external interfaces to communicate with other hardware IPs.

- AMBA APB interface for configuration
- AMBA AXI interface for data transfer



3.2 Block Diagram

DMAC v1.1 has the following blocks inside.

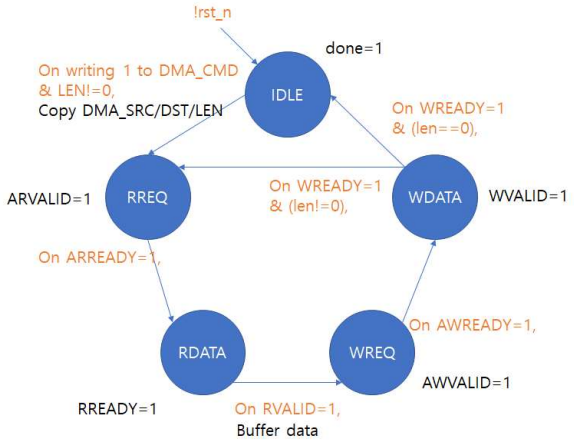


3.3 Configuration Register (lab2)

This block receives read/write requests from the APB and configures the registers describes in Section 2.4.

3.4 Finite State Machine (lab3)

DMA engine utilizes the following state machine to control operations.



State	Major outputs					Next State	Next state transition condition	Notes
	ARVALID	RREADY	AWVALID	WVALID	done			
IDLE	0	0	0	0	1	RREQ	(DMA_CMD.start is written as 1) and (DMA_LEN.byte_len!=0)	On moving out, - Copy DMA_SRC to ARADDR. - Copy DMA_DST to AWADDR - Copy DMA_LEN to internal counter
RREQ	1	0	0	0	0	RDATA	ARREADY=1	On moving out, - Increment ARADDR by 4
RDATA	0	1	0	0	0	WREQ	RVALID=1	On moving out, - Buffer RDATA into the data buffer
WREQ	0	0	1	0	0	WDATA	AWREADY=1	On moving out, - Increment AWADDR by 4 - Decrement the internal counter by 4
WDATA	0	0	0	1	0	RREQ	(WREADY=1) & (counter!=0)	
						IDLE	(WREADY=1) & (counter==0)	

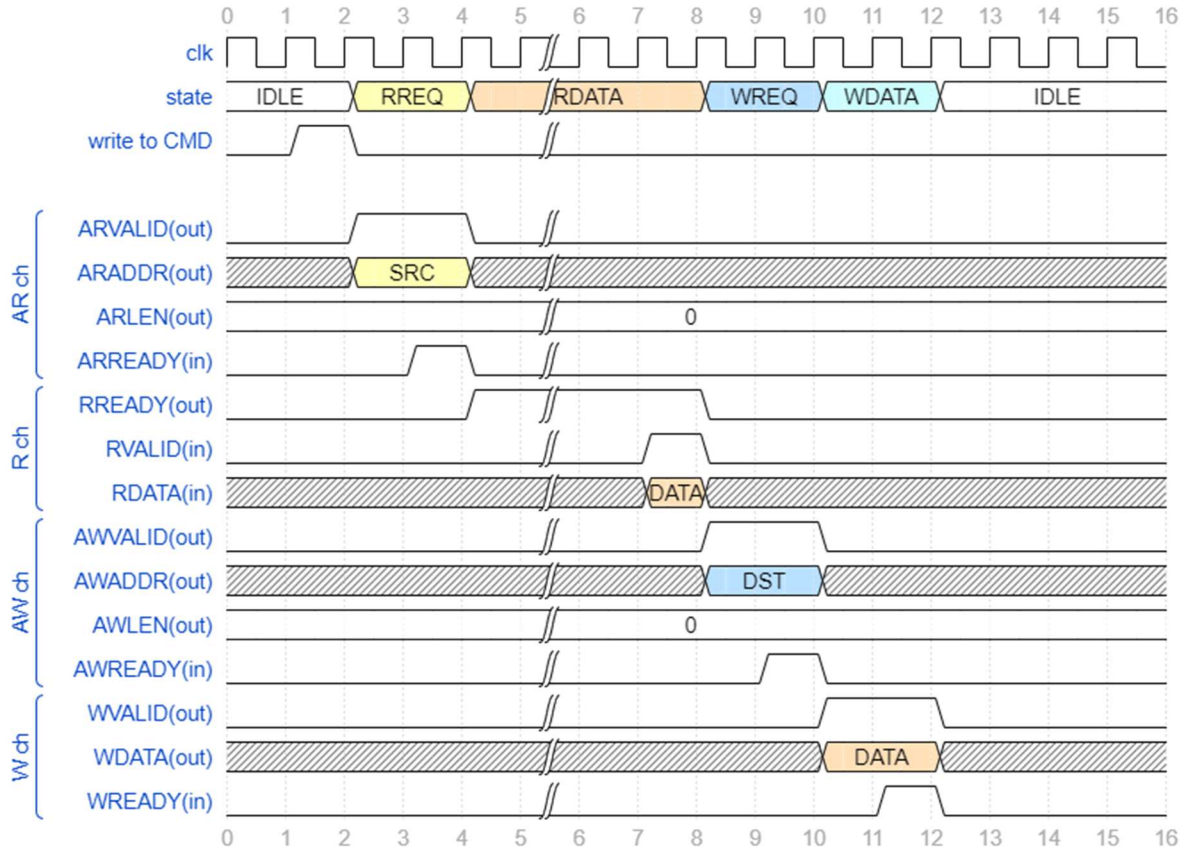


그림 1. DMA operation with microarchitecture v1.1

4 Micro-architecture v1.2 Specification (lab4)

A problem with microarchitecture v1.1 is that it reads/writes data one-by-one. As memory read takes some time, DMAC v1.1 will suffer from poor performance with a long memory read latency (그림 2). We will improve the microarchitecture to transfer a *burst* of data to minimize performance degradation.

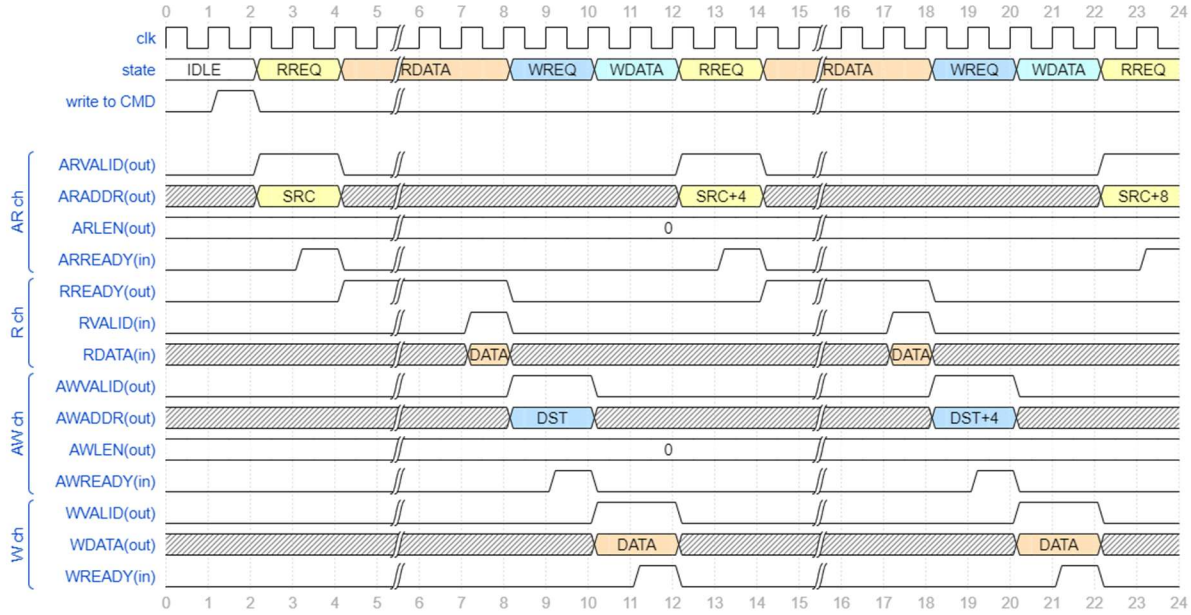


그림 2. DMA operation with microarchitecture 1.1. At a time, it transfers single burst of data

In Microarchitecture version 2, DMAC transfers up to 16 cycles of data with a single access. This can significantly reduce execution time by transferring data in bursts (그림 3).

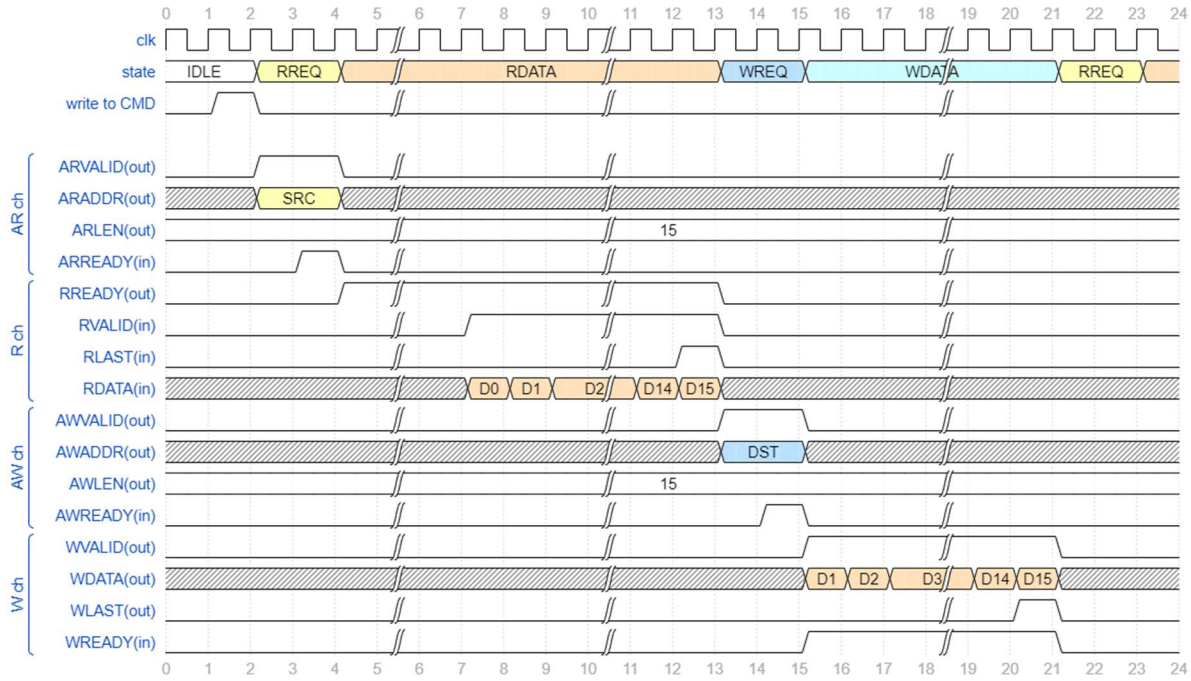


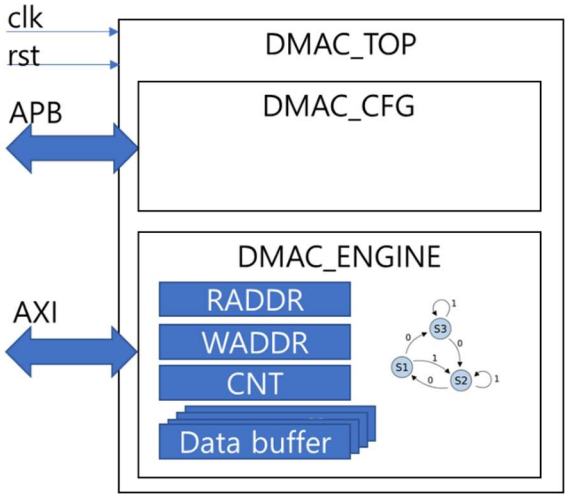
그림 3. DMA operation with burst transfers. At a time, a request reads/writes 16 cycles of data.

4.1 External Interface

Same as v1.1.

4.2 Block Diagram

Compared to v1.1, v.1.2 has multiple data buffers to hold a burst of read data, until they are transferred out for write.



4.3 Finite State Machine

Micro-architecture 1.2 slightly modifies the FSM in microarchitecture v1.1 to have the following state machine.

State	Major outputs					Next State	Next state transition condition	Notes
	ARVALID	RREADY	AWVALID	WVALID	done			
IDLE	0	0	0	0	1	RREQ	(DMA_CMD.start is written as 1) and (DMA_LEN.byte_len!=0)	On moving out, - Copy DMA_SRC to ARADDR. - Copy DMA_DST to AWADDR - Copy DMA_LEN to internal counter
RREQ	1	0	0	0	0	RDATA	ARREADY=1	On moving out, - Increment ARADDR by 4*16
RDATA	0	1	0	0	0	WREQ	RVALID=1	On moving out, - Buffer RDATA into the data buffer
WREQ	0	0	1	0	0	WDATA	AWREADY=1	On moving out, - Increment AWADDR by 4*16 - Decrement the internal counter by 4*16
WDATA	0	0	0	1	0	RREQ	(WREADY=1) & (counter!=0)	
						IDLE	(WREADY=1) & (counter==0)	

5 Change Log

Date	By	Description
2021-09-02	Jungrae Kim	Initial version