An Optimized Scheme for High-speed Data Interaction Based on TI-C6678 Multi-core DSP

Baoyu Su, Zhiyong Xu and Renjie Niu
Institute of Optics and Electronics Chinese Academy of Sciences, Sichuan Province, China
{baoyu su}baoyusu@126.com

Abstract - The performance of data interaction between different memories has become a significant factor in the complex embedded systems with the huge increase of processing needs, especially between internal chip-on memory and external memory. This paper advances a constructive optimized scheme named global direct memory access (GDMA) for high-speed data interaction in the multi-core digital signal processor (DSP) systems. Furmore, we give important recommendations to actualize software programming optimization of GDMA from three aspects.This scheme is based on the key technique of enhanced direct memory access versions3 (EDMA3), quick direct memory access (QDMA) and internal direct memory access (IDMA). This scheme can enhance the peak speed of data interaction between local memories by 53.8% and 2.5% averagely for some situations using QDMA compared EDMA3. By establishing the bridge memory area, the speed from level-1 data memory to external memory is optimized by 15.3% maximumly. The multi-core DSP system can achieve the performance of high-speed data interaction at around 5GBps to meet user expectations.

Index Terms – Data interaction, GDMA, High-speed, Multi-core DSP, Bridge memory.

1. Introduction

The signal processing currently has the characteristics of massive processing data and strict real-time requirement, which is heavy laden for the embedded systems. Hence, the evolution to multi/many-core architectures while being enabled by technology advancers, is also a solution to achieve more performance at less energy costs [1]. For example, the digital signal processor (DSP) system of this paper has received the level of 8 cores on a single chip, and the 32-core DSP is also predictable in the near future. Realizing high-speed data interaction is much more difficult because the multi-core system has more complex memory hierarchy and a great many shared resources for all cores [2]. Since the differences of working frequency lead to different speeds, we must find an effective scheme to balance these gaps. In conclusion, realizing high-speed data interaction is essential to enhance entire performance of the multi-core DSP system and other complex embedded systems.

Direct memory access (DMA) technology is a usual practice to ensure data interaction in embedded systems. The primary function of DMA is to move data without central processing unit (CPU) intervention because it has individual bus architecture and CPU is designed to execute algorithms operation rather than simple and repetitious data movement for a long time. Up to now, DMA has been developed several types. The embedded engineers usually use single-type DMA to actualize data transfer by investigating the practical applications in some laboratories. Although this method can meet currently requirement, they are solicitous to find a more effective mechanism. Furthermore, single-type DMA does not perfectly match with all scenes of data interaction and reduces the efficiency of our software coding.

The TMS320C6678 is a high-performance 8-core DSP whose cores and peripherals are interconnected using a bus called TeraNet (a dedicated switching network) [1]. In the following we will refer to this chip as simply TI-C6678. These cores are capable of running at 1.0 GHz (gigahertz), up to 1.25 GHz [3] [4]. For the EVM6678LE based on TI-C6678, the memory architecture consists of a two-level internal memory, external memory, and external extended memory, shown in Fig. 1. Each core has 32-kbyte level-1 program (L1P) memory, 32-kbyte level-1 data (L1D) memory, and 512-kbyte local level-2 (L2) memory [5]. In addition, the TI-C6678 has 4-Mbyte multi-core shared memory (MSM) and supports an external extended memory addressing space of up to 8GBytes for massive data storage [6].

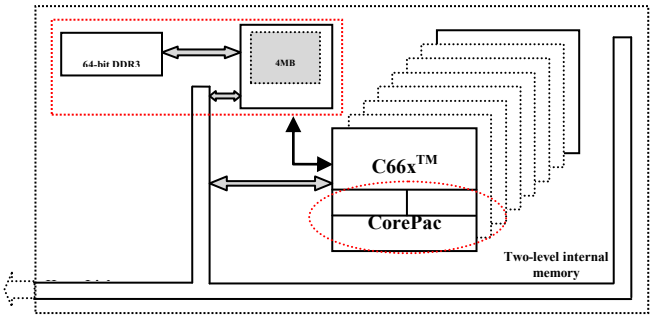

Fig. 1 Memory architecture of TI-C6678 System.

The remainder of this paper is outlined as follows: section 2 details the key technology related with our scheme. Section 3 presents the design and implementation. This is followed by test results and analysis in section 4. This paper closes with conclusions and future work in section 5.

2. The Key Technology

Our optimized scheme is based on three-type DMA. EDMA3 is discussed in section 2-A, QDMA discussed in section 2-B, and IDMA which is discussed in section 2-C.

A. EDMA3

Enhanced direct memory access version3 (EDMA3) is widely used to move data in the embedded systems. There are 144 general channels in the TI-C6678 and they all can be triggered by manual synchronization, event synchronization and chain synchronization. EDMA3 controller consists of EDMA3 transfer controller (EDMA3TC) and EDMA3 channel controller (EDMA3CC) [7]. EDMA3TC processes the tasks of data transfer and EDMA3CC is the most important part of user programming. EDMA3 support fully orthogonal 3-dimensional (3-D) transfer with independent indexes on source and destination address [8]. EDMA3 channels are configured in a parameter table. The table is a 2KBytes block of parameter RAM (PaRAM) located at the EDMA3CC. The PaRAM table consists of six-word parameter entries of 24 bytes each [7]. Each parameter entry of an EDMA3 event is organized into six 32-bit words, shown in Table I. EMDA3 controller can provide the ability to chain several EDMA transfer requests from one event that is driven by a peripheral or external device.

TABLE I Type Size for Papers

31 0	
Channel Option Parameter(OPT)	
Channel Source Address(SRC)	
Array/frame count (FRMCNT)	Element count (ELECNT)
Channel Destination Address(DST)	
Array/frame index (FRMIDX)	Element index (ELEIDX)
Element count reload (ELERLD)	Link address (LINK)

B. QDMA

QDMA is used to transfer a segment of data controlled by software code. QDMA's configuration and start is relatively simple and quick. QDMA only needs 1 to 5 CPU cycles to submit a transfer request [9]. QDMA is triggered by manual-programming registers but not event. QDMA can start and begin quickly to move data once this WORD is written after setting a specific WORD as the trigger word. The parameters of QDMA channel remain unchanged after completing data movement once, which is convenient to the next programming because we only need to change small part of registers. So, using QDMA can upgrade coding efficiency. QDMA always uses frame synchronization or block synchronization, which means that QDMA always requests a whole data frame (1-D) or data block (2-D) transmission.

C. IDMA

The IDMA technology is used to perform high-speed block transmission between any two of two-level internal memory. The IDMA controller allows rapid data paging between all local memories to cores. But IDMA does not moving data to internal mapped memory registers (MMRs). Typically, IDMA is used to transfer between slower level-2 memory and faster level-1 memory [10]. Compared the cache, the latency of IDMA is lower when the data interaction takes place in the background, concurrently with CPU operations. To fully support this function, IDMA consists of two orthogonal channels capable of working concurrently. The two channels are:

1) IDMA Channel 0: Intended for quick programming of configuration registers through external peripheral configuration port (CFG) of DSP. It is noticed that IDMA Channel 0 only can accesses external configuration registers but internal configuration registers.

2) IDMA Channel 1: Intended for data and program paging between local memory, i.e., for data transfer. In addition, IDMA Channel 1 also fills specific data to some blocks of local memory. It is controlled by four configuration registers. We pay attention to test the data interaction speed in this paper, so we focus on the IDMA Channel 1.

3. Design and Implementation

This section is designing the test scenarion of data interaction to get the compelling experimental results for our analysis and supporting our optimized scheme. The Part A is about overview of programming and Part B introduces the test scenario.

A. Overview of Programming

We choose EVM6678LE as the hardware platform for verifying and comparing the data transfer speed in many situations. The integrated development environment (IDE) is Code Composer Studio version 5.3 (CCSv5.3) compatibly with TI-C6678. The application programming interfaces (APIs) are based on CSL (Chip Support Library) module which makes programming easier. In order to accurately record the time of data interaction, we can get the number of CPU cycles between two breakpoints by enabling and viewing clock function of CCSv5.3 or using the clock register. Then the data interaction speed is equal to the ratio of data size and cycle number. This paper focuses on 4 kinds of SRAM (Static Random Access Memory), respectively L1D, L2, MSM and DDR3, because the frequency of data exchange between them is higher. It is taken notice that the L1D and L2 must be configured as SRAM not cache and their size can be set up according to the factual applications. Their start address is respectively 0x00F00000, 0x00800000, 0x0C000000 and 0x80000000.

B. Test Scenario of Data Interaction

We choose four of eight cores as the test group to verify data movement speed based on the three DMA modes. The test scenario of data interaction is shown in the Fig. 3. Core n (n is 0 up to 3) denotes the DSP core to run the program that realizes data movement by multi-mode. IDMA test is implemented between L1D and L2 on the internal memory. We have completed a test of the two-way data transfer between different memories for each mode.

For the realization of EDMA3, the detailed parameters of EDMA3 channel are ACNT, BCNT, CCNT, SRC, DST, etc. They together determine the amount and mode of data that to be transferred. ACNT is the size in bytes of an array, BCNT is the number of the array in a frame, and CCNT is the number of the frame in a data block [11]. Because QDMA is very similar

with EDMA3 except for ignoring the RLD register which consists of ELERLD domain and LINK domain, we only need to focus on the setting of trigger WORD for to realize QDMA. For the realization of IDMA, we need to notice that the start address and the destination address should be included in the internal memory area. The size of data segment should be suitable for the space of SRAM.

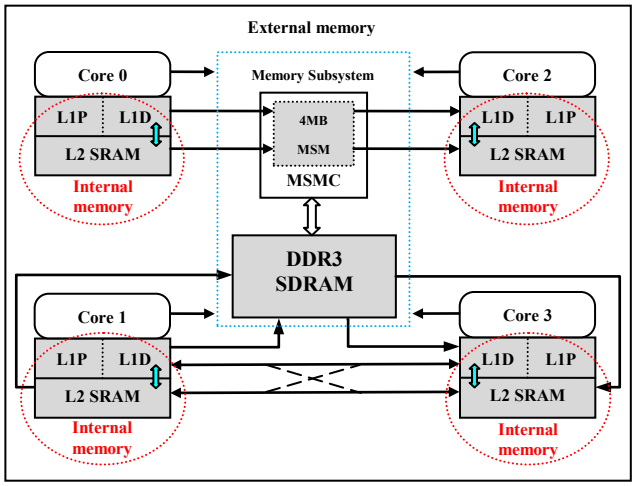

Fig. 2 Test scenario of data interaction.

4. Test Results and Analysis

We download the corresponding program to the core-group and ensure them to be carried out successfully on the EVM6678LE. Then, we get the test results described in section 4-A. Section 4-B presents our analysis.

A. Test Results

We get the speed of data transmission between memories using different modes, as shown in the Table II.

TABLE II Speed of Data Interaction

SRC \ DST	L1D SRAM	L2 SRAM	MSM SRAM	DDR3 SDRAM	DMA Mode
L1D SRAM	3.225	4.049	4.408	4.410	EDMA3
	3.319	4.167	4.557	4.534	QDMA
	3.763	7.227	※	※	IDMA
L2 SRAM	4.047	4.699	4.915	5.140	EDMA3
	4.182	4.844	5.072	5.253	QDMA
	7.220	3.268	※	※	IDMA
MSM SRAM	4.405	4.927	4.915	5.213	EDMA3
	4.551	5.073	5.087	5.262	QDMA
	※	※	※	※	IDMA
DDR3 SDRAM	4.225	5.140	5.198	3.396	EDMA3
	4.377	5.251	5.264	3.485	QDMA
	※	※	※	※	IDMA

The unit of speed is gigabit bytes per second (GBps) and the symbol, that is "※", indicates unsupported IDMA mode. Red font indicates the peak-bandwidth for EDMA3 mode, green font for QDMA mode, and blue font for IDMA mode.

B. Analysis

Based on the results, IDMA mode has a great advantage to move data between local memories especially two different memories, and the peak-bandwidth of IDMA mode for internal memory is increased by 53.8% and 50.9% compared with that of EDMA3 and QDMA mode. Consequently, the data interaction of this domain should be improved. But the disadvantage of IDMA mode is that using region is smaller than EDMA3 and QDMA mode. For the external memory, the average speed of QDMA mode is increased by 2.5% compared with EDMA3 mode, thereby gaining excellent performance for some data transmission situations by optimizing the selection of DMA mode. But QDMA do not support data reload and linking, which causes that it is not suitable for repeating data movement. The peak data bandwidth of EDMA3 mode is 5.213GBps between MSM and DDR3 when the frequency of DDR3 is running at 1333MHz and the width of data is 32 bits for our application. So, the result is very close to the theoretical speed, which is 5.332GBps. For a 1080*960 image, its size is 3.1MB and the transmission time is only 0.59 ns (nanosecond). Generally, there are 30 frames to process per second in the field of image processing, so the total time is 30*0.59 ns, i.e. 17.7 ns. Therefore, EDMA3 mode can satisfy the practical application at the present time. In addition, we take the situation transferring data between L1D and external memory into account. The peak-bandwidth is 4.557GBps but the peak-bandwidth between L2 and external memory is 5.253GBps, up 15.3% than the former. Hence, this situation should be optimized.

5. Conclusions and Future Work

We advance a cutting-edge optimized scheme for the multi-core DSP system, named global direct memory access (GDMA), aiming at achieving the best performance of data interaction to satisfy the user expectations. GDMA is demonstrated in Section 5-A, and Section 5-B presents software programming optimization of GDMA. Section 5-C unrolls the future work.

A. An Optimized Scheme: GDMA

The architecture of GDMA is shown in the Fig. 3. The entire bandwidth of GDMA can attain around 5GBps, which is very valuable and close to the theoretical bandwidth of TeraNet bus. GDMA divides memory into 3 areas, called internal memory, external memory, and bridge memory. Besides that, we label the most suitable DMA mode for these areas in the Fig. 3.

1) Internal Memory Area: For data transmission between local on-chip memories, we take use of IDMA mode. The peak transmission bandwidth of IDMA is increased respectively by 54.8% and 50% compared with that of EDMA3 and QDMA in this area. What is more, the realization code of IDMA is simpler and submitting task request is quicker than other modes.

2) External Memory Area: This area consists of DDR3 SDRAM and MSM SRAM. The characteristic of data interaction between them is massive and repeatability, so this case is more compatible to EDMA3 mode. EDMA3 can provide enough capability matching with the processing

requirement based on our analysis. It is most important that EDMA3 support linking, chaining and data reload, being convenient for the continuous tasks.

3) Bridge Memory Area: The reason of naming bridge memory is that this area plays a role of a bridge between L1D and external memory. When data interaction is required, GDMA uses bridge memory as the intermediate transfer-bridge to attain performance improvement by around 15% compared with direct access between L1D and external memory. This area belongs actually to the on-chip memory, which is the reason of not including DDR3 SDRAM. Though the speed between L2 and DDR3 is a little better than that between L2 and MSM, but they are further possibly leading to much latency. This area should make use of mixed mode consisting of QDMA and EDMA3 to select for different application environments.

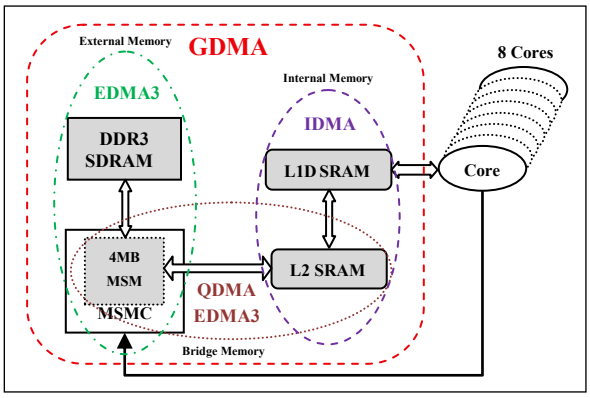

Fig. 3 Architecture of GDMA.

B. *Software Programming Optimization of GDMA*

GDMA give us an optimized scheme for high-speed data interaction, but it also brings some discomforts. Firstly, each DMA mode needs independent coding and many different API functions. Secondly, this scheme requires the support of many modules, which results in larger program to occupy shared memory space after compilation. Thirdly, these inconveniences not only affect the progress of programming, but also affect the optimization of the entire system. Therefore, we have to do further optimization about software programming. This work is very necessary and also very meaningful for ultra-high-speed data transmission in the future. In this subsection, we give personal recommendations to actualize software optimization from three aspects.

1) GDMA Registers: The primary task is to complete the definition of GDMA registers, because we must firstly understand the definition and set rules of relevant registers for the development of any program. In this subsection, we select the DMA type select register (we will refer to this as simply DMASEL) as an example to illustrate our thoughts about GDMA registers. Because we can use three types of DMA, we define two bits for to select the corresponding DMA type. For example, 00b represents using the IDMA mode, 01b represents using QDMA mode, and 10b represents using EDMA3 mode. The remaining bits can be defined as other purposes. Of course, the other method is that the GDMA registers also can be mapped to the registers of EDMA3, QDMA and IDMA to call for the corresponding DMA type.

2) GDMA APIs: APIs are some pre-defined functions and its purpose is to provide application developers the ability to access a set of routines based on a software or hardware, but you do not access to source code, or to understand the internal working mechanism of the details. The architecture of GDMA APIs must contain full functionality of three types of DMA, but we should optimize the similar functions, especially those between EDMA3 and QDMA. The number of GDMA APIs should be much smaller than the sum of EDMA3, QDMA and IDMA APIs.

3) GDMA Module: The modularity of GDMA can be better to handle complex systems by combining with other modules. The GDMA module should be able to clearly indicate its function, logic, state, and communication interfaces with other modules. The function, state, and communication interfaces can reflect external characteristic of GDMA module, and the logic reflects internal characteristic of GDMA module.

C. *Future Work*

GDMA is a very meaningful and challenging task for ultra-high-speed data interaction in the future. For the detailed implementation process, we should do further study and regard it as the main focus of our future work.

References

[1] Bernhard H.C. Sputh, Andrew Lukin and Eric Verhulst, "Transparent Programming of Many/multi Cores with OpenComRTOS: Comparing Inter 48-core SCC and TI 8-core TMS320C6678," in *The 6th MARC Symposium*, pp. 52–58, July 2012.

[2] L. Karam, I. AlKamal, A. Gatherer, G. Frantz, D. Anderson, and B. Evans, "Treads in Multicore DSP Platforms," *Signal Processing Magazine*, *IEEE*, vol. 26, no. 6, pp. 38–49, 2009.

[3] TMS320C6678 Multicore Fixed and Floating–Point Digital Signal Processor, Literature Number: SPRS691C, February 2012.

[4] Murtaza Ali, Eric Stotzer, Francisco D. Igual, Robert A.van de Geijn, "Level-3 BLAS on the TI C6678 multi-core DSP," *IEEE Computer Society, IEEE*, DOI 10.1109/SBAC-PAD.2012.26, pp. 179–186, 2012.

[5] TMS320C66x DSP Cache User Guide, Literature Number: SPRUGY8, November 2010.

[6] Chengfei Gu, Xiangyang Li, Wenge Chang, Gaowei Jia and Haishan Tian, "Matrix Transposition Based on TMS320C6678", in *GSMM*, pages 29-32, May 2012.

[7] Guolong Zhang and Xiaosu Xu, "High-speed and Real-time Communication Controller for Embedded Integrated Navigation System," in *IHMSC*, pp. 331–334, August 2009.

[8] Enhanced Direct Memory Access Controller User Guide, Literature Number: SPRUGS5A, December 2011.

[9] Fengqin Yu, "TMS320 C6000 structure principle and hardware design," Beijing: Beihang University Press, 2008, pp. 90–92.

[10] Jieming Ma, "Software-based Ultrasound Phase Rotation Beamforming on Multi-core DSP," University of Washington, 2012 pp. 6–9.

[11] Shanshan Xue, Jian Wang, Yubai Li, and Qiqiong Peng, "Parallel FFT Implementation Based on Multi-core DSPs", *ICCP Proceeding*, pp. 426–430, April 2011.