A Predictable Communication Assist

Ahsan Shabbir[1]
a.shabbir@tue.nl

Sander Stuijk[1]
s.stuijk@tue.nl

Akash Kumar[1,2]
akash@nus.edu.sg

Bart Theelen[3]
bart.theelen@esi.nl

Bart Mesman[1]
b.mesman@tue.nl

Henk Corporaal[1]
h.corporaal@tue.nl

[1]Eindhoven University of Technology Eindhoven, The Netherlands
[2]National University of Singapore, Singapore
[3]Embedded Systems Institute, The Netherlands

ABSTRACT

Modern multi-processor systems need to provide guaranteed services to their users. A communication assist (CA) helps in achieving tight timing guarantees. In this paper, we present a CA for a tile-based MP-SoC. Our CA has smaller memory requirements and a lower latency than existing CAs. The CA has been implemented in hardware. We compare it with two existing DMA controllers. When compared with these DMAs, our CA is up-to 44% smaller in terms of equivalent gate count.

Categories and Subject Descriptors

B.4.3 [**Hardware**]: Input/Output and data communication—*Interconnections,interfaces*

General Terms

Design, Performance

Keywords

CA, Predictable, FPGAs, Communication, MP-SoC, DMA

1. INTRODUCTION AND RELATED WORK

The number of applications which is executed concurrently in an embedded system is increasing rapidly. To meet the computational demands of these applications, a multi-processor system-on-chip (MP-SoC) is used. In [2], a multi-processor platform is introduced that decouples the computation and communication of applications through a communication assist (CA). This decoupling makes it easier to provide tight timing guarantees on the computation and communication tasks that are performed by the applications running on the platform.

Several CA architectures [4, 5, 6] have been presented before. These CAs use separate memory regions for storing data which needs to be communicated and data which is being processed (i.e., separate communication and data memories). This enables these CAs to provide timing guarantees on their operations, but at the cost of relatively high latencies and large memory requirements.

The problem of large memory requirement has been solved by a number of DMA architectures [1, 3, 7]. These DMAs transfer data between neighbouring tiles and between tiles and the main memory. However, DMA controllers do not provide any guarantees on their timing behaviour. A DMA controller is a piece of hardware which performs memory transfers on its own. A CA can be seen as an advanced distributed DMA controller [5]. Distributed means in this context that the CAs at both ends of the connection are working together to execute a block transfer, using a communication protocol on top of the network protocol.

CF'10, May 17–19, 2010, Bertinoro, Italy.
ACM 978-1-4503-0044-5/10/05.

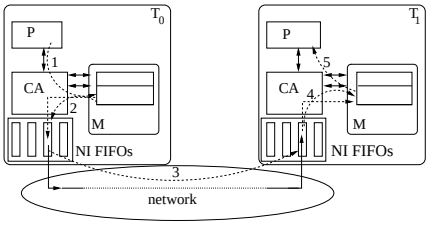

Figure 1: Proposed CA-based platform.

In this paper, we introduce a novel CA architecture in which a single memory region is used for data which is communicated and data which is processed. This leads to an up-to 50% lower memory requirement as compared to the CA design presented in [4]. At the same time, our CA architecture requires 44% less area when compared to existing DMA architectures.

The rest of the paper is organized as follows. Section 2 introduces our CA in more detail. Section 3 presents architectural details of our CA. The results of the hardware implementation are presented in Section 4 and Section 5 concludes the paper.

2. COMMUNICATION ASSIST

Figure 1 shows the global view of our CA. It receives data transfer requests from the processor (step 1 in Figure 1), moves the data to the Network Interface (NI) FIFOs (step 2). The data goes through the network (step 3) and the CA at the receiving tile copies it into the local memory of the tile (step 4). The processor P in tile T_1 processes the data and subsequently releases the space (step 5) so that the CA can re-use this space for further transfers.

The CA presented in [4] has a separate data memory and communication memory. These separate memories not only cost additional area but also latency as the processor has to move the data from the data memory to the communication memory and vice versa. Our CA does not require a sep-

arate communication memory resulting in a lower memory requirement and latency. Following are the basic functions of our CA:

1. It accepts data transfer requests from the attached processor and splits them into local and remote memory requests.
2. Local memory requests are simply bypassed to the data memory.
3. Remote memory requests are handled through a round robin arbiter. Every two cycles, a 32-bit word is transferred from the buffer in the memory to an NI FIFO channel or vice versa.
4. The buffers implemented in the memory are circular buffers. The number of NI FIFO channels can be greater than or equal to number of buffers in the data memory. Our CA is programable, so the same buffer in the memory can be used as input and output depending on the port to which it is connected.

Our CA acts as an interface that provides a link between the NoC and the sub systems (processor and memory). It also acts as a memory management unit that helps the processor keep track of its data. As a result, it decouples communication from computation and relieves the processor from data transfer functions.

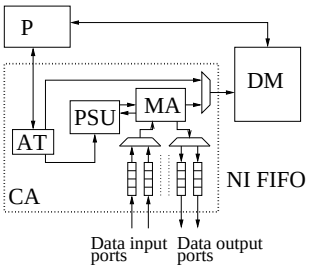

Figure 2: CA architecture.

Content	Offset
base address of the buffer	0x00
size of the buffer	0x02
NI FIFO ID, direction,	0x04
Write Start, W_S	0x06
Write End, W_E	0x08
Read Start, R_S	0x0A
Read End, R_E	0x0C

Figure 3: Context registers of a data buffer.

3. CA ARCHITECTURE

Figure 2 depicts the hardware components of our CA. The CA is connected to the network through input/output ports. Each data port has a FIFO buffer (NI FIFO) that connects the Memory Arbiter (MA) to the network. The NI FIFOs can be driven by two clocks: 1) the network clock and 2) subsystem clock. Separate clock domains allow the integration of subsystems with different clock frequencies. Following are the main components of our CA.

The **Address Translation Unit (AT)** is connected to the processor of a subsystem. The AT monitors the address bus of the processor and distinguishes between the local memory accesses and buffer memory accesses, it passes the local memory accesses to the DM, translates the virtual address of buffer into physical memory address.

The **Pointer Store Unit (PSU)** contains a set of registers (called buffer context) describing the status of each buffer. A buffer context consists of 6 registers as shown in Figure 3. The PSU selects one of the buffer contexts as indicated by the MA, sends the selected context to the MA and updates the registers for management of the circular buffers. Possible configurations of the PSU include the *size* of the buffer, the *base address* of the buffer in physical memory, and the *id* of the connected NI FIFO.

The **Memory Arbiter (MA)** receives an active context from the PSU and executes it. The MA executes the data transfer by generating a memory address, memory control signal and NI FIFO control signals according to the received context. The MA switches context every two clock cycles and checks the next buffers' context.

Every context belongs to a buffer such that the MA transfers one word between the NI FIFO and the buffer and then moves on to the next buffer. The transfers are performed in the same number of clock cycles every time and this gives us a CA with predictable timing behaviour.

4. HARDWARE IMPLEMENTATION

The CA compares favorably to classical DMA controllers. Table 1 shows the gate count (NAND2 equivalent) comparison of our CA with other architectures. The CA is synthesized for a clock frequency of 200 MHz. The design is implemented using Synopsis Design Compiler and 0.18μm Standard Chartered library. The results show that the our CA is 44% smaller than a commercial DMA [1]. The hardware results for the CA by [4] are not available in the literature. Note that our CA does not require complex functionality like "scatter and gather"; this makes our CA light weight when compared with the architectures shown in Table 1. All of the designs have 8 channels.

Table 1: Gate count comparison with other DMAs.

Property	our CA	MSAP [7]	PrimeCell [1]
queue config. (word)	32bit*8	32bit*8	32bit*4
gate count	36.3k	68k	82k

The MSAP presented in [7] is very similar to our CA. It uses a control network for the hand-shake between the processors, before the actual data transfer. Our CA does not require a control network as it uses "backpressure" as a flow control mechanism. This makes our CA more area efficient when compared to [7].

5. CONCLUSION

This paper introduces a programmable CA which uses a shared data and buffer memory. This leads to lower memory requirement for the overall system and to a lower communication latency as compared to CAs in literature. The CA is up-to 44% smaller in terms of area when compared with similar architectures and commercial DMA controllers.

6. REFERENCES

[1] ARM. Arm primecell^{TM} DMA controller, `http://www.arm.com/armtech/PrimeCell?OpenDocument`.

[2] Culler, D.,et al. Parallel computer architecture: a hardware/software approach. Morgan Kaufmann Publishers, Inc.

[3] Dave, C., and Charles, F. A scalable high-performance DMA architecture for DSP applications. In *ICCD '00:*,p. 414.

[4] Moonen, A., et al. A multi-core architecture for in-car digital entertainment. In *Proc. of GSPx Conference* (2005).

[5] Niewland, et al. The impact of higher communication layers on NOC supported MP-SoCs. In *NOCS '07* (2007), pp. 107–116.

[6] Nikolov, H., et al. Multi-processor system design with ESPAM. In *Proc. of CODES+ISSS* (06),pp. 211–216.

[7] Sang-Il, H., et al. An efficient scalable and flexible data transfer architecture for multiprocessor soc with massive distributed memory. In *DAC '04*, pp. 250–255.