A FPGA-based Soft Multiprocessor System for JPEG Compression

Sun Wei *Technical University Eindhoven, the Netherlands*
sunwei388@gmail.com

Abstract

To achieve a balance between high performance and energy efficiency embedded systems often use heterogeneous multiprocessor platforms tuned for a well-defined application domain. However, due to extremely high design cost and NRE for deep submicron IC, not many applications can afford that [1]. An alternative solution is FPGA-based multiprocessor system. In this way, both high programmability and low risk can be obtained. Recent research shows such a system which loses only 2.6X in performance (normalized) compared to a dedicated ASIP for IP packet forwarding application [2]. In this paper, we demonstrate the design flow of an FPGA-based multiprocessor system for high performance multimedia application and explore different on-chip interconnects for multiprocessor system. We construct a JPEG encoder on multiprocessor on Xilinx Virtex-II Pro FPGA. The design can compress a BMP image into a JPG image in high speed. We also implement different interconnects between processors, including bus, dual-port memory, FIFO and DMA controller, and explore the trade-off between them.

1. Introduction

FPGA is a programmable device consisting of logic blocks, memory blocks, programmable interconnections and sometimes processor cores. A soft processor is a programmable processor made from these programmable elements on the FPGA and a soft multiprocessor system is a network of soft processors. So the user can customize the processing unit, memory layout, interconnections and dedicated hardware accelerator for a specific application.

Xilinx provides tools and libraries for developing soft multiprocessor system on the Virtex family of FPGAs [3]. It enable user to integrate IBM PowerPC 405 cores, Microblaze soft processors, peripherals and customized hardware onto an FPGA chip. However, Inter-processor interconnections are not immediately available.

The soft multiprocessor solution proposes to implement a multiprocessor platform on FPGA instead of designing a new chip. The advantage of FPGA approach is (1) low design cost and turnaround time which means low investment and risk. (2) Designers can customize the multiprocessor system for a target application, especially by interconnection and hardware accelerators. (3) Retains multiprocessor programming model and provides an easy way to map application from existing code base. (4) Support for system iteration design method. Designers can iteratively optimize the system on the real, existing system instead of simulation and estimation. The disadvantage of FPGA approach is that soft multiprocessor system would lose a performance factor compared to ASIP if the ASIP fits well for the application. So it depends on the volume. For high volume ASIP is the preferred choice and for low volume FPGA is chosen. There is a 'break-even volume' where production cost compensates for the NRE DSM design cost. Since the latter increases rapidly there is a trend that the 'break-even volume' shifts to larger volumes which makes FPGAs more attractive. Furthermore, as the market changing faster and faster, it's getting more and more difficult to design ASIPs fit very well the market a few years later.

We address the following questions in this paper: (a) What's the design flow of soft multiprocessor system on FPGA. (b) What's the impact of different types of communication on FPGA? To demonstrate the design flow and interconnections, we build up a JPEG encoder as a multiprocessor system on a Xilinx Virtex-II Pro FPGA. In the second part of the study, different types of interconnections are evaluated and compared.

2. Experimental Study: JPEG Encoder on a Soft Multiprocessor Platform

2.1 Microblaze Soft Processor

Microblaze is a soft, 32-bit RISC processor designed by Xilinx for their FPGAs. Compared to other general purpose processors, it's quite flexible with a few configurable parts and capable of being extended by customized co-processors. There are a number of on-chip communication strategies available including a variety of memory interfaces. Following is the core block diagram of Microblaze processor. [4]

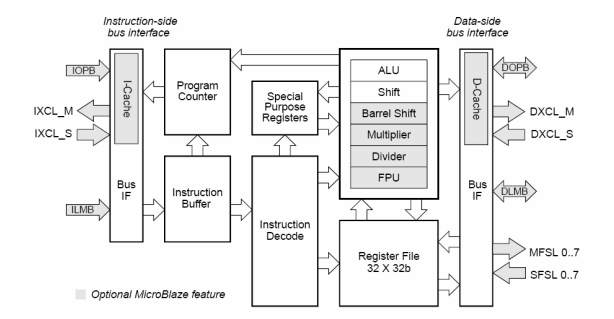

Fig.1. Microblaze processor core block diagram

Similar to most of RISC processors, Microblaze processor has an instruction decoding unit, 32x32b general purpose register file, arithmetic unit and special purpose registers. In addition, it has an instruction pre-fetch buffer. The arithmetic unit is configurable, as shown in core block diagram. The Barrel Shift, Multiplier, Divider and FPU are optional features. Microblaze processor has a three- stage pipeline: fetch, decode and execute. For most of instructions, each stage takes one clock cycle. There is no branch prediction logic. Branch with delay slot is supported to reduce the branch penalty.

Microblaze is a Harvard-architecture processor, with both 32-bit I-bus and D-bus. Cache is also an optional feature. Three types of buses, FSL, LMB and OPB are available. FSL bus is a fast co-processor interface. LMB is one-clock-cycle, on-chip memory bus while OPB is a general bus with arbitration. A typical single-core Microblaze system is as follows and a JPEG encoder has been mapped onto it [11]. A cache can be put between processor and external SDRAM. It's not shown on the following diagram because cache is considered as part of the Microblaze processor component in EDK.

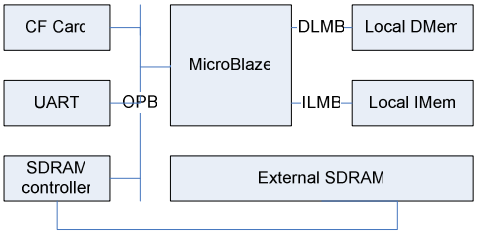

Fig.2. Typical single-core Microblaze system

2.2 Soft Multiprocessor System on Xilinx FPGA

We implement JPEG encoder on a Xilinx Virtex-II Pro 2VP30 FPGA with Xilinx Embedded Development Kit (EDK). For the entire system, including I/O, we use Xilinx XUP2Pro board, with Compact Flash (CF) card interface and external memory [5]. The 2VP30 FPGA consist of 13696 slices and 2448Kbits on-chip Block RAM (BRAM), 136 hardware multiplier and two PowerPC 405 cores. The estimated price for the chip is $557 @ 100pcs. [6]

The Microblaze soft core takes around 450 slices (3.2% of 2VP30 area) [4]. Nevertheless, one Microblaze processor typically needs at least 8KByte on-chip BRAM as data and instruction memory and a few memory controllers. It takes some slices and BRAMs in addition. Due to project schedule, the IBM PowerPC cores are not used in this design.

The soft multiprocessor system consists of four Microblaze processors, BRAMs, peripherals, external memory and interconnections as shown below. Besides FIFO interconnection, three other types of inter-connections, OPB bus, dual port memory and DMA, are evaluated later.

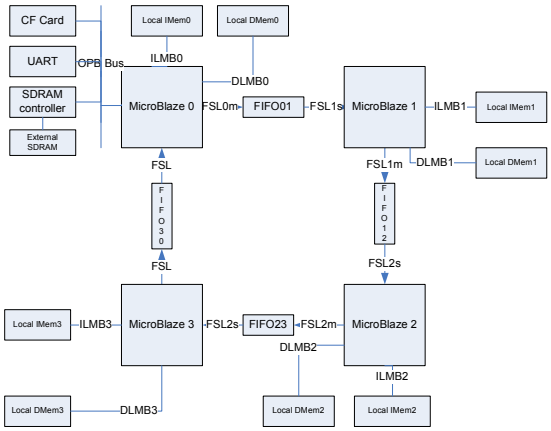

Fig.3. A soft multiprocessor system

Microblaze 0 in the system is used for I/O, external memory access and debugging while the rest three processors do the computation. External DDR memory is used as image buffer because CF card access is slow. The system runs at 100Mhz due to the limitation of OPB bus.

2.3 JPEG Encoder Application

We implement a baseline JPEG encoder application with color conversion and subsampling on the multiprocessor platform. The software reference code for the algorithm is written by Joris van Emden and Marcel Lauwerijssen from Technical University Eindhoven [7].

Except for file I/O and bootstrap, the JPEG encoder algorithm includes BMP and JPG header parsing, color conversion, DCT, zigzag scan, quantization and variable-length encoding. Following is the data flow of JPEG encoder.

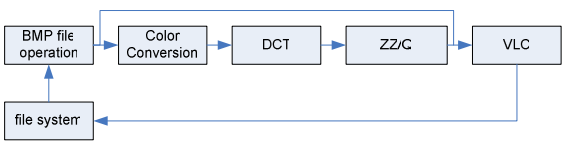

Fig.4. JPEG encoder data flow

2.4 Partitioning

These tasks are partitioned onto four processors, for instance in FIFO interconnection, as follows.

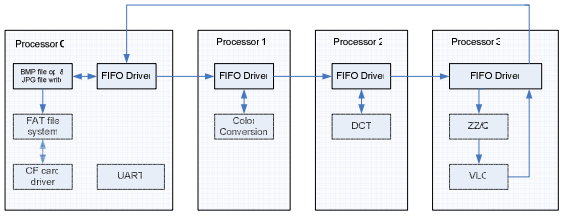

Fig.5. JPEG task partitioning

The table is a detailed description including input and output of every processor.

P#	Function	Input	Output
0	dedicated I/O	JPG bitstream	BMP macro block (RGB) + image size + end of image indication
1	color conversion	BMP macro block (RGB)	BMP macro block (YUV) + image size + end of image indication
2	DCT	BMP macro block (YUV)	f-domain macro block+ image size + end of image indication
3	ZZ/Q + VLC	f-domain macro block	JPG bitstream

Table1. Detailed task partitioning with input and output

The advantage of this partitioning is (a) low memory requirement. Actually Microblaze 1, 2 and 3 needs to store only a few macro blocks which is 16x16 pixels each. (b) easy to improve performance by dedicated hardware accelerators because every processor is dedicated to a well-defined task.

2.5 Streaming Programming Model

The programming model is modified from a shared memory model to a streaming model. In the reference code, all tasks share the same address space and communicate via shared memory. However, in order to maximize the throughput, these four processors need to run in parallel and therefore a streaming model is better. The inter-processor communication is adapted to a message-oriented model as well. Compared to shared memory, explicit message passing is easier to deploy, monitor and debug. The code now looks in this way:

processor 1 (color conversion)

```
for (…) {
        wait message from processor 0
        get message
        do color conversion
        generate a message containing the converted macro block
        check if there is space in the output FIFO
        send current message to processor 1
        ……
        }
```

processor 2 (DCT)

```
for (;;) {
        wait message from processor 1
        get message
        do DCT
        generate a message containing the transformed macro block
        check if there is space in the output FIFO
        send current message to processor 2
        ……
}
```

In addition, the communication link between BMP file operation and VLC is removed. Because there is only a small amount of data for image configuration transferred through this link, it doesn't make sense to make one more link for that. Instead, it is forwarded by processor 2 and 3. FIFO drivers are used to drive FIFO between processors and provide synchronization. For other interconnections, different drivers are employed.

3. Design Tools and Flow

Design tools and flow is an important factor with respect to design cost and time. Most of work is done with Xilinx EDK and ISE tools. EDK supports high level component based design. The design flow is also straightforward. There is little dependence between hardware flow and software flow so they can be designed and iterated independently.

3.1 System Design Flow

The system design flow is shown as follows.

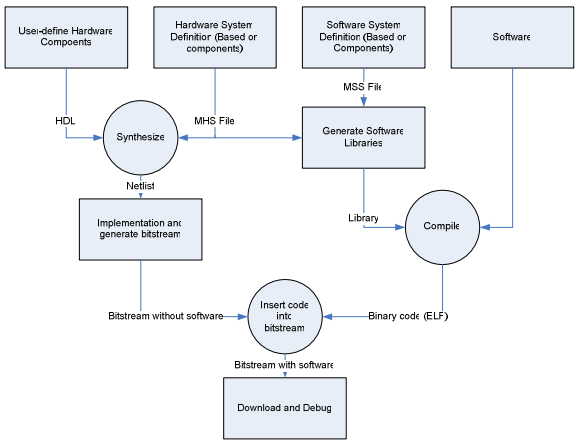

Fig.6. System Design Flow

On the hardware side (left), designers need to specify all needed hardware components, including components provided by Xilinx, like processor and memory and customized hardware components, like dma_controller and fifo_link component in this project. For customized hardware, designers need to provide source code or netlist. Within EDK, all these components are synthesized and invoke ISE afterwards to implement and generate a bitstream. Nevertheless, this bitstream is not the bitstream downloaded to FPGA because it contains hardware only. At the same time, on the software side, all needed software components, like drivers or operating system need to be specified as well. Based on these definition and hardware components definition, EDK can generate libraries for this system which is later linked to object files compiled from application code. The result is an ELF file. The detailed hardware and software flow is described in the following section.

The last step is to integrate software and hardware. Xilinx provides a tool called *data2mem* which can insert the binary software code in the ELF file into the bitstream generated from hardware flow. The setting of location and inserting method is already extracted during hardware flow. The resulting bitstream contains both hardware and software. It can therefore be downloaded into FPGA to run and debug.

3.2 Hardware Design Flow

The hardware system is defined on the component level with a Xilinx proprietary language in a .MHS file [3]. Basically it lists all components of the system, parameters and interconnections. A component can be a processor, a bus, a memory controller, a memory block, some peripheral or a custom hardware component. In EDK, Xilinx provides libraries for the Microblaze processor as well as a rich set of bus, memory and peripherals. In most of cases, it's enough to build a system. Most of them are provided in a netlist with a wrapper provided. Connections can be defined on both bus level and port level. On bus level, a group of signals are connected together. It's always preferable if possible. On port level, a signal is connected one by one. Every connection is called a port and defined a port name. For all memory components or memory-mapped peripherals, it's necessary to specify an address range. The next step is to synthesize. All components, both Xilinx provided and customized are synthesized together to generate a netlist for the whole system. Afterwards, the designer can start to implement and he can generate a bitstream consisting of the hardware configuration. A few more files are generated after synthesis, for instance, a memory mapping file. They are used for the software flow and the system flow later.

It's also practical to extend EDK by customized hardware components. To define a new component, the designer needs to specify the interface as well as the component entity. In EDK, there is a tool to generate the component template and the bus interface. Except for editing MHS file manually, there is a GUI interface, called "Base system generator" to generate XHS file for a simple system. For a multiprocessor system or complex system, it can be used as a good starting point.

3.3 Software Design Flow

The software is defined in a similar way. At the top level, components are specified. Designers can also specify bootstrap, operating system, file system, network stack, drivers and board support package if necessary. If some components are not provided by Xilinx, it's designer's responsibility to write them. Normally it's no longer written as a component like in hardware flow. It can be part of the application code.

In EDK package, Xilinx provides an alternative way to develop software with Eclipse initiated by IBM. Eclipse is nowadays becoming more and more popular and somehow industry standard of development environment. The Eclipse tool in EDK has been already customized for Microblaze processor or PowerPC and ready to use. The compiler and linker in EDK is a customized version of *gcc* tool chain. All *gcc* tools are available with *mb-* prefix. In some cases, especially in multiprocessor system, it's necessary to specify linkscripts to define heap and stack size, mapping of different component.

3.4 Debugging

After downloading the bitstream to the FPGA board, debugging starts. It's important and usually takes most of the design time. There are three ways of debugging, hardware debugging, software debugging and co-debugging.

For hardware debugging, there is a tool from Xilinx called *Chipscope* [8]. Basically Chipscope is an on-chip logical analyzer plus a user interface on PC. It can record timing information of multiple connections on chip. This information is read by the PC program afterwards via the JTAG port of the FPGA. A variety of triggers is available. The depth of trace is actually limited by the available memory on FPGA. To use Chipscope, the first step is to add a Chipscope component in the MHS file and to specify the connections. Normally at least two Chipscope components are required, chipscope_icon and chipscope_ila. The first one is the Chipscope controller and interface to the JTAG chain. The other one is the analyzer itself. The trace pins of the analyzer need to be connected to the ports to trace. After a bitstream is downloaded into the FPGA, the designer can start Chipscope Analyser on PC. This program is the interface to the user. It shares the JTAG connection with EDK and gets connected to the on-chip chipscope_icon automatically. By setting the necessary trigger condition, the designer can trace the signal he is interested in. Because the designer needs to go through the whole hardware flow if he adds some signals to trace, it's better to connect all possible signals to Chipscope as long as there is enough area left on the FPGA.

For software debugging, Xilinx provides a customized tool based on GNU *gdb*. To debug, simply start *XMD*, a backend server for gdb. After it connects to on-chip processor via JTAG, start gdb. Then you have full control of the processor. A customized version of *Insight*, a graphical shell of gdb is also available. Designers can also debug with Eclipse with a better interface. However, the mechanism is the same. To use gdb, it's necessary to enable the hardware debug module of the Microblaze processor. The debug module is connected to the JTAG interface of the FPGA and connected to XMD finally. For multiprocessor debugging, it's necessary to enable the debug module to every processor of interest. When starting XMD, you can choose the processor you like to debug and attach to it.

An easy alternative for software debugging is to add "printf "inside the software code. The information is dumped to a UART. The disadvantages are (a) printing on UART is slow; (b) only one processor can dump via UART due to the conflict of UART drivers between processors.

Co-debugging is often the most difficult part. In general we need to use Chipscope and gdb together. Moreover, designers need to synchronize hardware tracing and software debugging. He can use software to trigger Chipscope tracing and read the tracing data.

4. Interconnection Exploration

On-chip interconnection between processors is getting more and more important as the technology goes to deep-submicron because wires become dominant in delay and energy consumption. Four types of interconnections are implemented and compared afterwards [9].

4.1 Bus Interconnection

An easy way to connect four processors is via a bus. Xilinx provides OPB bus with arbitration. All processors, external memory and peripherals can just be connected to the OPB bus and it works. The hardware architecture of four-processor system connected by bus is as follows.

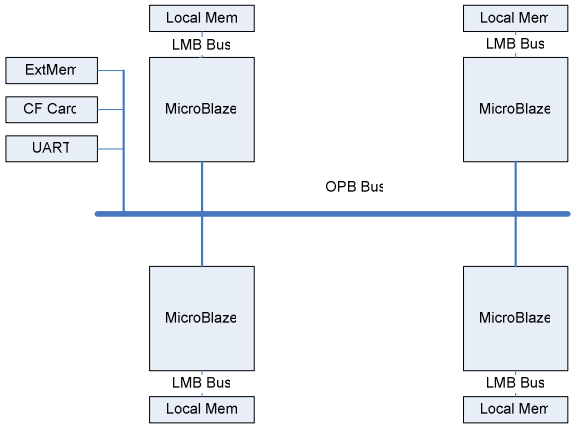

Fig.7. Hardware architecture of four-processor system connected by bus

The bus is shared by four processors, peripherals and external memory. Therefore it's a bottleneck of the system. It's very difficult for four processors to archive full-parallel running with bus interconnection. It may be used for a starting point for multiprocessor platform design.

4.2 Dual Port Memory Interconnection

Because all on-chip memory blocks on Xilinx FPGA are dual port memories, it's easy and efficient to employ dual port memory as communication channel between processors. The hardware and software architecture of four-processor system connected by dual port memory blocks is as follows.

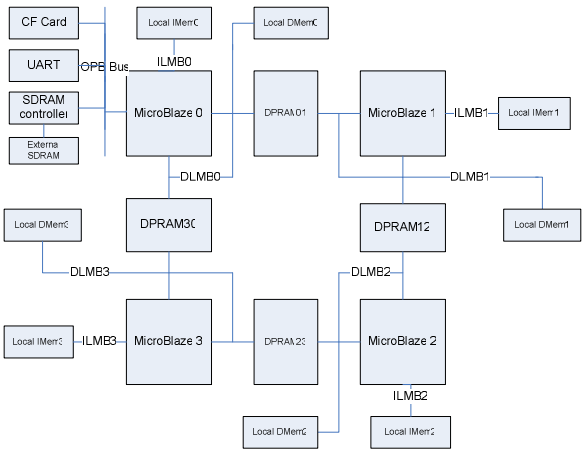

Fig.8. Hardware architecture of four-processor system connected by dual port memory blocks

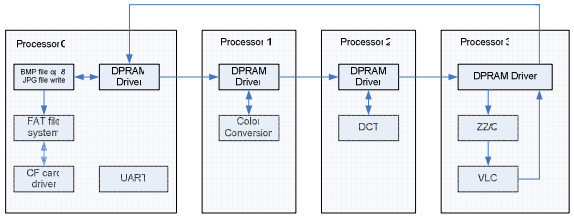

Fig.9. Software architecture of four-processor system connected by dual port memory block

Similar to the general architecture, every processor has two LMB buses, I-LMB bus and D-LMB bus. However, the data LMB bus here is connected to two dual port memory blocks in addition to data memory block. Each port of every dual memory block is connected to the data LMB bus of two different processors and therefore constitutes a communication channel. Every dual port memory is assigned to its dedicated address space as well. Processors can access dual port memory via normal memory access. The access is one-cycle-access and predictable because it's connected to LMB bus.

There is no inter-processor synchronization directly supported by dual port memory interconnection. It needs to be implemented through additional code, for instance, a flag. A flag can be a word in dual port memory. The pseudo code for a binary flag manipulation is as follows. If the flag is set more than once before it's cleared, it's only counted as once.

```
#define FLAG_UNSET 0
#define FLAG_SET  1

void flagSet (int* flag) {
        *flag++;
        }

flagClear (int* flag) {
        *flag=0;
}

flagWait (int* flag) {
        for (;;*flag) dummy_op();
}
```

The flag provides one-directional synchronization. If two-directional synchronization is needed, two flags can be used. The advantage of this implementation is simplicity. In fact it's quite straightforward. The disadvantage is that it can lock the processor and consume unnecessary energy because processor is still running when waiting for flag (busy-wait). An alternative is to design an additional hardware handshake component for synchronization.

4.3 FIFO Interconnection

Another often-used communication channel in a multiprocessor system is a FIFO. Compared to the last implementation, dual port memory blocks are replaced by FIFOs. The hardware and software architecture of four-processor system connected by FIFO is as follows.

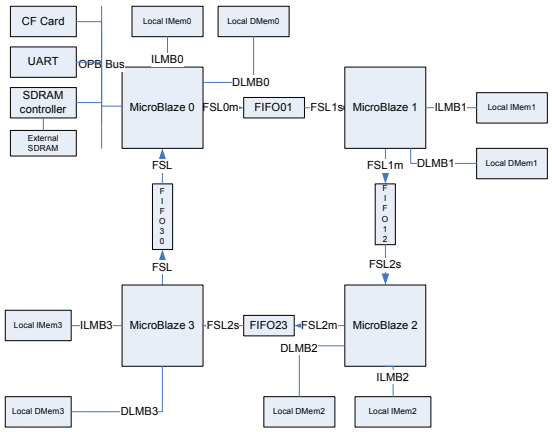

Fig.10. Hardware architecture of four-processor system connected by FIFO

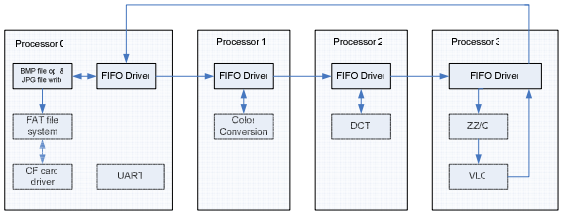

Fig.11. Software architecture of four-processor system connected by FIFO

FIFO is connected to processor via FSL bus. So there are two more buses for every processor, FSL master and FSL slave. FSL has build-in FIFO capacity. It's an ideal solution for FIFO implementation. Furthermore, there is hardware synchronization mechanism build in which is easy and efficient. If there is no data in the FIFO, the processor can be stopped without any extra energy

consumption.

In software, DPRAM driver is replaced by FIFO driver, which provides the same communication and synchronization mechanism. The partitioning and architecture of the application code is intact.

4.4 DMA Interconnection

DMA controller has its advantage in multiprocessor systems and is getting more and more deployed [10]. Compared to dual port memory and FIFO, it's an active component. So it can move data in parallel to processors without any attention from processor. Furthermore, all memory blocks on Xilinx FPGA are dual port memory blocks. That means the DMA controller can be simple and efficient because there is no need for complex arbitration circuit. The hardware and software architecture of four-Microblaze system connected by FIFO and DMA is as follows.

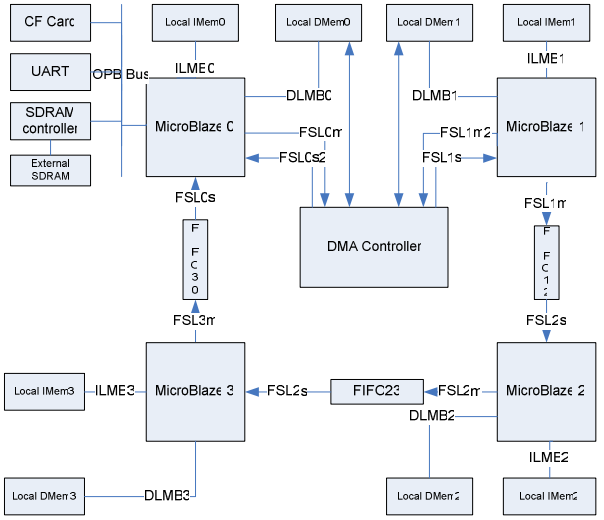

Fig.12. Hardware architecture of four-processor system connected by FIFO and DMA

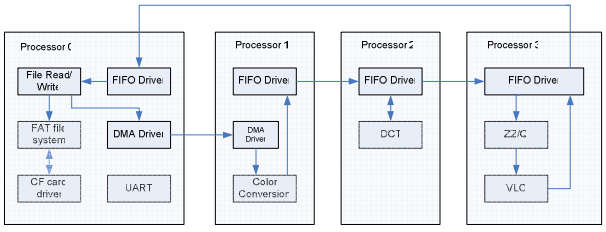

Fig.13. Software architecture of four-processor system connected by FIFO and DMA

Compared to the previous system, the FIFO between processor 0 and processor 1 is replaced by a DMA controller. The DMA controller has two sets of interfaces, to processor 0 and processor 1 respectively. For each interface, there is a memory bus connected directly to local data memory of the processor. It reads directly from the local data memory or writes directly to it. Besides that, the processor can configure and read back status via FSL master and slave bus. There is one channel inside the DMA controller. The processor only needs to set starting address, ending address, size of data block and go. It can move data autonomously. No CPU intervention need. After data is moved, a status byte can be read back by processor to indicate the result. Synchronization is also provided by controller and if no data moved by DMA, the processor can stop as well.

The software is similar to software in the previous system. The only modification is that DMA driver is added for the code running on processor 0 and 1. However, FIFO driver on the two processors have to be kept because there is still FIFO connection to other processors.

5. Conclusions

Fast design of high performance embedded systems based on soft multiprocessor cores with off-the-shelf tools is feasible. The tools and the flow are efficient. The software flow can be separated from the hardware flow. So a software update doesn't need re-implementing the whole chip. Most of the software code can be reused on the multiprocessor platform while it may require some modification for the programming model. Design cost and time can be significantly reduced.

There is a trade-off between different types of on-chip interconnections and therefore they should be deployed depending on application. (a) a bus is easiest to implement but poor in performance. It's not scalable either. It may be used for fast system prototyping and verification; (b) dual port memory is easy to implement. It's efficient and supports bidirectional communication. The disadvantage is that its resource consuming, inflexible due to the fixed topology, not scalable and that it needs an additional synchronization mechanism. (c) FIFO is also easy to implement and it uses build-in synchronization mechanism. But it's inflexible due to the fixed topology and not scalable either. Furthermore it's less efficient because it requires the processor to copy data into the FIFO. (d) DMA controller is flexible, scalable, and efficient. The controller actually can move data in parallel to processor. The disadvantage is the complexity of the controller.

Because of the after-manufacturing programmability of FPGA, the best interconnection is a combination of these interconnection types in a topology that targets for the application.

6. Acknowledgements

We thank Prof. Jef van Meerbergen for his guidance and reviewing of this paper. We thank Joris van Emden and Marcel Lauwerijssen for providing high quality JPEG encoder reference code and support afterwards. Prof. Henk Corporaal provides Xilinx board and FPGA for practical work. That's very important for our project. We also thank Jos Huisken, Akash Kumar and other members in ES group for their comments and discussions.

7. References

[1] Dr. Handel H. Jones, "How to Slow the Design Cost Spiral", *Electronics Design Chain*, 2002.

[2] Kaushik Ravindran, … "An FPGA-based Soft Multiprocessor System for IPv4 Packet Forwarding", 2005

[3] Xilinx Inc., "Platform Studio and EDK", *http://www.xilinx.com/ise/embedded_design_prod/platform_studio.html*

[4] Xilinx Inc., "Microblaze Microcontroller Reference Design User Guide", Sep, 2005.

[5] Xilinx Inc., "Xilinx XUP Virtex-II Pro Development System", *http://www.xilinx.com/univ/xupv2p.html*.

[6] Avnet Inc., *www.avnet.com.*

[7] Joris van Emden, Marcel Lauwerijssen, Sun Wei, Cristina Tena, "Embedded JPEG Codec Libarary", *http://sourceforge.net/projects/mb-jpeg/.*

[8] Xilinx Inc., "Xilinx Chipscope Pro", *http://www.xilinx.com/ise/optional_prod/cspro.htm*.

[9] Jeroen A. J. Leijten, Jef van Meerbergen, … "Stream Communication between Real-Time Tasks in a High Performance Multiprocessor"

[10] James Allan Kahle, IBM, "DMA Prefetch", *US Patent 7010626*, Mar, 2006.

[11] Joris van Emden, Marcel Lauwerijssen, Sun Wei, Cristina Tena, "JPEG Codec Libarary base don Microblaze processor", *http://www.opencores.org/projects.cgi/web/mb-jpeg/overview*