A Configurable High Speed Crossbar Network on FPGA

Zhongfu Guo, Xingming Zhang, Gao Yanzhao
National Digital Switching System Engineering & Technological R&D Center, Zhengzhou Henan 450002, China
gzf0309@mail.dlut.edu.cn

Abstract. The MPSoC platform plays a vital role in the design of parallel processor architectures. However, it poses a great challenge to design a configurable high-speed network regarding as the processors growing isomerization and reconfigurable. This paper proposes a configurable crossbar on-chip interconnection method based on FPGA, which can provide high speed high speed through Xilinx Fast Simplex Link (FSL) for processing elements include processors and hardware IP. We built a prototype system on FPGA to evaluate the transfer time and hardware costs of the crossbar network architectures. The experimental results show that each word can be reduced by 10 cycles, and the entire transmission delay only accounts for 6% of the total system operation time.

1. Introduction
Multi-core technology has evolved from homogeneous multi-core to heterogeneous multi-core. Because heterogeneous MPSoC integrate multiple types of on-chip resources and move toward diversification, How to provide a specific application-oriented system quickly and effectively has become an urgent problem to be solved. Combining reconfigurable technology with on-chip multi-core systems is the development trend of current embedded computing systems. However, interconnected communication between processors is the bottleneck of reconfigurable MPSoC design [1].

Considering the traditional bus-based architecture is clearly not configurable, it is not possible to change the connection as the reconfigurable processor changes while providing a high bandwidth connection. Therefore, a new on-chip interconnect solution is needed. This paper implements a flexible configurable Crossbar on-chip interconnect structure based on FPGA. This method is demonstrated using the FSL high-speed bus channel and programming interface aim for higher efficiency. At the same time, if the processor in the platform undergoes a reconfiguration change, the interconnection between computing resources can be dynamically configured. As a result, this solution provides configurable flexibility and provides efficient connectivity between processing elements.

The rest of this paper is organized as follows. Section 2 related work, Section 3 describes the crossbar network architecture and implementation on MPSoC systems. In Section 4, the experimental results and discussion are presented to clarify the architecture operations. Section 5 draws some conclusions and summarizes the paper.

2. Related Work
With the continuous increase of chip integration, the interconnection between processors elements directly affects the computing performance of the system-on-chip. A variety of studies have already been proposed. The current interconnection structure mainly includes on-chip bus, star interconnection, crossbar switch, NoC[2] and so on. Bus-based interconnection, different processors can’t access the bus
at the same time, need arbitration mechanism, affect the parallelism of the system, in order to build a flexible embedded MPSoC system, reconstruction technology has been proved to be an effective method. The interconnection structure changes with the change of computing resources, providing specific interconnections for specific computing scenarios, and crossbar interconnects and on-chip networks can be configured for greater flexibility. For example, Chen et al. [3] created a reconfigure Network-on-chip to create a “tidal lane” by changing the direction of the single-channel path to cope with the congestion problem of the on-chip network. However, due to the low granularity of reconstruction, the application scenario is limited. For example, Karata et al. [4] and others’ star interconnect architecture can dynamically adapt to changes in the number of processors, with high flexibility and low system overhead, but it is only applicable to single-core modules, considering the existence of on-chip interconnect topologies. Kinds, such as a typical symmetric processor. Wei et al. in Resu[m] considered that the crossbar is not linear with the increase of the number of cores, so the hierarchical structure of the interconnected architecture, although considering the interconnection of PE, has a strong reference for MPSoC. Liu et al. [6] proposed the application of the TDMA idea to the on-chip interconnect scheme. But this article focuses on the design process and the software tool chain.

In order to solve the above problems, it is necessary to design a more general reconfigurable interconnection strategy. This paper proposes reconfigurable crossbar interconnect architecture, the on-chip interconnect structure can be dynamically reconfigured for high flexibility can both achieve great performances.

3. Architecture and Concepts

3.1. Crossbar Network Architecture

Our architecture is to apply the configurable crossbar interconnect strategy to the design of MPSoC. The target heterogeneous multi-core platform mainly includes the following features: The computing resources in the system include embedded processors and hardware IP cores. The processor and the IP core are connected by a configurable crossbar. Software tasks run on a general purpose processor and hardware tasks run on an IP core. Tasks run on hardware IP with higher performance than they are running on.

![Crossbar Network Architecture](image)

Figure 1 Crossbar Network Architecture

Figure 1 shows the Crossbar Network Architecture. The processor core and the hardware IP core are connected by a crossbar. The communication data paths between different processing elements can be reconfigured. The processing elements can establish a connection according to the interaction of the tasks. The system includes a controller that considers the dependencies between the tasks while they are running. Two processing elements that perform more relevant tasks will guide the crossbar configuration connections. Considering that the processing elements are limited, a queue module is provided in the system to buffer the task request, and the current system state is recorded by the semaphore.
3.2. FSL based Organization

In order to ensure the efficiency and scalability of the system operation, we use a point-to-point bus interconnection mechanism to realize the data transmission between the processing elements and the crossbar.

(A) The communication architecture is based on a FIFO-based point-to-point bus. Microblaze needs to transfer tasks and data to the crossbar. Compared to traditional bus architectures, FIFO-based mechanisms provide continuous data transfer to increase interconnect throughput. The Xilinx FSL bus provides a communication interface that allows Microblaze to directly access the FSL’s FIFO buffer. At the same time, FSL is a point-to-point unidirectional link, so it is divided into two parts: the main module and the slave module. Figure 2 shows the FSL interface.

![Figure 2 FSL Bus Signals](image)

(B) Based on the FSL bus, this paper reuses the transmitted signals. Since the system communicates data through the crossbar, it requires two phases: the reconstruction phase and the data transmission phase. Therefore, a special communication protocol needs to be developed for the transmission. In this paper, the master-slave signals of FSL are multiplexed and a transmission protocol is designed. The main process is as follows:

Firstly, in consideration of the cross-switch configuration is first made for different requests of the application when the transmission is modified, both the configuration signal and the transmission signal need to be sent from the main module. This article mainly uses two interfaces FSL_M_Control and FSL_M_Data to multiplex to specify the number of the slave module. Other related signals directly represent the transmission signals associated with the bus transmission.

Secondly, after the crossbar receives the task request sent from the processor, it first determines whether the target IP core of the task request is in an idle state. If it is in a busy state, data transmission cannot be carried out; Otherwise, crossbar will be configured and the data can be transmitted directly. Similarly, when a hardware task finishes execution on the IP core, the results of the task’s execution are transmitted to the processor side via a cross switch, and subsequent tasks are run.

Thirdly, the system uses a buffer queue mechanism to cover resource conflict. Since only limited mission resources can be integrated on the platform, it is possible to cause resource competition. Therefore, an arbitration module must be implemented in the system to solve the problem of resource competition between IP cores. When a new request arrives at a processor, if the system is busy (i.e., the target IP core is already occupied by a task), the current task information (including the processor ID, target IP core ID, etc.) is stored in the queue buffer. After the current task is executed, the hardware IP core resources are released. The first task in the queue header will be fetched and used for scheduling. The crossbar must be reconfigured at the task runtime. The task is cleared from the queue when the first task in the queue header is sent to the corresponding hardware IP core for execution.

3.3. The algorithm process

The crossbar scheduling algorithm considers two parts. Firstly, the task calculation request. After receiving the task request, it checks the status of the target IP. If it is occupied, it puts the request in the queue. Secondly, it is the crossbar configuration part, and the data transmission after configuration. When finished, it will return the signal that the computing resource is idle and delete the completed request in the task queue. The scheduling Algorithm is outlined below.
The Scheduling Algorithm
1. Initialize IP_State, IP_ID, BUFFERRequest
2. Input the request signal into the crossbar BUFFERRequest
3. if (BUFFERRequest is not empty) then
4.   IP_ID ← IPH_Request
5.   if (IPH_Request State = Unoccupied) then
6.     check the IP_ID configure the crossbar
7.   if (crossbar configure completed)
8.     Path transmission data
9.     if (transmission completed)
10.    del the header request
11. end if
12. end if
13. else if
14. Return please wait
15. end if

Each processor on the platform can configure the crossbar and the configuration information is transmitted via the FSL bus interface. The IP_ID is represented by multiplexing the FSL_M_Control signal. The processor has different configuration strategies for different tasks. There are two situations to consider (1) if each general purpose processor is only connected to an IP core. In this case, all general purpose processors are isomorphic, so the task can choose to run on either processor. No data path is required between the general purpose processors for the transfer of tasks and data. (2) General-purpose processors can also be directly connected. This situation is more complicated and requires the design of a communication interface for both hardware and software tasks. This paper mainly studies and implements the first scheme.

![Finite State Automata](image)

Figure 3 finite state automata

3.4. Task partition and Interfaces
The tasks on different processors are designed independently, and the applications running on each processor are different. A static initialization assignment of the application is required at system setup. Software tasks running on a single processor access the FSL bus through the FSL bus call interface provided by the general purpose processor. Access interfaces are divided into software and software,
software and hardware, hardware and hardware. The design of the task communication interface has a greater relationship with the task division strategy.

In the first case, tasks running on different processors can only interact through shared storage; the processor and hardware modules are loosely coupled, and the hardware module can be regarded as a functional module of the processor. Therefore, the above three communication interfaces are regarded as interfaces between software and software, and data communication is performed by means of shared storage.

For the second case, communication between hardware modules is independent of the software programming interface. When a hardware module finishes running, the result is transferred directly to the next functional module through the data path.

4. Experiment

4.1. Experiment Setup

This article builds a prototype system on the Digilent XUPV5 development board and uses the Virtex UltraScale VU440 as a physical platform for on-chip reconfigurable resources. Two different types of hardware IP cores are integrated into the platform. The accumulator module mainly performs the function of integer summation. The module receives four 64-bit wide data in four clock cycles and returns the result after summing the input data. In addition to the accumulator, this article implements a one-dimensional IDCT module. The input to the IDCT module is also four 64-bit wide data, returning the result of the conversion.

The above functional modules are packaged in an FSL format IP core and connected to the crossbar via the FSL bus. The read and write timing of the access is consistent with the FSL bus protocol.

Implement a prototype system on an FPGA. The Microblaze processor is used in the system as a general purpose processor. A total of three Microblaze processor cores, one accumulator, one Inv, one IDCT module, and some necessary peripheral modules are integrated into the chip.

4.2. Hardware implementation overheads

Crossbar switch interconnect strategy can provide better service performance, so this section tests it as a realistic hardware overhead, configures different bus widths, and integrates them in ISE and EDK tools.

![Figure 4 Crossbar Hardware Cost](image)

Figure 4 shows the change in module hardware overhead for the crossbar as the data bit width increases. I/O resources are the main overhead of the crossbar module. When the data is greater than 64, for example, considering the beginning of 128, the I/O resources occupied by the matrix have exceeded the hardware resources that the FPGA itself can provide, so the hardware synthesis cannot be completed. The Virtex UltraScale VU440 FPGA chip used in this article provides only 1456 user I/Os and is already a very advanced model on the market. When the data bit width is 128, considering that the FSL bus is a
single-row bus, each module requires two point-to-point buses to be connected to the crossbar, so the total required signal is $128 \times 2 = 256$. For a system-on-chip with integrated 6 IP cores, at least $256 \times 6 = 1536$ I/O resources are required, considering that in addition to data transmission, other control signal connections are required.

Figure 5 Hardware Cost comparison

Figure 5 shows the ratio of the hardware resources occupied by the crossbar switch and other modules. It can be seen from the figure that the crossover switch occupies more resources than the three IP cores of the accumulator, Inv and IDCT, especially in terms of I/O. Since the other three IPs are mainly based on calculations, they occupy less I/O.

Figure 6 Crossbar transfer time

4.3. Transfer Time Result

The transmitted data is increased from four 64-bit data to 128 data, and the average transmission time in each case is measured separately. Figure 6 shows the results of the transmission over the Crossbar bus. As can be seen from the figure, as the transmission data increases from 4 to 128, the total transmission time increases from 56 cycles to 529 cycles. As a result, the transmission of a single data can be reduced from 14 clock cycles to 4 clock cycles, which shows that a single data transmission time is significantly shortened.

In order to analyze the running time of Crossbar in the whole system, we compare the transmission time and the running time. This paper transplants the EEMBC test set on the signal inversion and IDCT program to the Microblaze processor, and further realizes the IP of Inv and IDCT. Core and encapsulated in the form of an FSL bus module.
By analyzing the execution process of the test set, the data bit width is 32 bits, and the whole execution process is: adder(4 cycles), Inv(1433 cycles), IDCT(1226 cycles)
\[ T_{exec} = T_{adder} + T_{inv} + T_{IDCT} = 2663 \text{cycles} \]
We can get 143 cycles of hardware execution. In addition, since we encapsulate the computational module for the FSL core, we will bring an extra period of \(16 \times 3 = 48\) for data buffering. So the total time for hardware execution is
\[ T_{hardware} = T_{exec} + T_{packet} = 2711 \text{cycles} \]
The total transmission period of the data is
\[ T_{total-hardware} = T_{transfer} + T_{hardware} = 2903 \text{cycles} \]

| Table 1. IDCT IP Core Execution Time |
|-------------------------------------|
| **Type**    | **Hardware (cycles)** | **Software** |
|------------|-----------------------|--------------|
| IDCT       | 1226                  | 371005       |
| INV        | 1443                  | 437229       |

The proportion of transmission time is only 6% of the system running time. Considering that the mission model is implemented in hardware IP, when the general-purpose processor and hardware IP are jointly executed, the time ratio will be further reduced.

5. Conclusions
In this paper, we propose a configurable crossbar switch on-chip interconnect structure to accommodate the on-chip interaction requirements of reconfigurable MPSoC. All PEs are encapsulated as FSL interfaces. We evaluated the hardware overhead of the entire system implementation. It is measured that the transfer time can be as low as 4 cycles as the task increases. At the same time, the proportion of the system occupied by the transmission time is only 6% of the system running time. Considering the higher performance of the crossbar switch, the IO resource overhead is large, but the overall design can greatly improve the on-chip connection performance of the reconfigurable MPSoC system.

Acknowledgments
This work was financially supported by the High security level network infrastructure key equipment core chip and software development (2017ZX01030301).

References
[1] Benini, Luca, and Giovanni De Micheli. "Networks on chips: A new SoC paradigm." Computer-IEEE Computer Society- 35.EPFL-ARTICLE-165542 (2002): 70-78.
[2] Fernandez-Alonso, Eduard, et al. "Survey of NoC and programming models proposals for MPSoC." International Journal of Computer Science Issues 9.2 (2012): 22-32.
[3] Chen Saojie, Lan Yingcheng, Reconfigurable Network-on-chip, Springer; 2012.
[4] Karaata, Mehmet Hakan, and Thamer Alsulaiman. "An optimal inherently stabilizing routing algorithm for star P2P overlay networks." Concurrency and Computation: Practice and Experience 28.17 (2016): 4405-4428.
[5] Wei Shaojun, Liu Leibo, and Yin Shouyi. "Reconfigurable computing processor technology." Chinese Science: Information Science 42.12 (2013): 1559-1576(in Chinese).
[6] Zheng, Liu, et al. "Hybrid communication reconfigurable network on chip for MPSoC." Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010.
[7] Rosinger, Hans-Peter. "Connecting customized IP to the MicroBlaze soft processor using the Fast Simplex Link (FSL) channel." Xilinx Application Note (2004).