Heterogeneous Hardware Accelerators Interconnect: An Overview

Cuong Pham-Quoc, Zaid Al-Ars, Koen Bertels

Delft University of Technology, the Netherlands
Email: {P.PhamQuocCuong, Z.Al-Ars, K.L.M.Bertels}@tudelft.nl

Abstract. In this paper, we present an overview of interconnect solutions for hardware accelerator systems. A number of solutions are presented: bus-based, DMA, crossbar, NoC, as well as combinations of these. The paper proposes analytical models to predict the performance of these solutions and implements them in practice. Measurement results show that the NoC solution combined with a bus-based system provides the best performance as predicted by the analytical models. The NoC solution achieves a speed-up of 2.4× compared to the bus-based system, while consuming the least amount of energy. However, the NoC has the highest resource usage of up to 20.7% overhead.

1 Introduction

With the rapid development of technology, it is possible to integrate more than 7 billion transistors [1] into one system. However, the more transistors are integrated in the system, the more challenges need to be addressed such as power consumption, thermal emission and memory bottleneck. Therefore, homogeneous and heterogeneous multi-core are being increasingly used to overcome these issues.

Hardware/software co-design is one of the important approaches for multi-core design. In such systems, there is usually one traditional general purpose processor (GPP) and one or more hardware accelerators that function as co-processors to speed up the processing of special kernels of applications running on the GPP. With the rising number of cores, the communication between cores increases the requirements on interconnect parameters such as low-latency and area-efficiency.

Although bus systems are usually used as interconnect in most heterogeneous hardware accelerator systems due to their certain advantages [2], they become inefficient when the number of cores rises. Moreover, in data intensive applications, such as multimedia computing, HD digital TVs, etc., a large amount of data needs to be transferred from core to core. Therefore, data communication is usually a primary anticipated bottleneck for system performance. Obviously, optimization of the interconnect taking the data communication into account is an essential demand.

In this work, we present an overview on the interconnect solutions used for hardware accelerator systems. To improve the performance of bus-based interconnects, a DMA, a crossbar and a combination of both are used to consolidate the bus-based architecture. Moreover, Network on Chip (NoC), a state-of-the-art interconnect approach, can be used to improve the communication between hardware accelerators. In this work, we present the interconnect solution models to estimate the performance improvement of each interconnect compared to the bus-based interconnect. The experimental results show that the best system in terms of execution time and energy consumption is the system with a bus and a NoC, where the bus is used for the data exchange between the GPP and the hardware accelerators while the NoC is responsible for the data communication between the hardware accelerators. However, such system take a toll of up to 20.7% additional hardware resource compared to the bus-based interconnect system.

The rest of the paper is organized as follows. Section 2 briefly describes the related work. Section 3 presents in detail different interconnect solutions used in the heterogeneous hardware accelerator systems and their comparison. We implement an experiment to validate the comparison between the interconnect solutions in Section 4. The discussion on the different interconnect solution is presented in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

In this section, we discuss the different interconnect techniques available in Section 2.1, followed by the way these techniques are used at the system level in Section 2.2.

2.1 Interconnect techniques

Point-to-point interconnect is considered as the simplest interconnect solution for a system-on-chip (SoC). In a *point-to-point* interconnect architecture, the producer processing element (PE) is directly connected to the consumer PE. However, the biggest drawback of this architecture is the large number of wires required. This leads to difficulty in routing. Designs using this architecture are reported in [3], [4].

The bus architecture is a low cost interconnect for SoCs. The two standard and well-know bus architectures are AMBA developed by ARM [5] and CoreConnect developed by IBM [6]. Only CoreConnect has been adopted in Xilinx Virtex FPGA families. The main disadvantage of the bus architecture is the competition among modules (GPP, IO, memory controllers, etc) to access the bus introducing arbitrary latencies. This competition potentially degrades the performance of the system.

The crossbar is a well-known architecture for providing a high-performance and minimum latency interconnect. The main drawback of a crossbar is its cost. An $n \times n$ crossbar can quickly become prohibitively expensive as its cost increases by n^2. To reduce the cost, many studies focusing on application-specific crossbars have been reported such as in [7], [8].

In recent years, many Network-on-Chip architectures for FPGA have been reported such as DyNoC [9], FLUX [10] and CuNoC [11]. For low-latency applications-specific NoCs, driven by task graph, ReNoC [12] and Skip-links [13] are used. Scalability is the main advantage of NoC. Moreover, NoCs are emerging as a high level interconnect solution ensuring parallelism and high performance. However, there are still several issues that need to be addressed such as latency, power consumption and especially high area cost.

2.2 System-level interconnect solutions

The *Molen* architecture [14], is a heterogeneous, shared memory multicore system for software/hardware co-design. The Molen architecture consists of two types of processing elements (PEs): one *General Purpose Processor* (GPP) and one or more *Reconfigurable Processor(s)*, also so-called Custom Computing Unit(s) (CCUs). GPP has the main memory to contain application data while each CCU has each local memory (CCUMem) to contain its local data. The CCU exchanges parameters with GPP by exchange registers (CCUXreg) through an on-chip standard bus. While the GPP can access the main memory and the accelerator local memories, the accelerators can access only its local memory. The GPP and the accelerator local memories are also connected through an on-chip bus. When accelerator functions are needed, the GPP transfers data from the main memory to the local memory of the accelerator and copies the result back to the main memory.

The *MORPHEUS* architecture [15] has an ARM9 embedded RISC processor taking care for the control flow and synchronization, and three *heterogeneous reconfigurable engines* (HREs) for accelerating application kernels. The control infrastructure is done via an AMBA AHB bus which connects HREs and the ARM9 processor. The control flow is also performed via exchange registers, similar to the Molen architecture. A NoC is used to transfer data among HREs, main memory and off-chip memory. The data transfers via the NoC may be triggered by a Direct Network Access (DNA) hardware module. The MORPHEUS platform is implemented using STMicroelectronics CMOS090 technology. Although the platform shows very good simulation results, the NoC takes a huge resource toll up to 944Kgate.

A *Warp processor* [16] consists of a main general purpose processor, an *efficient on chip profiler*, *an on-chip computer aid design module* (CAD) and *a warp-oriented FPGA* (w-FPGA). The main processor executes the software part of an application while the critical software regions are synthesized and mapped onto the w-FPGA. The selection, synthesis and mapping the critical software kernels are done automatically by the profiler and the CAD module. The w-FPGA and the processor share the main data cache by using a mutually exclusive execution model. The main process, CAD module and the w-FPGA are connected together through an on-chip standard bus to configure the w-FGPA as well as to provide a mechanism for communication and synchronization between the main processor and the w-FPGA.

LegUp [17] is an open source high-level synthesis tool for FPGA-based processor/accelerators systems. The target system contains a processor connecting with custom hardware accelerators through a standard on-chip bus interface. The current version is implemented on the Altera Cyclone II FPGA with an Altera Avalon Bus as the interface for processor and accelerators communication. In this version, a shared memory architecture is used for exchanging variables between the processor and the accelerators. The shared memory uses an on-FPGA data cache and off-chip memory. The authors indicate that limitations of the bus system need to be further investigated.

3 Different Interconnect Solutions

In this section, we introduce different interconnect solutions used in heterogeneous hardware accelerators and give a comparison between them in terms of the total execution time of the hardware accelerators. In this work, we mainly focus on the data communication between the hardware accelerators.

3.1 Definitions and assumptions

Before presenting different interconnects used in heterogeneous hardware accelerator systems, we need to define some equations used to compare the quality of the interconnect techniques. The following terminology is used:

– **Hardware accelerator function**: A hardware accelerator function is defined by $Function(H, D_i^G, D_i^H, D_o^G, D_o^H)$; where H is the computation time of the hardware accelerator, D_i^G and D_i^H are the total amount of data input generated by the GPP and the other hardware accelerators, respectively. Similarly, D_o^G and D_o^H are the total amount of data output consumed by the GPP and the other hardware accelerators, respectively. All the amount of data is in byte. All values are considered for one execution of the hardware accelerator.
– **Data communication**: A communication between two functions is defined by $C_{ij}(F_i, F_j, D_{ij})$; where F_i and F_j are the producer and the consumer function, respectively, and D_{ij} is the total amount of data in bytes transferred from F_i to F_j. The functions F_i and F_j can be accelerated on hardware as well as run on the GPP.
– **The average time** taken by the GPP for transferring 1 byte from the main memory to a hardware accelerator local memory or vice versa is t_g, and the average time for transferring 1 byte from a hardware accelerator local memory to another one on the bus using direct memory access (DMA) is t_d. These values are platform dependent, however $t_d < t_g$.

The above mentioned actual amount of data can be measured by using profiling tools such as the QUAD toolset [18].

Hardware accelerator systems, such as Molen and LegUp, usually use a heterogeneous memory hierarchy in which the main memory is connected to the GPP while each hardware accelerator has its local memory to store data. In this work, we assume that the memory hierarchy is as follows: 1. GPP can access the main memory as well as the local memories of hardware accelerators through a standard on-chip bus; and 2. Hardware accelerators can access their local memory only.

In this work, we use the word "local memory" to refer to the local memory of a hardware accelerator. The word "main memory" is used for the main memory of the system which is connected to the GPP.

3.2 Bus-based interconnect

The bus system has some certain advantages compared with other interconnect techniques such as being compatible with most Intellectual Property (IP) blocks including GPPs [2]. Therefore, the bus system is considered as interconnect for most heterogeneous hardware accelerator systems. In these systems, GPP uses the bus to transfer data between the main memory and the local memories. Figure 1 depicts an architecture using the bus system as interconnect.

Consider $HW_1(H_1, D_{1i}^G, D_{1i}^H, D_{1o}^G, D_{1o}^H)$ and $HW_2(H_2, D_{2i}^G, D_{2i}^H, D_{2o}^G, D_{2o}^H)$ accelerators communicating together with the communication $C_{12}(F_1, F_2, D_{12})$. In most hardware accelerator systems, such as Molen and LegUp, whenever the hardware

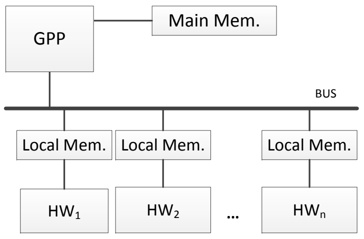


Fig. 1: The bus is used as interconnect

accelerator is executed, the GPP transfers input data from the main memory to the local memory and copies back the result of the hardware accelerator from the local memory to the main memory. Following this model, the total execution time of the two hardware accelerators is shown in (1). We refer to this model as the baseline model that we compare other interconnect solution with.

$$T_b = H_1 + H_2 + (D_{1i} + D_{1o} + D_{2i} + D_{2o})t_g \tag{1}$$

where $D_i = D_i^G + D_i^H$ and $D_o = D_o^G + D_o^H$.

The main advantage of the bus-based interconnect is that the system is simple. The bus-based system can be implemented on most hardware platforms. However, the biggest disadvantage of this system is that the communication between hardware accelerators is not taken directly into consideration but has to go through the main memory. This leads to a high volume of data needed to be transferred through the bus. Additionally, the data movement performed by the GPP with the bus is usually very slow. The higher the amount of data communication performed, the lower is the achieved performance of the system.

In the next sections, we introduce techniques used to consolidate the bus to improve the performance of such systems.

3.3 Bus-based with a consolidation of a DMA

DMA is a technique that allows to access system memory independently of the GPP. DMA is usually shared the bus with the GPP and other local memories. The main advantage of DMA is that while DMA transfers data, the GPP can do other work. Moreover, DMA usually takes less time than the GPP for moving the same amount of data. The main disadvantage of DMA is the bus competition because it shares the bus with GPP and local memories. In addition, hardware resource overhead is also a disadvantage of DMA.

In this model, a DMA is used to consolidate a bus. DMA is responsible for transferring data from one local memory to another local memory. Different from the baseline model, a communication profiling is used to improve the data communication operation. Consider the 2 above hardware accelerators HW_1 and HW_2, the outputs D_{1o}^H and D_{2o}^H of the hardware accelerators are transferred to other hardware accelerators by DMA rather than being written back to the main memory. In other words, the GPP is only responsible for transferring D_{1i}^G and D_{2i}^G from the main memory to the local memories as well as D_{1o}^G and D_{2o}^G from the local memories to the main memory. Other data movement is performed by DMA. Following this model, the total execution time for the two hardware accelerators is as follows.

$$T_d = H_1 + H_2 + (D^G_{1i} + D^G_{1o} + D^G_{2i} + D^G_{2o})t_g + (D^H_{1i} + D^H_{2i})t_d \tag{2}$$

The time needed to transfer the results of HW_1 and HW_2 (D^H_{1o} and D^H_{2o}) to other local memories is not considered in this equation since it is taken into account by the execution time of the other hardware accelerators.

The total reduction in time compared to the baseline model is as follows.

$$\Delta_d = (D^H_{1o} + D^H_{2o})t_g + (D^H_{1i} + D^H_{2i})(t_g - t_d) \tag{3}$$

3.4 Bus-based with a consolidation of a crossbar

Crossbar is a high-performance and minimum latency interconnect technique. Although the cost of crossbar increases by n^2 where n is the number of inputs, small crossbar is area-efficient and delay-optimized. In this model, we consider a 2×2 crossbar to share the local memories of the two hardware accelerators which communicate together. Figure 2aa depicts the system using a crossbar as a consolidation to the bus. The main advantage of the crossbar is that there is no need to move data between the two hardware accelerators connected to the crossbar. The main disadvantage of the crossbar is the two hardware accelerators cannot be executed in parallel due to the fact that a conflict can incur at the shared memories.

Consider the 2 hardware accelerators HW_1 and HW_2 above. With the crossbar, HW_1 can access not only its local memory but also local memory of HW_2. Therefore, neither GPP nor DMA is needed to transfer D_{12} from local memory of HW_1 to local memory of HW_2. The GPP is responsible for transferring other data. Based on this model, the total execution time of the two hardware accelerators is as follows.

$$T_c = H_1 + H_2 + (D^G_{1i} + D^G_{1o} + D^G_{2i} + D^G_{2o} + D^H_{1i} + D^H_{2i} - D_{12})t_g \tag{4}$$

Similar to the DMA model, The time needed to transfer the results of HW_1 and HW_2 (D^H_{1o} and D^H_{2o}) to other local memories is not considered in this equation since it is taken into account by the execution time of the other hardware accelerators.

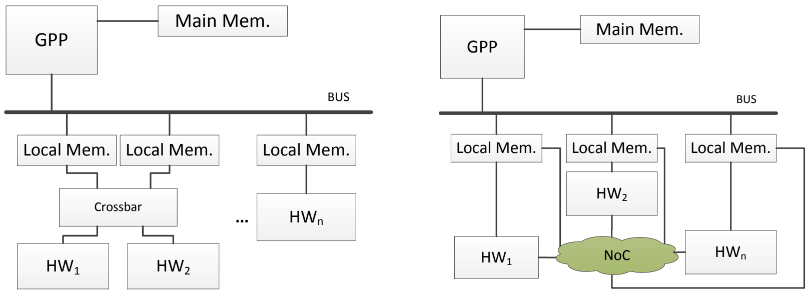
The total reduction in time compared to the baseline model is as follows.

$$\Delta_c = (D^H_{1o} + D^H_{2o} + D_{12})t_g \tag{5}$$

3.5 Bus-based with both a DMA and a crossbar

Due to the advantages of both DMA and crossbar, they can be considered as consolidations to the bus to improve the performance of the system at the same time. Consider again the 2 above hardware accelerators HW_1 and HW_2 and their communication. With the DMA, the data input for the two hardware accelerators from other hardware accelerators (D^H_{1i} and D^H_{2i}) are done by DMA. The GPP can do other work while the DMA performs the data movement. The total execution time of the two hardware accelerators is as follows.

$$T_{dc} = H_1 + H_2 + D^G t_g + D^H t_d \tag{6}$$



(a) (b)

Fig. 2: (a) The crossbar is used as a consolidation to the bus; (b) The NoC is used as interconnect of the hardware accelerators

where $D^G = D^G_{1i} + D^G_{1o} + D^G_{2i} + D^G_{2o}$ and $D^H = D^H_{1i} + D^H_{2i} - D_{12}$.

Similar to the DMA model, The time needed to transfer the results of HW_1 and HW_2 (D^H_{1o} and D^H_{2o}) to other local memories is not considered in this equation since it is taken into account by the execution time of the other hardware accelerators.

The total reduction in time compared to the baseline model is as follows.

$$\Delta_{dc} = (D^H_{1o} + D^H_{2o})t_g + (D^H_{1i} + D^H_{2i})(t_g - t_d) + D_{12}t_d \tag{7}$$

3.6 NoC-based interconnect

NoC is emerging as a high level interconnect solution ensuring parallelism and high performance. Although there are some certain disadvantages such as area overhead and the compatibility with processing cores, a well designed NoC can be used as the interconnect among the hardware accelerators. In this model, we use both the bus and the NoC as the interconnect. The NoC is used to transfer data from one local memory to another while the bus is used to exchange data between the GPP and the hardware accelerators. Figure 2b shows a system using a NoC as interconnect of the hardware accelerators. Using only the NoC as interconnect is an alternative solution. However, this solution will incurr a higher hardware overhead for the network interface at the GPP and higher delay in the communication between the GPP and the local memory compared to the bus.

With the NoC, the communication among the hardware accelerators is done in parallel with their execution. In other words, the output of one hardware accelerator is sent directly to the local memory of the consuming hardware accelerator through the NoC. Therefore, neither GPP nor DMA is required for data movement among the local memories. Consider the 2 above hardware accelerators HW_1 and HW_2. The total execution time of the two hardware accelerators is as follows.

$$T_n = H_1 + H_2 + (D^G_{1i} + D^G_{1o} + D^G_{2i} + D^G_{2o})t_g \tag{8}$$

The total reduction in time compared to the baseline model is as follows.

$$\Delta_n = (D^H_{1i} + D^H_{1o} + D^H_{2i} + D^H_{2o})t_g \tag{9}$$

However, the compatibility of the NoC and the hardware accelerators as well the the local memories needs to be addressed. The network interfaces should be developed to encapsulate the data and address generated by the hardware accelerators to the network package at the hardware accelerator side and to decode the network package to the data and address at the local memory side.

4 Experiments

4.1 Experimental setup

In this section, we introduce the application and the way we implement the experiment considering all the aforementioned interconnect solutions. We use the Molen architecture as the base system. Xilinx ML510 board [19] containing a xc5vfx130t FPGA device is used as our hardware system. In this experiment, we use the jpeg application from powerstone benchmark [20]. The QUAD toolset is used to generate the data communication profiling for the application first. We then choose the most suitable functions to accelerate on hardware. Figure 3 shows the communication profiling graph for the jpeg application.

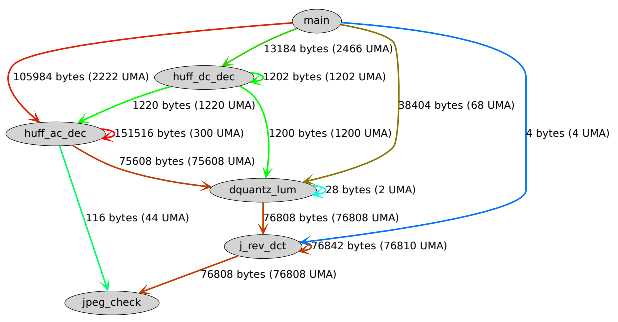


Fig. 3: The communication profiling graph generated by QUAD tool for the jpeg application

In this experiment, 4 functions (*huff_ac_dec*, *huff_dc_dec*, *dquantz_lum* and *j_rev_dct*) are accelerated on hardware. The application is implemented using the Molen architecture first. The DWARV tool [21] is used to compile the functions from C code to VHDL code. In the Molen architecture, only the bus system is used as interconnect. The GPP is the PowerPC embedded in the FPGA device and hardware accelerators are mapped on to the reconfigurable area. The PowerPC is run at 400MHz while the hardware accelerators are executed at 100MHz. BRAMs is used as local memories. We then extend the system with the DMA, the crossbar, both the DMA and the crossbar, and the NoC.

In the extended systems, we develop our 2×2 crossbar to share the local memories of the two hardware accelerators as depicted in Figure 2a. The Xilinx DMA IP core is used for DMA. A 3×2 NoC developed by Karlsruhe Institute of Technology, Germany [22] is adapted as the NoC in the experiment. We implement network interfaces (NIs) for the communication between the hardware accelerators as well as the local memories and the NoC. Table 1 presents the hardware resource utilization for each interconnect component and the maximum frequency.

Table 1: Hardware resource utilization (#LUTs/#Registers) for each interconnect component and the frequency

Component	**Resource utilization**	**Max. frequency**
Bus	1048/188	345.8MHz
DMA	700/556	252.7MHz
Crossbar	201/200	N/A
NoC	1854/2122	150MHz
NI HW Accelerator	396/426	422.5MHz
NI local memory	60/114	874.2MHz

4.2 Experimental results

In this section, we present the results for the jpeg application with different interconnect scenarios. We name the scenarios as Bus-based, DMA, Crossbar, DMA+Crossbar and NoC-based for the bus-based interconnect, bus with a DMA, bus with a crossbar, bus with both DMA and crossbar and NoC-based interconnect, respectively. In the jpeg application, we use two crossbars between *huff_ac_dec* and *huff_dc_dec* as well as between *dquantz_lum* and *j_rev_dct*.

Table 2 shows the computation time, the communication time and the total execution time for the hardware accelerators of the jpeg application. These numbers are measured by the real execution using the FPGA board mentioned above. The computation time is the time for the hardware accelerator processing input data while the communication time is the time for data movement between components. The execution total time of a hardware accelerator is the sum of the computation time and communication time. As shown in the table, the computation time does not change in different scenarios. The NoC-based scenario is the most efficient interconnect since it reduces the communication time by 74.3% compared to the bus-based model. Hence, the total execution time in the NoC-based scenario results in a 2.4× speed-up compared to the bus-based scenario. Based on the models presented in Section 3 and the information from the communication profiling graph in Figure 3, the communication time of hardware accelerators for each scenrios is computed theoretically. This theoretical communication time is shown in Figure 4a normalized to the software time of the hardware accelerators. The figure also compares the execution time normalized to the software time of the hardware accelerators in different scenarios. As shown in the figure, the theoretical communication time matches closely the measured communcation time.

Table 2: Execution time of hardware accelerators

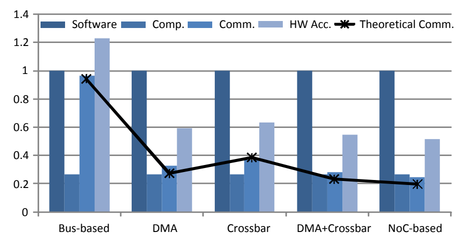
Scenario	**Computation**	**Communication**	**Total**
Bus-based	2.07ms	7.52ms	9.59ms
DMA	2.07ms	2.54ms	4.61ms
Crossbar	2.07ms	2.87ms	4.94ms
DMA+Crossbar	2.07ms	2.20ms	4.27ms
NoC-based	2.07ms	1.93ms	4.00ms

Table 3 gives the speed-up of the hardware accelerators and the overall application with respect to the software (the whole application is executed by the

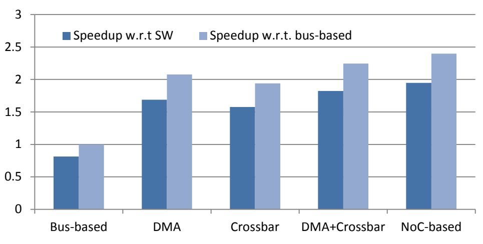
GPP only) and the bus-based model. The results show that the NoC-based model achieves a speed-up of up to 2.3× and 1.86× when compared to the bus-based model and the software, respectively. The table also shows that the performance of the bus-based model is even slower compared to the software due to the large communication time between the GPP and the hardware accelerators. Figure 4b shows the speed-up of hardware accelerators in different scenarios with respect to the software and bus-based model.

Table 3: Speed-up of hardware accelerators

Scenarios	**HW accelerators**		**Overall Application**	
	w.r.t Software	w.r.t Bus-based	w.r.t Software	w.r.t bus-based
Bus-based	0.81×	1.00×	0.81×	1.00×
DMA	1.69×	2.08×	1.64×	2.02×
Crossbar	1.58×	1.94×	1.54×	1.90×
DMA+Crossbar	1.82×	2.25×	1.75×	2.16×
NoC-based	1.95×	2.40×	1.86×	2.30×



(a)



(b)

Fig. 4: (a) Comparison between execution times normalized to software time; (b) Speed-up of hardware accelerators with respect to software and bus-based model

Table 4 shows the hardware resource utilization for all scenarios. The results show that the NoC-based model requires additional 20.7% resources (which takes 2.9% of FPGA resources) compared to bus-based model. Figure 5 compares the hardware resource utilization and the energy consumption in different scenarios normalized to bus-based model. As shown in the figure, although the hardware resource utilization is the largest in NoC-based scenario, it is the smallest energy consumption scenario. The energy consumption is calculated as power consumption (estimated with Xilinx PowerAnalyzer) multiplied by the overall application execution time. For all scenarios, the power consumption is almost identical, with a slight increase following the increasing hardware resource utilization.

Table 4: Hardware resource utilization (#LUTs/#Registers)

Scenario	**Accelerator**	**Interconnect**	**Total**
Bus-based	10707/11722	1048/188	11755/11910
DMA	10707/11722	1748/744	12455/12466
Crossbar	10707/11722	1249/388	11956/12110
DMA+Crossbar	10707/11722	1949/944	12656/12666
NoC-based	10707/11722	3490/2850	14197/14572

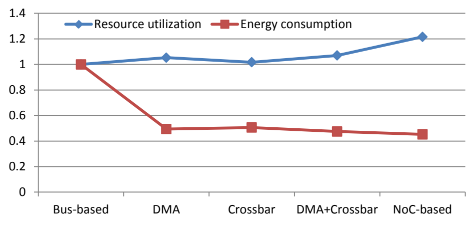


Fig. 5: Comparison of resource utilization and energy consumption normalized to bus-based model

5 Discussion

In previous sections, we presented the five different interconnect architecture heterogeneous hardware accelerator systems. We implemented an experiment with the jpeg application. In this section, we discuss the interconnect architectures in terms of hardware resource utilization, hardware accelerator speed-up and the energy consumption.

Based on the models as well as the experiment, the NoC-based model is the best in terms of execution time but it uses the most hardware resource when compared to others. The more resources are used the more power consumption is needed. On the other hand, the bus-based with consolidation of DMA, crossbar or a combination of both has a moderate improvement in speed-up and uses a limited amount of hardware resources.

Although modern devices, such as FPGA, contain a abundant amounts of resources, we have to choose trade off the number of resources and the price of the device. Moreover, the energy consumption is one of the main issues needed to be taken into consideration especially in battery-based systems. Energy consumption depends not only on power consumption but also the total execution time.

Based on the models, the designers can choose which interconnect solution is the most optimized for their systems. The designers have to choose trade off between the performance and the resource utilization. Depending on the requirements of the application as well as the resources available, the decision is made.

6 Conclusion

This paper presented an overview of interconnect solutions for hardware accelerator systems. The paper investigated the impact of augmenting the solutions to an existing bus-based infrastructure. Performance models for bus-based, DMA, crossbar, DMA+crossbar and NoC systems were discussed. Measurements made using these system match the predicted analytical performance models. The NoC solution provides the highest performance achieving a speed-up of 2.4× compared to the bus-based system, while consuming the least amount of energy. However, the NoC has the highest resource usage of up to 20.7% overhead.

References

1. NVIDIA, “NVIDIA Kepler GK110 Architecture Whitepaper,” 2012.

2. P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” in *DATE*, 2000.
3. C. Dick, “Computing the discrete fourier transform on FPGA based systolic arrays,” in *International Symposium on Field-programmable gate arrays*, 1996, pp. 129–135.
4. ARM Limited, “Multi-layer AHB overview,” 2001.
5. ——, “AMBA specification (rev 2.0),” 1999.
6. IBM, “Coreconect bus architecture,” 1999.
7. J. Y. Hur, T. Stefanov, S. Wong, and S. Vassiliadis, “Systematic customization of on-chip crossbar interconnects,” in *Reconfigurable computing: architectures, tools and applications*, 2007, pp. 61–72.
8. S. Murali, L. Benini, and G. De Micheli, “An application-specific design methodology for on-chip crossbar generation,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 7, pp. 1283 –1296, july 2007.
9. C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen, “DyNoC: A dynamic infrastructure for communication in dynamically reconfugurable devices,” in *Field Programmable Logic and Applications*, 2005, pp. 153–158.
10. S. Vassiliadis and I. Sourdis, “FLUX interconnection networks on demand,” *J. Syst. Archit.*, vol. 53, no. 10, pp. 777–793, oct 2007.
11. S. Jovanovic, C. Tanougast, S. Weber, and C. Bobda, “CuNoC: A scalable dynamic NoC for dynamically reconfigurable FPGAs,” in *Field Programmable Logic and Applications*, 2007, pp. 753–756.
12. M. B. Stensgaard and J. Sparso, “ReNoC: A network-on-chip architecture with reconfigurable topology,” in *International Symposium on Networks-on-Chip*, 2008, pp. 55–64.
13. C. Jackson and S. J. Hollis, “Skip-links: A dynamically reconfiguring topology for energy-efficient NoCs,” in *International Symposium on System on Chip*, 2010, pp. 49–54.
14. S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, “The MOLEN polymorphic processor,” *Computers*, vol. 53, no. 11, pp. 1363–1375, 2004.
15. M. Kuhnle, M. Hubner, J. Becker, A. Coppola, L. Pieralisi, R. Locatelli, G. Maruccia, T. DeMarco, F. Campi, A. Deledda, C. Mucci, and F. Ries, “An interconnect strategy for a heterogeneous, reconfigurable SoC,” *Design Test of Computers*, vol. 25, no. 5, pp. 442 –451, 2008.
16. R. Lysecky and F. Vahid, “Design and implementation of a microblaze-based warp processor,” *ACM Trans. Embed. Comput. Syst.*, vol. 8, pp. 1–22, 2009.
17. A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, “LegUp: high-level synthesis for FPGA-based processor/accelerator systems,” in *International Symposium on Field programmable gate arrays*, 2011, pp. 33–36.
18. S. A. Ostadzadeh, R. J. Meeuws, C. Galuzzi, and K. Bertels, “QUAD: a memory access pattern analyser,” in *Reconfigurable computing: architectures, tools and applications*, 2010.
19. Xilinx, “Ml510 reference design,” June 23 2009.
20. J. Scott, L. H. Lee, J. Arends, and B. Moyer, “Designing the low-power M•CORE architecture,” in *ISCA Workshop*, 1998.
21. R. Nane, V. Sima, B. Olivier, R. Meeuws, Y. Yankova, and K. Bertels, “DWARV 2.0: A CoSy-based C-to-VHDL hardware compiler,” in *Field Programmable Logic and Applications*, 2012.
22. J. Heisswolf, R. Koenig, and J. Becker, “A scalable NoC router design providing QoS support using weighted round robin scheduling,” in *Parallel and Distributed Processing with Applications Workshops*, 2012.