A Comparison of Five Different Multiprocessor SoC Bus Architectures

Kyeong Keol Ryu, Eung Shin, and Vincent J. Mooney
Georgia Institute of Technology
Electrical and Computer Engineering
Atlanta, GA 30332
{kkryu, eung, mooney}@ece.gatech.edu

Abstract

The performance of a system, especially a multiprocessor system, heavily depends upon the efficiency of its bus architecture. In System-on-a-Chip (SoC), the bus architecture can be devised with advantages such as shorter propagation delay (resulting in a faster bus clock), larger bus width, and multiple buses. This paper presents five different SoC bus architectures for a multiprocessor system: Global Bus I Architecture (GBIA), Global Bus II Architecture (GBIIA), Bi-FIFO Bus Architecture (BFBA), Crossbar Switch Bus Architecture (CSBA), and CoreConnect Bus architecture (CCBA). The performance of these architectures is evaluated using applications from wireless communications - an Orthogonal Frequency Division Multiplexing (OFDM) transmitter - and from video processing - an MPEG2 decoder. To increase performance, these bus architectures employ a pipelined scheme, resulting in improved throughput. While all five bus architectures perform well, we find that BFBA and CSBA perform the best for the OFDM transmitter and the MPEG2 decoder, respectively.

1. Introduction

Most of the current prevailing buses such as VME [1] and PCI [2] were designed for system level buses to connect to discrete devices on a Printed Circuit Board (PCB). However, there are many restrictions on a PCB, not least of which is the number of pins allowed. A System-on-a-Chip (SoC) allows designers to overcome the drawbacks of PCBs by implementing many or most parts of a system on a board on a single silicon chip. SoC technology allows one to take advantage of increased bus speed and decreased area compared with a PCB.

In the implementation of a multiprocessor SoC, the bus architecture comes to the forefront because the performance of the system is not dependent only on the CPU speed but also on the bus architecture which may cause in the system. An efficient bus architecture and arbitration for reducing contention plays an important role in maximizing the performance of the system. In this paper, five kinds of different multiprocessor SoC bus architectures are modeled and compared with detailed simulation: Global Bus I Architecture (GBIA), Global Bus II Architecture (GBIIA), Bi-FIFO Bus Architecture (BFBA), the Crossbar Switch Bus Architecture (CSBA), and CoreConnect bus architecture (CCBA).

For the bus performance evaluation, we apply these architectures to two application programs, one from wireless communications: an Orthogonal Frequency Division Multiplexing (OFDM) transmitter, and the other from video processing: an MPEG2 decoder. The pipelined operation in an SoC multiprocessor system allows significant improvements in the data throughput.

The paper is organized as follows. Section 2 shows the background and our motivation. Section 3 presents the detailed description about the five SoC bus architectures. In Section 4, we explain the two applications used for these architectures, including job assignments in each “Compute Node” (CN). Each job assignment is a unit of data processing in the multiprocessor system. Our experiment and simulation results are described in Section 5. Finally, we conclude the paper in Section 6.

2. Background and Motivation

Most of the recent designs of on-chip buses borrow their ideas from standard buses, especially VME and PCI, which were designed for PCB systems. The bus architecture for an SoC should be different from a PCB bus architecture because an SoC has a faster transfer rate due to shorter propagation delays and no restrictions on numbers of pins due to packaging or signaling constraints.

The most popular bus architectures utilize hierarchical levels of buses. For example, CoreConnect has three levels of hierarchy: Processor Local Bus (PLB), On-chip Peripheral Bus (OPB), and Device Control Register (DCR) [3,4]. PLB provides a high performance and low latency processor bus with separate read and write transactions, while OPB provides low speed with separate read and write data buses to reduce bottlenecks caused by slow I/O devices such as serial ports, parallel ports, and UARTs. The daisy-chained DCR offers a relatively low-speed data path for passing status and configuration information. The Advanced Micro controller Bus Architecture (AMBA) from

ARM has two levels of hierarchy: the Advanced High performance Bus (AHB), similar to PLB, and the Advanced Peripheral Bus (APB), similar to OPB [5]. CoreConnect and AMBA, which are pipelined buses, both require bridges between the high performance bus and the low speed bus for data transfer between the buses. CoreFrame from Palmchip Company is a nonpipelined bus which also has two independent bus types: Mbus for memory transfer and Palmbus for I/O devices [6].

The user configurable Triscend bus architecture utilizes a bus FIFO to enhance bus pipelining between masters and slaves [7]. The arbiter logic is relatively simple because the FIFO is both the single master for the slave side and also the single slave for the master side. The FIFO, however, requires additional memory and makes it difficult to predictably satisfy real-time constraints as compared to prioritized buffers. The Silicon Backplane from Sonic Inc. guarantees fixed bandwidth and latency by Time Division Multiplexed Access (TDMA) based arbitration [8].

As mentioned above, most current bus architectures for SoC have focused on increasing the communication efficiency between the high speed processor bus and the low speed peripheral bus. Yet, for many applications, the performance of multiprocessor systems relies more on the efficient communication among processors and a balanced distribution of the computation among the processors. The goal of this paper is the design and evaluation of different types of high-speed processor buses for multiprocessor system on a single chip. All five bus architectures we modeled have a separate 32-bit address bus and 64-bit data bus. GBIA and GBIIA are implemented similar to the conventional bus architecture of PCBs while BFBA, CSBA, and CCBA are more aggressive approaches.

3. Five Bus Architectures for SoC

We model five different bus architectures for a multiprocessor SoC. When we refer to a "Compute Node" (CN), we are referring to a processor together with a local SRAM for program memory (instruction cache) and a local SRAM for data memory (data cache). In this paper, we use the Motorola PowerPC (MPC) 750 for our processor core. Optional registers are added to CNs depending on the architecture and are described in the following subsections.
In GBIA, each CN shares a global bus and is synchronized with handshaking using shared registers between CNs. In GBIIA, an arbiter is added for all CNs to share the global bus. The third architecture is the Bi-FIFO bus architecture in which there are Bi-FIFOs to pass data between CNs. In the fourth architecture, we use a Crossbar Switch to provide multiple data paths among CNs. Finally, in the fifth architecture, we utilize the IBM CoreConnect [4] on-chip bus. The main differences lie in the way of synchronization and which CNs can communicate with which other CNs. For synchronization, GBIA and BFBA use shared registers, while GBIIA, CSBA, and CCBA utilize memory partitioning to preserve the data from the preceding CN. Only the adjacent CNs can communicate to each other in GBIA and BFBA while all CNs can pass data to any other CN in GBIIA, CSBA and CCBA. We model four CNs in all our multiprocessor architecture examples.

3.1 Global Bus I Architecture (GBIA)

Figure 1 illustrates GBIA in detail; as can be seen, CNs have a dedicated local bus (e.g., CPU Bus A) and a shared global bus. There are two registers, DONE_OP and DONE_RV, for handshaking between the communicating CNs. Each CN sets a flag in these registers after the data processing or the data receipt from the corresponding CN. The any CN can access the memory of upper adjacent CN through the segmented global bus. BB_x blocks (BB_1, BB_2, etc.) are bus bridges which allow different processors to access data memory. Note that while the GBIA is capable of allowing communication between lower adjacent or non-adjacent CNs, the arbitration protocol we employ limits the communication to upper adjacent CNs.

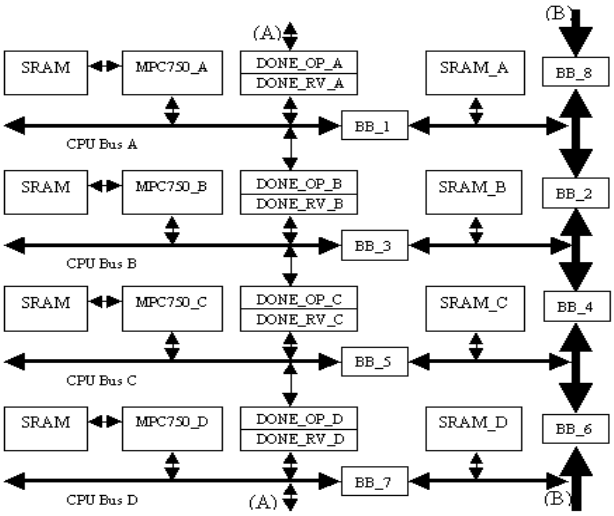

Figure 1: The diagram of global bus I architecture

Example 1: *Suppose that MPC 750_A writes to SRAM_A. Then the address decoder of MPC 750_A makes BB_1 connect to SRAM_A, and BB_2 and BB_8 block the access from any other CN. Next, MPC 750_B reads from SRAM_A. While the address decoder of MPC 750_B makes BB_1 disconnect from CPU Bus A, BB_2 and BB_3 are connected to CPU Bus B by the control of the address decoder. For the handshake operation between CN A and CN B, after MPC750_A sets DONE_OP_B at the completion of its operation, MPC750_B resets DONE_OP_B and then reads SRAM_A. When MPC 750_B finishes reading from SRAM_A, MPC 750_B sets DONE_RV_B to "1". MPC 750_A then resets*

DONE_RV_B to zero and begins processing the next packet.

3.2 Global Bus II Architecture (GBIIA)

In GBIIA, all CNs share one global bus as shown in Figure 2. Since the global bus can serve only one CN at a time, an arbiter is required to allocate the global bus to a CN when two or more CNs are contending for the global bus. Currently the arbiter grants the global bus in a FIFO fashion.

Twenty four memory locations (double words) in SRAM_D are reserved to store data ready flags. Data ready flags are written by the previous CN in the pipelined operation to indicate that its operation is complete, and data is available for the next CN to read and use. The CNs keep checking the data ready flags in SRAM_D and start their operation when the appropriate data ready flag value becomes one.

The arbitration protocol and data ready flag technique are also applied to the crossbar switch architecture.

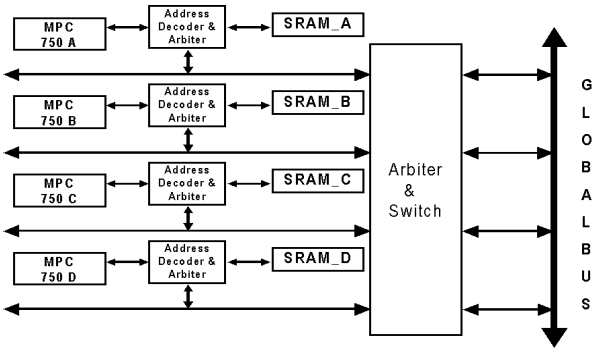

Figure 2: The diagram of global bus II architecture

3.3 Bi-FIFO Bus Architecture (BFBA)

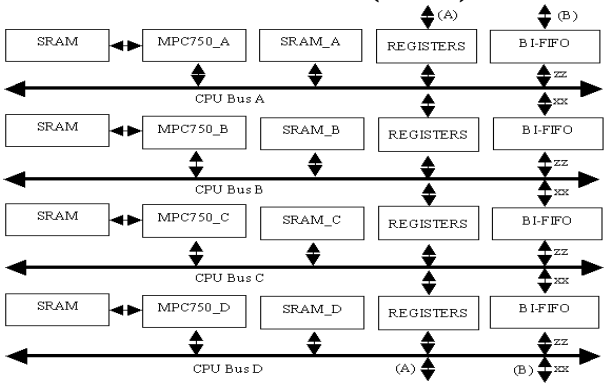

Figure 3: The diagram of Bi-FIFO bus architecture

Figure 3 shows the Bi-FIFO bus architecture. The data output by each CN can be exchanged through the Bi-FIFOs located between CNs. Each CN has two ports to access the Bi-FIFO as shown in Figure 3: the upper port, ZZ, and the lower port, XX. One CN can push the result data after the completion of the assigned operation to a Bi-FIFO, and the adjacent CN can read the data from the Bi-FIFO. For this operation, the user defines high and low threshold values for Bi-FIFO operation. A high threshold value indicates the Bi-FIFO is full while a low threshold value indicates the Bi-FIFO is empty. An interrupt signal is generated to the next CN when the data in a Bi-FIFO reaches the high threshold. The interrupted CN reads the data from Bi-FIFO until the data of the Bi-FIFO reaches the low threshold.

The synchronization issue between communicating CNs is resolved with the interrupt function and two flag registers, TX_DONE and RV_DONE, for handshaking. These two registers, plus the threshold registers, are contained in the "REGISTERS" block in Figure 3.

3.4 Crossbar Switch Bus Architecture (CSBA)

CSBA is an extended version of GBIIA. An array of transmission gates provides paths between all CNs and shared SRAMs as shown in Figure 4. In this architecture, each CN is composed of an MPC 750 and a local software SRAM (instruction cache). Each CN can access any shared SRAM A, B, C, or D at the same time if there is no competition for accessing the same SRAM block. When memory competition for the same shared SRAM occurs, an arbiter resolves this situation in a FIFO fashion.

To solve the synchronization problem between communicating CNs, SRAM_D has 24 data ready flags as was done in GBIIA.

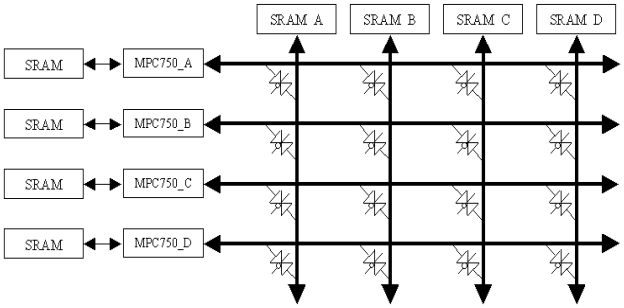

Figure 4: The diagram of crossbar switch bus architecture

3.5 IBM CoreConnect Bus Architecture (CCBA)

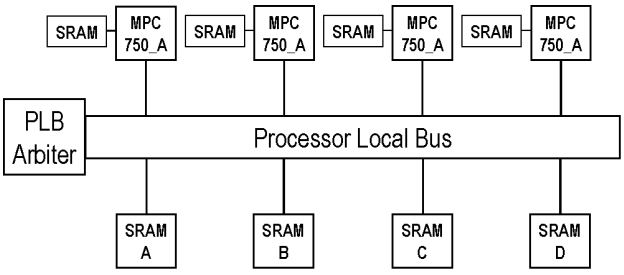

Figure 5: The diagram of IBM CoreConnect

Finally, we utilize the IBM CoreConnect [4] standard on-chip bus to compare the performance with our previous four bus designs. Since we focus on high performance, we just use the IBM CoreConnect Processor Local Bus (PLB) core as shown in Figure 5. MPC 750s are connected to the PLB through separate address, read, and write data buses with many control signals. SRAMs are also attached to the PLB. We designed the proper interfaces for MPC 750s and SRAMs. The PLB has separate read and write data buses similar to the approach of Winegarden [7]. We employ a fixed priority arbitration scheme.

We use the PLB core with 64-bit wide separate read and write buses provided by IBM under the license agreement.

4. Application Example

4.1 OFDM Transmitter Application

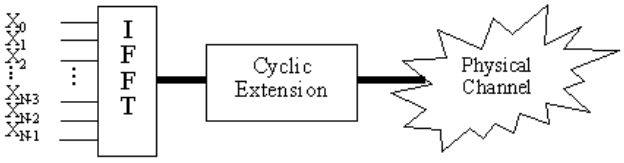

Figure 6: The block diagram of an OFDM transmitter

Five kinds of bus architectures for SoC were modeled and simulated with four-compute nodes (CNs): GBIA, GBIIA, BFBA, CSBA, and CCBA. For our first example, we use a wireless communication protocol, Orthogonal Frequency Division Multiplexing (OFDM)[9]. Specifically, we utilize the OFDM transmitter on all five bus architectures for their performance evaluation. Each CN executes the assigned OFDM functions, and the computed results are transferred to next CN as input data. The required data transactions occur with different methods according to the bus architecture. The bus performance is heavily dependent upon the arbitration scheme and the architecture to handle and reduce the bus contention among CNs.

OFDM employs several parallel channels with low bit rates whose main lobes of carriers are orthogonal and side lobes of carriers are overlapping one another. This is an efficient way of carrying several sub channels in a fixed bandwidth. The sub carriers are not separated by bandwidth but rather overlap their side lobes with each other. The frequency spacing between the sub carriers is arranged such that they become orthogonal. A Fast Fourier Transform (FFT) is used for digital modulation/demodulation of each sub channel.

Figure 6 shows the block diagram of an OFDM transmitter. The subchannels are modulated by an Inverse FFT (IFFT), and then a cyclic extension is added to avoid inter symbol interference caused by the physical channel. This extension is called the guard signal. From the physical channel's point of view, the extended OFDM symbol appears periodically.

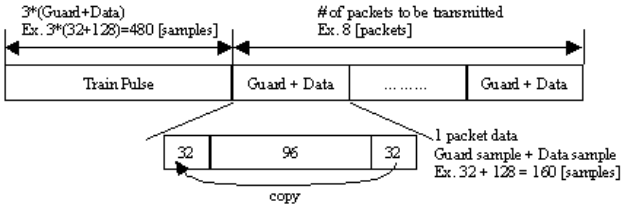

Figure 7: OFDM data format

Figure 7 shows the OFDM data format being transmitted. The OFDM data starts with a train pulse. The train pulse allows for the channel estimation and data synchronization at the receiver side. Guard and data packets follow the train pulse block. One packet of OFDM data we simulated here contains a 128-complex valued sample and a 32-complex valued guard signal. The guard data is usually a quarter of the data block.

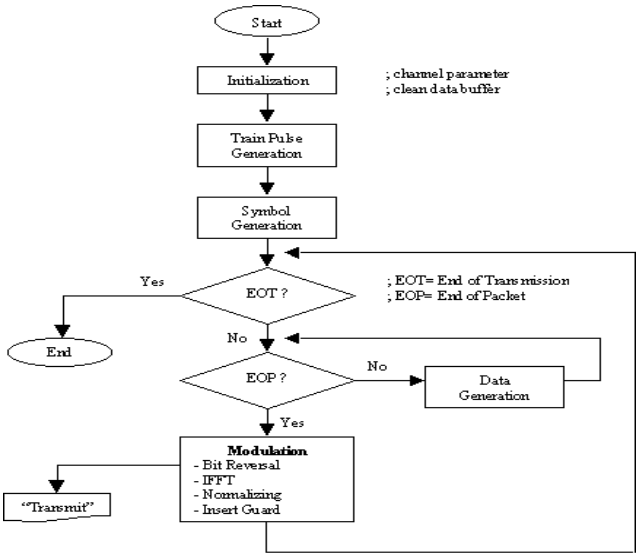

Figure 8: The flowchart of OFDM transmitter

Figure 8 shows the flow chart of the OFDM transmitter. The first three blocks are for train pulse generation and symbol generation which maps the original data to a symbol to be transmitted. The first three blocks (Initialization, Train Pulse Generation, and Symbol Generation) in Figure 8 are excluded in calculating throughput since these routines are executed only once at the startup. The End of Packet (EOP) loop controls data generation or data reading from an external device, which generates data to be transmitted. This EOP loop is repeated as many times as the size of the data packet, meanwhile, the outer loop is also repeated as many times as there are new data packets to be transmitted. The generated data is fed into the modulation

block, which executes bit reversal, IFFT, normalizing, and insertion of the guard signal, sequentially.

The job assignment in each CN proceeds after careful analysis of OFDM transmitter functions based on the flowchart. Table 1 outlines the assignment in each CN. The job on CN A seems heavier, but it is not the bottleneck of system because the first three functions listed for CN A (italicized in Table 1) are executed only once. Only data generation, symbol mapping and bit reversal functions are iterated in CN A. The job on CN B, IFFT, unfortunately is difficult to split up due to the structure of the IFFT.

Table 1: The function assignment in each compute node

Compute Node	Assigned Functions
A	*Initialization (channel parameter, etc)* *Train Pulse Generation* *Symbol Generation* Data Generation & Symbol Mapping Bit Reversal for Inverse FFT
B	Inverse FFT
C	Normalizing IFFT
D	Normalization Insertion of Guard Signal

4.2 MPEG2 Decoder Application

MPEG2 video is an ISO/IEC standard that specifies the syntax and the semantics of encoded video bit streams. These include parameters such as bit rates, picture sizes, and resolutions that may be applied, and how the video bit stream is decoded to reconstruct the picture.

Figure 9 shows input video frames and their processing on CNs. The video stream data is assumed as follows: M=1, N=2, where M is a period either of intra frame (I) or of predictive frame (P), and N is the number of pictures in a group of pictures. Each frame size is specified with 16 pixels by 16 pixels to reduce the simulation time in this application. All video frames that are input to CN A are distributed to each CNs, and each decoded frame is handed over to CN D at the end.

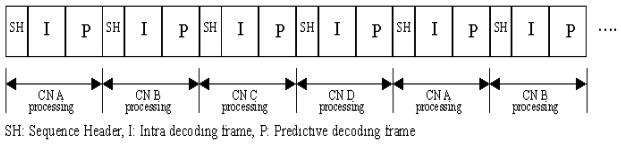

Figure 9: The input video frame of MPEG2

5. Experimental Results

For the bus architecture modeling and simulation, we use Seamless/CVE, the hardware/software co-simulator from Mentor Graphics [10], together with VCS, the Verilog HDL simulator from Synopsys [11]. Four MPC 750s from Seamless are employed for CPUs.

Each CN employs an MPC 750 with an 83.33 MHz CPU external clock speed, SYSCLK. The maximum frequency of SYSCLK, which dictates the maximum bus speed, is limited to 83.33 MHz in the PowerPC Hardware Specification (note that the internal clock speed can be much faster, e.g., 400 MHz) [12]. However, our results are equally applicable to much faster bus clock speeds.

Due to the sequential execution of functions among CNs, synchronization is required between communicating CNs. In this experiment, handshaking using shared registers between CNs, the generation of interrupt signals, and memory-partitioning techniques are adopted for synchronization.

5.1 Global BUS I Architecture (GBIA)

Figure 10 exhibits an eight-packet OFDM transmitter simulation waveform, and Figure 11 draws the pipelined operation. Figures 10 and 11 show CN A, CN B, CN C and CN D from top to bottom, respectively. There is a global bus at the bottom of both Figures 10 and 11 demonstrating the data transaction between CNs. After long processing of the train pulse and symbol generation in CN A, the first data packet is processed in CN A. CN B waits for the completion of CN A. The IFFT on CN B is then executed with the result from CN A in parallel with a new data packet being processed by CN A. CN C and CN D proceed with their assigned execution after receiving data from previous CNs. In this fashion, the pipelined operation achieves a significant increase in final data throughput. The output data packet is generated every 403,000 cycles which corresponds to 4.8360 ms/packet and 2.1175 Mbps.

Figure 10: The waveform of OFDM transmitter in GBIA

Note that in Figure 11, “w” refers to writing data to SRAM_{A, B, C, D} and “r” refers to reading data from SRAM_{A, B, C, D}. Note the pipelined flow of packets through the architecture.

The MPEG2 decoder is also simulated in Seamless and VCS. Each frame is decoded for 527,545 cycles which is same as 63305 ms/frame and 0.4852 Mbps.

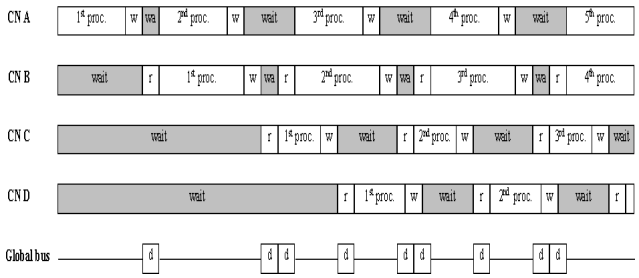

Figure 11: The pipeline operation of OFDM on GBIA

5.2 Global BUS II Architecture (GBIIA)

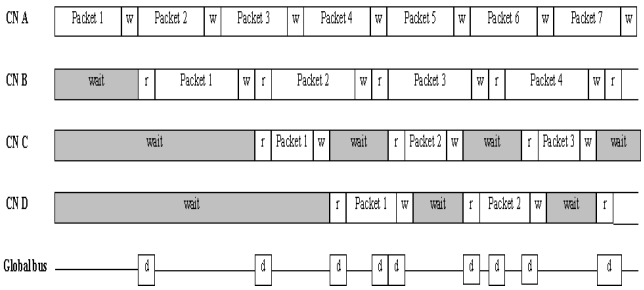

Figure 12: The pipeline operation of OFDM on GBIIA

Figure 12 shows the pipelined operation of OFDM on GBIIA. The bottleneck of the system arises from the IFFT function on CN B after it reads the result from CN A. CN C has to wait until CN B has completed the IFFT, and CN D is delayed until CN C finishes. The second bus transaction in the global bus at the bottom of Figure 11 shows that CN B and CN C utilize the bus alternately each reading a word at a time. This does not degrade the performance because each SRAM takes two cycles to access data, leaving, in general operation by one CPU, alternate cycles unused. An output data packet is generated every 381,061 cycles which corresponds to 4.5727 ms/packet and 2.2394 Mbps.

In MPEG2 application, each frame is decoded for 377,562 cycles which is equivalent to 4.5307 ms/frame, and the throughput is 0.6780 Mbps.

5.3 Bi-FIFO Bus Architecture (BFBA)

Figure 13 illustrates the pipelined operation of this architecture and shows the four CN operations, CN A, CN B, CN C, and CN D from top to bottom. Interrupt signals at the bottom are generated when the data in a Bi-FIFO reaches the high threshold so that the next CN starts to read the Bi-FIFO. An output data packet is produced every 378,348 cycles, which is equivalent to 4.5402ms/packet and 2.2554 Mbps.

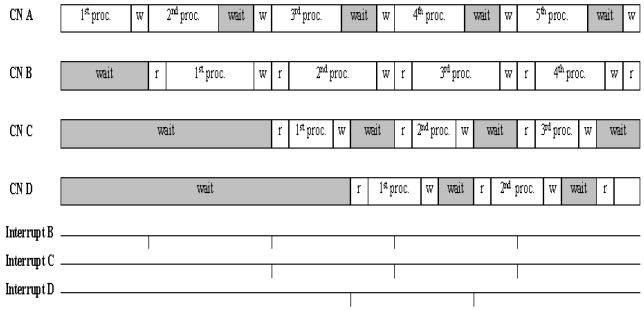

Figure 13: The pipeline operation of OFDM on BFBA

In the MPEG2 application, each frame is decoded for 507,853 cycles which corresponds to 6.0942 ms/frame and 0.5041 Mbps.

5.4 Crossbar Switch Bus Architecture (CSBA)

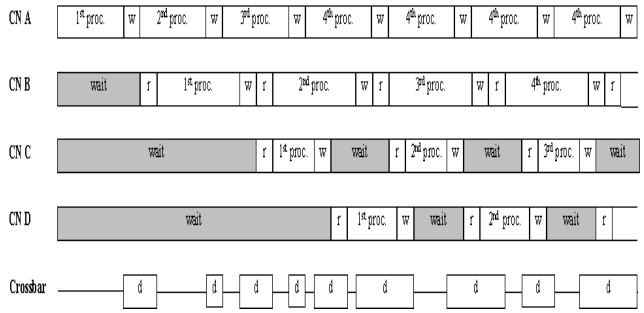

Figure 14: The pipeline operation of OFDM on CSBA

Figure 14 illustrates the pipelined operation of OFDM executing on CSBA. Since IFFT causes the bottleneck in the system performance, CSBA does not outperform any other bus architectures as much as we expected. This is because the pipelined operation is done at the function level rather than the data block level. The distinct feature of this architecture is the reduction in bus contention due to multiple point-to-point data paths such that any CPU can access any available SRAM on the crossbar switch. The output data packet is generated every 380,199 cycles, which is equivalent to 4.5624 ms/packet and 2.2444 Mbps.

In the MPEG2 application, each frame is decoded for 377,548 cycles which is equivalent to 4.5306 ms/frame, and the throughput is 0.6781 Mbps

5.5 IBM CoreConnect Bus Architecture (CCBA)

Figure 15 shows the pipelined operation of a current available standard on-chip bus architecture, CoreConnect from IBM [4]. The throughput we have measured from the simulation is 2.24156 Mbps. We use the 64 bit IBM

CoreConnect core with eight masters. This core requires an arbitration overhead of 2 cycles due to the complexity of the architecture. This architecture allows a data packet to be produced every 380, 686 cycles, equivalent to 4.5682 ms/packet and 2.2416 Mbps.

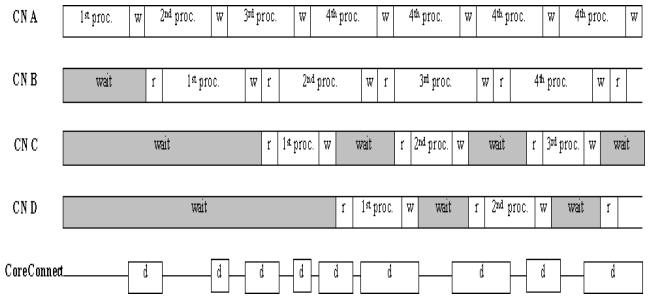

Figure 15: The pipeline operation of OFDM on CoreConnect

In the MPEG2 decoder, each frame is decoded for 378,141 cycles which is equivalent to 4.5382 ms/frame, and the throughput is 0.6769 Mbps.

5.6 Comparison of Results

Using OFDM with the same data input, Table 2 shows the result of our performance comparison for the bus architectures. As shown in Figure 16, all five bus architectures have almost the same throughputs since the OFDM application depends more on CPU processing time than on bus architecture. The IFFT on CN B is the bottleneck of the system.

The performance difference between BFBA and CSBA lies in the method of pointer increment. In BFBA, the Bi-FIFO controller moves the pointer to the next memory block, while in CSBA the pointer increment is done by the application code. The pointer increment done by the hardware is much faster than that by the software. Thus, read and write times in BFBA are 2675 cycles per packet less than those in CSBA, yielding a 0.0055 Mbps throughput difference.

The performance differences between GBIA and GBIIA arise from the synchronization protocols. In GBIA, there is a gap during the reading times from the next CN between write and read in the current CN. This can be seen in the gray "wait" boxes in Figure 11. Note especially the gray "wait" boxes between a "w" and a "r" for CN B which are not present for CN B in Figures 12, 13, 14, and 15. This allows GBIIA to outperform about 0.0609 Mbps in throughput.

The performance of the on-chip standard bus, CoreConnect from IBM, is in between the performance of CSBA and GBIIA. Table 2 demonstrates that the performance of a bus architecture does depend on the data transfer paths among communication entities. However, the performance improvement most heavily depends upon the application and the distribution of the application algorithm among the CNs.

Table 2: The performance comparison of bus architectures in OFDM transmitter

Bus Architectures	Exe. Cycle/Packet	Exe. Time/Packet	Throughput
BFBA	378,348	4.5402 ms	1.1277 Mbps
CSBA	380,199	4.5624 ms	1.1222 Mbps
CCBA	380,686	4.5682 ms	1.1208 Mbps
GBIIA	381,061	4.5727 ms	1.1197 Mbps
GBIA	403,000	4.8360 ms	1.0588 Mbps

Reference: 128 data samples and 32 guard samples per packet

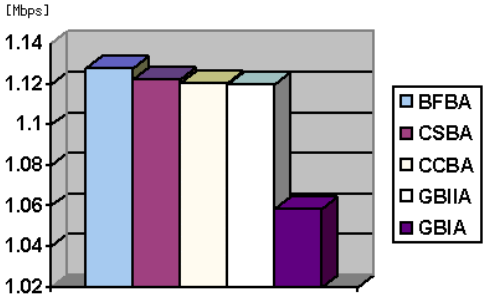

Figure 16: The throughput comparison of bus architectures in OFDM transmitter

Table 3: The performance comparison of bus architectures in MPEG2 decoder

Bus Architectures	Exe. Cycle/Frame	Exe. Time/Frame	Throughput
CSBA	377,548	4.5306 ms	0.6781 Mbps
GBIIA	377,562	4.5307 ms	0.6780 Mbps
CCBA	378,181	4.5382 ms	0.6769 Mbps
BFBA	507,853	6.0942 ms	0.5041 Mbps
GBIA	527,545	6.3305 ms	0.4852 Mbps

Reference: picture_size/frame=16pixel x16pixel, M=1, N=2

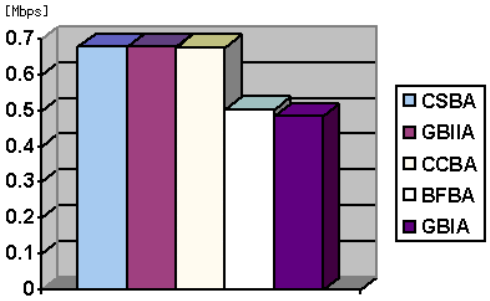

Figure 17: The throughput comparison of bus architectures In MPEG2 decoder

In the MPEG2 decoder shown in Table 3 and Figure 17, CSBA, GBIIA, and CCBA show better performance than BFBA and GBIA. The reason is the former three bus

architectures support global resource access, but, in the latter two, data has to be passed from CN A to each CN sequentially to supply the global data to be processed in each CN.

In the above comparison, the performance of a bus architecture is affected by the algorithm of application. For example, BFBA shows the best performance for an algorithm that has many local variables, small loops, and strong data dependency between functions because they can sequentially process functions with fast memory pointer increments between CNs. OFDM is a good example for these architectures. On the other hand, the MPEG2 decoder is a good example of an algorithm that needs to access many global variables, and has a big loop with its nested loops and hierarchy data structure. For those kinds of applications, CSBA, GBIIA and CCBA are the better choices than either BFBA or GBIA.

6. Conclusion

In this paper, we have presented five different multiprocessor SoC bus architectures: GBIA, GBIIA, BFBA, CSBA, and CCBA. We have compared performance employing an OFDM transmitter and MPEG2 decoder application. Each architecture can significantly increase performance with pipelined operation among CNs. However, the overall system performance is limited by the bottleneck of one CN executing IFFT in OFDM transmitter, resulting in an unbalanced job distribution in the pipelined operation.

As seen in OFDM part of section 5.6, the Bi-FIFO and Crossbar Switch bus architectures outperform GBIA, GBIIA and CCBA although the performance difference is not significant. We would expect an example with less regularity to perform much better on GBIIA and CBSA since GBIIA and CSBA allow direct communication between non-adjacent CNs. In the MPEG2 decoder application, CSBA, GBIIA, and CCBA outperform BFBA and GBIA because they allow global resources access unlike BFBA and GBIA. From the above comparisons, bus architectures for a certain system must be determined by the type of application, and we provide a certain level of practical guidelines for the selection of a bus architecture in section 5.6.

The main contribution of this paper is the exploration of high performance multiprocessor SoC bus architectures, which has not been explored in previous papers [3-8]. For our future work, we intend to examine more diverse applications, carry out more detailed studies on high speed bus design including interconnect delay, and propose a new bus architecture combining the best characteristics of the five designs presented in this paper.

7. Acknowledgement

This research is funded by the State of Georgia under the Yamacraw initiative [13] and by NSF under INT-9973120, CCR-9984808 and CCR-0082164.

We also acknowledge software donations from Mentor Graphics and Synopsys as well as hardware donations from Sun and Intel.

8. References

[1] VITA-VME bus International Trade Association, http://www.vita.com

[2] The PCI Special Interest Group, http://www.pcisig.com

[3] A. M. Rincon, W.R. Lee, and M. Slattery, “The Changing Landscape of SoC Design,” Proceedings of IEEE 1999 Custom Integrated Circuits Conference, May 1999, pp. 83-90.

[4] CoreConnect Bus Architecture, http://www.chips.ibm.com/products/coreconnect

[5] P. J. Aldworth, “System-on-a-Chip Bus Architecture for embedded Applications,” International Conference on Computer Design (ICCD’99), 1999, pp. 297 –298.

[6] B. Cordan, “An Efficient Bus Architecture for System-on-a-Chip Design,” Proceedings of IEEE Custom Integrated Circuits Conference, May 1999, pp. 623-626.

[7] S. Winegarden, “Bus Architecture of a System on a Chip with User Configurable System Logic,” IEEE Journal of Solid State Circuits, March 2000, Vol. 35, No. 3, pp. 425-433.

[8] D. Wingard and A. Kurosawa, “Integration Architecture for System-on-a-Chip Design,” Proceedings of IEEE 1998 Custom Integrated Circuits Conference, May 1998, pp. 85-88.

[9] D. Kim and G. L. Stüber, "Performance of Multiresolution OFDM on Frequency-selective Fading Channels," IEEE Transaction on Vehicular Technology, vol. 48, no. 5, pp. 1740-1746, September 1999.

[10] Seamless co-verification, http://www.mentor.com/seamless

[11] VCS data sheet, http://www.synopsys.com/products/simulation/vcs_ds.html

[12] MPC 750A RISC Microprocessor Hardware Specification, http://www.mot.com/SPS/PowerPC/library/750_hs.pdf

[13] Yamacraw, http://www.yamacraw.org