

FLEXBUS: A High-Performance System-on-Chip Communication Architecture with a Dynamically Configurable Topology *

Krishna Sekar
Dept. of ECE
UC San Diego, CA 92093
ksekar@ece.ucsd.edu

Kanishka Lahiri
NEC Laboratories America
Princeton, NJ 08540
klahiri@nec-labs.com

Anand Raghunathan
NEC Laboratories America
Princeton, NJ 08540
anand@nec-labs.com

Sujit Dey
Dept. of ECE
UC San Diego, CA 92093
dey@ece.ucsd.edu

ABSTRACT

In this paper, we describe FLEXBUS, a flexible, high-performance on-chip communication architecture featuring a dynamically configurable topology. FLEXBUS is designed to detect run-time variations in communication traffic characteristics, and efficiently adapt the *topology* of the communication architecture, both at the system-level, through *dynamic bridge by-pass*, as well as at the component-level, using *component re-mapping*. We describe the FLEXBUS architecture in detail and present techniques for its run-time configuration based on the characteristics of the on-chip communication traffic. The techniques underlying FLEXBUS can be used in the context of a variety of on-chip communication architectures. In particular, we demonstrate its application to AMBA AHB, a popular commercial on-chip bus. Detailed experiments conducted on the FLEXBUS architecture using a commercial design flow, and its application to an IEEE 802.11 MAC processor design, demonstrate that it can provide significant performance gains as compared to conventional architectures (up to 31.5% in our experiments), with negligible hardware overhead.

Categories and Subject Descriptors: B.4.3 [Input/Output and Data Communications]: Interconnections (Subsystems)

General Terms: Algorithms, Design, Performance

Keywords: Communication architectures, On-chip bus

1. INTRODUCTION

The on-chip communication architecture has emerged as a critical determinant of overall performance in complex System-on-Chip (SoC) designs, which makes it crucial to customize it to the characteristics of the traffic generated by the application. Since application traffic can exhibit significant diversity, configurable communication architectures that can be easily adapted to an application's requirements are desirable.

As shown in this paper, complex SoC designs can benefit from communication architectures that can be *dynamically configured* in order to adapt to changing traffic characteristics. Most state-of-the-art communication architectures provide limited customization opportunities through a *static* (design time) configuration of architectural parameters, and as such, lack the flexibility to provide high performance in cases where the traffic profiles exhibit significant dynamic variation.

In this paper, we describe FLEXBUS, a flexible communication architecture that detects run-time variations in system-wide communication traffic characteristics, and efficiently adapts the *logical connectivity* of the communication architecture and the components connected to it. This is achieved using two novel techniques, *bridge by-pass*, and *component re-mapping*, which address configurability at the system and component levels, respectively. These techniques provide op-

portunities for dynamically controlling the *spatial allocation* of communication architecture resources among different SoC components, a capability, which if properly exploited, can yield substantial performance gains. We also describe techniques for choosing optimized FLEXBUS configurations under time-varying traffic profiles. We have conducted detailed experiments on FLEXBUS using a commercial design flow to analyze its area and performance under a wide variety of system-level traffic profiles, and applied it to an IEEE 802.11 MAC processor design. The results demonstrate that FLEXBUS provides up to 31.5% performance gains compared to conventional architectures, with negligible hardware overhead.

1.1 Related Work

The increasing importance of on-chip communication has resulted in numerous advances in communication architecture topology and protocol design [1]-[5]. However, these architectures are based on fixed topologies, and therefore, are not fully optimized for traffic with time-varying characteristics. The use of sophisticated communication protocols [3], [6] and techniques for customizing these protocols, both statically and dynamically [7], [8], [9], have been proposed. Protocol and topology customization are complementary, and hence, may be combined to yield large performance gains. A number of automatic approaches have been proposed to statically optimize the communication architecture topology [10], [11]. While many of these techniques aim at exploiting application characteristics, they do not adequately address dynamic variations in communication traffic profiles.

2. BACKGROUND

Communication architecture *topologies* can range from a single shared bus, to which all the system components are connected, to a network of bus segments interconnected by *bridges*. *Component mapping* refers to the association between components and bus segments. Components can be either *masters* (e.g., CPUs, DSPs), which can initiate transactions (reads/writes), or *slaves*, (e.g., memories, peripherals), which only respond to transactions initiated by a master. *Communication protocols* specify conventions for the data transfer (e.g., arbitration policies, burst transfer modes). *Bridges* are specialized components that enable transactions between masters and slaves located on different bus segments. "Cross-bridge" transactions execute as follows: the transaction request from the master, once granted by the first bus, is registered by the bridge's slave interface. The bridge forwards the transaction to its master interface, which then requests access to the second bus. On being granted, the bridge executes the transaction with the destination slave and returns the response to its slave interface, which finally returns the response to the original master.

3. MOTIVATIONAL EXAMPLE: IEEE 802.11 MAC PROCESSOR

In this section, we use an IEEE 802.11 MAC processor design as an example to illustrate the concepts behind the FLEXBUS architecture. The functional specification of the IEEE 802.11 MAC processor consists of a set of communicating tasks, shown in Figure 1(a) (details of the 802.11 MAC protocol are available in [12]). For outgoing frames, the LLC task receives frames from the Logical Link Control Layer. The Wired Equivalent Privacy (WEP) task and the Integrity Checksum Vector (ICV) task work in conjunction to encrypt and compute a checksum of the frame, respectively. The HDR task

*This work was supported by the UC Discovery Grant (grant# com02-10123), the Center for Wireless Communications (UC San Diego), and NEC Laboratories America.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2005, June 13-17, 2005, Anaheim, California, USA.

Copyright 2005 ACM 1-59593-058-2/05/0006 ...\$5.00.

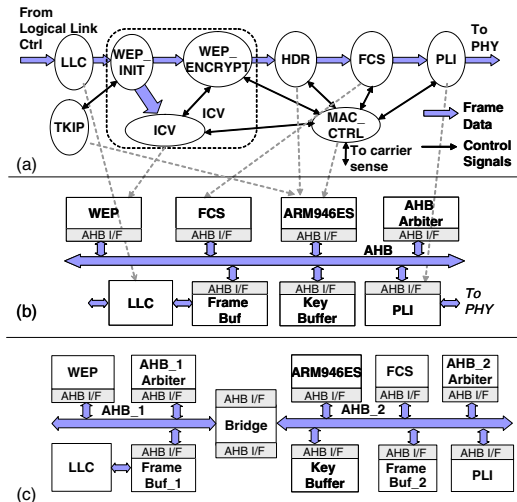


Figure 1: IEEE 802.11 MAC processor:(a) functional specification, mapping to a (b) single shared, (c) multiple bus architecture

generates the MAC header. The Frame Check Sequence (FCS) task computes a CRC-32 checksum over the encrypted frame and header. MAC_CTRL implements the CSMA/CA algorithm, determines frame transmit times, and signals the Physical Layer Interface (PLI) task to transmit the frames. The Temporal Key Integrity Protocol (TKIP) generates a sequence of encryption keys dynamically. If TKIP is disabled, the key is statically configured.

Figure 1(b) shows the set of components to which the above tasks are mapped in our design. An embedded processor (the ARM946ES) implements the MAC_CTRL, HDR, and TKIP tasks, while dedicated hardware units implement the LLC, WEP, FCS, and PLI tasks. In the next two sub-sections, we focus on only the tasks that result in the majority of the communication traffic: WEP, FCS, and TKIP.

3.1 Statically Configured Topologies

Example 1: We first consider an architecture where a single AMBA AHB bus segment [2] integrates all the system components (Figure 1(b)). The MAC frames are stored in `Frame_buf`, and processed in a pipelined manner: the WEP component encrypts a MAC frame, and then signals the FCS component to start computing a checksum on the encrypted frame, while it starts encrypting the next frame. When the keys are statically configured (the TKIP task is disabled), the on-chip communication traffic is largely due to the WEP and FCS components. In such a scenario, simultaneous attempts by the WEP and FCS hardware to access the system bus lead to a large number of bus conflicts, resulting in significant performance loss. Experiments indicate that the maximum data rate under this architecture is 188 Mbps. When TKIP is enabled, additional traffic between the processor and the key buffer further degrades the data rate to 158 Mbps. ■

This bus topology can fail to provide high performance by failing to exploit parallelism in the communication transactions. This can be addressed by using an architecture that uses multiple bus segments.

Example 2: Figure 1(c) presents a version of the MAC processor architecture which consists of two AHB segments connected by a bridge. The WEP component reads frame data from memory `Frame_buf1`, encrypts it, and then transfers the encrypted frame into `Frame_buf2`. The FCS component processes the frame from `Frame_buf2`, while WEP starts processing the next frame from `Frame_buf1`. We first assume that the keys are statically configured. The parallelism offered by the multiple bus architecture enables the FCS and WEP tasks to process frame data stored in their local frame buffers concurrently. However, the frame copy phase from `Frame_buf1` to `Frame_buf2` incurs large latencies due to the complex nature of cross-bridge transactions (Section 2). Experiments show that the data rate achieved by this architecture is 201 Mbps, only a 7% improvement over the single shared bus. When the TKIP task is enabled, the improvement over the shared bus is 11% (176 Mbps). ■

In summary, we observe that when the proportion of cross-bridge traffic is low, the multiple bus architecture performs well, whereas at other times, the single shared bus architecture is superior.

3.2 Dynamic Topology Configuration

We next consider the execution of the IEEE 802.11 MAC processor under two variants of the FLEXBUS architecture.

Example 3: We first assume that the keys are statically configured. Under FLEXBUS, the architecture operates in a multiple bus mode during intervals that exhibit localized communications, and hence enables concurrent processing of the WEP and FCS tasks. In intervals that require low latency communications between components located on different bus segments, a *dynamic bridge by-pass* mechanism temporarily transforms the architecture into a single shared bus topology. The measured data rate under this architecture was 248 Mbps, a 23% improvement over the best statically configured architecture. ■

The bridge by-pass mechanism provides a technique to make *coarse-grained* (system level) changes to the communication architecture topology. However, at times, the ability to make more fine-grained (component level) changes to the topology is also important, as illustrated next.

Example 4: When the TKIP task is enabled, the FLEXBUS architecture with dynamic bridge by-pass achieves a data rate of 208 Mbps, which is 18% better than the best conventional architecture. We next consider the application of a *dynamic component re-mapping* capability. Using this technique, while the overall architecture remains in the multiple bus configuration, the mapping of the slave `Frame_buf2` is dynamically switched between the two buses. It is mapped to AHB.2 as long as the FCS and WEP components are processing frame data, and to AHB.1 at times when the most recently encrypted frame needs to be efficiently transferred from `Frame_buf1` by the WEP task. Under this architecture, we observed a maximum data rate of 224 Mbps, a 27% improvement over the best static architecture. ■

In summary, the above illustrations establish that by recognizing dynamic variations in the spatial distribution of communication transactions, and correspondingly adapting the communication architecture topology (both at the system and component levels), large performance gains can be achieved. In the next two sections, we describe how this principle is realized in the FLEXBUS architecture.

4. FLEXBUS ARCHITECTURE

FLEXBUS consists of a dynamically configurable communication architecture topology. The techniques underlying FLEXBUS are independent of specific communication protocols, and hence can be applied to a variety of on-chip communication architectures. In our work, we demonstrate its application to the AMBA AHB [2], a popular commercial on-chip bus. FLEXBUS provides two different topology customization opportunities: (i) dynamic bridge by-pass (Section 4.1), which enables system level customization through run-time fusing and splitting of bus segments, and (ii) dynamic component re-mapping (Section 4.2), which enables component level customization through run-time switching of components from one bus segment to another.

The key technical challenges in developing FLEXBUS are: (i) maintaining compatibility with existing on-chip bus standards; (ii) minimizing timing impact; (iii) minimizing logic and wiring complexity (hardware overhead); (iv) providing low reconfiguration penalty. The rest of this section provides details on how FLEXBUS provisions for dynamic configurability keeping the above goals in mind.

4.1 Dynamic Bridge By-pass

Figure 2 shows the hardware required to support dynamic bridge by-pass for a system consisting of two AMBA AHB bus segments, connected by a bridge. AHB1 (primary bus) has two masters and one slave, and AHB2 (secondary bus) has one master and one slave.

Operating Modes: The system can be operated in (i) multiple bus mode and (ii) single shared bus mode, by disabling or enabling bridge by-pass, respectively, via the `config_select` signal. In the multiple bus mode, the signals shown by the dotted arrows are inactive. The two bus segments operate concurrently, with each arbiter resolving conflicts among the masters in its own bus segment. Transactions between masters on AHB1 and slaves on AHB2 go through the bridge using the conventions described in Section 2. In the single shared bus mode, the bridge is “by-passed” by setting `config_select = 1`, which activates the signals shown by the dotted arrows. In this mode, the inputs of the bridge master and slave interfaces are directly routed to the outputs, by-passing the internal bridge logic (using multiplexers). This allows

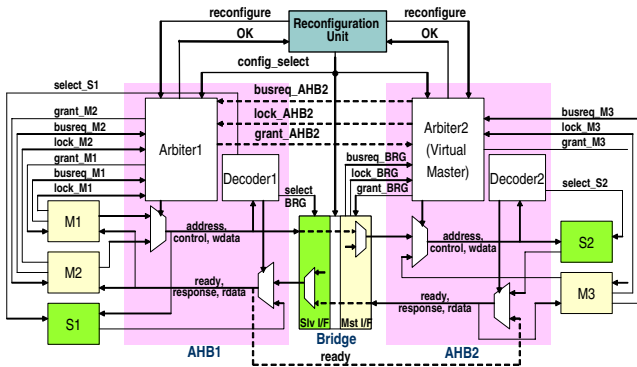


Figure 2: Dynamic bridge by-pass capability in FLEXBUS

transaction requests from masters on AHB1 to slaves on AHB2 (and the corresponding slave responses) to reach within one clock cycle.

Arbitration: In the multiple bus mode, more than one master may have transactions executing in parallel, whereas in the single bus mode, only one master can be granted access to FLEXBUS at any given time. Clearly, the arbitration mechanisms of the multiple bus mode need to be adapted for the single bus mode. For this, we use a *distributed arbitration mechanism* in the single bus mode, in which one of the arbiters behaves as a *virtual master* that is regulated by the other arbiter (Figure 2). On receiving one or more bus requests from masters on AHB2, Arbiter2 immediately sends a bus request to Arbiter1 using the *busreq_AHB2* and *lock_AHB2* signals. Arbiter1 arbitrates among the received bus requests from masters on AHB1 as well as the virtual master. In parallel, in order to reduce arbitration latency, Arbiter2 arbitrates among its received bus requests. However, it grants AHB2 to its selected master only when it receives the *grant_AHB2* signal from Arbiter1. This guarantees that only one master is granted access to FLEXBUS when in the single bus mode. Finally, the *ready* bus signal, which in the AMBA AHB bus protocol indicates the state of the bus to all the components, is routed from AHB1 to AHB2 to ensure correct system operation in the single bus mode.

Address Decoding: The address decoders on the two bus segments require no change to enable dynamic bridge by-pass.

Run-time Configuration: The Reconfiguration Unit (Figure 2) is responsible for selecting the bus configuration at run-time (using policies such as described in Section 5), and for ensuring correctness of system operation while switching between the two configurations. It initiates bus reconfiguration by asserting the *reconfigure* signal to the arbiters, which then terminate their current transactions, deassert all grant signals, and assert the *OK* signal. On receiving the *OK* signal from both the arbiters, the Reconfiguration Unit toggles the *config_select* signal. The worst case overhead of bus reconfiguration for the two bus segment AMBA bus system is 17 cycles (assuming the bus is not locked and all slaves have single cycle response).

4.2 Dynamic Component Re-mapping

Figure 3 shows a two bus segment AHB architecture, wherein master M2 and slave S2 can be dynamically re-mapped to AHB1 or AHB2.

Operating Modes: The mapping of M2 and S2 is selected by the signals, *config_select_M2* and *config_select_S2*, respectively. The signals of a re-mappable master or slave are connected to both AHB1 and AHB2. However, the switch boxes, SWITCH_M and SWITCH_S, activate the signals to and from only one of the buses, depending on the configuration. Note that, only a subset of the master and slave signals require to be switched in the AMBA AHB protocol.

Arbitration: The arbiters require no change, as master requests are only sent to the arbiter on the bus to which they are currently mapped.

Address Decoding: The address decoders on the two bus segments are reconfigured to generate the correct select signal for the re-mappable slave. For example, when S2 is mapped to AHB1, on observing an address belonging to S2 on the address bus, Decoder1 asserts the *select_S2* signal, while Decoder2 asserts the *select_BRG* signal. The situation is reversed when S2 is mapped to AHB2. This is done by switching the mapping of S2's address space between BRG and S2 in the decoder, based on the *config_select_S2* signal.

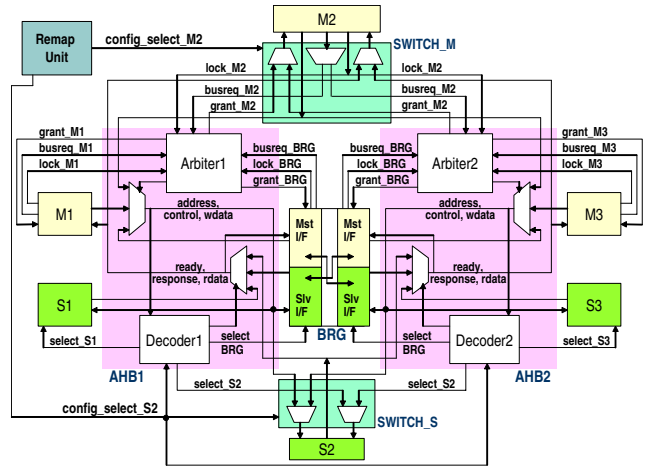


Figure 3: Dynamic component re-mapping capability in FLEXBUS

Run-time Configuration: The master and slave configuration select signals are generated by the Remap Unit. In order to ensure correct operation of the system during re-mapping, the Remap Unit monitors the master's *busreq* and slave's *select_S2* signals to determine if they are currently active on the bus. If not, the corresponding *config_select* signal is toggled. The rest of the bus operates without interruption.

5. DYNAMIC CONFIGURATION POLICIES

The problem of dynamically choosing the optimum configuration of the FLEXBUS architecture based on an observation of the characteristics of the communication traffic can be addressed using several approaches. In the past, similar problems in the domain of dynamic power management have been addressed using stochastic policies [13] and history-based heuristic techniques [14]. In our work, we examine a history-based technique in detail.

Let us consider a FLEXBUS system featuring dynamic bridge by-pass between two bus segments, *BUS1* and *BUS2*. Let N_{BUS1} , N_{BUS2} and N_{BRG} represent the number of local transactions on *BUS1*, number of local transactions on *BUS2* and number of transactions between the two bus segments, respectively, during an observation interval. A transaction refers to one bus access (e.g., a burst of 5 beats constitutes 5 transactions). The time taken to process this traffic under the single bus mode, T_{Single} , is $(N_{BUS1} + N_{BUS2} + N_{BRG}) \times C_L \times t_{SB}$, where C_L is the average number of cycles for a local bus transaction, and t_{SB} is the clock period in the single bus mode, since all transactions are on the same bus. Similarly, the time taken under the multiple bus mode, $T_{Multiple}$, is approximated by $\max(N_{BUS1}, N_{BUS2}) \times C_L \times t_{MB} + N_{BRG} \times C_B \times t_{MB}$, where C_B is the average number of cycles for a cross-bridge transaction, and t_{MB} is the clock time period in the multiple bus mode. If $T_{Single} > T_{Multiple}$, then the single bus mode is preferred, else the multiple bus mode is better. At run-time, the system observes the two bus segments over a time period TP and records the number of bus transactions of each type. At the end of the time period, it selects the new configuration based on the above criterion.

The choice of an appropriate configuration time period, TP , is crucial. Smaller time periods enable the policy to be more responsive to variations in the traffic characteristics. However, if the traffic profile changes rapidly, this might lead to excessive oscillations between the configurations, thus potentially degrading the performance due to the reconfiguration overhead. Therefore, we propose the use of an adaptive time period, TP , which is selected as follows. Let C denote the number of configurations of the bus over τ clock cycles. If $C/\tau > \lambda_1$, then the time period, TP , is doubled, if $C/\tau < \lambda_2$, then TP is halved, else it is unchanged. λ_1 and λ_2 represent two thresholds, and depend on the reconfiguration overhead. In our experiments, we observed an average reconfiguration time of 10 cycles, for which a τ value of 250 cycles, a λ_1 value of 0.0025, and a λ_2 value of 0.001 proved effective. In general, these parameters should be carefully set, based on an analysis of the traffic characteristics of the application.

Similar policies can also be designed for dynamic component re-mapping by monitoring, for each re-mappable master and slave, the fraction of traffic to or from components on the different bus segments.

Bus Architecture	Area (sq. mm)	Delay (ns)	Frequency (MHz)
Single Shared Bus	82.12	4.59	218
Multiple Bus	84.27	3.79	264
FLEXBUS (single bus mode)	82.66	4.72	212
FLEXBUS (multiple bus mode)		3.93	254

Figure 4: FLEXBUS hardware implementation results

6. EXPERIMENTAL RESULTS

In this section, we present experimental studies that evaluate the hardware implementation of FLEXBUS, followed by an analysis of its performance using (a) synthetic traffic profiles, and (b) traffic generated by the IEEE 802.11 MAC processor.

FLEXBUS was implemented by enhancing reference AMBA AHB RT-level implementations available in the Synopsys Designware library [15]. For the synthetic workload experiments, we used a system consisting of eight masters and eight slaves, equipped with programmable VERA bus-functional models [15] for traffic generation. The MAC processor was implemented using an instruction set model for the ARM processor, and Verilog for the remaining hardware. Performance results were obtained using ModelSim [16] simulations. For accurate chip-level area and delay comparison, we generated floorplans of all the systems [17] for a commercial 0.13 μ m technology [18]. The floorplanner was modified to report global wirelengths for different communication architectures. Global wire delays were calculated assuming delay optimal buffer insertion [19] and Metal 6 wiring. The designs were annotated with these wire delays and Synopsys Design Compiler [20] was used for delay estimation.

The area and timing analysis methodology described above was applied to the example eight master and eight slave system under (i) FLEXBUS with dynamic bridge by-pass, (ii) single shared bus, and (iii) multiple bus architectures. Figure 4 shows the results of these studies. For the FLEXBUS, the critical path delay in the multiple bus mode is smaller than in the single bus mode since many long paths of the single bus mode are false paths in the multiple bus mode. The static multiple bus architecture has smaller delay than the single shared bus due to shorter wirelengths and lesser bus loading. FLEXBUS incurs a small delay penalty (on average 3.2%) compared to the statically configured architectures, due to extra wiring and logic delay.

To analyze FLEXBUS's performance under synthetic traffic profiles, we generated traffic profiles using a two-state Markov model, where each state corresponds to either local traffic (*i.e.*, traffic on the same bus segment) or cross-bridge traffic (*i.e.*, traffic between different bus segments). Varying the state transition probabilities allowed us to vary the granularity with which the two traffic types are interleaved. Figure 5(a) shows a representative traffic profile consisting of a mix of local and cross-bridge traffic. Figure 5(b) shows the run-time configuration decisions taken by the policy (Section 5). Figure 5(c) plots the cumulative latency for the different architectures. We observe that FLEXBUS successfully adapts to frequent changes in the traffic characteristics, achieving significant performance benefits over the static single shared bus (21.3%) and multiple bus (17.5%) architectures.

Finally, we examine the performance of FLEXBUS and conventional architectures for the IEEE 802.11 MAC processor described in Section 3. We considered two variants of FLEXBUS, featuring (i) dynamic bridge by-pass, and (ii) dynamic component re-mapping (Frame_buf2 is the re-mappable slave). The policy described in Section 5 is used to select the bus configuration. All the buses were operated at 200 MHz. Table 1 shows the average time taken to process a single frame (of size 1 KB) under the different bus architectures. From the table, we see that the times required by both variants of FLEXBUS are significantly smaller compared to the conventional architectures. The data rate increase due to FLEXBUS over the single shared bus is 31.5% and over the multiple bus is 23%. The table also shows the upper bound on performance, obtained using an ideal reconfigurable bus (no reconfiguration overhead) with an ideal reconfiguration policy (full knowledge of future bus traffic). We observe that for this system, FLEXBUS and its associated policies perform close to the ideal case.

7. CONCLUSIONS

In this paper, we presented FLEXBUS, a novel dynamically configurable on-chip communication architecture, and described techniques for efficiently adapting FLEXBUS at run-time. In future work, we plan to extend FLEXBUS to more complex communication architectures.

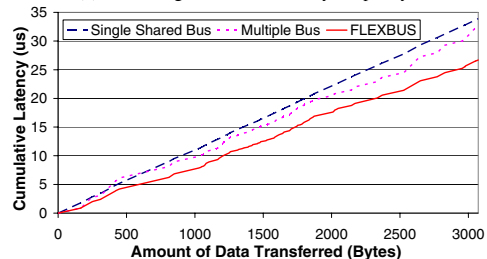
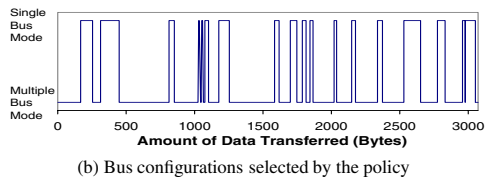
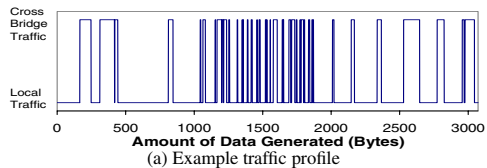


Figure 5: FLEXBUS performance under synthetic traffic profiles

Table 1: Performance of the IEEE 802.11 MAC processor under different communication architectures

Bus Architecture	Computation Time (ns)	Data Transfer Time (ns)	Total Time (ns)
Single Shared Bus	42480	-	42480
Multiple Bus	26905	12800	39705
FLEXBUS (Bridge By-pass)	27025	5290	32315
FLEXBUS (Comp. Re-mapping)	27010	5270	32280
Ideally Reconfigurable Bus	26905	5120	32025

8. REFERENCES

- [1] "CoreConnect Bus Architecture." <http://www.chips.ibm.com/products/coreconnect/>.
- [2] "AMBA 2.0 Specification." <http://www.arm.com/armtech/AMBA>.
- [3] D. Wingard and A. Kurosawa, "Integration Architecture for System-on-a-Chip Design," in *Proc. Custom Integrated Circuits Conf.*, pp. 85–88, 1998.
- [4] A. Adriahtenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network," in *Proc. Design Automation & Test Europe (DATE) Conf.*, pp. 70–73, 2003.
- [5] S. Han, A. Baghdadi, M. Bonaciu, S. Chae, and A. A. Jerraya, "An Efficient Scalable and Flexible Data Transfer Architecture for Multiprocessor SoC with Massive Distributed Memory," in *Proc. Design Automation Conf.*, pp. 250–255, June 2004.
- [6] R. Yoshimura, K. T. Boon, T. Ogawa, S. Hatanaka, T. Matsuoka, and K. Taniguchi, "DS-CDMA Wired Bus With Simple Interconnection Topology for Parallel Processing System LSI's," in *Proc. Int. Solid-State Circuits Conf.*, pp. 370–371, 2000.
- [7] K.Lahiri, A.Ragunathan, and S.Dey, "Design of High-Performance System-on-Chips Using Communication Architecture Tuners," *IEEE Trans. on CAD*, vol. 23, no. 6, pp. 919–932, 2004.
- [8] T. Meyerowitz, C. Pinello, and A. Sangiovanni-Vincentelli, "A Tool for Describing and Evaluating Hierarchical Real-Time Bus Scheduling Policies," in *Proc. Design Automation Conf.*, pp. 312–317, June 2003.
- [9] J. Hu and R. Marculescu, "DyAD — Smart Routing For Networks-on-Chip," in *Proc. Design Automation Conf.*, pp. 260–263, June 2004.
- [10] A. Pinto, L. P. Carloni, and A. Sangiovanni-Vincentelli, "Constraint-Driven Communication Synthesis," in *Proc. Design Automation Conf.*, pp. 783–788, June 2002.
- [11] S.Pasricha, N.Dutt, and M.B.Romdhane, "Fast Exploration of Bus-based On-chip Communication Architectures," in *Proc. Int. Symp. HW/SW Codesign*, Sept. 2004.
- [12] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." IEEE Computer Society LAN/MAN Standards Committee, IEEE Std 802.11-1999 Edition.
- [13] L.Benini, A. Bogliolo, G.Paleologo, and G.D.Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Trans. on CAD*, vol. 18, pp. 813–833, June 1999.
- [14] F. Douglas, P. Krishnan, and B. Bershad, "Adaptive Disk Spin-Down Policies for Mobile Computers," in *USENIX Symp. Mobile and Location Independent Computing*, pp. 121–137, Apr. 1995.
- [15] "Synopsys DesignWare Intellectual Property." <http://www.synopsys.com/products/designware/designware.html>.
- [16] "Modelsim 5.7e." <http://www.model.com>.
- [17] W. Dai, L. Wu, and S. Zhang, "UCSC Floorplanning Tool." <http://www.soe.ucsc.edu/research/surf/GSRC/progress.html>.
- [18] "CB-12." <http://www.necel.com/cbic/en/cb12/cb12.html>.
- [19] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley, Menlo Park, CA, 1990.
- [20] "Design Compiler 2003.12, Synopsys Inc." http://www.synopsys.com/products/logic/design_compiler.html.