Automated Throughput-driven Synthesis of Bus-based Communication Architectures

Sudeep Pasricha†, Nikil Dutt†, Mohamed Ben-Romdhane‡

†Center for Embedded Computer Systems
University of California, Irvine, CA
{sudeep, dutt}@cecs.uci.edu

‡Conexant Systems Inc.
Newport Beach, CA
m.benromdhane@conexant.com

Abstract – **As System-on-Chip (SoC) designs become more complex, it becomes increasingly harder to design communication architectures which satisfy design constraints. Manually traversing the vast communication design space for constraint-driven synthesis is not feasible anymore. In this paper we propose an approach that automates the synthesis of bus-based communication architectures for systems characterized by (possibly several) throughput constraints. Our approach accurately and effectively prunes the large communication design space to synthesize a feasible low-cost bus architecture which satisfies the constraints in a design.**

I. Introduction

The performance of System-on-Chip (SoC) designs today is heavily dependent on the efficiency of their communication architectures. Increasing SoC complexity, however, has made it harder to design communication architectures which meet performance constraints. Bus-based communication architectures, which are widely used in SoC designs today, have customizable topologies, arbitration protocols, pipeline depths, buffer sizes, DMA burst modes, bus widths and speeds, all of which combine to create a vast exploration space. Manually evaluating all possible implementation alternatives thus becomes practically infeasible. Therefore communication synthesis attempts to automatically determine a cost efficient communication architecture implementation which meets all performance constraints.

Typically, systems are characterized by performance constraints which are highly dependent on the nature of the application. *Throughput* of communication connections is a good measure of the performance of a system. Several modern application domains such as broadband, networking, tele-communication and image processing have average throughput constraints which must be satisfied in order to avoid bottlenecks and to function correctly [3].

In this paper, we propose an approach for automated synthesis of low cost bus-based communication architectures for systems characterized by (possibly several) throughput constraints. We chose bus-based communication architectures like AMBA [11] because of their widespread use in SoC designs today. Our approach attempts to prune the vast communication design space and uses a fast simulation engine [6] to quickly analyze interesting combinations of communication parameters. The novelty of our approach is in the ability to automatically satisfy multiple throughput constraints while synthesizing a feasible low-cost configuration of a standard bus-based communication architecture (such as [11]) which is commonly used in SoC designs. We not only synthesize the bus topology, but also determine values for communication architecture parameters such as arbitration strategies, bus widths, bus speed, Out-of-order (OO) buffer sizes [12] and DMA burst sizes. To demonstrate the usefulness of our approach, we present an interesting case study of an AMBA based SoC subsystem from the broadband communication application domain. Using our approach, we were able to synthesize a feasible low-cost bus architecture which satisfied all throughput constraints for the SoC subsystem in a matter of a few hours. Performing such an exploration manually would have taken a designer several days or even weeks.

II. Related Work

There is already a significant body of research in the area of bus architecture synthesis. Early work was aimed at minimizing bus width [13], interface synthesis and simple synchronization protocol selection [8] and topology generation for simple busses without arbitration [3]. Ryu et al. [1] performed studies to find optimal bus topologies for a SoC design. Pinto et al. [4] proposed an algorithm for constraint-driven topology synthesis under the assumption that relative positions of components were fixed. Lyonnard et al. [2] proposed a synthesis flow which supported shared bus and point to point connection templates. But these templates need to be parameterized manually, which makes the process time consuming and cumbersome. Lahiri et al. [5] designed communication architectures after exploring different solutions using fast performance simulation. However, they assumed the bus topology to be given. Thepayasuwan et al [7] and Drinic et al [10] propose approaches which takes into consideration an estimate of the final layout of the design to generate a bus topology. However, neither of these approaches considers the effect of different communication parameters on system performance during synthesis. Shin et al. [14] used a genetic algorithm for automating the generation of bus architecture parameters to meet performance requirements. However, they do not focus on bus topology synthesis. Our approach differs from these existing approaches in the way we automate the synthesis of not only the bus topology, but also the generation of values for bus architecture parameters such as arbitration strategies, bus widths, bus speed, OO buffer sizes and DMA burst sizes, while satisfying performance constraints.

III. Automated Bus Architecture Synthesis

We now describe our approach for automated throughput-driven bus architecture synthesis. First we formulate the problem and present our assumptions. Next we give an overview of the strategies we use to meet throughput constraints. Finally we present our automated bus architecture synthesis approach in detail.

A. Problem Formulation

We are given a SoC with several components (IPs) that communicate with each other. The standard bus-based communication architecture (e.g. CoreConnect, AMBA etc.), for which the bus topology and communication parameter values must be synthesized, is also specified. It is assumed that hardware software partitioning has taken place and that the appropriate functionality has been mapped onto hardware and software. The IPs are assumed to be standard "black box" components which cannot be modified during the synthesis process, except for the memory blocks. We are also given one or more throughput constraints for the system which must be met. These constraints can involve communication between two or more IPs. Fig. 1 shows a communication throughput graph *CTG = G(V,A)* which is a directed graph, where each vertex *v* represents a component in the system, and an edge a_{ij} connects components *i* and *j* that need to communicate with each other. An edge is associated with a throughput constraint $\tau(a_{ij})$ if it lies within a throughput constraint path. The figure shows a constraint path involving IPs *MEM1*, *S1* and *M2*, for which average throughput of data streaming out of the master *M2* must not fall below 360 Mbps (Megabits per second). A throughput constraint path, in general, has a single master for which data throughput must be maintained and other masters, slaves and memories which are in the critical path that impacts the maintenance of the throughput.

The problem then is to generate a bus topology and determine parameter values for the selected standard bus-based communication architecture, for which all throughput constraints in the system are satisfied. Additionally, the synthesized bus architecture must be cost effective, having the least number of busses, and the lowest values for bus widths and speeds, while still satisfying the constraints.

B. *Strategies for Meeting Throughput Constraints*

Fig. 1 shows a *CTG* of a system and a simple bus mapping for it. All the bus masters and high performance slaves and memories are part of the main bus, while the high latency, low bandwidth slaves and memories are part of the peripheral bus. Most standard bus communication architectures (e.g. AMBA [11]) follow a similar bus classification scheme. The shared bus structure shown in Fig. 1 may or may not violate the throughput requirement of the system. In case there is a violation, we must transform and customize the bus architecture till the throughput requirement is met. We classify the changes to be made to the bus architecture into two categories, discussed below.

a. Architecture Transformations

(i) Splitting Memories: If different masters access non-overlapping regions (in memory space) of a memory block it is beneficial to split the memory, to allow multiple concurrent access and thus improve performance. This is also beneficial for an efficient bus split transformation.

(ii) Dedicated Slaves: Memories and other slaves which are only accessed by a single master can be removed from the bus and made private to the accessing master, to prevent unnecessary traffic on the bus.

(iii) Splitting Busses: If the accesses of multiple masters on the same bus overlap frequently in time, performance can be improved by separating the masters (and the IPs they interact with) to different busses. This increases bus bandwidth available to the masters and reducing arbitration conflicts that degrade performance. The major cost of splitting busses is the addition of a bridge for inter-bus access and arbiter and decoder units for the new bus.

(iv) Increasing Memory Ports: For memories for which requests from masters overlap both in space and time, performance can be improved by adding additional ports.

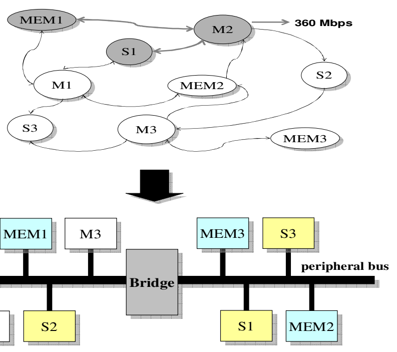


Fig. 1. Example of Communication Throughput Graph (CTG) with corresponding simple bus mapping

b. Parameter Customizations

(i) Data Bus Width: The width of the data bus can impact the throughput of the system. Doubling the width, for instance, effectively doubles the theoretical bus bandwidth.

(ii) Arbitration Protocols: Shared busses require an arbitration protocol to determine which master gets control of the bus when multiple masters request for access to the bus simultaneously. There are several arbitration protocols such as static priority, round robin (RR), random and time division multiplexed access (TDMA) which can effectively control performance for the masters on the bus.

(iii) DMA burst size: Changing DMA burst size can have varying effects on system performance [5]. Increasing burst size on a shared bus can improve throughput for certain masters while limiting it for others.

(iv) OO Buffer Size: Out of order (OO) buffers are used by slaves that support out-of-order transaction completion [12]. The buffer size determines how many out of order requests can be simultaneously handled, and directly affects performance in systems that support OO completion.

(v) Bus Speed: Finally, increasing bus speed can improve throughput. However, this parameter is limited by process technology and cannot be increased beyond a certain value.

C. *Synthesis Approach*

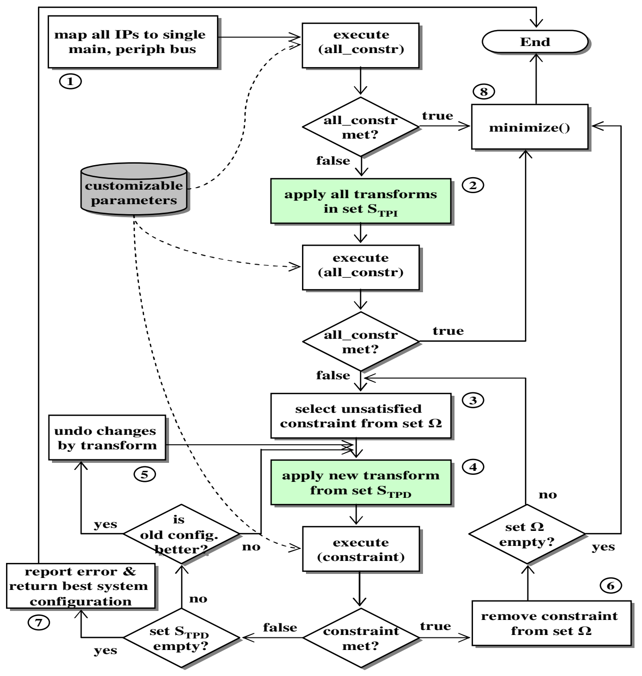
This section describes our synthesis approach. First, we classify the architectural transformations discussed earlier into two categories. The first category consists of all the transformations which improve performance of the entire system and not just for the throughput constraint paths. We call these the *Throughput Path Independent* (TPI) transforms. The set of these transforms is defined as $S_{TPI} = \{T_{sm}, T_{ds}\}$ where T_{sm} is the split memory and T_{ds} is the dedicated slave transformation described earlier. The second category consists of all the transformations which improve performance for the throughput constraint paths. These are the *Throughput Path Dependent* (TPD) transforms. The set of these transforms is defined as $S_{TPD} = \{T_{smb}, T_{spb}, T_{imp}\}$ where T_{smb} is the split main bus, T_{spb} is the split peripheral bus and T_{imp} is the increase memory port transformation. Next we define $\Pi = \{P_{wd}, P_{sp}, P_{dma}, P_{arb}, P_{oo}\}$ as a superset of sets corresponding to all the parameter customizations, where each set element contains a list of valid values for the corresponding customizable parameter. For instance, P_{wd} can contain the values 16, 32 and 64 which represent the allowed data bus widths that can be selected during synthesis.

We will now explain our automated synthesis flow. The inputs to the flow (from the designer) include a CTG graph, a target communication architecture (e.g. AMBA [11]) and a parameter customization superset Π. The general idea is to map all the components from the CTG to a simple bus topology and then systematically performing architectural transformations on it till all constraints are satisfied. We first perform all *Throughput Path Independent* transforms in order to improve performance of the entire system. Next we select a throughput constraint path and focus on it, performing different *Throughput Path Dependent* transforms till the constraint is satisfied. The latter process is repeated for every constraint, until all constraints are satisfied.

Fig. 2 depicts the flow. In the first step, all components from the CTG are mapped to a simple topology having a single *main* and *peripheral* bus. We then call the *execute* function which simulates the simple design for the various combinations of customizable parameters. If all constraints are satisfied, we proceed directly to Step 8 and call the *minimize* function which attempts to reduce the cost of the final system. If all constraints are not satisfied then we proceed to Step 2 and apply all the transformations in the set S_{TPI}. If the constraints are still not met, we proceed to Step 3 to select a throughput constraint from set Ω (which is a superset of all throughput constraints in the system), and then randomly select and apply a transformation from the set S_{TPD} (Step 4). If the constraint is not satisfied, we check to see how the best result after the current transformation compares with the best result from before the transformation. If the result is worse, we undo the effect of the transformation (Step 5) before returning to Step 4 to try another transformation from S_{TPD}. If the constraint is not satisfied even after all the transformations in S_{TPD} have been applied, we return the best result for the unsatisfied constraint path to the designer and exit (Step 7). If the constraint is satisfied, we remove the satisfied constraint from set Ω (Step 6) and return to Step 3 to select the next constraint to be satisfied. We repeat the sequence of steps, till all constraints are satisfied or we encounter a constraint which cannot be satisfied.

Ideally, we would like to test every possible combination of the parameters specified by the designer when we call the *execute* function. However, the large exploration space makes this prohibitive and we must prune the design space to achieve realistic run times. The *execute* function incorporates design pruning by assuming the maximum allowed value for the bus speed, bus width and OO buffer size, as specified in P_{wd}, P_{sp} and P_{oo} respectively. These values give the best performance and a greater likelihood that the throughput constraint will be met. This leaves us with a design space requiring combinations of *(i)* arbitration protocol and *(ii)* DMA burst size. To prune the arbitration protocol space, one of the many optimizations used is to only consider combinations of static priorities for masters in throughput constraint paths, and not care about other masters, as long as they get a lower priority. This is a reasonable assumption because masters in critical paths require greater bus access priority. To prune the DMA burst size space, one of the strategies used is to consider the maximum valid DMA burst size (if the DMA is part of a constraint path being considered), because larger burst sizes generally entail lesser transfer protocol overhead. Thus the customizable parameter space is greatly restricted, speeding up the synthesis process.

Once a bus topology and a set of communication parameter values are found which satisfy the throughput constraints, we call the *minimize* function. This is a simple function that attempts to minimize the 'optimistic' values we selected for the bus widths, speeds and OO buffer sizes, to reduce the cost of the final system. The aim is to arrive at the lowest values for these parameters which still allow the design to meet all constraints. For more details on the synthesis approach, please refer to [15].



map all IPs to single main, periph bus
1
execute (all_constr)
End
8
all_constr met?
true
minimize()
false
customizable parameters
apply all transforms in set S_{TPI}
2
execute (all_constr)
all_constr met?
true
false
select unsatisfied constraint from set Ω
3
undo changes by transform
5
apply new transform from set S_{TPD}
4
no
yes
is old config. better?
no
execute (constraint)
set Ω empty?
yes
report error & return best system configuration
7
no
6
set S_{TPD} empty?
yes
false
constraint met?
true
remove constraint from set Ω

Fig. 2. Automated Synthesis Flow

IV. Case Study

In order to demonstrate the usefulness of our approach, we consider a case study of a broadband communication subsystem shown in Fig. 3. The *ASIC1* block is an encryption accelerator which performs standard encryption such as DES, 3DES, SHA-1 and AES in hardware. The *ARM926* processor runs communication protocol stack software. Additionally, the *SDRAM* interface supports OO transaction completion for improved memory access performance. There are two throughput constraints that must be satisfied in this system. The first involves the encryption engine, where *ASIC1* needs to process data from *RAM3* once triggered by the *ARM* processor, and send it to the external interface (*EXT_IF*) component at a minimum rate of 100 Mbps. The second throughput constraint involves the *USB* subsystem. Data packets received at the *USB* must be routed to *RAM1*, from where the *DMA* engine transfers the data to an external memory interface (*SDRAM_IF*), at a minimum rate of 480 Mbps. Table 1 gives the allowed values for the customizable parameters, set initially by the designer. Additionally, we assume that only single port memories are available.

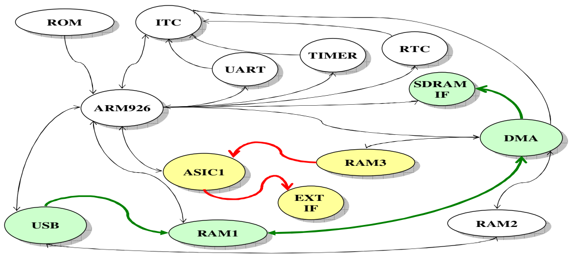


Fig. 3. Broadband SoC Subsystem

Table 1. Customizable Parameter Set

Set	Values
P_{wd}	16, 32, 64
P_{sp}	33, 66, 100, 133, 166
P_{dma}	2, 4, 8, 16
P_{arb}	static, RR, random
P_{oo}	1-8

The target communication architecture for the automated synthesis is the AMBA3 AXI high performance bus [12] and a low bandwidth APB bus [11]. For the purposes of system simulation, we use the fast transaction based simulation models proposed in [6]. The output of our automated synthesis engine is shown in Fig. 4. The values for customizable parameters are given in Table 2.

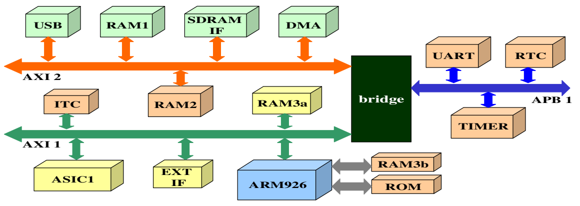


Fig. 4. Synthesized Architecture

Table 2. Communication Parameter Values

Parameter	Values		
	AXI 1	AXI 2	APB
Bus width	32	32	32
Bus speed	66	133	33
Arb. scheme	static ASIC1>DMA>USB>ARM		
DMA size	16		
OO buffer	4		

There are a few important observations here. Firstly, we see that the synthesis engine splits *RAM3*, creating a dedicated memory for the *ARM926* processor (*RAM3b*) in the process, which reduces the load and conflict on the main bus. The *ROM* is also made private to the processor. Secondly, we find that the main AXI bus has been split, so that components which are part of the *USB* throughput constraint path and the *RAM2* component now have a dedicated bus. The rest of the components remain on the original AXI bus. The APB bus is not split because it is not directly involved in any constraint path. The appropriate communication parameter values allow us to merge the components in the *ASIC1* throughput constraint path with other components that consume bus bandwidth, such as the *ARM926*. A manual refinement effort to obtain a bus architecture satisfying both constraints would be inclined to create an additional bus, separating the *ARM926* processor from the components in the *ASIC1* throughput constraint path. Our synthesis approach finds a lower cost solution, and this is made possible by integrating communication parameters in the synthesis flow. The entire automated synthesis process took a few hours to complete. Manually exploring such a complex design space to generate a bus topology and parameters values that satisfy throughput constraints would take a designer several days or even weeks.

V. Conclusion and Future Work

In this paper we presented an automated approach for synthesizing bus-based communication architectures to meet throughput constraints in a design. Our approach synthesizes not only the bus topology, but also generates values for communication parameters such as arbitration strategies, bus widths, speeds, DMA burst sizes and OO buffer sizes, while satisfying several throughput requirements and minimizing system cost. Results from the automated synthesis of a bus architecture for the broadband communication subsystem case study show the usefulness of our approach.

Acknowledgements

This research was partially supported by grants from Conexant Systems Inc. and UC Micro (03-029).

References

[1] K. K. Ryu, Vincent J. Mooney III "Automated Bus Generation for Multiprocessor SoC Design", *In Proceedings of DATE 2003.*

[2] D. Lyonnard, S. Yoo, A. Baghdadi, A. A. Jerraya **"**Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip", *In Proceedings of DAC 2001*

[3] M. Gasteier, M. Glesner "Bus-based communication synthesis on system level", *In ACM TODAES, January 1999*

[4] A. Pinto, L. Carloni, A. L. Sangiovanni-Vincentelli "Constraint-driven communication synthesis", *In Proceedings of DAC 2002*

[5] K. Lahiri et al, "Efficient exploration of the SoC communication architecture design space", *In Proceedings of ICCAD 2000*

[6] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration", *In Proceedings of DAC 2004.*

[7] N. Thepayasuwan, A. Doboli "Layout Conscious Bus Architecture Synthesis for Deep Submicron Systems on Chip", *In Proceedings of DATE 2004*

[8] J. Daveau et al, "Protocol selection and interface generation for HW-SW codesign", *In IEEE Trans. on VLSI Systems, March 1997*

[9] R. B. Ortega and G. Borriello, "Communication synthesis for distributed embedded systems", *In Proceedings of ICCAD 1998*

[10] M. Drinic et al. "Latency-guided on-chip bus network design", *In Proceedings of ICCAD 2000*

[11] D. Flynn. "AMBA: enabling reusable on-chip designs". *In IEEE Micro, 17(4):20--27, July-Aug 1997*

[12] AMBA AXI Specification www.arm.com/armtech/AXI

[13] S. Narayan and D. Gajski, "Synthesis of system level bus interfaces", *In Proceedings of DATE 1994*

[14] Chulho Shin, et al "Fast Exploration of Parameterized Bus Architecture for Communication-Centric SoC Design", *In Proceedings of DATE 2004.*

[15] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Automated Synthesis of Bus Architectures for Systems with Throughput Constraints," *CECS Technical Report 04-20, August 2004.*