Automated Synthesis of Bus Architectures for Systems with Throughput Constraints *

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

†Center for Embedded Computer Systems
University of California Irvine
Irvine, CA 92697-3425, USA
1 (949) 824-2248
{sudeep, dutt}@cecs.uci.edu

‡Conexant Systems Inc.
4000 Mac Arthur Blvd
Newport Beach, CA 92660 USA
1 (949) 483-4600
m.benromdhane@conexant.com

CECS Technical Report #04-20
August, 2004

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

Automated Synthesis of Bus Architectures for Systems with Throughput Constraints *

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

†Center for Embedded Computer Systems
University of California Irvine
Irvine, CA 92697-3425, USA
1 (949) 824-2248
{sudeep, dutt}@cecs.uci.edu

‡Conexant Systems Inc.
4000 Mac Arthur Blvd
Newport Beach, CA 92660 USA
1 (949) 483-4600
m.benromdhane@conexant.com

CECS Technical Report #04-20
August, 2004

Abstract

As System-on-Chip (SoC) designs become more complex, it becomes increasingly harder to design communication architectures which satisfy designer constraints. Manually traversing the vast communication design space for constraint-driven synthesis is not feasible anymore. In this report 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 all throughput constraints. We present a case study of a broadband SoC subsystem, for which we were able to synthesize a bus architecture in a matter of hours, instead of days or even weeks it would have taken for a manual effort.

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

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. Changing just one of these parameters requires a reevaluation of the others due to their highly interdependent nature. For instance, increasing DMA burst size on a shared bus can improve performance for one part of the system, but it can degrade the performance of another master on the same shared bus, requiring a change in arbitration strategy or bus topology to preserve its performance. As a result of this complex interdependence, it becomes impossible to manually evaluate all possible implementation alternatives. 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. It can be used to characterize the whole system (system throughput) or parts of the system (subsystem throughput). 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 report, 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. We assume that hardware/software partitioning has already taken place, and appropriate functionality has been mapped onto hardware IPs and software code. 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. Our approach is easily portable across different standard bus-based communication architectures such as CoreConnect [13], Wishbone [14] and OCP [15], and can be extended to automatically synthesize other communication architecture specific parameters as well. 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.

The rest of the report is organized as follows. Section II discusses related work in the area of bus-based communication architecture synthesis. Section III formulates the problem and presents our approach for automated throughput-driven bus architecture synthesis. Section IV describes a case study of a broadband communication SoC subsystem where we used our approach to automatically synthesize the bus architecture. Finally Section V concludes the report and gives directions for future work.

II. Related Work

There is already a significant body of research in the area of bus-based communication architecture synthesis. Early work from Narayan et al [16] was aimed at determining a minimum bus width when mapping several communication channels on one bus. Daveau et al [8] propose an algorithm for interface synthesis and simple synchronization protocol (e.g. handshakes, FIFO) selection during communication synthesis. Gasteier et al [3] describe the generation of a low cost communication topology after analyzing statically scheduled data transfers. However their approach synthesizes only simple busses without arbitration, to minimize cost.

Ryu et al [1] generate five different custom bus templates and compare throughput performance for applications mapped on them. Here the designer is limited to selecting busses from these simple pre-designed templates which lack the high performance features found in standard bus architectures such as in [12]. Pinto et al [4] propose a general purpose algorithm for constraint-driven communication synthesis. The goal is to minimize a communication cost function and assumes that relative positions of cores in a SoC are fixed, which is not possible when performing exploration early in the design flow (which is usually the case). Lyonnard et al [2] propose a synthesis design flow which supports two generic communication templates – shared bus and point to point connections. These templates need to be parameterized manually, which makes it cumbersome and time-consuming for the designer to select appropriate combinations of parameters to meet design requirements. Lahiri et al [5] design communication architectures after exploring different solutions using fast performance simulation. However, they assume 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.

Our approach is different from existing approaches because we focus on satisfying throughput constraints while automatically synthesizing low cost bus architectures. And unlike existing approaches, we not only synthesize the bus topology but also generate values for the complex interdependent communication architecture parameters such as arbitration strategies, bus widths, bus speed, OO buffer sizes and DMA burst sizes.

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 architecture with several components (IPs) that communicate with each other. The bus-based communication architecture (e.g. OCP, CoreConnect, AMBA etc.), which determines the pins at the IP interface and for which the bus topology and communication parameter values must be synthesized, is also assumed to be specified. It is assumed that hardware software partitioning has taken place and that the appropriate functionality has been mapped onto hardware IPs (either standard IPs or ASIC blocks) and software (scheduled to run on a processor IP). 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) where the vertices represent cores and the edges connect cores that communicate with each other. 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, possibly a DMA and can have any number of slaves and memories. The problem then is to generate a bus topology and determine communication parameter values for the selected standard communication architecture, which enables all valid inter-IP communication and satisfies every throughput constraint in the system. Additionally, the synthesized bus architecture which satisfies all constraints must be as low cost as possible. This means that, for instance, if we have a choice between two bus architectures that satisfy all constraints – one with a lower bus width than the other, then we will choose the architecture with the lower bus width.

B. Strategies for Meeting Throughput Constraints

Fig. 1 shows a communication throughput graph 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 follow a similar bus classification scheme. AMBA [11] for instance calls the main bus an Advanced High Performance Bus (AHB). The peripheral bus is called Advanced Peripheral Bus (APB).

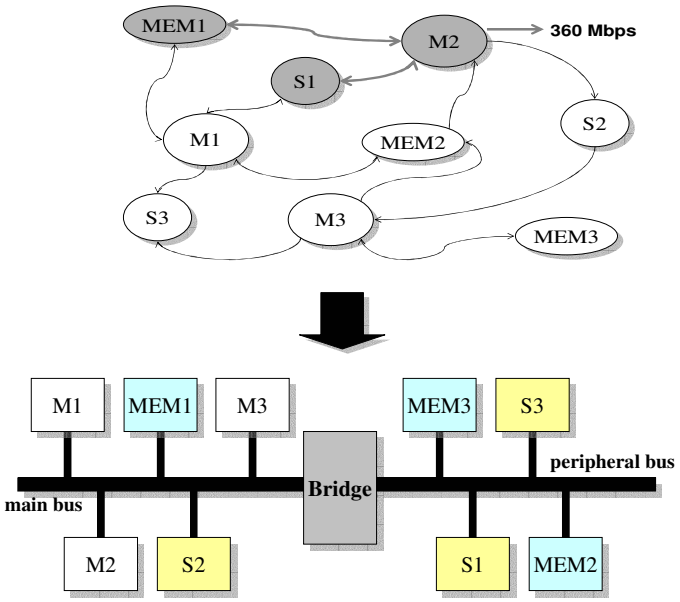


Fig. 1. Example of Communication Throughput Graph (CTG) with corresponding simple bus mapping

The shared bus structure shown in Fig. 1 may or may not violate the throughput requirement of the system. In the case that there is a violation, we need to 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 – *Architecture Transformations* and *Parameter Customizations*. These are discussed below.

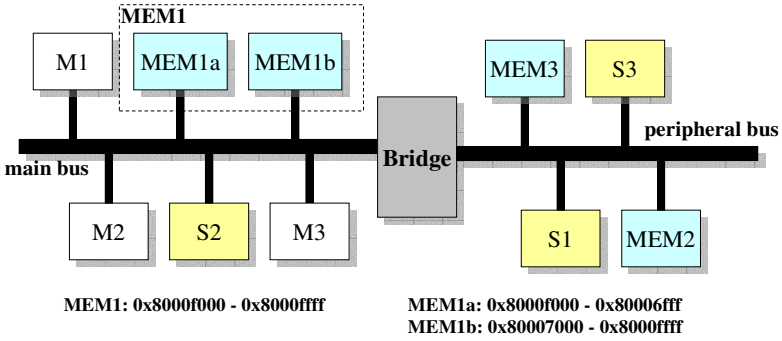
a. ***Architecture Transformations***

These transformations change the memory architecture and transform the bus topology by adding or removing busses to the existing bus architecture, and redistributing components on them. The aim of these transformations is to improve system performance so that throughput constraints are satisfied. For the purpose of our synthesis algorithm, we identified five such transformations. Note that this is not an exhaustive list, and can be extended to include additional transformations.

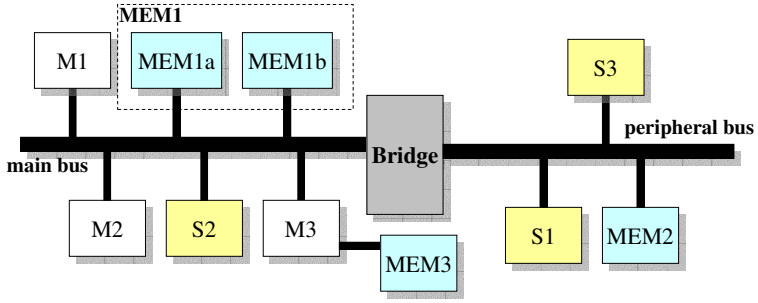
(i) *Splitting Memories:* It is possible that different masters access non overlapping regions (in memory space) of a memory block. If the access times for these masters overlap in time (i.e. simultaneous access), only one of these masters can get access to the memory while the others must wait till the transfer is complete. In such a case, it is beneficial to split the memory, to improve performance. Fig. 2(a) shows this transformation for the system in Fig. 1. MEM1 is split into MEM1a and MEM1b, which now allows masters to gain access to the separate regions in MEM1 without having to wait for the other master to complete its operation. The splitting of memories is also beneficial for an efficient bus split transformation, presented later.

(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. This prevents unnecessary traffic on the bus due to transfers between the master and the slave. Fig. 2(b) shows how the MEM3 is made private to master M3

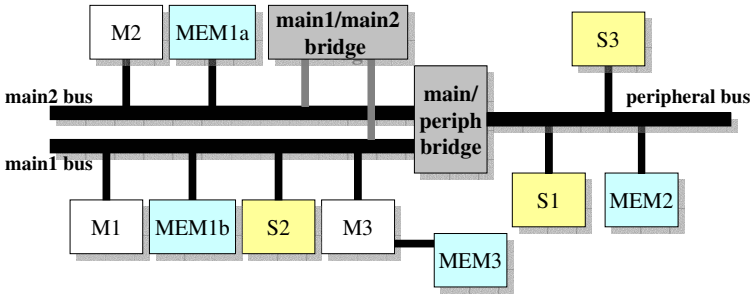
which is the only master that accesses it. Any slave on the main bus or the peripheral bus can be made private. This frees up bandwidth on the bus and improves performance.



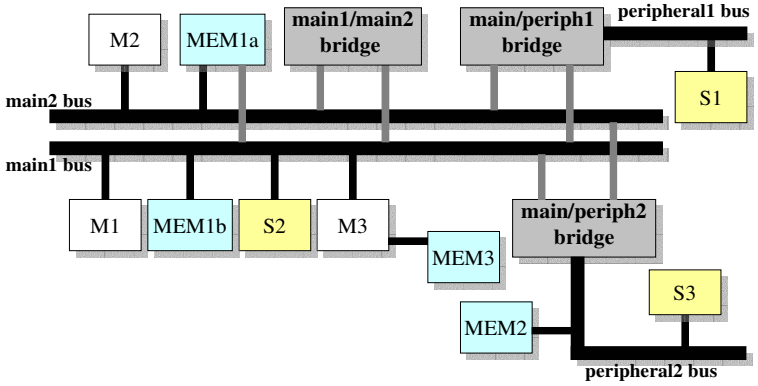
(a) Splitting Memory



(b) Making Dedicated Memory/Slave



(c) Splitting Main Bus



(d) Splitting Peripheral Bus and Extending Memory Ports

Fig. 2. Architectural Transformations

(iii) *Splitting Main Bus*: If the accesses of multiple masters overlap frequently in time, performance can be improved by assigning the masters to different busses. This has the effect of increasing bus bandwidth available to the masters and reducing arbitration conflicts that degrade performance. Fig. 2(c) shows how master M1 is assigned to a separate bus from the one which has masters M1 and M3 on it. Such a switch requires that other IPs on the shared bus also be allocated to the appropriate bus. This decision depends on how frequently a master interacts with these IPs. To improve performance it makes sense to transfer IPs to the bus where they are accessed the most, because inter-bus accesses encounter the overhead of the bridge, which can actually degrade performance. It is possible that an IP is accessed frequently by masters on the two separate busses, and due to the bridge overhead the performance gets degraded. In such a case, splitting busses is not the best option. However, if the set of IPs accessed by masters on separate busses is mostly disjoint, then performance improves substantially with bus splitting. The major cost of splitting the main bus is the addition of a bridge for inter-bus access.

(iv) *Splitting Peripheral Bus*: If slaves on the peripheral bus are accessed simultaneously by several masters, only one master can gain access while the others must wait for the operation to finish. This can degrade performance. To overcome this bottleneck, the peripheral bus can be split so that slaves which are accessed simultaneously are on different peripheral busses. Fig. 2(d) shows the case when slave S1 is separated from the rest of the peripherals on the peripheral bus, and attached to a newly created peripheral bus. The major cost of splitting the peripheral bus is the addition of bridges, just like in the case of splitting the main bus.

(v) *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. 2(d) shows how Memory MEM1a can be simultaneously accessed by more than one master because it has multiple ports.

b. ***Parameter Customizations***

Certain communication parameters can have a significant impact on system performance. We have identified five such customizable parameters which we use in our synthesis approach. Just like the architecture transformations, this list is not exhaustive and can be extended to include additional parameters.

(i) *Data Bus Width*: The width of the data bus can significantly impact the throughput of the system. Changing the width from 32 to 64 bits for instance effectively doubles the theoretical bus bandwidth, allowing more data to be transferred per unit time.

(ii) *Arbitration Protocols*: Shared busses require an arbitration protocol to determine which master gets control of the bus when multiple masters request access to the bus simultaneously. There are several arbitration protocols that can be used, such as static priority, round robin (RR), random and time division multiplexed access (TDMA). The arbitration protocol can effectively control the frequency of allowed accesses and 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]. OO transaction completion allows a variable latency slave to signal the completion of a read or write transaction regardless of the order in which the transaction was received. Normally, if OO completion is not supported, a transaction cannot complete before other transactions issued before it have completed. Thus OO completion improves performance. 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 the bus speed can improve throughput performance. However, this parameter is technology dependent and cannot be increased beyond a certain value. Also, increasing speed has several costs associated with it such as the addition of buffer blocks to synchronize core speeds with bus speeds, and the addition of register slices to meet timing closure, which actually increases latency in the design.

C. Synthesis Approach

This section describes our synthesis approach. We start with a few definitions. A communication throughput graph, $CTG = G(V,A)$ 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. Furthermore, each arc is associated with an average throughput constraint $\tau(a_{ij})$ if it lies within a throughput constraint path. For simplicity, we assume that an edge can only be part of one throughput constraint path, and have only one value for $\tau(a_{ij})$. Fig. 1 shows a communication throughput graph with arcs $a_{MEM1\backslash M2}$ and $a_{S1\backslash M2}$ between components MEM1 and M2, and S1 and M2 respectively having a throughput constraint value of 360 Mbps, corresponding to the constraint path they belong to.

Standard communication architectures [11-15] classify IPs as high performance, low latency components which must be added to the high performance (main) bus, or as low performance, high latency peripherals which must be added to the slower peripheral bus. We take this classification into account and define $b(v)$ which indicates whether a component belongs to the main bus or the peripheral bus. This allows us to determine valid moves for the component in our automated synthesis approach.

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 Γ_k as the k^{th} throughput constraint set, which contains all the vertices (components) that are part of a throughput constraint path. Then set Ω is a superset of all n throughput constraints in the system, and is defined as

$$\Omega = \bigcup_{k=1}^{n} \Gamma_k$$

Let $\Lambda = \{A_{sm}, A_{ds}, A_{smb}, A_{spb}, A_{imp}\}$ be a superset of sets corresponding to all the architecture transformations, where each set element contains a list of components which are eligible for the particular transformation. For instance, A_{imp} will hold the list of all memory blocks for which the number of ports can be increased. This ensures that the transformation occurs only on those memory blocks.

Similarly, let $\Pi = \{P_{wd}, P_{sp}, P_{dma}, P_{arb}, P_{oo}\}$ be 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, constraint superset Ω, architecture transformation superset Λ and 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. 3 depicts the flow. In the first step, all components from the CTG are mapped to a simple topology having a single main bus and a single peripheral bus. We then call the *execute* function which simulates the simple design for the various combinations of customizable parameters. The function takes a parameter which indicates the constraints that it needs to check for, during simulation. The value *all_constr* which is passed to the function in this case, indicates to the function that we want to check for all constraints in the system. The function then returns a true value if all throughput constraints are satisfied for some combination of communication parameters, and false otherwise. If all constraints are satisfied, we proceed directly to Step 8 and call the *minimize* function which attempts to reduce the cost of the system. If all constraints are not satisfied then we proceed to Step 2 and apply all the transformations in the set S_{TPI}. We call *execute* again to check if all constraints are met for some combination of communication parameters, after the transformations. If the constraints are still not met, we select a throughput constraint from set Ω (Step 3), and then randomly select and apply a transformation from the set S_{TPD} (Step 4). Next, we call *execute* again, this time to check if the selected constraint was satisfied after the transformation. If the constraint is not satisfied, we check to see how the best result (defined as the largest throughput for a combination of communication parameter values, which is still lesser than the desired throughput) 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 CS (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, every time we call the *execute* function to explore the effect of customizable parameters on the bus architecture we would like to test every possible combination of the parameters specified by the designer. However, this makes the exploration space prohibitively large and too time consuming to traverse while searching for a solution to satisfy the throughput constraints in the system. Therefore we must prune the design space to achieve realistic run times. The *execute* function incorporates design pruning for customizable parameters, and is shown in Fig. 4. For every combination of customizable parameter values for which the system must be simulated, we assume a single value for the bus speed, bus width and OO buffer size, which is the maximum allowed value for these parameters specified in P_{wd}, P_{sp} and P_{oo}. These values give us 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. We can reduce the exploration space further by intelligently pruning values from their parameter sets P_{arb} and P_{dma}. For instance, consider the case of a throughput constraint path with a single master which is given the maximum static arbitration priority on a bus with other masters. If the constraint is not satisfied for any combination of communication parameters, we can ignore all other static priority combinations for which this master has the maximum priority, because they will always produce inferior results. This is one way in which the arbitration protocol space can be pruned. Similarly, the DMA burst size exploration space can also be restricted. Once a DMA burst size is found to satisfy a constraint (which is part of a combination of communication parameter values satisfying the constraint), we can effectively eliminate all burst size values lower than the selected value, when we proceed to satisfy other constraints. 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 all 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. In the function, we first select the bus speed set P_{sp}, and repeatedly simulate the design for values of bus speed lower than the one currently selected. The aim is to arrive at the lowest value of bus speed for which all constraints are still met. The values of the other parameters are not changed and the reduction is performed for all busses in the bus architecture. We repeat this process for bus widths and OO buffer sizes. The end result is a low cost system with the lowest acceptable communication parameter values that still allow the design to meet all constraints.

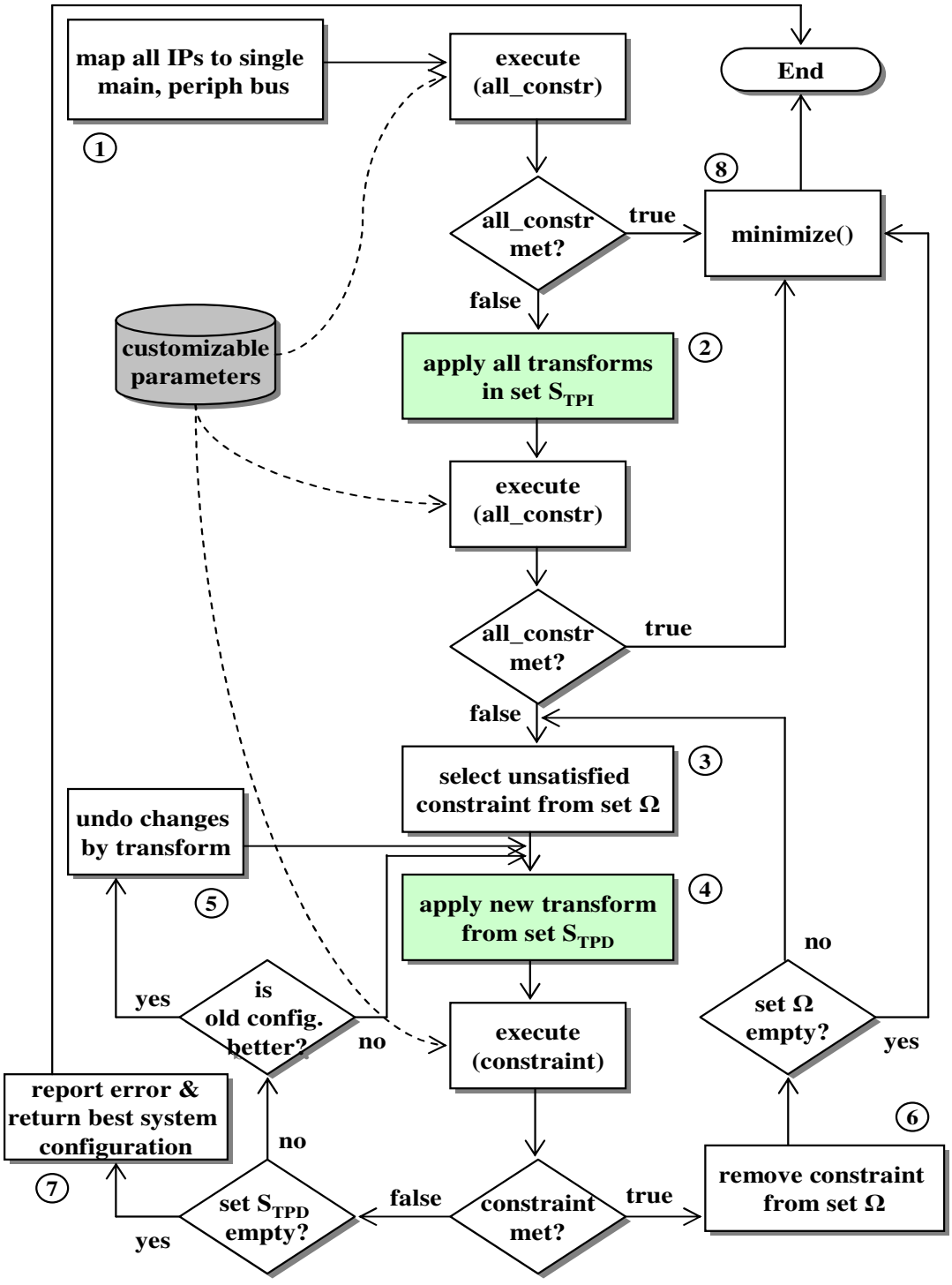


Fig. 3. Automated Synthesis Flow

```
function execute(constr_to_check)
begin
Step 1 : bus_width ⇐ max(Pwd) ;
Step 2 : bus_speed ⇐ max(Psp) ;
Step 3 : OO_buffer_size ⇐ max(Poo) ;
Step 4 : select new combination of (i) pruned arbitration
         protocol space and (ii) pruned DMA burst size space;
         if none exists then return false ;
Step 5 : simulate design ;
Step 6 : if (constr_to_check == all_constr)
          { if all constraints are met, return true } ;
         else
          { if specified constraint is met, return true } ;
Step 7 : go to Step 4 ;
end
```

Fig. 4. *execute* function

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. 5. 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 and also performs control operations at the system level. The DMA engine handles packet forwarding and routing. 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, involving the ASIC1 block. The ASIC1 block 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.

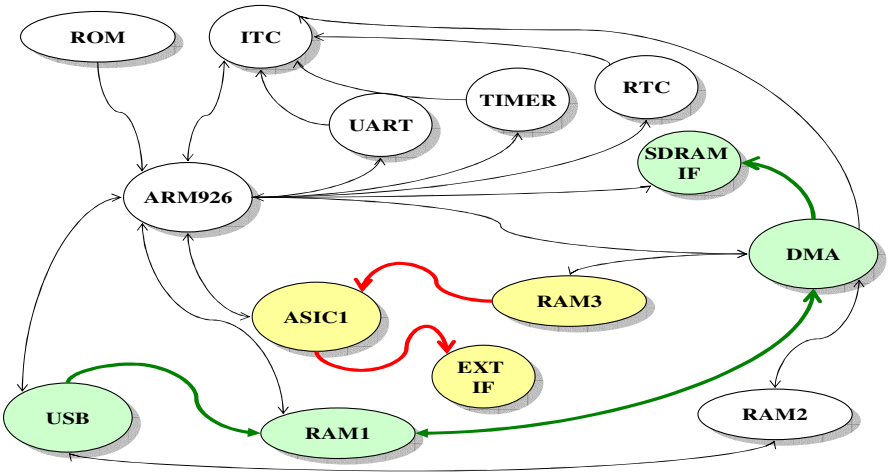


Fig. 5. Broadband SoC Subsystem

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 first proposed in [6], which allow simulation speeds well in excess of 100K cycles/sec while running embedded software on the processor ISS. The output of our automated synthesis engine is shown in Fig. 6. The values for customizable parameters are given in Table 2.

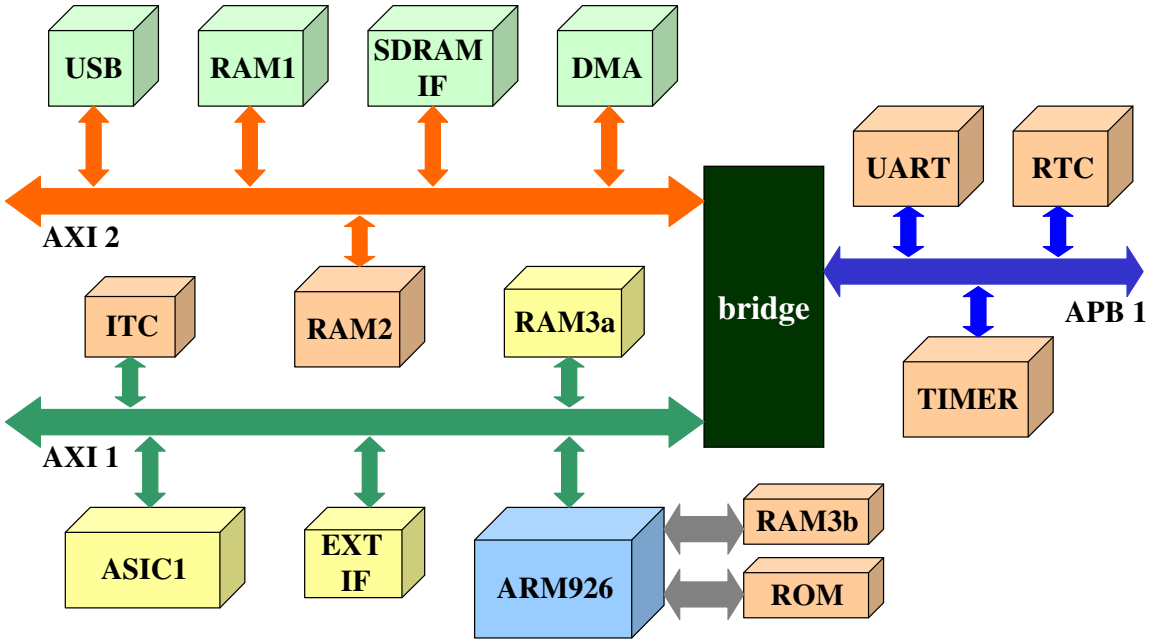


Fig. 6. Synthesized Architecture

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.

Our second case study involves a variant of the broadband communication subsystem from Fig. 5, shown in Fig. 7. Here, additional functionality in the form of a packet switch module (SWITCH) and memory (RAM4) have been added to the system. While the constraint paths and throughput requirements remain the same, note the dependence of the newly added SWITCH component on the external interface (EXT IF).

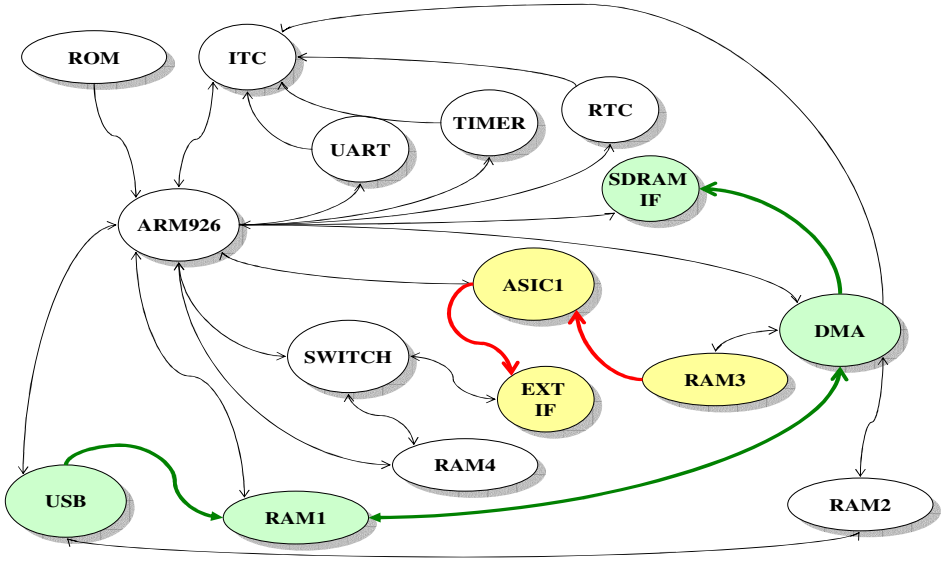


Fig. 7. A Variant Broadband SoC Subsystem

Table 3. Customizable Parameter Set

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

Table 3 gives the allowed values for the customizable parameters, set initially by the designer. Like in the previous case, the target communication architecture for the automated synthesis is the AMBA3 AXI high performance bus [12] and a low bandwidth APB bus [11]. The output of our automated synthesis engine is shown in Fig. 8. The values for synthesized communication parameters are given in Table 4.

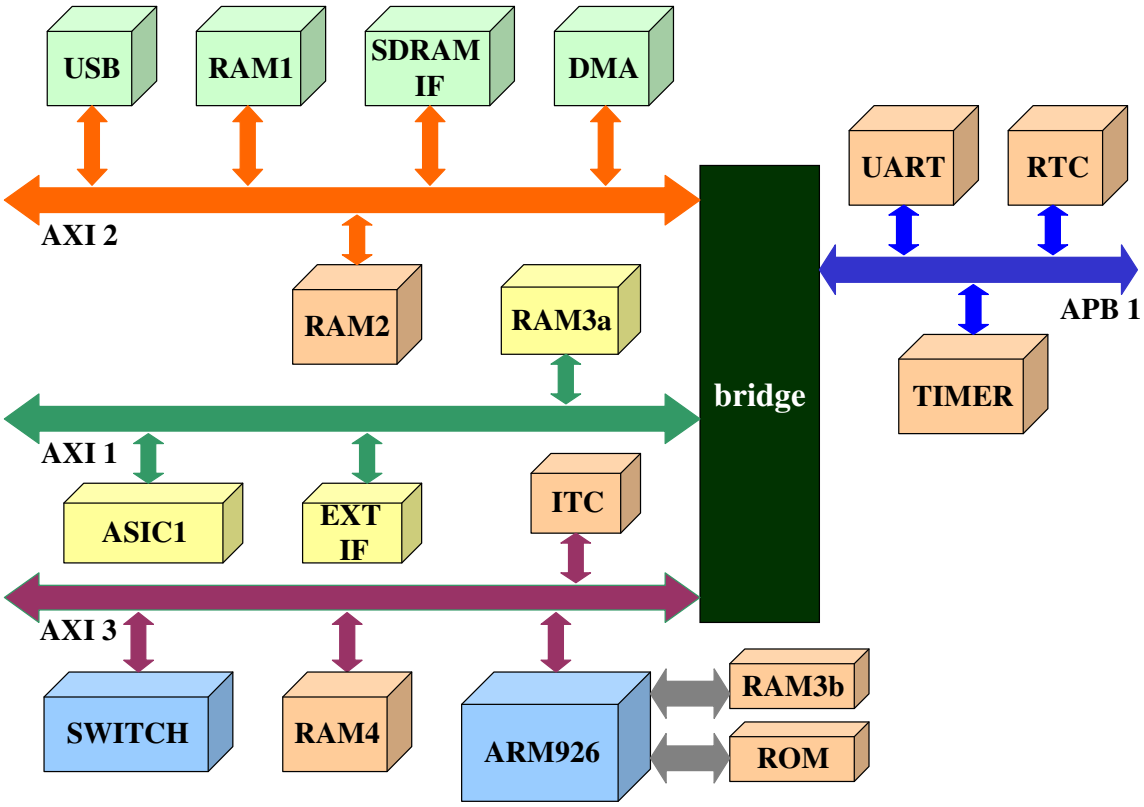


Fig. 8. Synthesized Architecture

Table 4. Communication Parameter Values

Parameter	Values			
	AXI 1	AXI 2	AXI 3	APB
Bus width	32	32	32	32
Bus speed	66	133	66	66
arb scheme	static ASIC1>DMA>USB>ARM>SWITCH			
DMA size	16			
OO buffer	4			

From the synthesized architecture, we observe that the ARM processor and the SWITCH components are assigned to a separate bus (AXI 3). With the increased activity of the ARM processor due to the addition of the SWITCH, the processor can no longer share the same bus as ASIC1. Likewise, the SWITCH interacts frequently with the memory RAM4 and the ARM processor, and therefore it is attached to the AXI 3 bus. Even though the EXT IF module is accessed frequently by the SWITCH component, it remains attached to AXI 1 since it is part of the ASIC1 constraint path. Due to the lower arbitration priority of the SWITCH (Table 4) as compared to ASIC1, the constraint path is not adversely affected by accesses to EXT IF from the SWITCH component.

The entire automated synthesis process for each case study took a few hours to complete. It should be noted that manually exploring such a complex design space to generate a bus topology and values for communication parameters that satisfy all throughput constraints would take a designer several days or even weeks.

V. Conclusion and Future Work

In this report 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 architecture 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 studies show the usefulness of our approach. This report is part of ongoing research. Future work will focus on speeding up the simulation engine and developing heuristics to handle more complex systems having intersecting throughput constraint paths.

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] Sudeep Pasricha, Nikil Dutt, Mohamed 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, G. F. Marchioro, T. Ben-Ismail, A. A Jerraya, "Protocol selection and interface generation for HW-SW codesign", *In IEEE Trans. on VLSI Systems, Vol. 5, No. 1, 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] IBM Coreconnect www.chips.ibm.com/products/powerpc/cores

[14] Wishbone Specification www.silicore.net/wishbone.htm

[15] Open Core Protocol International Partnership (OCP-IP). OCP datasheet, http://www.ocpip.org

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