Floorplan-Aware Automated Synthesis of Bus-based Communication Architectures

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

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

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

ABSTRACT

As System-on-Chip (SoC) designs become more complex, it is becoming harder to design communication architectures to handle the ever increasing volumes of inter-component communication. Manual traversal of the vast communication design space to synthesize a communication architecture that meets performance requirements becomes infeasible. In this paper, we address this problem by proposing an automated approach for synthesizing cost-effective, bus-based communication architectures that satisfy the performance constraints in a design. Our synthesis flow also incorporates a high-level floorplanning and wire delay estimation engine to evaluate the feasibility of the synthesized bus architecture and detect timing violations early in the design flow. We present case studies of network communication SoC subsystems for which we synthesized bus architectures, detected timing violations and generated core placements in a matter of hours instead of several days it took for a manual effort.

Categories and Subject Descriptors: J.6.1 [**Computer Applications**]: Computer-Aided Design

General Terms: Design, Performance

Keywords: Communication Synthesis, Systems-on-Chip

1. INTRODUCTION

Improvements in process technology have led to more and more functionality being integrated onto a single chip, which has in turn resulted in a sharp increase in the amount of overall on-chip communication volumes between the integrated components. In such highly integrated systems, on-chip communication is expected to become a major performance bottleneck [1]. Already, increasingly demanding performance requirements from the next generation of multimedia, broadband and network applications are making interconnect design a challenging proposition.

Bus-based communication architectures [2-4] remain a popular choice for handling on-chip communication in SoC designs today, because they are simple to design and take up very little area. However, selecting and reconfiguring standard bus-based communication architectures such as AMBA [2] and CoreConnect [3], to meet application specific performance requirements, is a very time consuming process. This is due to the large exploration space created by customizable bus topologies, arbitration protocols, DMA burst sizes, data bus widths, bus clock speeds and buffer sizes, all of which significantly impact system performance [5][12][26].

To counter the challenge of ever increasing on-chip bandwidth requirements and a vast communication exploration space, early planning of the interconnect architecture at the system level must become an integral part of a SoC design process. However the complex interplay between communication architecture parameters is hard to analyze effectively even at the system level. Previous research in this area (discussed in Section 2) has been limited to identifying and automating the exploration of small subsets of this design space. Very often, designers end up evaluating the communication design space by creating simulation models annotated with detail based on experience, and manually iterating through different design configurations. Such an effort remains time consuming and produces systems which are generally overdesigned for the application at hand.

To address this problem, we propose a bus architecture synthesis approach in this paper, which automates the generation of a cost effective communication architecture for a SoC. We make use of SystemC [23] to quickly capture components at the behavioral system-level and automate the bus architecture synthesis for the design. The novelty of our approach is in the ability to automatically satisfy performance constraints and detect bus cycle time violations, while synthesizing a feasible, low-cost configuration of a standard bus-based communication architecture (such as [2]) which is commonly used in SoC designs. Our approach synthesizes the bus topology, as well as values for bus architecture parameters such as arbitration priority orderings, data bus widths, bus clock speeds and DMA burst sizes. Additionally, we make use of a high-level floorplanning engine to generate estimates of core placements on the chip. Typically, once the system architecture is frozen, it takes several months before a floorplan of the design becomes available. Violations of timing constraints, detected at this late stage, can require changes in the architecture which can severely impact time-to-market. Our high-level floorplanning and wire delay estimation engines detect these timing violations early in the design flow at the system level, where architectural modifications and tradeoff analysis can be performed quickly and efficiently to eliminate such violations. To demonstrate the usefulness of our approach, we present case studies of network communication SoC subsystems, used for data packet processing and forwarding. Compared to a manual effort which took several days and produced overdesigned systems, our automated flow synthesized low-cost bus architectures, detected timing violations and generated core placements which satisfied performance constraints for the SoC subsystems in a matter of a few hours.

2. 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 [6], interface synthesis and simple synchronization protocol selection [7] and topology generation for simple busses without arbitration [8]. Ryu et al. [9] performed studies to find optimal bus topologies for a SoC design. Pinto et al. [10] proposed an algorithm for constraint-driven topology synthesis under the assumption that relative positions of components were fixed. Lyonnard et al. [11] proposed a synthesis flow which supported shared bus and point to point connection templates. These templates have to be parameterized manually, which makes the process time consuming. Lahiri et al. [12] designed communication architectures after exploring different solutions using fast performance simulation. However, they assumed the bus topology to be given. Shin et al. [13] 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, while also satisfying performance constraints.

A key component of our synthesis flow is the integrated floorplanner. There have been other approaches in the past which have made use of a floorplanning tool [14-18] in a synthesis flow, but for different reasons. Bergamaschi et al. [18] and Thepayasuwan et al. [14] used the floorplanner to generate an early core placement estimate. Drinic et al. [15] used the floorplanner to determine feasibility of the synthesized design by comparing estimates of wire length with an upper bound on wire length.

However an upper bound on wire length has the disadvantage of not accounting for varying capacitive loads of the components. Hu et al. [16] also used the floorplanner to estimate wire length, which they used to calculate energy consumption in point to point networks. Dick et al. [17] invoked the floorplanner repeatedly in their custom bus topology synthesis approach to obtain global wiring delays and ensure that real time deadlines were met. Unlike existing approaches, the floorplanner in our approach is used to identify bus cycle time violations and verify the feasibility of the synthesized bus architecture early in the design flow. We believe that this step will become increasingly important in the deep submicron era as clock speeds increase and lengthy propagation delays cause violations of timing constraints that will need to be detected and corrected early in the design flow.

3. AUTOMATED BUS SYNTHESIS

This section describes our approach for automated bus architecture synthesis. Section 3.1 formulates the problem and presents our assumptions. Section 3.2 discusses the simulation engine while Section 3.3 describes communication parameter constraints, which guide the bus synthesis process. Section 3.4 gives an overview of our floorplan and wire delay calculation engines used for detecting timing violations in the design. Finally, Section 3.5 presents our automated bus architecture synthesis approach in detail.

3.1 Problem Formulation

We are given a SoC having several components (IPs) that need to communicate with each other. The standard bus-based communication architecture (e.g. AMBA [2], CoreConnect [3]) which determines the pins at the IP interface and 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" library components which cannot be modified during the bus synthesis process, except for the memory components.

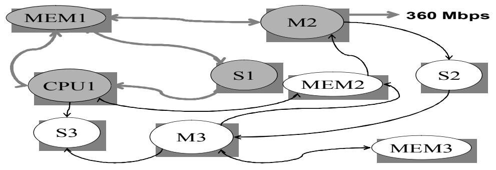


Figure 1. Communication Throughput Graph (CTG)

Typically, SoC designs need to satisfy performance constraints which are generally dependent on the nature of the application. The *throughput* of communication between components is a good measure of the performance of a system [8]. We assume that we are given one or more throughput constraints for the system that need to be satisfied. These constraints can involve communication between two or more IPs. Figure 1 shows a **Communication Throughput Graph (CTG)** 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. Each vertex *v* contains information about the component it represents, such as its area, dimensions (fixed width/height or aspect ratio), capacitive loads on output pins and which bus type it can be connected to – a *main* high bandwidth bus like AHB [2], a *peripheral* low bandwidth bus like APB [2] or both. An edge a_{ij} is associated with a throughput constraint $\tau(a_{ij})$ if it lies within a ***throughput constraint path (TCP)***. Figure 1 shows a TCP involving CPU1, MEM1, S1 and M2 components, where the rate of data packets streaming out of M2 must not fall below 360 Mbps. A TCP, 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 bus architecture parameter values for the selected standard bus-based communication architecture, while ensuring that all throughput constraints in the system are satisfied. In addition, we want to consider layout information of the chip to detect bus cycle time violations, early in the design flow. Finally, 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 all constraints.

3.2 Simulation Engine

Since communication behavior is characterized by unpredictability due to dynamic bus requests from cores, bus contention etc., a simulation based approach is necessary for accurate performance estimation. In our synthesis flow, we capture behavioral models of components and bus architectures in SystemC [23], and keep them in an IP library database. Since we were concerned about the speed of simulation, we chose a fast transaction-based, bus cycle accurate modeling abstraction, which averaged simulation speeds of 150–200 Kcycles/sec [5], while running embedded software applications on processor ISS models.

3.3 Communication Parameter Constraints

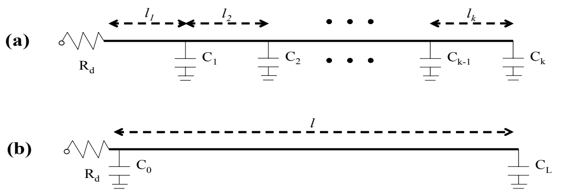
The exploration space for a typical SoC bus-based communication architecture such as AMBA [2] consists of combinations of bus topology configurations with communication parameter values for arbitration schemes, data bus widths, bus clock speeds and DMA burst sizes. If we allow these parameters to have any arbitrary values, an incredibly vast design space is created. The time required to simulate through all possible system configurations searching for one which satisfies every design constraint would become unreasonably large, even with the fast simulation engine. More importantly, once we manage to find such a system configuration, there would be no guarantee that the values generated for the communication parameters would be practically feasible. To ensure that our synthesis approach generates a realistic communication architecture configuration, we allow the designer to specify a *Communication Parameter Constraint set* (Ψ). These constraints are in the form of a discrete set of valid values for the communication parameters to be synthesized. A major motivation to allow this constraint specification is that it allows the designer to bias the synthesis process based on knowledge of the design and the technology being targeted. For instance, a designer might decide that the synthesized design should only have data busses with 16, 32 or 64 bit widths, because the IPs in the design cannot support larger widths effectively. Or a designer might set the allowable bus clock frequency to multiples of 33 MHz, with a maximum speed of 166 MHz, based on the operation frequency of the cores in the system and past experience of the clock generation mechanism. Such knowledge about the design is not a prerequisite for using our synthesis framework. As long as Ψ is populated with any discrete set of values for the parameters, our framework will attempt to synthesize a feasible communication architecture. However, informed decisions can greatly reduce the time taken for synthesis and help the designer generate a more practical system.

3.4 Floorplanning and Delay Estimation Engines

The floorplanning stage in a typical design flow arranges arbitrarily shaped, but usually rectangular blocks representing circuit partitions, into a non-overlapping placement while minimizing a cost function, which is usually some linear combination of die area and total wirelength. Our *floorplanning engine* is adapted from the simulated annealing based floorplanner proposed in [19]. The input to the floorplanner is a list of components and their interconnections in the system. Each component has an area associated with it (obtained from RTL synthesis). Dimensions in the form of width and height (for "hard" components) or aspect ratio (for "soft" components) are also required for each component. Additionally, maximum die size and fixed locations for hard macros can also be specified as inputs. Given these inputs, our floorplanner minimizes the cost function

$$Cost = w_1.Area + w_2.Bus_{WL} + w_3.Total_{WL} \qquad ... (1)$$

where *Area* is the area of the chip, Bus_{WL} is the wire length corresponding to wires connecting components on a bus, $Total_{WL}$ is total wire length for all connections on the chip (including inter-bus connections) and w_1, w_2, w_3 are adjustable weights which are used to bias the solution. The floorplanner outputs a non overlapping placement of components from which the wire lengths can be calculated by using half-perimeter of the minimum bounding box containing all terminals of a wire (HPWL) [20].



(c) $$C_O = \sum_{j=1}^{k} C_j - C_L \qquad C_L = \sum_{j=1}^{k} \frac{\sum_{i=1}^{j} l_i}{l} . C_j$$

Figure 2. Transforming multiple pin net into a two pin net

Once the wire lengths have been calculated, the *delay estimation engine* is invoked. The wire delay is calculated based on formulations proposed in [21]. The inputs to this stage are the wire lengths from the floorplanner and the capacitive loads (C_L) of component output pins (obtained from RTL synthesis). We can simplify the multiple pin problem (which is representative of a bus line) depicted in Figure 2(a) to a two pin problem shown in Figure 2(b). Then the delay for a wire of length *l*, with optimal wire sizing (OWS) [21], is given as

$$T = R_d C_o + \left(\frac{\alpha_1 l}{W^2(\alpha_2 l)} + \frac{2\alpha_1 l}{W(\alpha_2 l)} + R_d c_f + \sqrt{R_d r c_a c_f l} \right) . l \qquad \text{... (2)}$$

where $\alpha_1 = \frac{1}{4} r c_a$, $\alpha_2 = \frac{1}{2}\sqrt{\frac{r c_a}{R_d C_L}}$ and *W(x)* is Lambert's *W* function defined as the value of *w* which satisfies $we^w = x$. R_d is the resistance of the driver, *l* is the wire length, C_O and C_L are capacitive loads which are calculated as shown in Figure 2(c) and the rest of the parameters are dependent on the process technology used – *r* is the sheet resistance in Ω/sq, c_a is unit area capacitance in fF/μm^2 and c_f is unit fringing capacitance in fF/μm (defined to be the sum of fringing and coupling capacitances). The values for these technology dependent parameters are listed in Table 1, and have been calculated from [22].

Table 1. Parameters based on NTRS 97

Tech (μm)	0.18	0.15	0.13
r	0.068	0.073	0.081
c_a	0.060	0.054	0.046
c_f	0.064	0.054	0.043

The delay estimation engine is ultimately used to check for bus cycle time violations in the design. This is illustrated through an example. Figure 3 shows a floorplan for a system where IP1 and IP2 are connected to the same bus as ASIC1, Mem4, ARM, VIC and DMA, and the bus has a speed of 333 Mhz. This implies that the bus cycle time is 3 ns. For a 0.13 μm process and a driver resistance value R_d of 0.4 kΩ, the floorplanner finds a wire length of 9.9 mm between pins connecting the two IPs to the bus, with C_L = 2.936 pF and C_O = 0.988 pF for the wire. The wire delay, obtained by inserting these values in (2), is found to be 3.5 ns. This violates the clock cycle time constraint of 3 ns, and we thus conclude that the bus architecture is not feasible. In this way, our floorplanning and delay estimation engines determine if a synthesized design is feasible or not. As we will show later, our synthesis flow attempts to automatically eliminate such violations once they are detected.

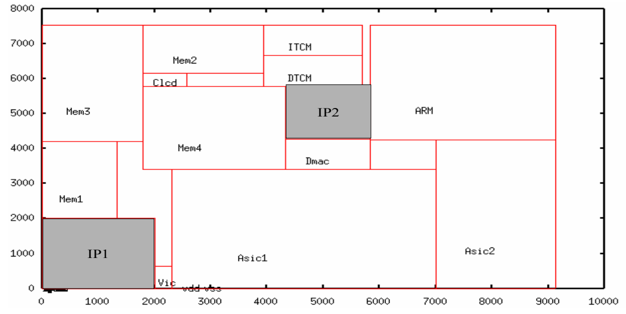


Figure 3. Example floorplan

3.5 Synthesis Approach

We now describe our automated synthesis approach in detail. Figure 4 gives a high level overview of the flow. The inputs to the flow include a Communication Throughput Graph (CTG), a target communication architecture (e.g. AMBA), a set of Communication Parameter Constraints (Ψ) and a library of behavioral IP models. The general idea is to first perform preprocessing transformations on the CTG to improve the performance of the entire system (*preprocess*) and then map all the components from the CTG to a simple bus topology. Then, we iteratively select a Throughput Constraint Path (TCP), starting from the TCP with the largest constraint and moving in descending order, from set Ω (which is a superset of all TCPs in the system) and search the communication parameter space for a suitable parameter configuration (*explore_params*) and possibly perform topology mutations if needed (*mutate_topology*) till all TCP constraints are satisfied. Once all TCP constraints are satisfied, we optimize the design (*optimize_design*) to lower the cost of the system and avoid possible timing violations. Next we invoke the floorplanning and delay estimation engines to detect bus cycle time violations. If timing violations are detected, we update Ω and repeat the topology mutation and parameter exploration phase, or proceed to output the synthesized system and floorplan if there is no violation.

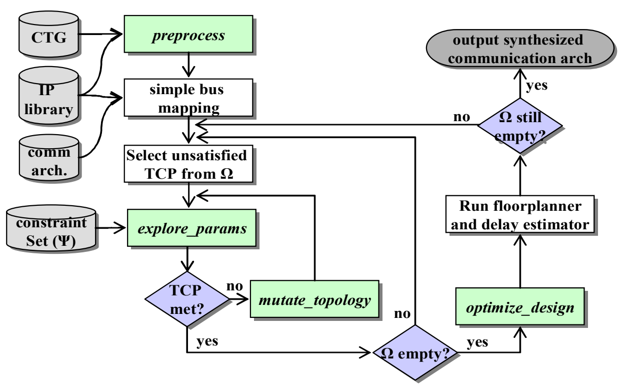


Figure 4. Automated Synthesis Flow

Figure 5 shows the pseudo code for the *preprocess* stage. In the first step we map the components in the CTG from the behavioral IP library database to a simple, protocol-independent, transaction-level simulation model in SystemC [24] having a virtual channel for every edge in the graph. This model has no contention since there are no shared channels and also because we assume infinite ports at IP interfaces. The purpose of this step is to obtain, through simulation, a memory usage profile (Step 2). Once we have obtained this profile, we attempt to split those memory nodes for which different masters access non-overlapping regions (Step 3). Finally we merge local slave nodes with their master nodes to reduce contention and loading on shared busses (Step 4). Note that we perform Step 3 before Step 4 because it allows us to generate local memories which can then be merged with their corresponding masters.

```
Step 1:   Map CTG to protocol-independent TLM
Step 2:   Simulate design
          Generate usage profile for memory modules
Step 3:   Split memory nodes wherever applicable
Step 4:   Merge local memory/slave nodes with master nodes
```

Figure 5. *preprocess* procedure

After the preprocessing stage, all the components in the enhanced CTG and the selected bus architecture are mapped from the IP library database to the fast transaction-based bus cycle-accurate simulation model (Section 3.2) with a simple bus topology – a single shared *main* and a single shared *peripheral* bus. As mentioned earlier, every node in a CTG has information relating to the type of bus it can be connected to, which guides the mapping process. Once the simple topology is created, we select the largest unsatisfied TCP constraint from set Ω and search for a suitable combination of communication parameter values to satisfy the constraint in the *explore_params* stage. Figure 6 gives the pseudo code for this procedure. *explore_params* searches for a suitable combination of parameter values which satisfies the TCP constraint under consideration, for the current topology. The parameter values are bounded by the constraint set Ψ specified by the designer. However, the exploration space arising from the combinations of the bounded values can still be very large. In the interest of achieving practical running times, we set the bus widths and speeds to the maximum allowed values set by the designer in Ψ (Step 1). We then select a combination of a valid arbitration priority ordering and DMA burst size, and proceed to simulate the design (Steps 2, 3). The best result configuration in Step 3 is the combination of parameters for which the least number of TCP constraints are violated and the throughput for the TCP being considered is the highest. The set of valid arbitration priorities is governed by the following rules: (a) priorities of masters in TCPs with larger throughput constraints are always greater than priorities of masters in TCPs with lower throughput constraints, (b) once a TCP constraint is satisfied, the relative arbitration priority ordering for masters in the TCP is updated (Step 5) and not changed anymore and (c) only combinations of priority orderings within the TCP under consideration need to be explored if the previous two rules are followed. These three rules reduce the large arbitration space and make it more manageable. The set of valid DMA burst sizes is governed by the following rule: (a) once a TCP constraint is satisfied, only those DMA burst size values which did not violate the satisfied TCP constraint are considered for subsequent TCPs. Thus, as TCP constraints are satisfied, the set of valid DMA burst size values shrinks, reducing the DMA burst size exploration space. Figure 6 shows how once a TCP constraint is satisfied, we simulate the design for different DMA burst size values to generate an updated set of allowed DMA burst sizes (Step 6) which will be used for subsequent TCP explorations.

```
Step 1:  Set bus speed, bus width to maximum allowed in set Ψ
Step 2:  Select a combination of valid arbitration priority ordering and
         valid DMA burst size. Exit if all valid combinations exhausted
Step 3:  Simulate design
         Update best result configuration
Step 4:  If TCP constraint not satisfied or previously satisfied TCP
         constraint violated, goto Step 2
Step 5:  Update Ω and arbitration priority ordering for masters in TCP
Step 6:  Simulate design for remaining DMA burst sizes and update
         allowed DMA burst size set.
```

Figure 6. *explore_params* procedure

If the TCP constraint is not satisfied for any combination of communication parameter values, we attempt to change the communication topology in the *mutate_topology* stage. Figure 7 shows the pseudo code for this procedure. To meet TCP constraints, we need to eliminate conflict on shared busses, and this can be done by creating new busses and migrating IPs, from the TCP being considered, iteratively to the new bus till the conflict is resolved. In *mutate_topology*, we first choose an unselected master in the current TCP, create a new bus and migrate the master to the new bus (Step 2). In subsequent iterations of *mutate_topology*, we migrate the slaves in the current TCP to the new bus (Step 3). Once all slaves in the current TCP have been considered for migration and the TCP is still not satisfied, we check for unselected masters in the current TCP (Step 4). If there are still unselected masters remaining, we undo all slave migrations since the last master migration, mark the slaves in the TCP as being unselected and migrate a randomly chosen previously unselected master to the new bus (Step 4a). In subsequent iterations we again migrate the slaves to the new bus (Step 3). After all masters and slaves in the current TCP have been moved to the new bus or at least considered for migration, it is possible that the TCP constraint is still not met. In that case, we mark all the master and slave IPs in the TCP as unselected, randomly select a master on the previously created bus and permanently assign it to that bus, create another bus and starting from a randomly selected master, iteratively migrate IPs to that bus (Step 4b). In this way, new busses are created till enough bandwidth is available to satisfy the TCP throughput constraint. Note that if a topology mutation causes the best result configuration from *explore_params* to violate any previously satisfied TCP constraints, we undo the mutation (Step 1). Otherwise we keep the mutation, even if it deteriorates current TCP performance slightly. This allows us to take into account the effect of local minima in the exploration phase.

```
Step 1:   If previous mutation caused any satisfied TCP constraint to be
          violated, undo mutation
Step 2:   If (no master in current TCP selected yet)
          Create new bus
          Goto Step 6
Step3:    If (TCP master selected) and (unselected TCP slaves remain)
          Goto Step 5
Step 4:   If (all TCP slaves already selected)
Step 4a:    If (unselected TCP master remains)
              Undo all slave migrations since last master migration
              Mark slaves in TCP as unselected
              Goto Step 6
Step 4b:    If (all TCP masters already selected)
              Mark slaves in TCP as unselected
              Mark all masters in TCP as unselected
              Randomly select master on last created bus, permanently
                      assign master to that bus
             Create new bus
             Goto Step 6
Step 5:   Randomly select an unselected slave.
          Migrate to new bus. Exit
Step 6:   Randomly select an unselected master.
          Migrate to new bus. Exit
```

Figure 7. *mutate_topology* procedure

Once all the TCP constraints are satisfied, we arrive at the *optimize_design* stage. The pseudo code for this stage is given in Figure 8. The purpose of this stage is to reduce the ‘pessimistic’ high values we selected for bus widths and bus clock speeds, to reduce the cost of the final system. Here we iteratively consider each bus in the system and attempt to lower the value for bus width (Step 2) and bus clock speed (Step 4), without violating any TCP constraints. Reducing the bus speed in this stage also helps prevent physical timing violations since it lengthens the bus cycle time.

```
Step 1:  Select previously unselected bus from generated bus arch.
Step 2:  Reduce data bus width to next lower value
         Simulate design
Step 3:  If (TCP constraint violation), undo, else goto step 2
Step 4:  Reduce bus speed to next lower value
         Simulate design
Step 5:  If (TCP constraint violation), undo, else goto step 4
Step 6:  If all busses examined, exit, else goto step 1
```

Figure 8. *optimize_design* procedure

Next we pass the optimized system through our floorplanning and wire delay estimator engine. If a timing violation is detected, the set Ω is updated with TCPs which have components on the busses with violations, and we again go back and attempt to select appropriate parameter value combinations and a different topology which resolves the timing violations. Changing the topology by migrating IPs to a new bus, in particular, reduces capacitive loading and consequently wire delay on the bus, which is one of the primary causes of timing violation in a design (Section 3.4). Finally, after any violations have been resolved and all TCP constraints satisfied, we output the final synthesized bus topology, parameter values for bus speeds, data bus widths, DMA burst size and arbitration priority ordering, along with the feasible floorplan.

4. CASE STUDIES

We applied our automated bus-based communication architecture synthesis approach on two industrial strength designs from the network communication domain. In the first case study, we selected a network communication SoC subsystem used for fast data packet processing and forwarding. Figure 9 shows the CTG for this system. There are two data manipulation related TCP constraints that must be satisfied in this system. The first TCP involves the *encryption engine* and includes the ARM926, ASIC1, RAM3 and EXT_IF blocks. The EXT_IF block fetches data and stores it in RAM3. The ASIC1 and ARM926 blocks fetch non overlapping sections of the data, process them and store them back in RAM3, from where the EXT_IF block fetches and streams them out at a minimum rate of 200 Mbps. The second TCP involves the *USB subsystem*. Data packets received at the USB are routed to RAM1. The ARM926 reads this data, processes it and stores it back to RAM1 from where the DMA engine transfers it to SDRAM_IF, which streams it out at a minimum rate of 480 Mbps. There is also a third subsystem which involves the SWITCH, RAM2 and ARM926 components. However, this is a very low priority data path which has no data rate constraint from the designer, and therefore we do not classify it as another TCP to be satisfied.

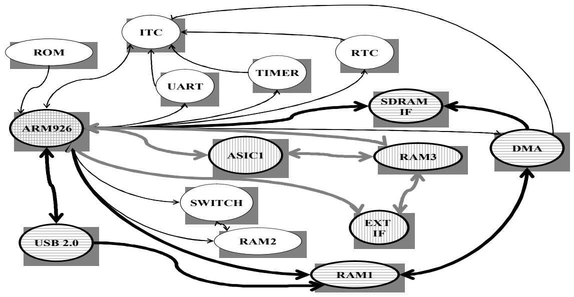


Figure 9. Network Communication Subsystem

Table 2 shows the *Communication Parameter Constraint* set (Ψ) for this case study. The target communication architecture for the automated synthesis is the AMBA2 high performance AHB bus and a low bandwidth APB bus [2]. For the floorplanner, we give maximum priority to minimizing wire length for components on a bus, and equal lower priorities for area and total wire length minimization.

Table 2. Customizable Parameter Set

Set	Values
bus width	8, 16, 32
bus speed	33, 66, 100, 133, 166, 200
DMA burst size	1, 2, 4, 8, 16
arbitration strategy	static priority

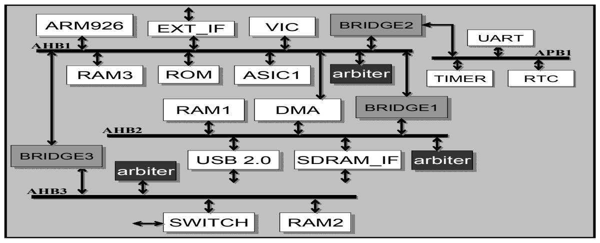


Figure 10. Synthesized SoC subsystem

Table 3. Communication Parameter Values

Parameter	Values			
	AHB1	AHB2	AHB3	APB1
bus width	32	32	32	32
bus speed	133	133	133	66
dma size	16			
arb priority	ARM>USB> DMA> EXT_IF>ASIC1>SWITCH			

Figure 10 shows the final output of our synthesis flow – a synthesized architecture which meets all throughput and timing constraints. The values for the generated communication parameters are given in Table 3 and the final floorplan for this system is shown in Figure 11. The automated synthesis engine initially created 2 AHB busses, with the SWITCH and RAM2 components connected to AHB1, which was assigned a clock speed of 200 Mhz to meet the encryption path throughput constraint. However, the floorplanning engine detected a cycle time violation for the bus due to excessive capacitive loading. The *topology_mutate* stage then split the shared AHB bus and assigned the ARM926, ASIC1 and EXT_IF masters and their associated slaves to one bus, and the SWITCH and RAM2 components to another AHB bus, to reduce capacitive loading. Finally, the *optimize_design* function reduced the bus speeds for the AHB busses from 200 Mhz to 133 Mhz and the APB bus to 66 Mhz, to lower the cost of the system. Both the throughput constraints were still met at these lower bus speeds. The synthesis engine made a simple assumption and assumed a 133 Mhz bus speed for AHB3 to simplify the design of BRIDGE3 to AHB1, but a designer can choose to further lower the AHB3 bus speed if a more complex bridge is acceptable.

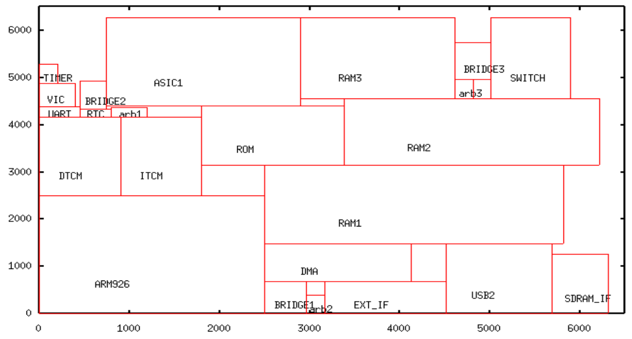


Figure 11. Floorplan for SoC subsystem

For our second case study, we considered a derivative of the network communication subsystem from Figure 9, which extends and partially modifies the functionality of the previous system. Figure 12 shows this derivative architecture, which has an additional TCP constraint involving the ARM926, SWITCH, RAM2 and two newly added components: a memory array (RAM4) and an ASIC block (ASIC2). In this TCP, data packets received from the SWITCH are stored in RAM2. These packets are retrieved by ASIC2 which reads and modifies some protocol header information before storing it back to RAM4 from where the SWITCH must stream it out at a minimum data rate of 3.2 Gbps. The ARM926 is used minimally, for directing data flow in this TCP.

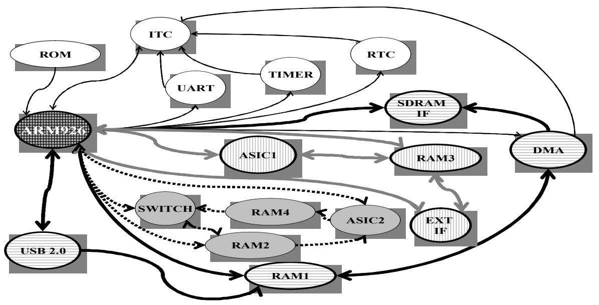


Figure 12. Network Communication Subsystem Derivative

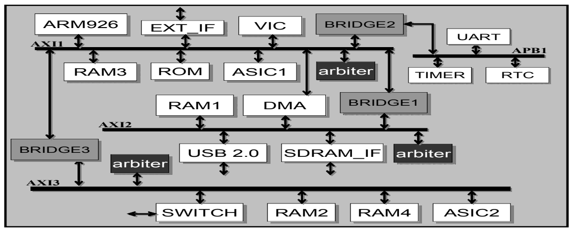


Figure 13. Synthesized subsystem for derivative architecture

Table 4. Communication parameter values (derivative arch.)

Parameter	Values			
	AXI1	AXI2	AXI3	APB1
bus width	32	32	64	32
bus speed	100	100	200	66
dma size	16			
arb scheme	SWITCH>ASIC2>ARM>USB>EXT_IF>DMA>ASIC1			

The *Communication Parameter Constraint* set (Ψ) is slightly modified from Table 2, with the addition of a larger data bus width value of 64, to handle the increased bandwidth requirements. Also, instead of using the AMBA2 AHB bus architecture, we modify the target communication architecture to AMBA3 AXI [25]. Our synthesis flow outputs the architecture shown in Figure 13. The values for the generated communication parameters are shown in Table 4 and the final floorplan is shown in Figure 14. Since AXI supports separate channels for reads and writes, the bus speeds required to maintain throughput are lower (100 Mhz). The AXI3 bus which supports the SWITCH TCP has a 64 bit data width and a high 200 Mhz bus clock speed in order to maintain the high data flow rate.

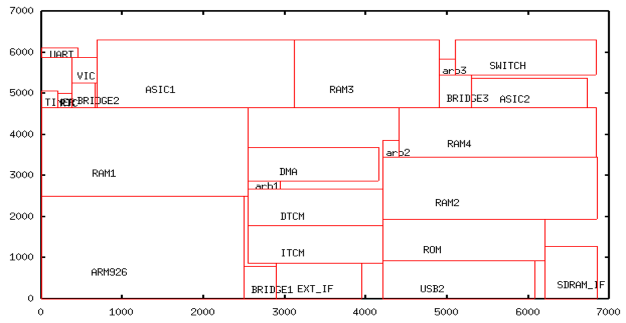


Figure 14. Final floorplan for derivative SoC subsystem

Table 5 compares the *final* synthesized designs for the two case studies with the results for the *initial* single main/peripheral shared bus mapped design, a synthesis flow *without floorplanner* and timing violation detection, and a *manual* synthesis effort by a designer. Compared with the initial design, the final synthesized design not only performs significantly better but also satisfies all constraints. For the flow without the floorplanning stage, although the number of busses is less and all throughput constraints are satisfied, the timing violations go undetected and the designs are not feasible. For the manual effort, although the performance is better than the final synthesized design, note that the number of busses is also more in both cases. The manual effort not only took longer (several days compared to a few hours for our automated flow) but also overdesigned the system and exceeded the requirements, ending up with a more expensive system.

Table 5. Synthesis Result Comparison

Case Study1 Designs	initial	w/o floorplanner	final	manual
Number of Busses	2	3	4	5
TCP constr. satisfied	0/2	2/2, but not feasible	2/2	2/2
Exec. cycles (millions)	49.76	24.51	20.32	18.8
Time to synthesize	~mins	~hours	~hours	~days

Case Study2 Designs	initial	w/o floorplanner	final	manual
Number of Busses	2	3	4	6
TCP constr. satisfied	0/3	3/3, but not feasible	3/3	3/3
Exec. cycles (millions)	88.48	47.63	29.10	26.58
Time to synthesize	~mins	~hours	~hours	~days

5. CONCLUSION

In this paper, we presented an approach for automating the synthesis of bus-based communication architectures for systems characterized by possibly several throughput constraints. Our approach synthesizes a low-cost bus topology and generates values for bus architecture parameters such as arbitration priority ordering, bus widths, bus speeds and a DMA burst size, required to meet the performance constraints in the design. In addition, we use a high level floorplanning and delay estimation engine to generate a layout of the components on the chip and detect bus cycle time violations early in the design flow. Results from the automated synthesis of AMBA based bus architectures for the network communication subsystem case studies show the usefulness of our approach. Our approach reduces the exploration and design time by at least an order of magnitude when compared to a manual effort, while also guaranteeing feasibility of physical design. Furthermore, our approach is easily portable across different standard bus-based communication architectures such as CoreConnect [3] and OCP [4], and can be extended to automatically synthesize other bus architecture specific parameters such as out-of-order (OO) buffer sizes as well. Our future work will deal with optimality analysis and understanding issues involved with synthesis for SoC subsystems having a larger number of components than the case studies presented here.

6. ACKNOWLEDGEMENTS

This research was partially supported by grants from Conexant Systems Inc., CPCC fellowship and UC Micro (03-029). We would also like to thank Prof. Markov and the PARQUET group at the Univ. of Michigan.

7. REFERENCES

[1] D. Sylvester and K. Keutzer, “Getting to the bottom of deep sub-micron”, *In Proc. of ICCAD 1998*

[2] Flynn, “AMBA: enabling reusable on-chip designs” *In IEEE Micro, 1997*

[3] IBM CoreConnect *http://www.chips.ibm.com/products/powerpc/cores*

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

[5] S. Pasricha, N. Dutt, M. Ben-Romdhane, “Fast Exploration of Bus-based On-chip Communication Architectures”, *In Proc. of CODES-ISSS 2004*

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

[7] J. Daveau, et al “Protocol selection and interface generation for HW-SW codesign”, *In IEEE Trans. on VLSI System, Vol. 5, No. 1, March 1997*

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

[9] K. K. Ryu, V. J. Mooney III “Automated Bus Generation for Multiprocessor SoC Design”, *In Proc. of DATE 2003*

[10] A. Pinto et al “Constraint-driven communication synthesis”, *DAC 2002*

[11] D. Lyonnard et al “Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip”, *DAC 2001*

[12] K. Lahiri et al, “Efficient exploration of the SoC communication architecture design space”, *In Proc. of ICCAD 2000*

[13] C. Shin, et al “Fast Exploration of Parameterized Bus Architecture for Communication-Centric SoC Design”, *In Proc. of DATE 2004*

[14] N. Thepayasuwan, A. Doboli “Layout Conscious Bus Architecture Synthesis for Deep Submicron Systems on Chip”, *In Proc. of DATE 2004*

[15] M. Drinic et al. “Latency-guided on-chip bus network design”, *In Proc. of ICCAD 2000*

[16] J. Hu et al, “System-Level Point-to-Point Communication Synthesis Using Floorplanning Information”, *In Proc. of ASP-DAC 2002*

[17] R. P. Dick, N. K. Jha “MOCSYN: multiobjective core-based single-chip system synthesis”, *In Proc. of DATE 1999*

[18] R. A. Bergamaschi et al “SEAS: a system for early analysis of SoCs”, *In Proc. of CODES-ISSS 2003*

[19] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design", *In IEEE Trans TVLSI, Dec. 2003*

[20] A. E. Caldwell, et al, “On Wirelength Estimations for Row-based Placement”, *In IEEE Trans. on ICCAD, vol.18, (no.9), IEEE, Sept. 1999*

[21] J. Cong, D. Z. Pan, “Interconnect Performance Estimation Models for Design Planning”, *In IEEE Trans. on ICCAD, Vol 20, No. 6, June 2001*

[22] Semiconductor Industry Association, “National Technology Roadmap for Semiconductors”, *SIA 1997*

[23] SystemC initiative. *www.systemc.org*

[24] S. Pasricha, “Transaction Level Modeling of SoC with SystemC 2.0”, *In Proc. of Synopsys User Group Conference (SNUG), 2002*

[25] AMBA AXI Specification *www.arm.com/armtech/AXI*

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