FABSYN: Floorplan-Aware Bus Architecture Synthesis

Sudeep Pasricha, *Student Member, IEEE*, Nikil D. Dutt, *Senior Member, IEEE*, Elaheh Bozorgzadeh, *Member, IEEE*, and Mohamed Ben-Romdhane, *Member, IEEE*

***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 floorplan-aware bus architecture synthesis (FABSYN) to synthesize cost-effective, bus-based communication architectures that satisfy the performance constraints in a design. Our synthesis approach incorporates a high-level floorplanning and wire delay estimation engine to evaluate the feasibility of the synthesized bus architecture and detect bus cycle time violations early in the design flow, at the system level. We present case studies of network communication SoC subsystems for which we synthesized bus architectures, detected and eliminated timing violations, and generated core placements in a matter of hours instead of several days for a manual effort.**

***Index Terms*—Bus architecture synthesis, high level floorplanning, on-chip communication architecture, system-on-chip (SoC).**

I. 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 complex 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 system-on-chip (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 the advanced microprocessor architecture (AMBA) [2] and CoreConnect [3] architectures, 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, direct memory access (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 an SoC design process. However, the complex interplay between communication architecture parameters is becoming hard to analyze effectively, especially at the system level. 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 floorplan-aware bus architecture synthesis (FABSYN) approach in this paper, which automates the generation of a cost effective communication architecture for an SoC. We make use of SystemC [23] to quickly capture components at the behavioral 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 clock 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. 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 bus clock cycle time constraints (described in more detail in Section III-E) detected late in the flow at the physical implementation stage can require changes in the architecture which can severely impact time-to-market. Since the bus architecture synthesis process determines the number and type of components assigned to each bus, which decides the cumulative load capacitance on a bus and which, in turn, has a direct impact on signal delay and bus clock cycle time constraint satisfiability (Section III-E), there is a need to make the synthesis process more physically aware. Our high-level floorplanning and wire delay estimation engines detect bus cycle time violations early in the design flow at the system level, during the syn-

Manuscript received June 21, 2005; revised September 14, 2005 and December 8, 2005. This work was supported in part by grants from Conexant Systems Incorporated, CPCC fellowship, and UC Micro under 03-029.

S. Pasricha, N. D. Dutt, and E. Bozorgzadeh are with the School of Information and Computer Science, University of California, Irvine, CA 92617 USA (e-mail: sudeep@cecs.uci.edu; dutt@cecs.uci.edu; eli@ics.uci.edu).

M. Ben-Romdhane was with Conexant Systems Inc., Newport Beach, CA 92660 USA. He is now with Newport Media Inc., Lake Forest, CA 92630 USA.

Digital Object Identifier 10.1109/TVLSI.2006.871763

thesis process, 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 and eliminated timing violations and generated core placements which satisfied performance constraints for the SoC subsystems in a matter of hours.

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 [6], interface synthesis and simple synchronization protocol selection [7], and topology generation for simple buses without arbitration [8]. Ryu *et al.* [9] performed studies to find optimal bus topologies for an 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 and eliminate bus cycle time violations early in the design flow. We believe that this step will become increasingly important in the deep-submicrometer era as clock speeds increase and lengthy propagation delays cause frequent violations of timing constraints that will need to be detected and corrected early in the design flow if shrinking time-to-market constraints are to be met.

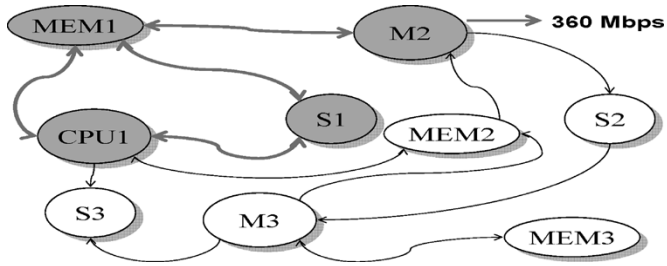


Fig. 1. Communication Throughput Graph (CTG).

III. Automated Bus Synthesis

This section describes our approach for automated bus architecture synthesis. Section III-A discusses how SoC performance requirements are represented in our approach. Section III-B presents our assumptions and states the problem description. Section III-C discusses the simulation engine while Section III-D describes communication parameter constraints, which guide the bus synthesis process. Section III-E gives an overview of our floorplan and wire delay calculation engines used for detecting timing violations in the design. Finally, Section III-F presents our automated bus architecture synthesis approach in detail.

A. SoC Performance Requirements

Typically, SoC designs need to satisfy performance requirements that are 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 that need to be satisfied for the system. These constraints can involve communication between two or more IPs.

Fig. 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 bounds on 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)***. Fig. 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 Mb/s. 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.

B. Problem Description

We are given an application for which we assume the HW/SW partitioning has already been performed. The resulting SoC design has possibly several hardware and software components (IPs) onto which application functionality has been mapped and which 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. 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.

The goal of the FABSYN communication architecture synthesis approach is to determine the number of buses and the allocation of SoC IPs on these buses (bus topology synthesis), and generate values for arbitration priorities, data bus widths, bus clock speeds, and DMA burst sizes (bus architecture parameter synthesis) for the selected standard bus-based communication architecture, while ensuring that all system throughput constraints are satisfied. In addition, we want to consider layout information of the chip to detect bus cycle time violations early in the design flow, so that we can modify the bus architecture to eliminate these violations which might otherwise take up costly design iterations later in the flow.

This leads us to our problem definition:

Problem Definition: A bus $\boldsymbol{B}$ can be considered to be a partition of the set of components $\boldsymbol{V}$ in a CTG, where $\boldsymbol{B} \subset \boldsymbol{V}$. Our primary objective is to determine a component to bus assignment for a hierarchical bus architecture, such that the partitioning of $\boldsymbol{V}$ onto $\boldsymbol{N}$ buses results in a minimal number of buses $|\boldsymbol{N}|$ and satisfies bus cycle timing constraints, while meeting all performance requirements in the design, represented by the TCPs in a CTG. As a secondary objective, we attempt to reduce the clock speeds and data widths of the buses in the synthesized solution.

C. Simulation Engine

Since communication behavior is characterized by unpredictability due to dynamic bus requests from cores, nondeterministic bus contention delays, buffer overflow delays 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 kHz [5], while running embedded software applications on processor instruction-set simulator (ISS) models.

D. 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 buses 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.

E. 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 nonoverlapping 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 bounds on 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

$$\text{Cost} = w_1.\text{Area} + w_2.\text{Bus}_{WL} + w_3.\text{Total}_{WL} \qquad (1)$$

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 nonoverlapping 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].

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 net problem (which is representative of a bus line) depicted in Fig. 2(a) to multiple two pin

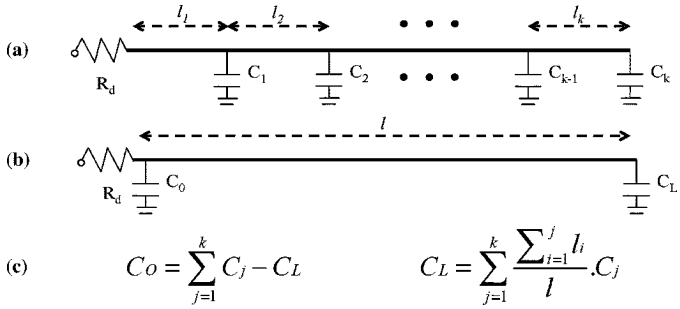


Fig. 2. Transforming multiple pin net into a two-pin net.

TABLE I
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

net problems, as shown in Fig. 2(b). Then the delay for a wire of length l, with optimal wire sizing (OWS) [21], is given as

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

where $\alpha 1 = (1/4) r c_a$, $\alpha 2 = (1/2)\sqrt{r c_a / R_d C_L}$, 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 Fig. 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 $fF/\mu\mathrm{m}^2$, and c_f is unit fringing capacitance in $fF/\mu\mathrm{m}$ (defined to be the sum of fringing and coupling capacitances). The values for these technology dependent parameters are listed in Table I, and have been calculated from [22].

The delay estimation engine is ultimately used to check for bus cycle time violations in the design. This is illustrated through an example. Fig. 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, which clearly violates the bus clock cycle time constraint of 3 ns. In this way, our floorplanning and wire delay estimation engines can determine if a synthesized design has buses with clock cycle timing violations. Typically, once such violations are detected at the physical implementation stage in the design flow, designers end up pipelining the buses by inserting latches, flip-flops, or register slices on the bus, in order to meet bus cycle time constraints. However, we found that such pipelining

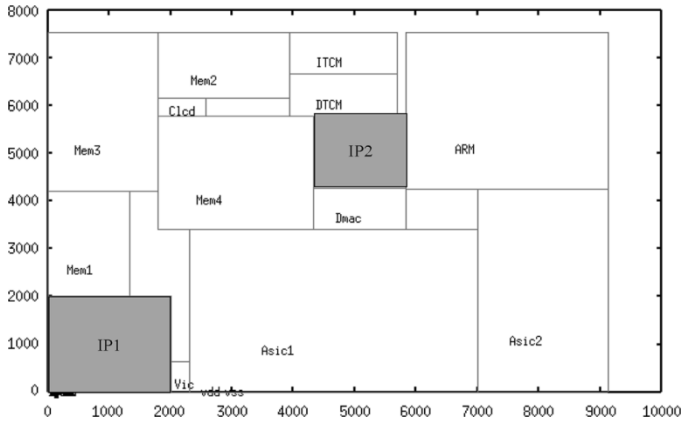


Fig. 3. Example floorplan.

of the bus can not only have an adverse effect on critical path performance, but also requires tedious manual reworking of RTL code and extensive reverification of the design, which can be very time consuming. As we will show later, our synthesis flow attempts to automatically eliminate such violations early in the design flow at the system level once they are detected.

F. Synthesis Approach

In this section, we describe our bus architecture synthesis approach. First, we will present a few definitions that will be used later when explaining the synthesis flow in more detail.

Definitions: Let $\mathrm{CTG} = G(V_{\mathrm{CTG}}, A_{\mathrm{CTG}})$ be a Communication Throughput Graph, where V_{CTG} is the set of vertices, each of which represents a component (a master or a slave) in the design, and A_{CTG} is the set of edges used to connect the components in V_{CTG} that need to communicate with each other. $\mathrm{SLV}_{\mathrm{CTG}}$ is the set of slave components in V_{CTG}, where $\mathrm{SLV}_{\mathrm{CTG}} \subset V_{\mathrm{CTG}}$. $\mathrm{MEM}_{\mathrm{CTG}}$ is the set of memory components in $\mathrm{SLV}_{\mathrm{CTG}}$, such that $\mathrm{MEM}_{\mathrm{CTG}} \subset \mathrm{SLV}_{\mathrm{CTG}}$. $L_{\mathrm{CTG}} = \{l_1, l_2, \ldots, l_n\}$ is a set of slave leaf components (i.e., slave components with a single incident edge connecting them to a single master component) in the CTG ($L_{\mathrm{CTG}} \subset \mathrm{SLV}_{\mathrm{CTG}}$), and where master($l_n$) refers to the master connected to the leaf component l_n. Next, let $\Omega = \{\mathrm{TCP}_1, \mathrm{TCP}_2, \ldots, \mathrm{TCP}_n\}$ be a superset of all throughput constraint paths (TCPs) in a CTG, where each TCP in Ω is itself a set of vertices representing the components that are part of the TCP, as discussed previously in Section III-B. $\mathrm{MAST}_{\mathrm{TCP}n}$ is the set of master components and $\mathrm{SLV}_{\mathrm{TCP}n}$ is the set of slave components in the constraint path TCP_n, such that $\mathrm{TCP}_n = \mathrm{SLV}_{\mathrm{TCP}n} \cup \mathrm{MAST}_{\mathrm{TCP}n}$.

We now describe our automated synthesis approach in detail. Fig. 4 gives a high level overview of the flow. The inputs to the flow include a Communication Throughput Graph, a target bus-based 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 of the target bus-based communication architecture. Then, we iteratively select a Throughput Constraint Path (TCP) from set Ω, starting from the TCP with the most stringent constraint, and search the

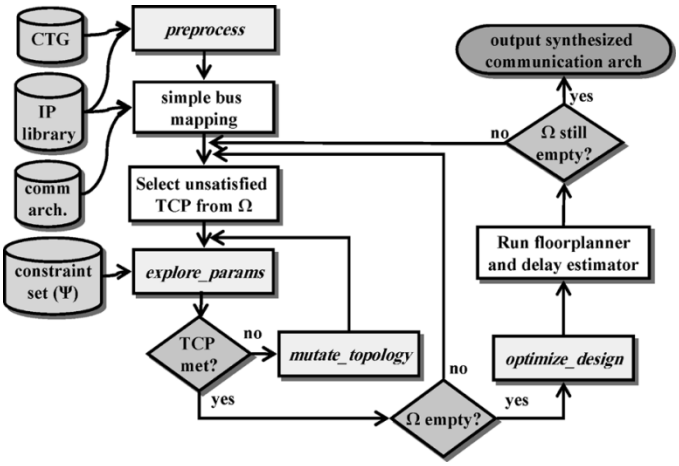


Fig. 4. FABSYN bus architecture synthesis flow.

Step 1: *map CTG to communication protocol-independent TLM*
Step 2: *simulate design ; record memory data traffic profiles*
Step 3: *for each mem_node* $\in$ MEM_{CTG}
if NonOverlapAccess(mem_node)
SplitNode(mem_node)
Step 4: *for each slave_node* $\in$ L_{CTG}
MergeNode(slave_node, master(slave_node))

Fig. 5. Preprocess procedure.

communication parameter space for a suitable parameter configuration (*explore_params*) and possibly perform topology mutations if needed (*mutate_topology*) till the TCP constraint is satisfied. Once all TCP constraints are satisfied, we optimize the design (*optimize_design*) to further lower the cost of the system. Next, we invoke the floorplanning and delay estimation engines to detect bus cycle time violations. If timing violations are detected, we update Ω with the TCPs having components on the buses with violations, and use a feedback loop to re-enter the flow to repeat the topology mutation and parameter exploration phase to eliminate these violations or proceed to output the synthesized system and floorplan once there are no violations.

Fig. 5 shows the pseudocode for the *preprocess* stage. In the first step we map the components in the CTG from the behavioral IP library database to a bus 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 nonoverlapping regions (Step 3). Finally we merge local slave nodes with their master nodes to reduce contention and loading on shared buses (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. Fig. 6(a)–(c) illustrates this process. The CTG shown in Fig. 6(a) is taken through the preprocess procedure and the MEM2 node is split, as shown in Fig. 6(b), into two nodes (MEM2a and MEM2b), since CPU1 accesses a region of memory which is distinct from that accessed by masters M2 and M3. Finally, the leaf slave nodes for CPU1 (slave nodes Mem2a and S4) are merged with CPU1 into a *hypernode*, as shown in Fig. 6(c).

After the *preprocess* 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 III-C) 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. A bus B can be considered to be a partition of nodes V_{CTG} in a CTG, such that $B \subset V_{\mathrm{CTG}}$. Fig. 6(d) shows the mapped components on the main and peripheral bus partitions, for the preprocessed CTG in Fig. 6(c).

Once the simple topology has been 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 (Fig. 4). Fig. 7 gives the pseudocode for this procedure. The *explore_params* procedure searches for a suitable combination of parameter values which satisfies the TCP constraint under consideration, for the current bus 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 must further prune this space.

We start by decoupling the bus widths and speeds from the arbitration schemes and DMA burst sizes. We set the bus widths and speeds to the maximum allowed values set by the designer in Ψ (Step 1). We do this because if TCP constraints are not met for the maximum values of bus widths and speeds, they will certainly not be met for lower values of these parameters. We cannot, however, set the DMA burst size to its maximum value and the arbitration priority to a fixed value, and make the same guarantee. Therefore, Step 1 allows us to quickly prune only the bus width and speed parameter space. Next, we select a combination of a valid arbitration priority ordering and DMA burst size, and then proceed to simulate the design (Steps 2 and 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: 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. Fig. 7 shows how once a TCP constraint is

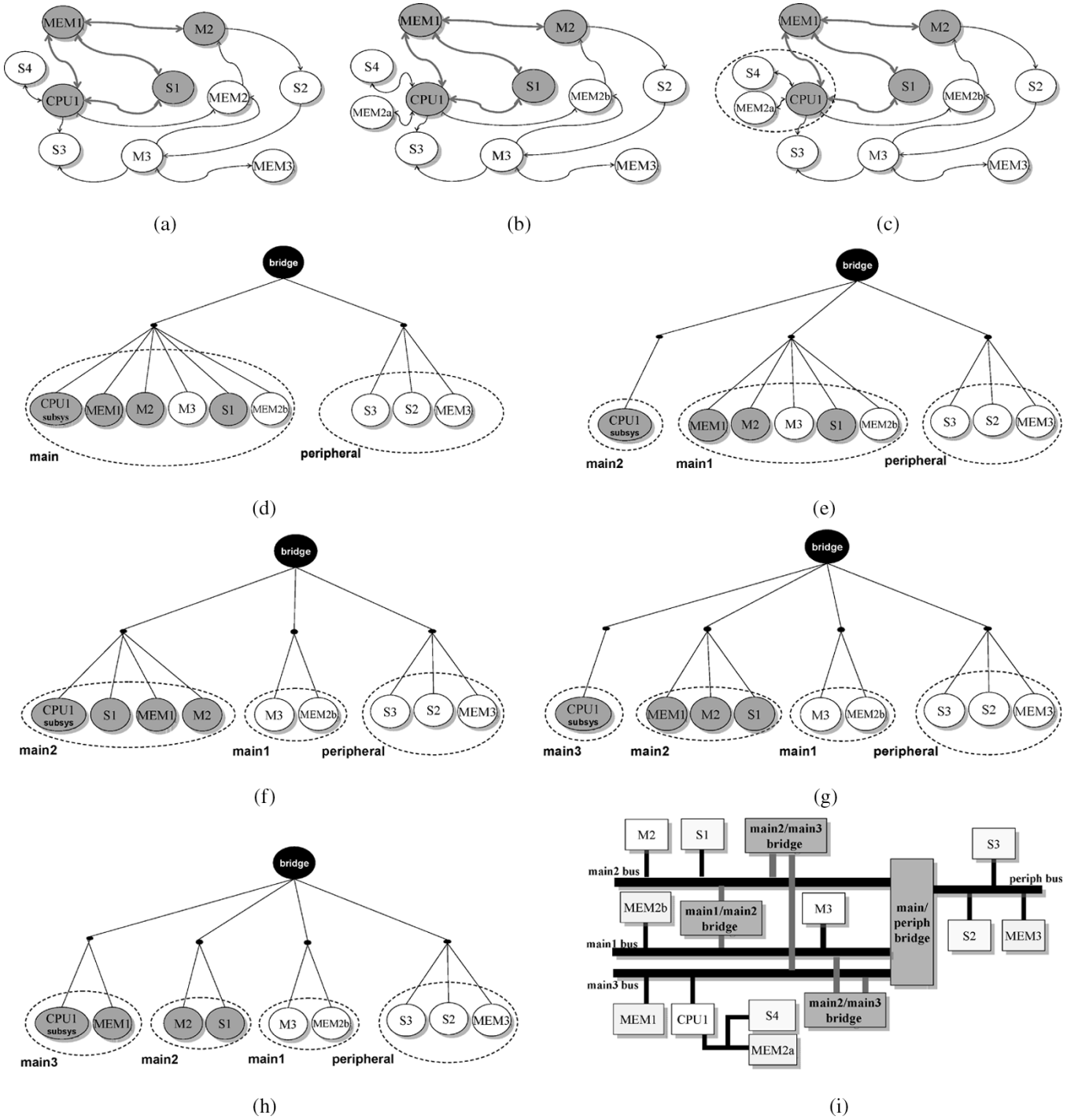


Fig. 6. FABSYN bus architecture synthesis illustration. (a) Initial system CTG, (b) after splitting MEM2, (c) merging local nodes to create *hypernode*, (d) after mapping nodes to main and peripheral buses, (e) after the first call to *mutate_topology*, (f) after migrating all nodes in TCP to main2, (g) creating a new bus and migrating CPU1 to it, (h) final topology with TCP satisfied, and (i) synthesized bus architecture.

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.

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. Fig. 8 shows the pseudocode for this procedure. To meet TCP constraints, we need to eliminate conflict on shared buses, and this can be done by creating a new bus and migrating IPs, from the TCP being considered, iteratively to the new bus until the conflict is resolved.

In *mutate_topology*, we first check to see if this is the first time that the procedure has been called, and if so, then we create a new bus, choose an unselected master at random, and migrate the master to the new bus (Steps 2 and 6). If it is the first time that the procedure has been called, then none of the masters in $\mathrm{MAST}_{\mathrm{TCP}}$ have been previously selected for migration, and the function call NoneSelected $(\mathrm{MAST}_{\mathrm{TCP}})$ returns a true

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*
Update allowed DMA burst size set.

Fig. 7. *explore_params* procedure.

Step 1: *if previous mutation violates a satisfied TCP constraint,*
UndoPreviousNodeMigration()
Step2: *if (NoneSelected(*$MAST_{TCP}$*))*
B = CreateNewBus()
Goto Step 6
Step 3: *if (!AllSelected(*SLV_{TCP}*))*
Goto Step 7
Step 4: *if ((!AllSelected(*$MAST_{TCP}$*))&&(AllSelected(*SLV_{TCP}*))*
UndoNodeMigration(SLV_{TCP}*)*
MarkUnselected(SLV_{TCP}*)*
Goto Step 6
Step 5: *if ((AllSelected(*$MAST_{TCP}$*))&&(AllSelected(*SLV_{TCP}*))*
MarkUnselected(SLV_{TCP}*)*
MarkUnselected($MAST_{TCP}$*)*
m_node = random(unselected($MAST_{TCP}$*))*
MarkSelectedPermanent(m_node)
B = CreateNewBus()
if (!AllSelected($MAST_{TCP}$*)*
Goto Step 6
else
Goto Step 7
Step 6: *m_node = random(unselected(*$MAST_{TCP}$*))*
MarkSelected(m_node)
MigrateNode(m_node, B)
exit;
Step 7: *s_node = random(unselected(*SLV_{TCP}*))*
MarkSelected(s_node)
MigrateNode(s_node, B)
exit;

Fig. 8. *Mutate_topology* procedure.

value. In subsequent invocations of *mutate_topology*, we iteratively migrate the slaves in $\mathrm{SLV_{TCP}}$ to the new bus (Steps 3 and 7). The function call AllSelected $(\mathrm{SLV_{TCP}})$ returns a false value if there are any remaining slaves in $\mathrm{SLV_{TCP}}$ which have yet to be selected for migration. Once all slaves in $\mathrm{SLV_{TCP}}$ 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 by calling UndoNodeMigration $(\mathrm{SLV_{TCP}})$, mark the slaves as being unselected, and migrate a randomly chosen previously unselected master to the new bus (Steps 4 and 6). In subsequent invocations of *mutate_topology*, we again migrate the slaves to the new bus (Steps 3 and 7). 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 (Step 5). In that case, we mark all the master and slaves 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 (or a randomly selected slave if there are no more masters to migrate), we iteratively migrate IPs to that bus (Steps 5 and 6). In this way, new buses are created until enough bandwidth is available to satisfy the TCP 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: *Select previously unselected bus from generated bus architecture*
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*

Fig. 9. *optimize_design* procedure.

Fig. 6(e)–(h) illustrates the topology mutation process, starting from the simple bus mapping in Fig. 6(d). The components in the TCP are shown in gray; $\mathrm{MAST_{TCP}} = \{\mathrm{CPU1}, \mathrm{M2}\}$ and $\mathrm{SLV_{TCP}} = \{\mathrm{S1}, \mathrm{MEM1}\}$. The result of the first invocation of *mutate_topology* is shown in Fig. 6(e), which depicts a newly created bus onto which the CPU1 master has been migrated. Subsequent calls to the procedure iteratively migrate the rest of the components in the TCP to the new bus. However, the TCP constraint is not satisfied for any of the intermediate topologies, due to data traffic conflicts on both the *main1* and *main2* buses, even when all the components in the TCP have been migrated to a separate bus, as shown in Fig. 6(f). Therefore, we proceed to create another bus (*main3*) and first migrate a master (CPU1) as shown in Fig. 6(g), followed by slaves in the TCP. For the configuration shown in Fig. 6(h), after MEM1 has been migrated to the new bus, the throughput constraint is found to be satisfied, and no more topology mutation is required, unless there is a timing violation detected by the floorplanning and wire delay estimation engine later in the flow (Fig. 4).

Once all the TCP constraints are satisfied, we arrive at the *optimize_design* stage. The pseudocode for this stage is shown in Fig. 9. The purpose of this stage is to reduce the maximum values we selected earlier for bus widths and bus clock speeds. Here we iteratively consider each bus in the system and attempt to lower the value for data bus width (Step 2) and bus clock speed (Step 4), without violating any TCP constraints. Reducing the bus width reduces the number of wires in a bus and lowers the cost of the system. Reducing the bus speed on the other hand, reduces the probability of a bus cycle time violations since it lengthens the bus clock cycle time period. The order in which the bus width or the bus speed is reduced is flexible and is left to the designer.

Next, we pass the optimized system through our floorplanning and wire delay estimator engine. For the system shown in Fig. 6, we pass the final modified CTG shown in Fig. 6(h) to the engine. If a timing violation is detected (as discussed in

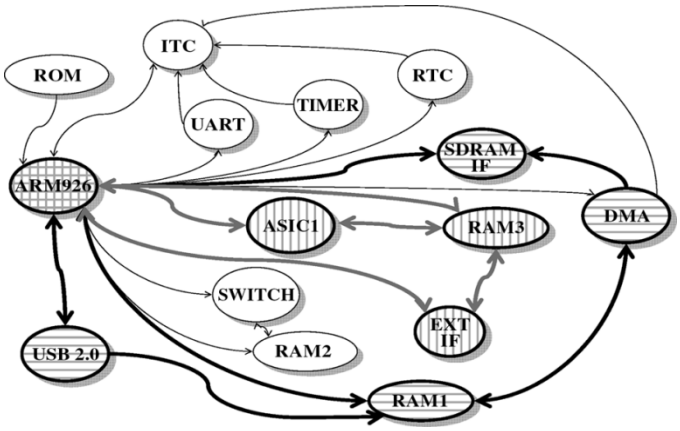


Fig. 10. Network communication subsystem.

Section III-E), the set Ω is updated with TCPs which have components on the buses with violations, and we use a feedback loop to go back and attempt to eliminate these violations. Since the cumulative capacitive load of components directly contributes to increasing signal propagation delay (Section III-E), we attempt to reduce the number of components on the bus having a violation. Therefore, when we go back into the flow using the feedback loop, we first select the TCP from Ω which has components on the violated bus with the largest load capacitance on its pins, and iteratively migrate them to another existing bus (or a new bus if migration to existing buses causes TCP constraint violations). If there is still a violation, we select another TCP from Ω and migrate components from that TCP away from the violated bus. We also give higher priority to reducing bus clock speed over reducing data bus width in the *optimize_design* stage, since reducing bus clock speed improves the probability of meeting the bus clock cycle period constraint. Note that the solution is guaranteed to converge when we use a feedback path. This is because in the worst case we end up creating a new bus (to migrate components away from the violated bus), which increases the cost of the system, but as a tradeoff we get improved system performance (even after we consider bridge overhead delays) and the ability to meet bus cycle time constraints.

Finally, after all 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. For the system shown in Fig. 6, the final synthesized architecture looks like the one shown in Fig. 6(i).

IV. Case Studies

We applied our automated bus-based communication architecture synthesis approach on three 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. Fig. 10 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 nonoverlapping sections of the data,

TABLE II
Customizable Parameter Set

Parameter	*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

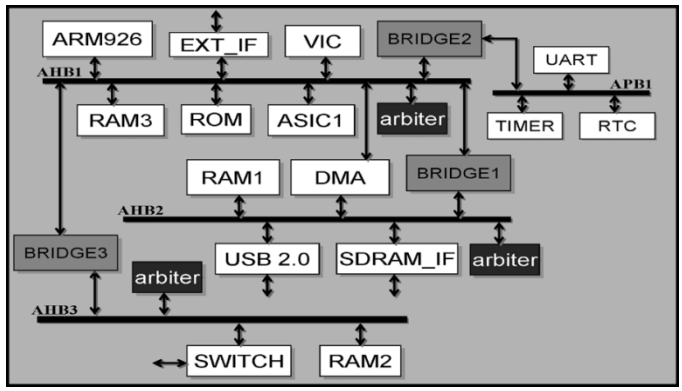


Fig. 11. Synthesized SoC subsystem.

TABLE III
Communication Parameter Values

Parameter	*Values*			
	AHB1	*AHB2*	*AHB3*	*APB1*
Bus width	32	32	32	32
Bus speed	133	133	133	66
DMA burst size	16			
Arbitration strategy	ARM>USB> DMA> EXT_IF>ASIC1>SWITCH			

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 Mb/s. 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 Mb/s. 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.

Table II 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.

Fig. 11 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 III and the final floorplan for this system is shown in Fig. 12. The automated synthesis engine initially

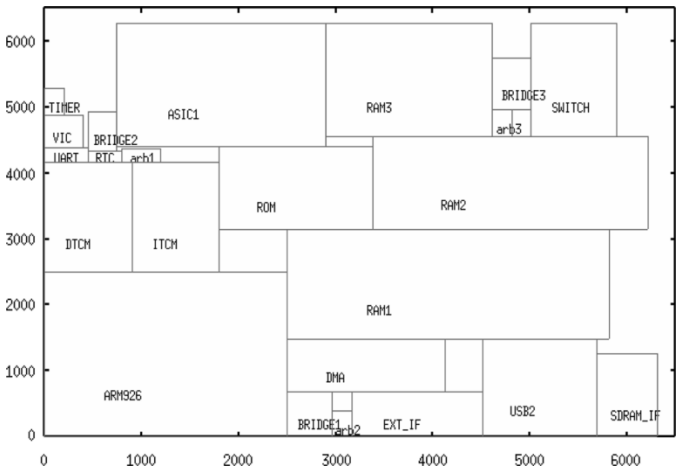


Fig. 12. Floorplan for SoC subsystem.

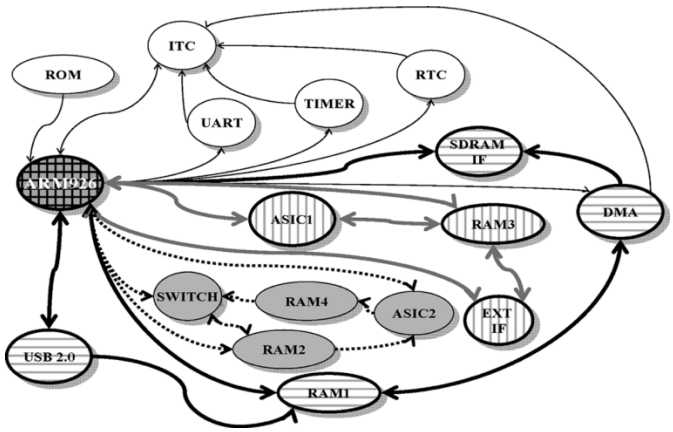


Fig. 13. Network communication subsystem derivative.

created 2 AHB buses, with the SWITCH and RAM2 components connected to AHB1, which was assigned a clock speed of 200 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 buses from 200 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.

For our second case study, we considered a derivative of the network communication subsystem from Fig. 10, which extends and partially modifies the functionality of the previous system. Fig. 13 shows this derivative architecture, which has an additional TCP constraint involving the ARM926, SWITCH, RAM2, and two newly added components: a memory array

TABLE IV
CUSTOMIZABLE PARAMETER SET

Parameter	*Values*
Bus width	8, 16, 32, 64
Bus speed	33, 66, 100, 133, 166, 200
DMA burst size	1, 2, 4, 8, 16
Arbitration strategy	static priority

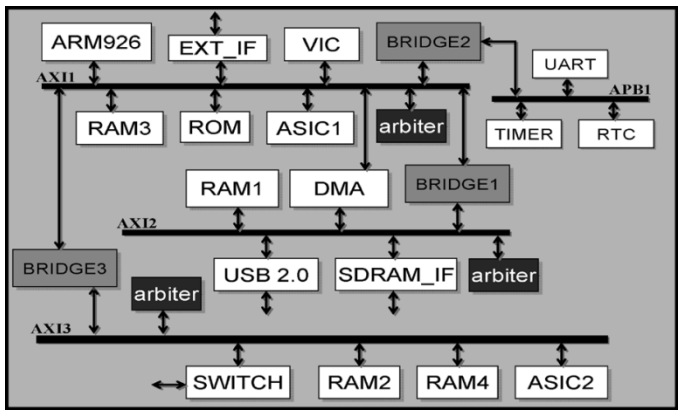


Fig. 14. Synthesized SoC subsystem for derivative architecture.

TABLE V
COMMUNICATION PARAMETER VALUES (DERIVATIVE ARCH)

Parameter	*Values*			
	AXI1	*AXI2*	*AXI3*	*APB1*
Bus width	32	32	64	32
Bus speed	100	100	200	66
DMA burst size	16			
Arbitration strategy	SWITCH>ASIC2>ARM>USB>EXT_IF>DMA>ASIC1			

(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 Gb/s. The ARM926 is used minimally, for directing data flow in this TCP.

The *Communication Parameter Constraint* set (Ψ) for this case study is shown in Table IV and is slightly modified from Table II, 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 like in the previous case, we modify the target communication architecture to AMBA3 AXI [25]. Our synthesis flow outputs the architecture shown in Fig. 14. The values for the generated communication parameters are shown in Table V and the final floorplan is shown in Fig. 15. 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

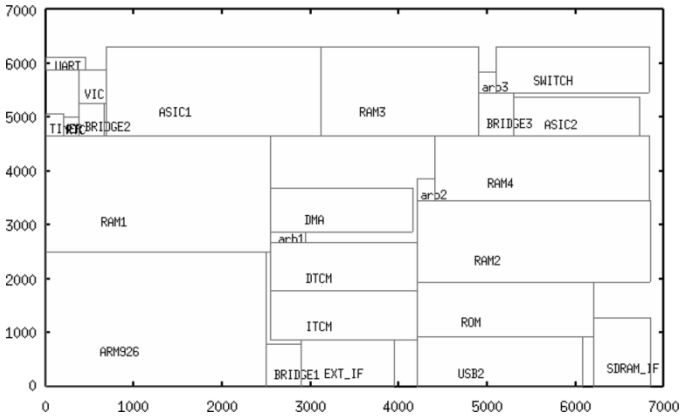


Fig. 15. Final floorplan for derivative SoC subsystem.

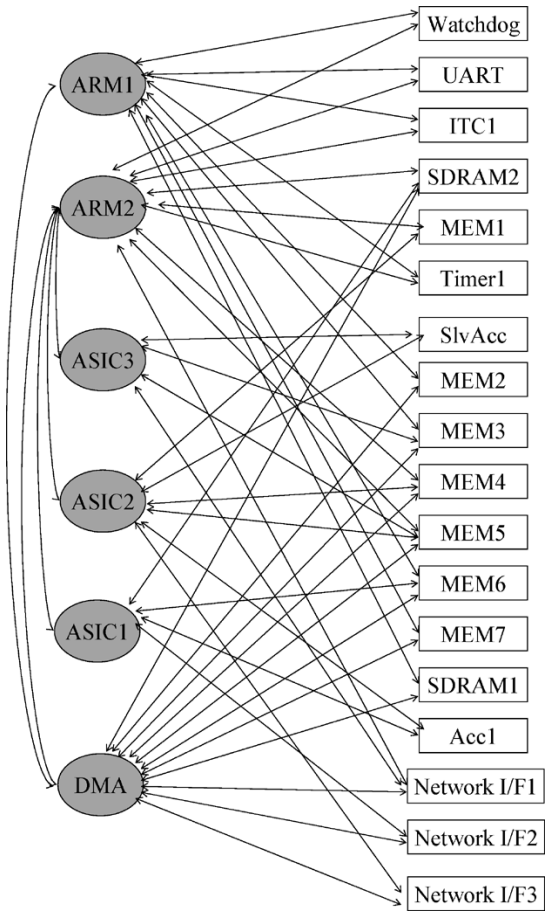


Fig. 16. MPSoC networking subsystem.

has a 64-bit data width and a high 200-MHz bus clock speed in order to maintain the high data flow rate.

For the third case study, we chose a multiprocessor system (MPSoC) networking subsystem. Fig. 16 shows the CTG for the system. For clarity, the TCPs are presented separately in Table VI. The *Communication Parameter Constraint* set (Ψ) is shown in Table VII. The target communication architecture for the synthesis process is the AMBA2 AHB bus architecture.

ARM1 is a protocol processor (PP) while ARM2 is a network processor (NP). The ARM1 PP is responsible for setting up and closing network connections, converting data from one protocol type to another and exchanging data with the NP using shared memory. The ARM2 NP directly interacts with the network ports and is used for assembling incoming packets into frames for the network connections, network port packet/cell flow control, keeping track of errors, and gathering statistics. The ASIC1 block performs hardware cryptography acceleration, while ASIC2 and ASIC3 are used for other data packet and frame processing. The DMA is used to handle fast memory to memory and network interface data transfers, freeing up the processors for more useful work.

TABLE VI
THROUGHPUT CONSTRAINT PATHS (TCPS)

IP cores in Throughput Constraint Path (TCP)	*Throughput Requirement*
ASIC1, Acc1, Network I/F2, MEM5, MEM6	1.2 Gbps
ARM2, MEM7, DMA, Network I/F1	360 Mbps
ASIC2, DMA, Network I/F3, MEM5, MEM1, SlvAcc	200 Mbps
ARM1, MEM2, DMA, SDRAM1, Network I/F1	640 Mbps
ASIC3, Network I/F1, MEM3, MEM4, Network I/F3	1 Gbps

TABLE VII
CUSTOMIZABLE PARAMETER SET

Parameter	*Values*
Bus width	**8, 16, 32, 64**
Bus speed	**50, 100, 150, 200, 250, 300**
DMA burst size	**1, 2, 4, 8, 16**
Arbitration strategy	**static priority**

The synthesis process first generated the system shown in Fig. 17(a). However, once we passed the architecture through the floorplanning and wire delay estimation stage, it was discovered that the system was not feasible because of the excessive cumulative load capacitance on the AHB1 bus, which caused a timing violation. Fig. 17(b) shows the floorplan layout for this configuration. The synthesis process records this violation, and resynthesizes the communication architecture to come up with the architecture shown in Fig. 17(c) with a reduced capacitive loading on AHB1 while still satisfying all TCP constraints. This architecture does not violate any bus cycle time constraints and the final floorplan is shown in Fig. 17(d). Note that the synthesis process splits the SDRAM2 and MEM4 components, moving portions of both these components to a local bus of the ARM2 processor. This reduces unnecessary traffic and capacitive loading on the shared AHB bus. The synthesized communication parameter values are shown in Table VIII. Since most of the streamed data was native 32 bits, a higher 64-bit bus width did not affect the performance significantly and the synthesized buses all have 32-bit data bus widths.

We now compare the quality of our synthesis process. Since none of the existing synthesis approaches are aimed at detecting bus cycle time violations early in the design flow, there is no direct point of comparison. We chose to compare the quality of our synthesized designs with an approach which maps all the components in the application to a single main/peripheral

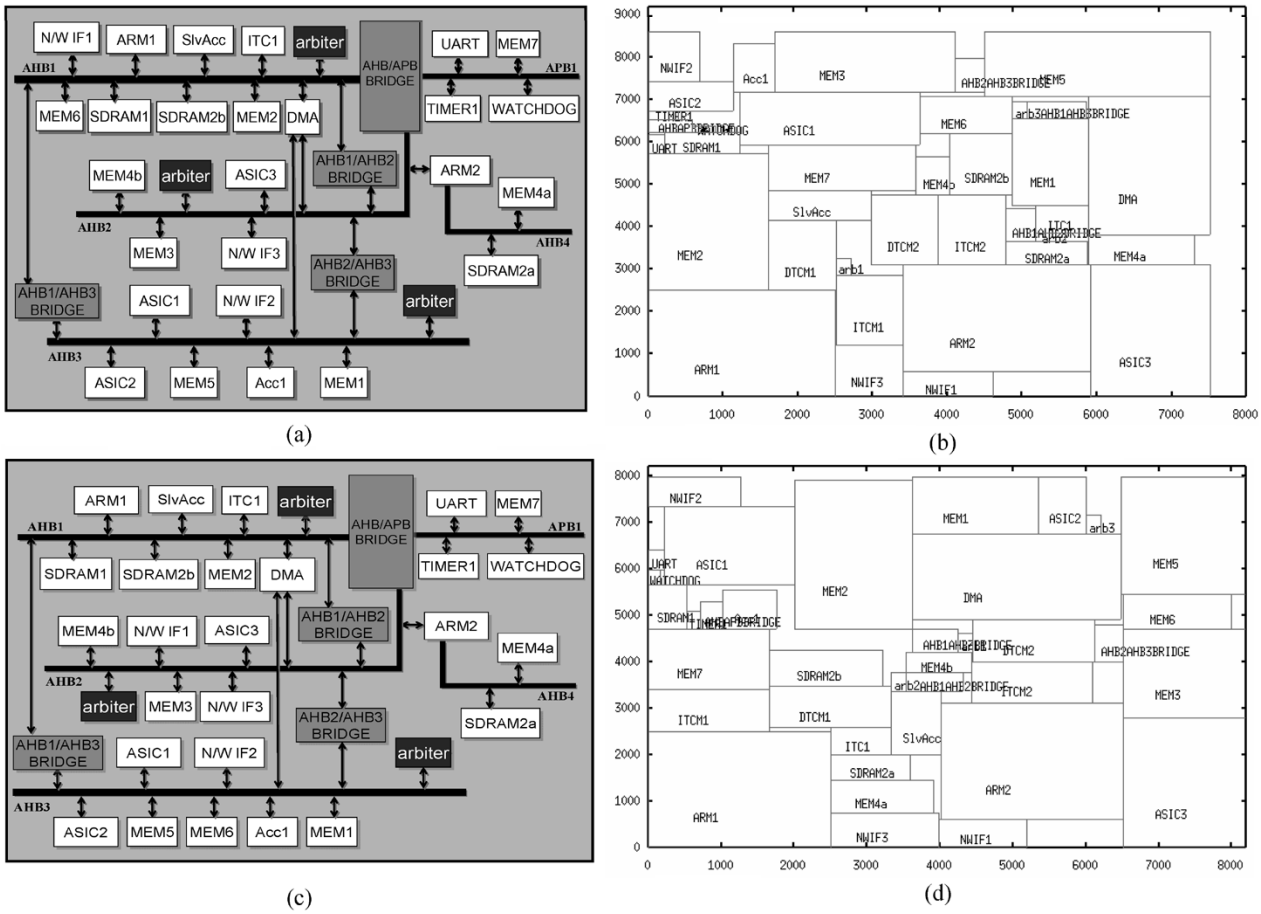


Fig. 17. Synthesis output for MPSoC networking subsystem: (a) intermediate MPSoC configuration, (b) floorplan for intermediate MPSoC configuration, c) final synthesized MPSoC subsystem, and (d) final floorplan for synthesized MPSoC subsystem.

TABLE VIII
COMMUNICATION PARAMETER VALUES

Parameter	*Values*				
	AHB1	*AHB2*	*AHB3*	*AHB4*	*APB1*
Bus width	32	32	32	32	32
Bus speed	150	300	250	50	50
DMA burst size	8				
Arbitration strategy	ARM2>ARM1> DMA> ASIC1>ASIC3>ASIC2				

shared bus (***initial***), an automated bus architecture synthesis flow which does not use a high level floorplanner (***ABS***), and a manually intensive, high level synthesis effort by a designer which also makes use of a floorplanning and wire delay estimation engine to detect timing violations (***manual***) just like our floorplan-aware automated bus architecture synthesis approach (***FABSYN***). The *manual* synthesis approach involves a designer manually selecting a combination of bus topology and communication parameter values, simulating the high level design models in SystemC and then iteratively modifying the bus architecture and parameter values based on the simulation results and designer intuition, until all constraints are found to be satisfied.

Table IX compares the results from our synthesis approach for the three case studies with the results from the other approaches. The *initial* approach is unable to satisfy any of the TCP constraints for all three of the case studies, because of excessive data traffic conflicts on its restricted number of buses. In contrast, the *ABS* approach does manage to satisfy TCP constraints for all the case studies, but in each case it synthesizes a bus architecture with bus clock cycle time violations that remain undetected, and, thus, the synthesized architecture is not feasible in each case. The *manual* approach satisfies all TCP constraints and is also able to detect and eliminate bus clock cycle time violations in the design, just like our *FABSYN* approach. However, there are a few key differences between the *manual* approach and our *FABSYN* approach. First, the *manual* approach generates bus architectures having a greater implementation cost (i.e., having a larger number of buses) when compared with architectures generated using our approach. This is because our automated flow is able to traverse a much larger communication parameter exploration space than the *manual* approach, and prevents us from making conservative decisions to create a new bus like in the *manual* approach, unless all suitable combinations of communication parameters are unable to meet the TCP constraint for the existing bus topology. Second, the performance of the architecture generated by the *manual* approach is actually found to be better than our *FABSYN* approach (except for the third

TABLE IX
SYNTHESIS RESULT COMPARISON

Case Study 1 Designs	*initial*	*ABS*	*manual*	*FABSYN*
Number of Busses	2	3	5	4
TCP constraints satisfied	0/2	2/2, **not feasible**	2/2	2/2
Execution cycles (millions)	49.76	24.51	18.80	20.32
Time to synthesize	~mins	~hours	~days	~hours

Case Study 2 Designs	*initial*	*ABS*	*manual*	*FABSYN*
Number of Busses	2	3	6	4
TCP constraints satisfied	0/3	3/3, **not feasible**	3/3	3/3
Execution cycles (millions)	88.48	47.63	26.58	29.10
Time to synthesize	~mins	~hours	~days	~hours

Case Study 3 Designs	*initial*	*ABS*	*manual*	*FABSYN*
Number of Busses	2	5	6	5
TCP constraints satisfied	0/5	5/5, **not feasible**	5/5	5/5
Execution cycles (millions)	170.11	98.20	94.50	92.69
Time to synthesize	~mins	~hours	~days	~hours

case study, where frequent bridge delay overheads reduce performance). This is because of the larger number of buses used by the *manual* approach, which reduces data traffic conflict and improves concurrency, at the cost of increasing the implementation cost. But it is important to note is that we are not really concerned about the absolute performance of the system. What is important to us is that we satisfy all TCP constraints and minimize the implementation cost of the synthesized architecture, and that we do so in a reasonable amount of time. The *manual* approach suffers from the major drawback that it takes several days for the designer to come up with a bus architecture which is typically overdesigned and exceeds the requirements (resulting in a more expensive system), whereas our *FABSYN* approach generates a better quality architecture in a matter of a few hours.

V. CONCLUSION

In this paper, we presented an approach for automating the synthesis of bus-based communication architectures for systems characterized by several possible 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 at the system level. 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. Future work will focus on extending the FABSYN approach to crossbar based communication architectures.

ACKNOWLEDGMENT

The authors would like to thank Prof. I. Markov and the PARQUET group at the University of Michigan for all their assistance.

REFERENCES

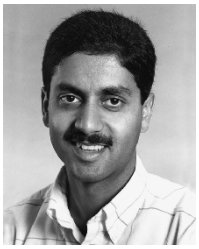
[1] D. Sylvester and K. Keutzer, "Getting to the bottom of deep sub-micron," in *Proc. of Int. Conf. Comput.-Aided Des.*, 1998, pp. 203–211.
[2] D. Flynn, "AMBA: Enabling reusable on-chip designs," *IEEE Micro*, vol. 17, no. 4, pp. 20–27, Jul./Aug. 1997.
[3] IBM CoreConnect [Online]. Available: http://www.chips.ibm.com/products/powerpc/cores
[4] Open Core Protocol International Partnership (OCP-IP), OCP Datasheet [Online]. Available: http://www.ocpip.org
[5] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Fast exploration of bus-based on-chip communication architectures," in *Proc. CODES-ISSS*, 2004, pp. 242–247.
[6] S. Narayan and D. Gajski, "Synthesis of system level bus interfaces," in *Proc. Eur. Des. Test Conf.*, 1994, pp. 395–399.
[7] J. Daveau, G. Marchioro, T. Ben-Ismail, and A. A. Jerraya, "Protocol selection and interface generation for HW-SW codesign," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 5, no. 1, pp. 136–144, Mar. 1997.
[8] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level," in *Proc. 9th Int. Symp. Syst. Synthesis*, Nov. 1996, pp. 65–70.
[9] K. K. Ryu and V. J. Mooney, III, "Automated bus generation for multiprocessor SoC design," in *Proc. Des. Automat. Test Eur. Conf.*, 2003, pp. 282–287.
[10] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, "Constraint-driven communication synthesis," in *Des. Automat. Conf.*, 2002, pp. 783–788.
[11] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," in *Des. Automat. Conf.*, 2001, pp. 518–523.
[12] K. Lahiri, A. Raghunathan, and S. Dey, "Efficient exploration of the SoC communication architecture design space," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2000, pp. 424–430.
[13] C. Shin, Y. Kim, E. Chung, K. Choi, J. Kong, and S. Eo, "Fast exploration of parameterized bus architecture for communication-centric SoC design," in *Proc. Des. Automat. Test Eur.*, 2004, pp. 352–357.
[14] N. Thepayasuwan and A. Doboli, "Layout conscious bus architecture synthesis for deep submicron systems on chip," in *Proc. Des. Automat. Test Eur.*, 2004, pp. 108–113.
[15] M. Drinic, D. Kirovski, S. Meguerdichian, and M. Potkonjak, "Latency-guided on-chip bus network design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2000, pp. 420–423.
[16] J. Hu, Y. Deng, and R. Marculescu, "System-level point-to-point communication synthesis using floorplanning information," in *Proc. ASP/DAC Des. Automat. Conf.*, 2002, pp. 573–579.
[17] R. P. Dick and N. K. Jha, "MOCSYN: Multiobjective core-based single-chip system synthesis," in *Proc. Des. Automat. Test Eur.*, 1999, pp. 263–270.
[18] R. A. Bergamaschi, Y. Shin, N. Dhanwada, S. Bhattacharya, W. Dougherty, I. Nair, J. Darringer, and J. Paliwal, "SEAS: A system for early analysis of SoCs," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign Syst. Synthesis*, 2003, pp. 150–155.
[19] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[20] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zelikovsky, "On wirelength estimations for row-based placement," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1265–1278, Sep. 1999.
[21] J. Cong and D. Z. Pan, "Interconnect performance estimation models for design planning," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 6, pp. 739–752, Jun. 2001.
[22] *"National Technology Roadmap for Semiconductors,"* Semiconductor Industry Association, 1997.
[23] SystemC initiative [Online]. Available: http://www.systemc.org
[24] S. Pasricha, "Transaction level modeling of SoC with systemC 2.0," in *Proc. Synopsys User Group Conf. (SNUG)*, 2002, pp. 55–59.
[25] AMBA AXI Specification [Online]. Available: http://www.arm.com/armtech/AXI
[26] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Extending the transaction level modeling approach for fast communication architecture exploration," in *Proc. Des. Automat. Conf.*, 2004, pp. 113–118.

Sudeep Pasricha (S'02) received the B.E. degree in electronics and communication engineering from Delhi Institute of Technology, Delhi, India, in 2000. He is currently working toward the Ph.D. degree in computer science at the University of California, Irvine.

From 2000 to 2002, he was a Design Engineer with STMicroelectronics, India. As part of the Embedded Systems Team at the Center for Research and Development, he worked on developing an Architecture Description Language (ADL) framework for processor/memory systems, and a Transaction Level Modeling (TLM) methodology for SoC exploration, verification and eSW development. From 2003 to 2004, he worked in collaboration with Conexant Systems, Newport Beach, CA, to develop a methodology for high-level exploration and synthesis of SoC communication architectures. His research interests include SoC communication architecture modeling, exploration and synthesis, system level modeling languages and methodologies, and CAD for embedded systems.

Mr. Pasricha received a best paper award nomination at DAC in 2005 and the best paper award at ASPDAC in 2006.



Nikil D. Dutt (S'81–M'89–SM'97) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, in 1989.

He was an ACM SIGDA Distinguished Lecturer during 2001 and 2002, and an IEEE Computer Society Distinguished Visitor between 2003–2005. He has served on the steering, organizing, and program committees of several premier CAD and Embedded System Design conferences and workshops, including ASPDAC, CASES, CODES+ISSS, DATE, ICCAD, ISLPED and LCTES. He is currently a Professor of Computer Science and Electrical and Electronic Computer Science at the University of California, Irvine, where he is affiliated with the following centers: CECS, CPCC, and CAL-IT2. His research interests include embedded systems design automation, computer architecture, optimizing compilers, system specification techniques, and distributed embedded systems.

Prof. Dutt received best paper awards at CHDL89, CHDL91, VLSIDesign2003, CODES+ISSS 2003, and ASPDAC-2006. He currently serves as Editor-in-Chief of the ACM *Transactions on Design Automation of Electronic Systems* (TODAES) and is an Associate Editor of the ACM *Transactions on Embedded Computer Systems* (TECS). He serves or has served on the advisory boards of ACM SIGBED and ACM SIGDA, and is Vice-Chair of IFIP WG 10.5.

Elaheh Bozorgzadeh (S'00–M'03) received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1998, the M.S. degree in computer engineering from Northwestern University, Evanston, IL, in 2000, and the Ph.D. degree in computer science from the University of California, Los Angeles, in 2003.

She is currently an Assistant Professor in the Department of Computer Science at the University of California, Irvine. Her research interests include CAD for FPGAs, reconfigurable computing, and design automation for embedded systems. She has authored 3 book chapters and more than 30 journal and conference papers.

Dr. Bozorgzadeh is a member of ACM/IEEE.

Mohamed Ben-Romdhane (M'90) received the B.S. and the M.S. degrees in electrical engineering from the Ecole National des Ingenieurs de Tunis (ENIT), Tunisia, in 1987 and 1989, respectively, and the Ph.D. degree in digital signal processing from the Georgia Institute of Technology, Atlanta, in 1995.

He served as Executive Director of SOC, IP, and Software for Conexant Systems Incorporated, Newport Beach, CA. He is currently the Vice President of Engineering for Newport Media Incorporated, Lake Forest, CA, where he is focusing on designing chips for mobile audio and video standards such as, DVB-H, ISDB-T, T-DMB, and DAB. His research interests include wireless systems, low-power design, embedded systems, DSP algorithms and implementation, and SOC design.