Latency-Guided On-Chip Bus Network Design

Milenko Drinić, *Member, IEEE,* Darko Kirovski, *Member, IEEE,* Seapahn Megerian, *Member, IEEE,* and Miodrag Potkonjak, *Member, IEEE*

***Abstract*— Deep submicron technology scaling has two major ramifications on the design process. First, reduced feature size significantly increases wire delay, thus resulting in critical paths being dominated by global interconnect rather than gate delays. Second, ultra-high level of integration mandates design of systems-on-chip that encompass numerous design blocks of decreased functional granularity and increased communication demands. The convergence of these two factors emphasizes the importance of the on-chip bus network as one of the crucial high-performance enablers for future systems-on-chip.**

We have developed an on-chip bus network design methodology and corresponding set of tools which, for the first time, close the synthesis loop between system and physical design. The approach has three components: a communication profiler, a bus network designer, and a fast approximate floorplanner. The communication profiler collects run-time information about the traffic between system cores. The bus network design component optimizes the bus network structure by coordinating information from the other two components. The floorplanner aims at creating a feasible floorplan; it also sends feedback about the most constrained parts of the network. We demonstrate the effectiveness of our bus network design approach on a number of multi-core designs.

***Index Terms*— Bus network design, latency, system synthesis, on-chip communication.**

I. Introduction

As applications have become more complex with increased levels of hardware and software sharing (communications, multimedia, video games, networking), designers have striven for more gates on chip as well as simplified and time-efficient design methodologies. Deep submicron (DSM) as a technology and reuse as a design methodology have recently emerged as means of overcoming the growing difficulty of rapidly designing and verifying highly integrated systems-on-chip. Due to design complexity and time-to-market pressure, it is expected that future systems-on-chip are designed as networks of virtual components. Virtual component (VC) is a core wrapped with logic that enables it to I/O data to the attached bus with an arbitrary bus protocol. Because of high integration levels (100s of millions of transistors), the network of cores is estimated to count hundreds of VCs, where each core is smaller than 50-100K gates. Wire latency in such cores is estimated to less than 25% of the maximum on-chip delay. This facilitates the timing closure within a core. The limitation on the gate count enables the usage of traditional design methodologies for VC design [1]. Such a design paradigm does not pose significant restrictions to application design, as most of the modern multimedia, graphics, and communications applications use building blocks (e.g., controllers, DSP processors, Viterbi decoding, DCT, Huffman codec, Reed-Solomon error correction, RSA and AES encryption).

Since decreased levels of module granularity in computation result in higher communication costs, it is expected that the performance of future core-based systems is greatly affected by inter-core communication. Communication among cores in DSM systems poses several design issues that can be classified as: *(i)* synchronization and *(ii)* performance optimization problems. While latency insensitive synchronization between cores can be resolved using relay stations and appropriate communication protocols [2]–[4], to the best of our knowledge, problems such as bus network design and core to bus assignment have not been addressed to date.

In this paper, we present a novel system-level design framework that, based on the communication profile of the modules involved, creates a single-chip bus network and assigns cores to buses such that the overall processing throughput of the system is maximized. The framework consists of three components: *(i)* a communication profiler, *(ii)* a bus network designer, and *(iii)* an approximate floorplanner. For a given set of applications and a fixed number of pre-synthesized cores, the designer initially simulates the communication behavior of the system modules and creates a profile of the connectivity and communication patterns among cores. The bus network designer uses the communication profile to arrange on-chip bus structures and core connectivity. Its goal is to create a communication network which results in maximized expected throughput. Note that the communication profiler provides only estimates for actual communication delays on any specific bus network. Therefore, the objective function is defined heuristically.

The created bus network is then fed to the approximate floorplanner which attempts to create a feasible layout. The feasibility of the layout is measured by comparing bus wirelengths to an upper bound constraint. In case of an unsuccessful search, the approximate floorplanner returns to the bus network designer a list of K best solutions with all unsatisfied bus constraints. In the next iteration, the bus network designer considers these solutions, their corresponding latencies, and tries to rearrange the bus network such that at least one of these solutions can be satisfied. The goal of the synthesis process is to explore the solution space by toggling between infeasible

Manuscript received March 20, 2004; revised November 11, 2004. This work was supported by the IEEE.

M. Drinić is with Center for Software Excellence, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052. E-mail: mdrinic@microsoft.com

D. Kirovski is with Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA. E-mail: darkok@microsoft.com.

S. Megerian is with Department of Electrical & Computer Engineering, Madison, WI 53706, USA. E-mail:megerian@ece.wisc.edu.

Miodrag Potkonjak is with the Computer Science Department, University of California, Los Angeles, CA 90095, USA. E-mail: miodrag@cs.ucla.edu.

and feasible solutions and trying to find bus networks which result in higher communication throughput.

The developed synthesis framework has been deployed in optimizing performance in a number of core based designs extrapolated from several applications running on the state-of-the-art (30+ modules) systems-on-chip. Since the optimization process estimates latencies and heuristically models communication patterns, we have confirmed the throughput improvement by simulating the communication of modules using optimized and non-optimized (ad-hoc) bus networks.

A. Motivational Example

We present several design trade-offs involved in bus network design using a design example presented in Fig. 1. Consider eight cores $C1, \ldots, C8$ which communicate with each other in four control cycles as presented in Fig. 1(a). Buses are connected using bridges. For simplicity and brevity, assume that an instance of communication referred to as a control cycle is a cycle on a bus necessary to complete transfer of a data word between two cores, two bridges or a core and a bridge connected to the same bus. Sending a message over two bus bridges (three buses) takes three bus control cycles. We have assumed a simple round robin arbitration scheme on each bus. There exist more complex and more efficient bus arbitration schemes. However, in cases when the number of components attached to a single bus segment is limited as in our case, round robin arbitration scheme yields comparable results to more complex ones. The optimization goal is to assign cores to buses, such that no bus has more than three cores, and that all messages are delivered in as few control cycles as possible.

A possible greedy approach would identify modules that communicate most frequently among each other and assign them to a single bus. An example of such a greedy strategy would identify cores $C1$, $C5$, and $C6$, and assign them to a single bus. The resulting core-to-bus assignment, presented in Fig. 1(b-A), would take *seven bus control cycles* to deliver all messages.

Let us consider the communication overlap among modules as part of our heuristic assignment strategy. First, we define a *Communication-Connectivity Graph* CCG as an undirected weighted graph with a set of nodes C representing cores and a set of edges E representing existence of communication between two cores. Next, we define *weight* ω_S *of a control cycle* S as the cardinality of the set of communications that occur at control cycle S. *Weight* $w(E)$ *of an edge* $E(C_i, C_j)$ in a CCG is defined as a sum of weights:

$$w(E) = \sum_{S \in CS} \omega_S \tag{1}$$

of all control cycles CS at which cores C_i and C_j communicate. A CCG that corresponds to the communication pattern in the design example is presented in Fig. 1(c).

The goal of assigning a set of modules M that *(i)* communicate frequently with each other and *(ii)* communicate to other cores $\{C - M\}$ at control cycles at which no other core communicates among M, can be modeled in the following way. We define a *bus* B as a partition of cores from $C, B \in C$. Next, we define an *objective function* $OF(B)$ *of a bus* B as the sum of edge weights among cores $C_i, C_j \in B$ minus the sum of edges adjacent to nodes in B and $C - B$:

$$\begin{aligned} OF(B) = & \sum_{\substack{C_i, C_j \in B \\ i \neq j}} w(E(C_i, C_j)) \\ & - \sum_{\substack{C_i \in B \\ C_j \in C - B}} w(E(C_i, C_j)) \end{aligned} \tag{2}$$

We heuristically denote a particular core-to-bus assignment as α*-optimal* with respect to the above mentioned optimization goal, if it represents a K-partitioning of C into K buses with maximal:

$$OF = \sum_{i=1}^{K} OF(B_i) \tag{3}$$

The objective function for core-to-bus assignment presented in Fig. 1(b-A) yields $OF = -15$. The corresponding partitions of the associated CCG are presented in Fig. 1(c-A).

Exhaustive search has identified the solution depicted in Fig. 1(b-C) as α-optimal, resulting in $OF = 3$, and only *four control cycles* required to deliver all messages among modules. Hence, even on such a small example we have demonstrated that optimal core-to-bus assignment, which involves a number of often counterintuitive trade-off considerations, may significantly improve on-chip communication performance. In this manuscript, we discuss a method for building and modeling of the communication profile of a core-based system, we introduce a viable set of heuristic objectives (α-optimality) which aim at considering the trade-offs involved in bus network design, and finally, we present interactive algorithms which enable effective search for the α-optimal core connectivity.

II. Related Work

A. The Effect of Deep Submicron on Design Strategies

While semiconductor researchers are announcing 0.04-micron nominal channel length technologies [1], [5], [6] as well as gate oxide implants as thick as a few atoms, both EDA community is taking steps to address the emerging synthesis problems associated with such technologies. For example, Intel has already announced that it has built a new SRAM chip with 500 million transistors using 65-nm technology. The chip is scheduled for production starting in 2005. Aggressive deep submicron (DSM) manufacturing technologies are expected to result in: *(i)* significant increase of wire latency due to increased RC (delay of a 0.1 micron wide interconnect is an order of magnitude greater than that of a 0.5 micron wire) [7], *(ii)* increased noise resulting in higher likelihood of signal crosstalk and delay uncertainty [8], *(iii)* current leakage through the gate oxide [9], and *(iv)* increased power dissipation [10]. Although most of the posed problems can be addressed at lower levels of design abstraction, the problem of increased wire latencies has significant impact

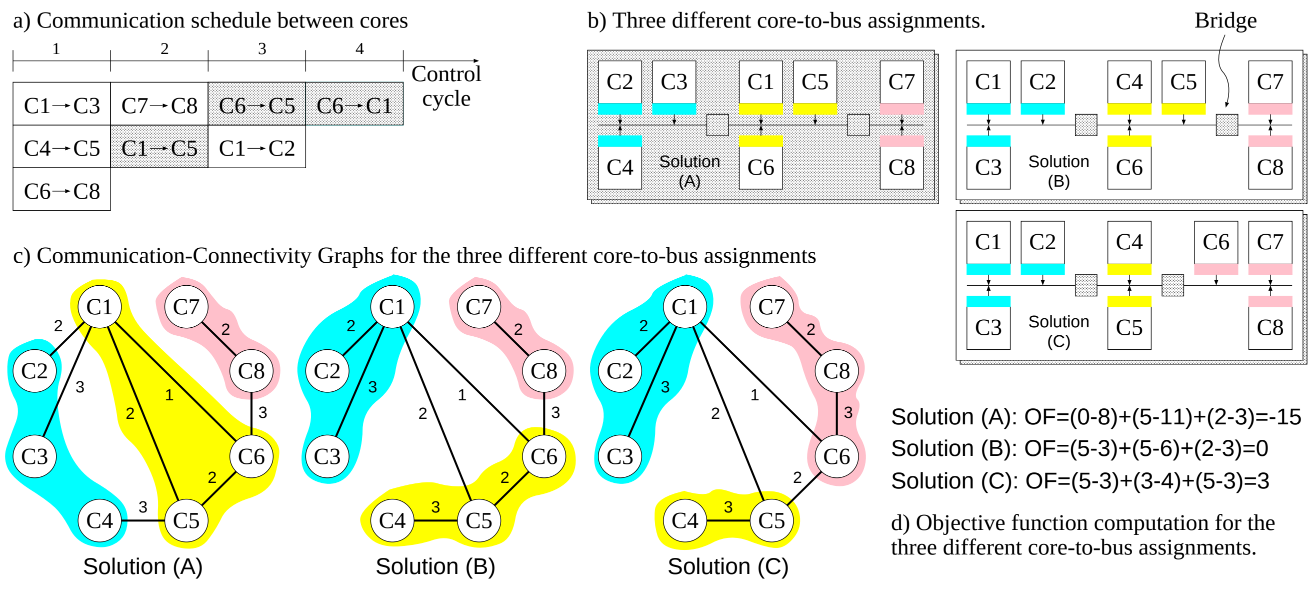

a) Communication schedule between cores
1
2
3
4
Control cycle
C1→C3
C7→C8
C6→C5
C6→C1
C4→C5
C1→C5
C1→C2
C6→C8
b) Three different core-to-bus assignments.
Bridge
Solution (A)
Solution (B)
Solution (C)
c) Communication-Connectivity Graphs for the three different core-to-bus assignments
Solution (A)
Solution (B)
Solution (C)
Solution (A): OF=(0-8)+(5-11)+(2-3)=-15
Solution (B): OF=(5-3)+(5-6)+(2-3)=0
Solution (C): OF=(5-3)+(3-4)+(5-3)=3
d) Objective function computation for the three different core-to-bus assignments.

Fig. 1. An example of system throughput performance obtained for three different core to bus assignments.

on the higher level design stages. At lower design levels, researchers have explored ways of improving interconnect performance by topology optimization [11], buffer insertion and sizing [12], [13], bus driver improvements [14], and optimal wire sizing [15]. As interconnect delay is estimated to dominate (up to 80% of) the circuit's critical path, high-level and logic synthesis optimization methods, which take into account this phenomenon, have to be deployed [1], [16]. Addressing the impact of slow interconnects at higher design stages requires interactive collaboration between the high-level design tools and placement and routing tools [1]. For example, such collaboration can result in floor and wire planning based on RTL descriptions [17]. A different path in high-level design for DSM is exploration of communication protocols and core I/O wrappers for functional insensitiveness to interconnect delays [2]–[4], [18]. An important problem of high-level design is the communication architecture synthesis. A number of approaches to solving this problem have emphasized the minimization of the application response time [19], core to bus assignments and network protocols for an improved communication throughput [20], [21], integration of communication protocol selection with hardware/software partitioning and co-simulation [22], [23], and guaranteeing quality of service for networks on silicon [24].

B. Floorplanning for DSM

Floorplanning in general, refers to finding the best physical placement of modules in a design, subject to area, wiring and other metrics. The traditional heuristic approaches to floorplanning are based on the min-cut method [25], [26], force-directed guidance [27], [28], rectangular dualization [29], and simulated annealing [30], [31]. A good survey of floorplanning techniques and involved trade-offs is presented in [32].

Recent advancements in floorplanning for DSM include a variation of the force-directed method that improves the previous results by considering several different forces to reduce cell overlaps and improve area. It has been demonstrated that the Wong-Liu floorplanning algorithm can be combined with module placement and interconnect route planning using more accurate interconnect cost models. Finally, the integration of floorplanning and high level synthesis can be used to improve storage requirements and data transfer performance of the system.

C. On-Chip Bus Standards

On-chip bus design has attracted little attention among academic researchers. However, there is a number of industrial initiatives, mainly within the VSI Alliance to initiate a set of guidelines for on-chip bus and bus wrapper design. The target of possible standardization is ease of attaching cores with arbitrary bus protocols to system buses. IBM has proposed an open on-chip bus architecture, CoreConnect, compliant with the VSI proposal. Similarly, the Parallel Intermodule (PI) bus has been proposed to address the demands of real-time and fault tolerant applications [33].

III. Global Design Flow

The complexity of modern application-specific systems has resulted in design flows which consist of a number of stages. The two most widely accepted design flows are the golden model and the waterfall model. The golden model is a copy of the design specification at some level of abstraction (usually RTL) at which most of the changes are performed [34]. The underlying concept behind the waterfall design process is a progression through various levels of abstraction with the intent of fully characterizing each level before moving to the next level. Designing for DSM involves a number of alterations to the traditional design flows [1]. Most of the changes are related to performing higher level design stages (such as logic synthesis) with approximation of effects caused by DSM [17]. Since obtaining those effects requires computing at least an approximate layout, commonly the design flow becomes iterative and interactive.

In this paper, we present a novel system-level design framework which, based on the communication profile of involved modules, creates a single-chip bus network and assigns cores to buses such that the overall processing throughput of the system is maximized. The framework consists of three components: *(i) a communication profiler*, *(ii) a bus network designer*, and *(iii) an approximate floorplanner*. The global design flow of this framework is presented in Fig. 2. For a given set of applications and a fixed number of pre-synthesized cores, the designer initially simulates the communication behavior of the system modules and creates a profile (CCG) of the connectivity and communication patterns among cores. The communication profiler takes into account temporal correlation of communication patterns among cores. Based on this communication profile, the design flow enters a synthesis loop which toggles between bus network design and approximate floorplanning.

The bus network designer rearranges the bus network by:

- removing or adding bridges between buses, or
- reassigning cores from one bus to another.

Its goal is to create core connectivity which results in maximized expected communication throughput. This objective is estimated heuristically by considering communication delays and overlap. Communication delay is proportional to the number of bridges (or replicators [2]–[4]) along the message path.

The created bus network (i.e., core connectivity) is then fed to the approximate floorplanning tool which attempts to create a feasible layout. The feasibility of the layout is measured by comparing bus wirelengths to an upper bound constraint. The approximation floorplanning tool is based on a modified simulated annealing algorithm which throughout its search memorizes a pool $\Pi(K)$ of K best solutions. In case of unsuccessful search, the tool returns to the bus network designer, the list $\Pi(K)$ of solutions with all unsatisfied wirelength constraints. In the next iteration, the bus network designer considers the pool $\Pi(K)$ of best solutions and tries to rearrange the bus network such that at least one of these solutions can be satisfied.

The designer starts the loop with an initial bus network solution which has a feasible layout and results in relatively low-quality performance. The goal of the synthesis process is to explore the solution space by toggling between infeasible and feasible solutions and try to find bus networks which result in higher communication throughput.

As the complexities of behavioral specifications increase, both design flows are becoming more vulnerable to the engineering change (EC) process due to the demand for updating design solutions. To address this issue, we have developed a generic EC methodology, applicable to all design stages, which facilitates constraint manipulation to augment the design with flexibility for future changes [35]. The EC is conducted by searching for a correction that induces minimal perturbation of the optimized solution.

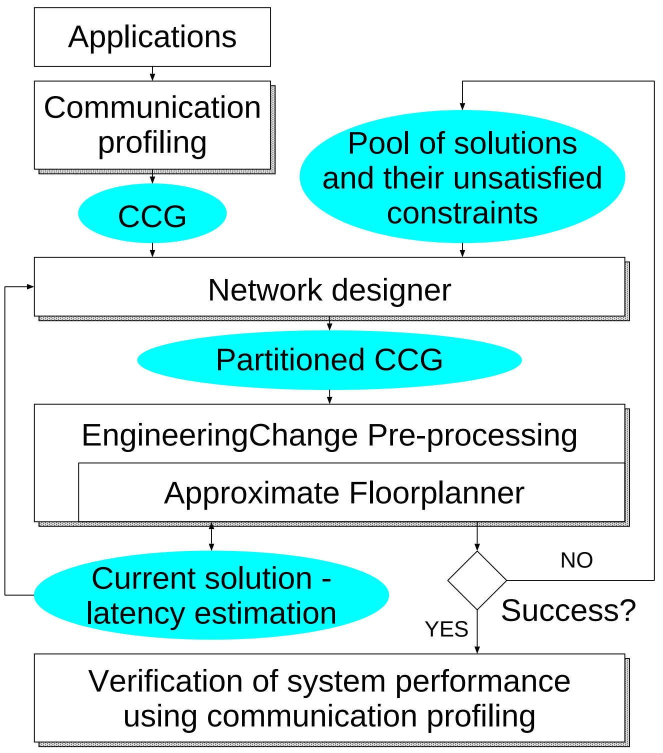

Applications
Communication profiling
Pool of solutions and their unsatisfied constraints
CCG
Network designer
Partitioned CCG
EngineeringChange Pre-processing
Approximate Floorplanner
NO
Current solution - latency estimation
Success?
YES
Verification of system performance using communication profiling

Fig. 2. Global design flow of the interactive bus network design process.

IV. Preliminaries

In order to position our work, in this subsection, we present the related hardware and communication model, and outline our communication profiling methodology. The generic hardware model that we have adopted, assumes the following set of constraints: cores have the property of being hard, i.e., with constant layout, and hence, with specific location of their bus interface. Cores are connected with buses of limited maximal length. This limitation is posed by the maximum length of the bus transaction control cycle. The bus network is switched using a topology of bus bridges, where a bus bridge is a crossover of maximally four buses with associated synchronization and buffering logic [2]–[4]. The area of a bus bridge is approximated at 5% of an average core size per bridge port. Such approximation stems from the complexity of modern bus interfaces [36]. Bus arbitration is performed statically with priorities being assigned using the round robin arbitration scheme [37]. We allow all interconnect optimizations targeted for DSM (see Section II). Bus latency is modeled using a second order polynomial of the wirelength adopted from. Message routing across the bus network is assumed to be deadlock free.

A. Communication Profiling

One of the crucial components of our synthesis framework is the communication profiler. A completely accurate communication information could be obtained if one conducted extensive, time consuming simulations for a given application over the set of all possible configurations of a bus network. Even if such a simulation was conducted, the very next set of input data could change communication patterns. Therefore, we decided to employ statistical techniques. Our goal is to estimate the chances that a given bus architecture is capable of executing a given application under the user specified timing constraints. This estimate is formed by extracting a CCG. The CCG is extracted assuming dedicated connectivity and distribution of each piece of communication data over a number of bus control cycles. Similar ideas have been widely used in synthesis literature [38]. In addition to our experimental results (Section VI), we justify our approach using at least three conceptual and intuitive reasons:

(*i*) majority of applications have high ratio of computation versus communication in terms of the number of operations normalized with the number of transfers between blocks;
(*ii*) this type of systems typically has bursty communication so that small timing fluctuations do not change the communication overlap patterns; and
(*iii*) strict timing constraints and deployed synchronization mechanisms further facilitate high predictability of relative timing of data transfers.

The communication profiler is executed as a pre-processing step with a goal to summarize the essential statistics of communication patterns of the implemented application. The input to the communication profiler is an instance of the communication model of our target system. The model consists of two types of cores: masters and slaves. The communication of a slave is modeled with the following operators: {init:ID}, {done:ID}, and a pattern P of {receive:ID:length}, {nop}, and {send:ID:length} signals, where ID represents a unique identifier of a core and length denotes the burstiness of the signal in control cycles. The communication of a master is modeled as a semi-infinite stream of statistically modeled multiplexed signals: {init:ID}, {done:ID}, and patterns P_i, $(i = 1, k)$, where each pattern P_i is defined as in the case of a slave core. Both masters and slaves stall at {init:ID}, {done:ID}, and {receive:ID:length}. In our experiments, we have used traffic patterns extracted and extrapolated from the MediaBench benchmark suite [39]. An example of a simple communication between a master and a slave is shown in Fig. 3.

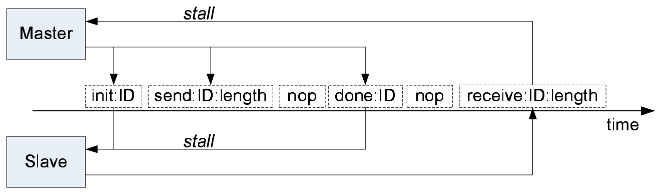


Fig. 3. An example of a simple communication between a master and a slave. The master initiates the transfer, sends a packet, closes the communication channel and waits for an acknowledgment.

The communication profiler collects run-time information according to the definitions presented in Subsection I-A. Weight $w(E)$ of a CCG edge $E(C_i, C_j)$ is computed as a sum of vector sums, where each vector sum is computed per $C_i - C_j$ transaction.

$$w(E) = \sum_{S \in CS} \sum_{i=-D}^{D} \omega_{S_i} \cdot pdf(i) \tag{4}$$

The vector sum is computed as a $pdf()$-weighted sum of constraints in the preceding and succeeding D control cycles to the control cycle S at which the transaction occurs. The CCG, which is built during a single run of the communication profiler, is post-processed by removing edges with small weights, typically smaller than the median of all weights in the CCG minus one standard deviation. By design, we maintain the network connected at all design stages (Subsection V-A). Therefore, we can safely remove low weight edges, improve tool's runtime and maintain valid connectivity of the bus network.

The intuition behind the use of a weighting function, $pdf()$, is similar to the intuition used in force directed scheduling [38]. In this scheduling algorithm, the first step is to determine ASAP and ALAP schedules. Force directed scheduling uses the pdf with uniform distribution of each operation between its ASAP and ALAP times as an estimate as to when a particular operation will eventually be scheduled. Note, that the activity at actual control steps heavily depends on the allocated hardware. However, the estimation is often reasonably accurate. If we replace the scheduling of an operation with the scheduling of transfers by using a bus network, it is easy to see that the use of the pdf facilitates robustness of estimation. The pdf distributes the weight over a certain time period such that the time fluctuations of the communication overlap are captured. While a number o different pdfs can be used, we have chosen a normalized Gaussian probability distributions in our methodology.

V. Latency-Guided Design of On-Chip Bus Networks

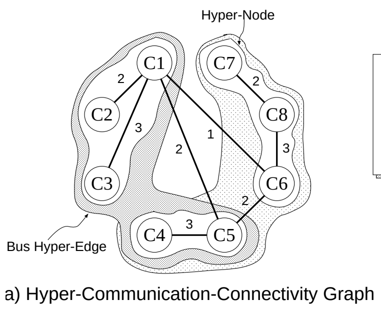
A. Network design techniques

In this subsection, we present the bus network designer. We start by introducing and formally defining the bus network design problem, and discussing its complexity. The main part of the section presents the algorithm for network design, followed by the description of the output it provides to the synthesis-driven floorplanner.

A typical system-on-chip is composed of a number of independent subsystems (cores) that exchange data. The goal of the bus network design algorithm is to create a bus network and a core to bus assignment such that the overall throughput of the system is maximized. In the first run of the algorithm, its input consists of a set of cores, a communication profile of the system applications represented as a CCG, and an initial ad-hoc solution (a starting bus network and bus-to-core assignment with a floorplan). In the subsequent runs, the ad-hoc solution as an input is replaced with the information from the floorplanner which quantifies the constraints that cannot be satisfied with respect to the solution obtained in the previous run of the bus network designer.

We start the formal description of the problem with a series of related definitions. We first define an extension to the CCG, the *Hyperedge Communication-Connectivity Graph* $hCCG$ as an undirectional graph where a *Hypernode* hN is a collection of nodes in CCG and a *Bus Hyperedge* BhE is a hyperedge that encompasses at most Q hypernodes, i.e., the maximum number of buses attached to a bridge. We define a relation $hN \propto BhE$ when hN is covered by BhE. Hypernode *hN* is *semi-free*, if it belongs to only one *BhE*. Hypernode hN is *seized* if $hN \propto BhE_i$ and $hN \propto BhE_j$, where $BhE_i \neq BhE_j$. According to its definition, an *hCCG* formally describes a bus architecture. While *hN* represents a bus segment, *BhE* represents a bridge and its relation to adjacent busses. A *Chain of BhEs* is a set of *BhE*s such that there exists no *BhE* that does not overlap with another *BhE* in the chain. A *hCCG* is *connected*, if there exists a chain of *BhE*s which encloses all nodes of *CCG*. Hypernode *hN* is *valid*, if it satisfies the relation $|hN| \leq M$. An *hCCG* is *valid*, if and only if *(i)* it is composed of valid *hN*s, *(ii)* if any two *BhE* overlap in maximum one *hN*, *(iii)* if no *hN* belongs to more than two *BhE*s, and *(iv)* if *hCCG* is connected. An example of a valid *hCCG* and its corresponding bus structure is shown in Fig. 4.

It is important to stress that technology-specific limitations for a desired control cycle (parasitic capacitance of the bus segment) are reflected through three design constraints: *(i)* λ_{max} - maximum length of a bus segment, *(ii)* M - maximum number of cores that can be attached to a bus segment (hypernode cardinality $|hN| \leq M$), and *(iii)* Q - maximum number of bus segments attached to a single brigde/router.



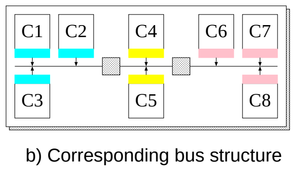


Fig. 4. An example of a *hCCG* and its corresponding bus structure.

We formally define the trade-offs involved in generation of bus networks and core to bus assignment using an objective function OF (its simplified version is presented in Subsection I-A, Equation 3):

$$OF = \sum_{i=1}^{|CCG|} \left[\sum_{C_j, C_k \in hN_i} w(E(C_j, C_k)) - \sum_{\substack{\forall C_j \in hN_i \\ C_k \in hCCG - hN_i}} w(E(C_j, C_k)) \cdot |\pi(C_j, C_k)| \cdot \Phi_{j,k} \right] \quad (5)$$

where $\pi(C_j, C_k)$ represents the routing path between cores C_j and C_k, and $|\pi(C_j, C_k)|$ represents the path length (latency is modeled proportional to the number of bridges between C_j and C_k). More formally, $|\pi(C_j, C_k)|$ is equal to the number of *BhE*s in the routing chain that connects the *hN*(s) which contain C_j and C_k. $\Phi_{j,k}$ is defined as the sum of bus segment workloads on the path from C_j to C_k ($C_j \rightarrow C_k$):

$$\Phi_{j,k} = \sum_{\forall hN \in C_j \rightarrow C_k} \phi(hN). \quad (6)$$

A workload $\phi(hN)$ of a bus segment hN is defined as the percentage of time when the bus segment is in use by cores associated to it. $\phi(hN)$ is determined from the statistical behavior of each core for a given application workload. As with other parameters of the communication profile, $\phi(hN)$ is an estimate based on the corresponding CCG.

The objective function has been derived with the following set of incentives: *(i)* cores which communicate frequently should be placed on the same bus, *(ii)* cores attached to different buses should communicate through as few as possible bridges, and *(iii)* congestion of individual buses should be minimized. Important features of the objective function are its *global* system's performance[1] characterization and *empirical correlation* to system's throughput (see Section VI). The search for an α-optimal design (maximized OF) can be abstracted as follows.

Problem: *Balanced Partitioning of an hCCG.*
Input: *An hCCG hypergraph and a real number* ***a****.*
Question: *Is there a balanced partitioning of hCCG into a set of hNs which results into a valid hCCG such that its OF is greater than* ***a****?*

The problem of Balanced Partitioning of an hCCG is computationally intractable as it can be straightforwardly restricted by imposing $M = 2$ and simplifying OF as presented in Subsection I-A, to the NP-complete balanced partitioning problem [40]. To address this difficult problem, we have used simulated annealing as a search algorithm. The algorithm is illustrated using the pseudo-code in Fig. 5.

The developed components of the algorithm that have been augmented into a traditional simulated annealing search engine are: a selection of atomic solution alterations, *moves*, and an engineering change pre-processing. We first describe the set of *move* functions. Moves can be classified into two categories: moving *CCG* nodes C across hypernodes hN and modifying $hCCG$ hyperedges *BhE*s. Moves do not affect the underlying *CCG* structure.

In the first category, we distinguish two different *move* actions: *(i)* $SwapNodes(C_i, hN_i, C_j, hN_j)$ node swapping between hypernodes and *(ii)* $MoveNode(C_i, hN_i, hN_j)$ node transfer from one hypernode hN_i to another hN_j. If $|hN_i|$ is equal to one before the $MoveNode(C_i, hN_i, hN_j)$ operation, hypernode hN_i can be removed from the list of hypernodes. Both $SwapNodes()$ and $MoveNode()$ are not performed if the resulting $hCCG$ is not valid. Examples of the $SwapNodes()$ and $MoveNode()$ moves are illustrated using Figures 6 and 7.

The second category of moves *MoveBhE()* modifies hyperedges and thus, the bus network structure. There are

[1]We interpret the global system performance in this context as the fulfillment of strict timing requirements and the overall communication throughput of the resulting network.

```
$T = T_0$; $currentPartition = initialPartition$
$EngineeringChangeProcedure(FloorplanConstraints)$
**While** $(T > T_F)$
  $currentPartition = bestPartition$
  **While** (cumulative improvement $\geq \sigma$)
    $a = random()$
    **Case** $(a > const1)$ :
        $currentPartition.SwapNodes(random())$
    **Case** $(a < const1)\&(a > const2)$ :
        $currentPartition.MoveNode(random())$
    **Case** $(a < const2)$ :
        $currentPartition.MoveBhE(random())$
    $\delta = currentSum - OF()$
    **If** $(\delta < 0)$ **then** *Accept modified currentPartition*
    **else** *Accept modified currentPartition with*
      *probability* $p = e^{-\frac{\delta}{T}}$
  **End while**
  *Decrease temperature*
**End while**
```

Fig. 5. Pseudo-code of the simulated annealing algorithm for balanced partitioning **CCG**.

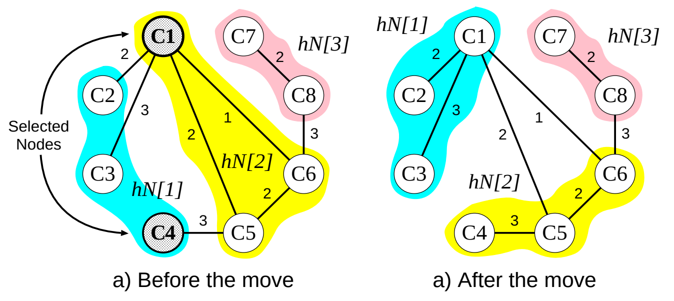


Fig. 6. An example of an $SwapNodes(C_1, hN_2, C_4, hN_1)$ move.

three possible alternations of this move. The first one is a removal of a selected BhE. In this case, other BhEs at an ϵ-distance equal to one, pseudo-randomly acquire all semi-free hypernodes from the removed BhE (see Fig. 8(2)). We define ϵ-distance of a hypernode hN, $hN \propto BhE$, as a set of hypernodes covered by BhEs which are included in all chains of length ϵ starting from BhE. The second variant of $MoveBhE()$ is regrouping as illustrated in Fig. 8(3). Here, the set HN of all seized $hN \propto BhE$ is released from BhE and a semi-free hN_x from another BhE_x is seized by HN to create a new hyperedge $BhE_y = hN_x \cup HN$. The third variant (see Fig. 8(4)) of *MoveBhE()* creates two new hyperedges BhE_x and BhE_y from a parent hyperedge BhE

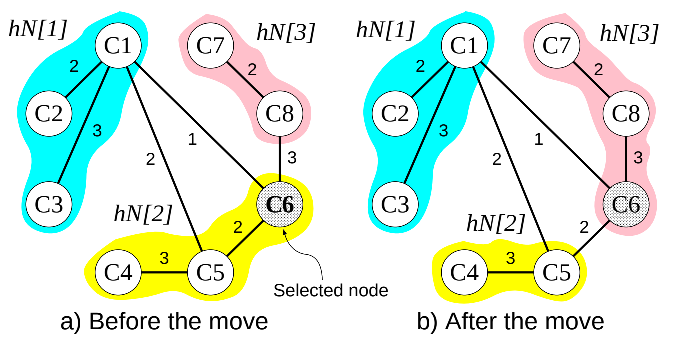


Fig. 7. An example of a $MoveNode(C_6, hN_2, hN_3)$ move.

by pseudo-random bipartitioning of BhE. If there exists only one seized $hN \propto BhE$, one of the partitions with only semi-free hypernodes is regrouped with a pseudo-randomly selected hyperedge $BhE_z \neq BhE$. *MoveBhE()* is only performed on higher search temperatures since it significantly changes the structure of the *hCCG*. *MoveBhE()* by construction preserves the validness of *hCCG*. Examples of all three variants of *MoveBhE()* and their corresponding bus structures are depicted in Fig. 8.

Although some moves (e.g., *MoveBhE()*) impact a bus network more than others (e.g., *SwapNodes()* and *MoveNode()*), in order to fully preserve the integrity of the simulated annealing optimization mechanisms, at all temperatures of the annealing process, all deployed moves were applied in the same proportion. This decision was further accentuated by the observation that computationally the most expensive component of simulated annealing is the random number generation which can be kept low, if all moves are equally likely at all temperatures. Finally, note that at different temperatures different subsets of moves were used.

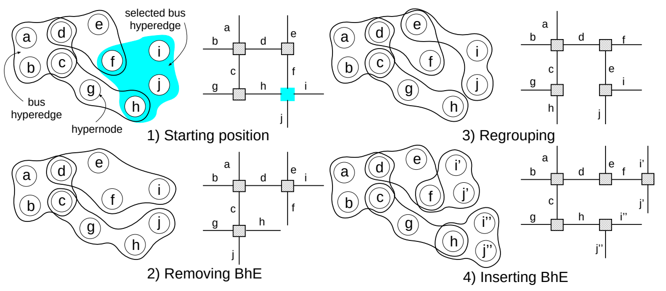


Fig. 8. An example of an *MoveBhE* move.

The interactivity of the bus network designer and the approximate floorplanner is enabled through an engineering change (EC) procedure *EngineeringChangeProcedure()*. The goal of this procedure is to mark the feasible portion of the bus network as unchangeable and to reduce the solution space and thus, search time for the new iteration of simulated annealing. Due to brevity, we do not present the developed EC technique as the key concepts have been adopted from a generic EC methodology [35].

The developed EC process restricts certain subdomains of the solution space by adding constraints based on floorplanner's report. This report contains quantitative information about the satisfiability of constraints posed by the output of the balanced partitioning of *hCCG*. The EC technique defines the likelihood $p(object)$ to participate in a *move* action for each node $C \in hCCG$ and hyperedge $BhE \in hCCG$. A core and/or a bus $object$, that often violated the λ_{max} technology dependent parameter, is set with high $p(object)$. The bus network designer outputs the final bus connectivity and core to bus assignment (the set of hypernodes $hN \in hCCG$, hyperedges $BhE \in hCCG$, and nodes $C \in hCCG$ of the partitioned $hCCG$) to the approximate floorplanner for technology parameter evaluation.

B. Synthesis-Driven Floorplanning

In this subsection, we present the approximate floorplanner. First, we characterize the contribution of our floorplanning algorithm. Then, we formally define the problem and the input and output data structures. Finally, we discuss the technical details of our algorithm and its implementation.

The quality of the resulting bus network is highly dependable on the information the floorplanner tools is providing via its feedback to the network designer. Therefore, we designed the floorplanner tool such that it better addresses the requirements of a bus network design. It considers higher granularity blocks rather than gates. Its primary goal is to optimize the length of the bus network and the length of each individual bus segment as opposed to optimizing individual wire length. One of the main features of our floorplanner is that it collects statistics during its runtime. Each move and each snapshot of the current floorplan are a potential indication of the actual distance from the solution that satisfies the given constraints. This information is essential for the network design tool such that the global design flow converges toward a solution of good quality.

At the heart of the floorplanning task lies the rectangle packing problem: given a set of rectangles C of arbitrary dimensions, place them with no overlap in the smallest possible bounding rectangle. The arrangement of rectangles C represents their floorplan F. One of the main problems associated with rectangle packing is that in two dimensions, rectangle placement solutions are continuous and infinite. To address this issue, we use the sequence pair based representation introduced in [31] which provides a compact representation that is proven to be P-admissible. We formally define our latency-guided approximate floorplanning problems as:

Problem: *Approximate latency-guided floorplanning.*
Input: *A set of cores* $C = \{C_i | i = 1, \ldots, |C|\}$, *a set of bridges* $BhE = \{BhE_i | i = 1, \ldots, |BhE|\}$, *an hCCG, a bus length constraint* λ_{max}, *and a real number Max_Area.*
Question: *Is there a floorplan F of C such that* $Area(F) \leq Max\_Area$ *and* $\forall hN_i \in hCCG$, $Length(hN_i) \leq \lambda_{max}$.

Due to the computational intractability of the rectangle packing problem [11], we have developed a variant of a traditional simulated annealing based approach to search the solution space described above. The algorithm is illustrated using the pseudo-code in Fig. 9.

At each step in our simulated annealing process, we calculate the minimum area required by each solution instance using the method presented in [31]. As each module $M_i \in C \cup B$, is assigned an exact placement coordinate $M_i(x_i, y_i)$ in the plane. To estimate bus lengths, we use 1/2 the perimeter of the smallest bounding box (BB_i) of each bus $hN_i \in hCCG$. The objective function that simulated annealing optimizes is a linear combination of the area and bus length requirements. By varying the coefficients of this function, one can increase or decrease the degree of importance of each constraint. The equation below illustrates the estimated cost that is optimized during the floorplanning process:

```
$T = T_0$; $M = C \cup B$; $currentFloorplan = Initial\_Floorplan(M)$
$EngineeringChangeProcedure(hCCG)$
While $(T > T_F)$
  $currentFloorplan = bestFloorplan$
  While (cumulative improvement $\geq \sigma$)
    $a = random(LongestBus, ShortestBus)$
    Case$(a > Const) : currentFloorplan.Greedy\_Move()$
    Case$(a \leq Const) : curerntFloorplan.Enabling\_Move()$
    $Current\_Area = Area(currentFloorplan)$
    For each $hN_i \in hCCG$
      If $(Length(hN_i) > \lambda_{max})$
        Add $BB_i$ to violated constraints list
    End For
    $Current\_Cost = b \cdot Current\_Area + d \cdot \sum_{i=1}^{|CCG(hN)|} BB_i$
    Decision: Accept or reject currentFloorplan.
    If $Current\_Cost \geq min(Cost(\Pi_i \in \Pi(K))$
      Add currentFloorplan to $\Pi(K)$.
  End while
  Decrease temperature
End while
```

Fig. 9. Pseudo-code of the simulated annealing algorithm for approximate latency-guided floorplanning.

$$Fpl\_Cost = b \cdot Current\_Area + d \cdot \sum_{i=1}^{|CCG(hN)|} BB_i, \quad (7)$$

where constants b and d are determined empirically.

The floorplanning tool outputs several important statistics to the network designer. During simulated annealing, we keep track of the K best solutions encountered ($\Pi(K)$). For each solution $\Pi_i \in \Pi(K)$, we report the area and the violated bus constraints. In addition to the best solution instances found, we also report the overall percentage of instances that each bus constraint has violated.

The standard simulated annealing process is augmented with solution transformation actions, *moves*. We define two types of moves: *greedy* and *enabling* moves. Depending on the lengths of the shortest and the longest buses, we assign the probabilities of taking either the greedy or enabling move.

In the *greedy* move, we select the longest bus and try to improve the placement of its modules. We calculate the center-of-mass C_m of the bus by averaging the x and y coordinates of each external connection of the bus. We then calculate a force vector **V** for the module that has the longest Manhattan distance from C_m. **V** is used to update the sequence pair strings such that the selected module is moved in the direction of **V** in proportion to the magnitude $|V|$.

Similarly, in the *enabling* move, we select the shortest bus and try to relax the placement of its modules. We calculate the center-of-mass C_m and force vector **V** as described above with the exception of selecting the module closest to the C_m. We update the sequence pair strings such that the selected module moves in the opposite direction of **V** in proportion to the magnitude of $|V|$.

A special case in both the greedy and the enabling moves arises when a bridge is selected to be moved. In this case, the force calculated for the bridge is an average of the forces acting on the bridge from all buses connected to the bridge. An example of a resulting floorplan which satisfies given bus wirelength constraints for a design with 100 cores is shown in Fig. 10.

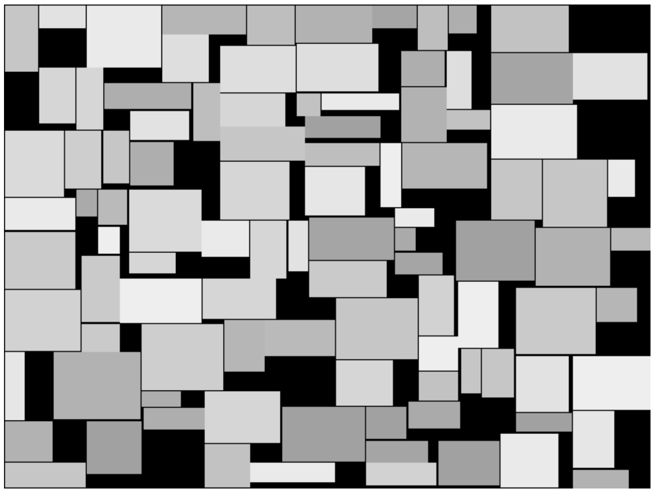


Fig. 10. An example of a resulting floorplan for a design with 100 cores.

C. Discussion

There is a number of other possible algorithmic approaches besides simulated annealing that can be applied to problems defined in Subsections V-A and V-B. Many types of heuristic, iterative improvement, probabilistic, integer linear and non-linear programming and other types of algorithms can be easily envisioned and realized for the addressed task. We decided to use a probabilistic approach mainly because of the complex structure of interacting constraints and complexity of modeling. The second aspect on which our decision was made is that simulation and extraction of estimation based on hCCG are time consuming. Therefore, even if run-time intensive optimization is conducted, the overall additional overhead to the overall approach is relatively low.

VI. Experimental Results

We have demonstrated the effectiveness of our synthesis paradigm on a set of synthetic designs. The design benchmark has been assembled using a library of intellectual property cores from [41]. The cores varied from controllers, crypto and DSP processors and functions, signal modulators, multimedia codecs, communications and peripherals ASICs, voice codecs, etc. The area for each core was estimated based on the core's gate count.

Currently, there are no established benchmarks for synthesis of systems on chips. Very few applications are available in public domain. We were able to obtain two large communication applications from our industrial partners. However, we are bound by non-disclosure agreements so we are not able to publish the obtained results. In order to circumvent this problem, we decided to extrapolate applications from MediaBench benchmark suite [39]. The extrapolation was conducted in the following way. We identified manually functional blocks that were targeted for implementation on a single core. Each block represented a single node in a hCCG. Then, we increased the number of blocks. Next, we connected new hCCG nodes to the remainder of the hCCG as well as one to another. The edges were added in such a way that each new hCCG node had $\frac{m}{n}$ % more edges with proportionally more weight, where m and n were the existing number of edges and nodes respectively.

The final step was the aggregation of multiple original and extrapolated single applications. That was accomplished by adding a single edge between two atomic applications. The edge had a weight that is proportional to the original output of the application with higher output. While, obviously, the synthetic applications were not complete replacements for generic system on chip applications, the developed model did have statistically similar properties. Application requirements were determined based on a statistical model of an application, however, with certain requirements for general-purpose, DSP, crypto, communications, and speech processing.

We demonstrate our approach on computationally intensive applications where the ratio of computation to communication is relatively high and where strict throughput and synchronization timing constraints are imposed. While, in principle, one can envision ways to generalize the approach to reactive, control-dominated and other types of applications, it is not clear to what extent the proposed approach would be effective. Therefore, we restrict our attention to computationally intensive application, such as one found in MediaBench [39].

The experimental results are presented using Table I. Columns 1-4 describe the tangible properties of each system: application emphasis, estimated number of gates [41], number of buses, and number of bridges. Columns 5 and 6 quantify the run-time properties of the synthesis framework: the number of complete iterations of the bus network design and floorplanning loop and the elapsed total run-time of the semi-automated process. The last two columns represent the following properties of the obtained solution: *(i)* optimized system throughput as a multiple of the throughput obtained by the initial ad-hoc system which is fed as a starting solution to the bus network designer and *(ii)* the median ratio of idle cycle time on the system buses. During our experiments, we have explored different strategies for selecting the initial solution: greedy heuristic, random elimination, and biased random elimination. Greedy heuristic starts with complete solution where we assign a dedicated interconnect between any two blocks that exchange data. It consequently at each step eliminates interconnect with the smallest amount of traffic and assigns its traffic to bus network that are able to accommodate additional traffic and have the highest utilization ratio. In the first variant of random elimination, we randomly selected which interconnect to eliminate and to which to assign traffic. In the second variant, biased random elimination, the selection of interconnect for elimination and traffic reassignment was also random. However, the probabilities for elimination were inversely proportional to traffic and the probabilities for assigning traffic were directly proportional to the traffic. Both of the randomized strategies were augmented with restart strategies. The termination criteria was that in 100 consecutive attempts no additional improvement with the respect to the

Design Specification	Cores	Gate count	Buses	Bridges	Synthesis loop		α-optimal Solution Properties	
					Iter.	Time	Throughput	Median bus idle time ratio
DSP+crypto	13	1M	4	2	3	15min	1.46	0.45
GPP+communications	30	1.8M	11	4	4	1h	1.57	0.71
GPP+DSP	75	4.2M	21	7	5	5h	2.03	0.43
Communications+speech	100	5.4M	31	12	5	11h	3.11	0.79
DSP+speech	125	7.5M	41	15	5	17h	2.17	0.40
GPP+peripherals+communications	150	9.6M	47	18	7	21h	2.42	0.43
GPP+crypto+peripherals	200	15M	62	23	9	35h	3.16	0.67

TABLE I

QUANTIFICATION OF THROUGHPUT IMPROVEMENTS USING THE DEVELOPED BUS NETWORK DESIGNER FOR A NUMBER OF BENCHMARK DESIGNS EXTRAPOLATED FROM REAL-LIFE APPLICATIONS [39].

best current solution was detected. Different initial solutions had no effect to the quality of the final solution. An example of a design and its resulting bus network is shown in Fig. 11. The throughput improvement for various designs ranged from 46% to 216%. We reported the throughput improvement with respect to the initial solution that yielded the best throughput. Run-times per synthesis loop were increasing from 15 minutes to 35 hours for designs that ranged from 1 million (13 cores and 2 bridges) to 15 million (200 cores and 23 bridges) gates.

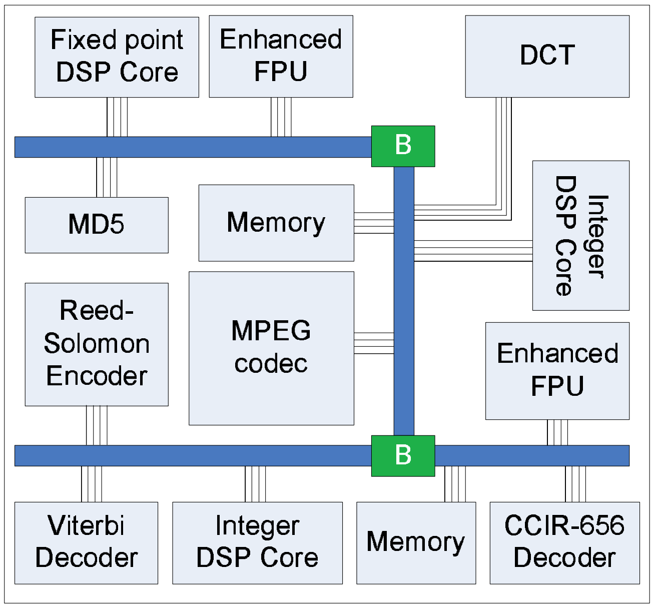


Fig. 11. An example of a design (labelled DSP+crypto in Table I) with 13 cores, 4 buses, and 2 bridges.

VII. CONCLUSION

In order to address the two major ramifications of DSM on the design process: increased interconnect delay and design reuse using blocks with decreased functional granularity and increased communication demands, we have developed a methodology for the design of on-chip bus network structures. The design methodology closes the synthesis loop between system and physical design by performing the following three procedures. First, the communication among cores is profiled to obtain run-time information about the traffic. Next, the bus network designer creates and optimizes the bus network structure by coordinating information from the profiler and the approximate floorplanner. The latter, as the final component of the design, aims at creating a feasible floorplan. In the case of an infeasible solution, the approximate floorplanner communicates the information about the most constrained parts of the network back to the bus network designer. The efficiency of the presented design methodology has been demonstrated on a set of multi-core designs extrapolated from a multimedia benchmark suite.

REFERENCES

[1] D. Sylvester and K. Keutzer, "Rethinking deep-submicron circuit design," *Computer*, vol. 32, no. 11, pp. 25–33, 1999.
[2] L. Carloni, K. McMillan, A. Sladanha, and A. Sangiovanni-Vincentinelli, "A methodology for correct-by-construction latency insensitive design," *International Conference on Computer Aided Design*, pp. 309–15, 1999.
[3] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T.-P. Fang, "Q-modules: Internally clocked delay-insensitive modules," *Transactions on Computers*, vol. C-37, no. 9, pp. 1005–8, 1988.
[4] J. T. Udding, "A formal model for defining and classifying delay-insensitive circuits," *Distributed Computing*, vol. 1, no. 4, pp. 197–204, 1986.
[5] B. Davari, "CMOS technology: Present and future," *Symposium on VLSI Circuits*, pp. 5–10, 1999.
[6] K. Goto, T. Sugii, and J. Matsuo, "High performance 0.04 μm PMOSFET," *Fujitsu Scientific and Technical Journal*, vol. 34, no. 2, pp. 135–41, 1998.
[7] R. Ho, K. Mai, H. Kapaida, and M. Horowitz, "Interconnect scaling implications for CAD," *International Conference on Computer Aided Design*, pp. 425–9, 1999.
[8] H. Zhou and D. F. Wong, "Global routing with crosstalk constraints," *Design Automation Conference*, pp. 374–7, 1998.
[9] M. C. Johnson, D. Somasekhar, and K. Roy, "Leakage control with efficient use of transistor stacks in single threshold CMOS," *Design Automation Conference*, pp. 442–5, 1999.
[10] J. Abraham, "Power calculation and modeling in deep submicron," *International Symposium on Low Power Electronics and Design*, pp. 124–6, 1998.
[11] J. Crenshaw, M. Sarrafzadeh, P. Banerjee, and P.Prabhakaran, "An incremental floorplanner," *Ninth Great Lakes Symposium on VLSI*, pp. 248–51, 1999.
[12] C. Alpert, A. Devgan, and S. Quay, "Buffer insertion for noise and delay optimization," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 11, pp. 1633–45, 1998.
[13] J. L. Wong, A. Davoodi, V. Khandelwal, A. Srivastava, and M. Potkonjak, "Wire-length prediction using statistical techniques," *International Conference on Computer Aided Design (ICCAD)*, pp. 702–5, 2004.
[14] J. Hu and S. S. Sapatneker, "FAR-DS: Full-plane AWE routing with driver sizing," *Design Automation Conference*, pp. 84–9, 1999.

[15] J. Cong and L. He, "Optimal wiresizing for interconnects with multiple sources," *ACM Transaction on Design Automation of Electronic Systems*, vol. 1, no. 4, pp. 478–511, 1996.

[16] A. Salek, J. Lou, and M. Pedram, "An integrated logical and physical design flow for deep submicron circuits," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 9, pp. 1305–15, 1999.

[17] R. Otten and R. Brayton, "Embedded tutorial: Planning for performance," *Design and Automation Conference*, pp. 122–7, 1998.

[18] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, and A. A. Jerraya, "A generic wrapper architecture for multi-processor SoC cosimulation and design," *Proceedings of International Workshop on Hardware-Software Codesign*, pp. 195–200, 2001.

[19] T.-Y. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," *International Conference on Computer Aided Design*, pp. 288–95, 1995.

[20] M. Drinic, D. Kirovski, S. Meguerdichian, and M. Potkonjak, "Latency-driven design of multi-purpose systems-on-chip," *ACM-IEEE Design Automation Conference*, pp. 27–30, 2001.

[21] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Application-driven synthesis of core-based systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 9, pp. 1316–26, 1999.

[22] P. Gerin, S. Yoo, G. Nicolescu, and A. Jerraya, "Scalable and flexible cosimulation of SoC designs with heterogeneous multi-processor target architectures," *Asia South Pacific Design Automation*, pp. 63–8, 2001.

[23] P. V. Knudsen and J. Madsen, "Integrating communication protocol selection with partitioning in hardware/software codesign," *International Symposium on Systems Synthesis*, pp. 111–6, 1998.

[24] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage, "Networks on silicon: Combining best-effort and guaranteed service," *Proceedings of the conference on Design, Automation and Test in Europe*, p. 423, 2002.

[25] D. Lapotin and S. Director, "Mason: A global floorplanning approach for VLSI design," *Transactions on Computer-Aided Design of Integrated Circuit Systems*, vol. 5, no. 4, pp. 477–89, 1986.

[26] D. Wong and C. Liu, "Floorplan design of VLSI circuits," *Algorithmica*, vol. 4, pp. 263–91, 1989.

[27] J. Gu and K. F. Smith, "A structured approach to vlsi circuit design," *IEEE Computer*, pp. 9–22, 1989.

[28] K. Udea, H. Kitazawa, and I. Harada, "CHAMP: Chip floor plan for hierarchical VLSI layout design," *Transactions on Computer Aided Design of Integrated Circuit Systems*, vol. 4, no. 1, pp. 12–22, 1985.

[29] E. Kuh and T. Ohtsuki, "Recent advances in VLSI layout," *IEEE Proceedings*, vol. 78, no. 2, pp. 237–62, 1990.

[30] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–80, 1983.

[31] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing-based module placement," *International Conference on Computer Aided Design*, pp. 472–9, 1995.

[32] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*. Dordrecht, Netherlands: Kluwer Academic Publishers, 1993.

[33] T. Gore, "PI-bus-making single-chip real-time solutions a reality," *Real-Time Magazine (no.4), Real-Time Consult*, pp. 6–10, 1995.

[34] J. Gateley, "Sun microsystems integrates emulation into the SPARC processor and workstation design process," *ASIC & EDA*, 1994.

[35] D. Kirovski, M. Drinic, and M. Potkonjak, "Engineering change protocols for behavioral and system synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 8, pp. 1145–55, 2005.

[36] G. Bischoff, K. Brace, S. Jain, and R. Razdan, "Formal implementation verification of the bus interface unit for the Alpha 21264 microprocessor," *International Conference on Computer Design*, pp. 6–24, 1997.

[37] E. Macii and M. Poncino, "Automatic synthesis of easily scalable bus arbiters with dynamic priority assignment strategies," *Computers & Electrical Engineering*, vol. 24, no. 3–4, pp. 223–8, 1998.

[38] Z. Zhang, Y. Fan, M. Potkonjak, and J. Cong, "Gradual relaxation techniques with applications to behavioral synthesis."

[39] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communications systems," *International Symposium on Microarchitecture*, pp. 330–5, 1997.

[40] M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[41] *Available online at http://www.mentor.com/inventra/cores/catalog/index.html*, 2000.

Milenko Drinic Milenko Drinić received his Ph.D. degree in computer science from the University of California, Los Angeles in 2003. In 2003 he joined Microsoft Research, Redmond, WA. His research interests include: static analysis, static and dynamic code optimization, data compression, system, security intellectual property protection, and VLSI CAD.

Darko Kirovski Darko Kirovski received his Ph.D. degree in computer science from the University of California, Los Angeles, in 2001. Since April 2000 he has been a researcher at Microsoft Research. His research interests include: system security, multimedia processing, and embedded system design. He has received the 1999 Microsoft Graduate Research Fellowship, the 2000 ACM/IEEE Design Automation Conference Graduate Scholarship, the 2001 ACM Outstanding Ph.D. Dissertation Award in Electronic Design Automation, and the Best Paper Award at the ACM Multimedia 2002. Contact info: Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA. Email: `darkok@microsoft.com`

Seapahn Megerian Seapahn Megerian received the BS degree in computer science and engineering in 1998 and the MS degree in computer science in 1999 from UCLA. He received the PhD degree in computer science from the University of California at Los Angeles in 2004. He is an assistant professor in the ECE Department at the University of Wisconsin at Madison. His main research areas are distributed embedded systems and wireless sensor networks. In addition, his research includes design automation of high performance communication architectures, computational security, and intellectual property protection

Miodrag Potkonjak Miodrag Potkonjak received his Ph.D. degree in Electrical Engineering and Computer Science from University of California, Berkeley in 1991. In 1991, he joined Computer & Communication Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been with Computer Science Department at UCLA, until 1998 as Assistant Professor, then as Associate Professor, and since July 2000 as Professor.

He received the NSF CAREER award, OKAWA foundation award, UCLA TRW SEAS Excellence in Teaching Award and a number of best paper awards. He has published a book and more than 250 papers in leading CAD and VLSI design, real-time systems, multimedia, signal processing, computational security, and communications, journals and conferences. He holds five patents. His research interests are focused on complex distributed systems, including embedded systems, computer aided design, ad hoc sensor networks, computational security, practical optimization and modeling techniques, and intellectual property protection.