On-Chip Bus Architecture Optimization for Multi-core SoC Systems*

Cheng-Min Lien[1], Ya-Shu Chen[1], and Chi-Sheng Shih[2]

[1] Department of Computer Science and Information Engineering
[2] Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106
cshih@csie.ntu.edu.tw

Abstract. With the significant driving force from the application domains, modern embedded systems are designed over heterogeneous multi-core SoC platforms. When more and more functions are integrated into one system, the designs of embedded systems have become more and more complicated. In particular, most of embedded multimedia applications are data intensive. Performance bottleneck are often caused by inappropriate bus architecture design within the system. In this paper, we present the algorithms for bus architecture optimization in MFASE. The algorithm takes the workloads in the system and their timing behavior requirements into account. The goal is to minimize the number of buses in the system without violating timing requirements. We prove that the minimzation problem is NP-hard and develop a heuristic algorithm. We evaluate the algorithm with extensive simulations. The performance results show that the algorithm reduce up to 80% of the bus cost and performs as well as optimal exponential algorithm does.

1 Introduction

Traditional system-level chip design aims at designing reliable single function systems or distributed embedded systems. Thanks to modern SoC platform, a multiple function system can be integrated onto a single chip. However, the design complexity exponentially grows as the number of functions on one chip increases. Traditional design approach is not suitable for such systems. Without proper system-level performance analysis tool and design tool, engineers rely on their experience of designing single function systems to design multiple function systems. The results are usually over-allocated resources such as bus, memory and processing elements. In addition, traditional design tools focus on the correctness of the systems. Timeliness of the systems is left for the application engineers and is ignored during system-level chip design. Traditional design method leads to great programming overhead and is not suitable for designing multi-function SoC systems.

* This work is supported in part by a grant from Academia Sinica Thematic program and part by a grant from the NSC program NSC 96-2752-E-002-008-PAE.

The design of SoC can become very complex due to the variety of software and hardware system blocks that need to be integrated. Mobile phone is one example. As the market gets more competitive, more and more features such as motion video capability and audio playback are integrated in mobile phones. A new challenge is how to find a communication architecture between the cost and performance trade-off efficiently for the mobile phone venders. It is because that if all processing elements are on one bus, the execution of every processing element becomes sequential. The application may be fail due to the bus contention. To solve the problem, we can use multi-bus architecture to increase the parallelism of the system.

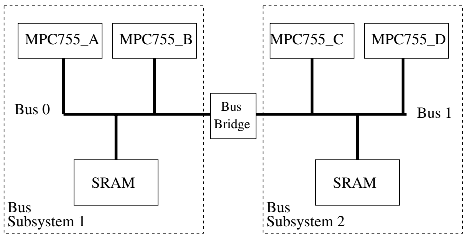


Fig. 1. An example of multi-bus architecture

Figure 1 shows an example of multi-bus architecture which are connected through a bus bridge to exchange data between them. Each bus subsystem has two MPC755s [1] and a memory block. Both bus systems in Figure 1 can operate at the same time without bus contention. In this way, the system performance will increase. On the other hand, the cost of the system also increases because of additional bridge, memory and bus routing for bus subsystems.

In last few decades, many researches have focused on the SoC communication system-level synthesis problem [2,3]. In the paper, we are concerned with the bus architecture synthesis for SoC platform. We use a directed acyclic graph (DAG) to describe the software property including bus transaction, transaction time, precedence constraint and timing constraint. We also demonstrate that the proper selection of the communication architectures which based on multi-bus. For this, we schedule the bus transactions of input software for the different architecture and select one.

There are numerous advantage for conducting system level co-synthesis such as shortening the design time, manufacture cost, die size and power dissipation [4,5]. The researches in system level co-synthesis address on two main issues. The first issue is to optimize the selection and mapping of the system's functional blocks onto a set of processing elements (PE), like CPUs, digital signal processors (DSPs) and application-specific cores, etc [6,7]. The second issue is to optimize the communication architecture between the processing elements [8,9,10,11]. The separation between computation and communication enables the system designers to explore the communication architecture independently of processing elements selection and mapping. The focus of this paper lies on the second problem of the system level co-synthesis design.

The targeted issue in this paper is to systematically determine on-chip bus architecture so as to minimize chip cost subject to the real-time performance constraints. The algorithm is a greedy algorithm. It starts with a most expensive architecture to conduct the feasibility test for the given task set. When the real-time performance constraints are not met, the algorithm terminates. Otherwise, the algorithm iteratively evaluates the design and reduces the number of buses on the chip. It terminates when the real-time performance constraints cannot be met. In this paper, we use a more practical model for the SoC bus design. Both high-level bus transactions as well as the effect of shared memory accesses are considered.

The remainder of this paper is organized as follows: In Section 2, we present related work in on-chip bus communication synthesis and formally define the problem. A two steps heuristic algorithm was proposed for on-chip bus communication architecture synthesis in Section 3. Section 4 presents the performance evaluation results for the developed algorithms. Finally, Section 5 summaries the paper.

2 Background and Formal Model

Our work is related to several on-chip bus synthesis researches. In [10], Lahiri *et al.* presented an algorithm to find a communication architecture after the system has been partitioned. They focused on how to use a set of bus architectural templates to connect processing elements. To do so, the algorithm assumes that the bus topology is given. In this paper, we relax this assumption. Specifically, we are interested in how to synthesis the bus topology when the bus transactions are given based on system-level design analysis. In [9], Kim *et al.* presented the problem for finding the on-chip bus topology and the allocation of shared memories. The proposed exploration technique is a three steps algorithm. In the first step, they use a static performance estimation technique, proposed in [8], to quickly evaluate each candidate design and prune the design space. The second step is to scatter communication traffic in conflict on a bus into different buses to reduce conflict and maximize concurrency. For this purpose, the algorithm selects a processing element one at a time, allocates it to a new bus, and produces different share memory allocation. This step is time consuming, when the number of processing elements are large. The last step determines the priorities of each processing elements.

In [11], Pandey *et al.* formulate the problem from a different perspective. The given input is a hardware communication lifetime interval graph (CLTI.) In CLTI, it is assumed that the computation time for each processing element is a constant but the bus transaction time, depending on the bus width, is not a constant. The contribution of the paper is to find an optimal bus width which minimizes the bus contention, and then to optimize the on-chip bus topology. Unfortunately, many bus protocols such as AMBA only support fixed bus width. Hence, this approach is only applicable for custom designed bus architecture.

In the following, we define the terms used in this paper and define the problem of interests. *Processing element*, denoted by PE_k where k is no less than 0, is a CPU, DSP, or an ASIC in the system. A *Task* is a sequence of works such as computation and file access to complete certain function. Tasks are denoted by

T_1, T_2, etc. When a task needs to send or retrieve data from other components in the system such as memory or processing elements, it triggers a bus access request. When the bus is free, the task occupies the bus to send or retrieve data. Otherwise, it may wait till the request is granted by bus arbitrator. Namely, a *bus transaction* is a data transmission over bus by a task to other devices in the system. The bus transactions for task T_i are denoted by $BT_{i,1}$, $BT_{i,2}$, etc. In this paper, we assume that the execution of a bus transaction cannot be interrupted. A bus transaction fails when it is interrupted during its execution. A bus transaction is defined by two parameters: requesting PE and bus transaction interval. *Requesting PE* is a processing element on which the task starts the bus transaction, and *bus transaction interval* is the amount of time needed to complete the transaction. Bus transaction interval for bus transaction $BT_{i,j}$ is denoted by $BI_{i,j} \in \mathbb{Z}^+$. We assume that the length of all bus transaction intervals are known *a priori*. It is because the HW/SW partitioning result has given. *Precedence constraint* is the execution order of bus transactions. A bus transaction cannot start until all of its preceding bus transactions complete.

Bus transaction graph, denoted by $BTG = (V, E)$ where V and E is the set of vertex and edges, is a labeled directed acyclic graph. A vertex, denoted by $v_{i,j}$, represents bus transaction $BT_{i,j}$; A directed edge from $v_{i,j}$ to $v_{i,k}$, denoted by $e_{i,j,k}$, represents that bus transaction $BT_{i,j}$ is the preceding bus transaction for bus transaction $BT_{i,k}$.

While designing a system-on-chip, we are often interested in a set of tasks. Hence, we can present all the bus transactions for the set of tasks by a set of bus transactions. *Common relative deadline* for a set of bus transactions, denoted by $D \in \mathbb{Z}^+$, is the maximum allowable response time for any of the bus transactions. The set of bus transactions meet its timing constraint when all the bus transactions complete before the common deadline. The rationale of meeting timing constraint is to assure that the tasks can meet their real-time performance requirements. Example for the real-time performance requirements are the playback rates for multimedia player and sampling rate for audio recorder.

Local memory for a bus transaction means that the memory and requesting PE are on the same bus; *Remote memory* for a bus transaction menas that the memory and requesting PE are not on the same bus. Hence, when the bus transaction is executed, it will occupy at least two buses to access the memory. *Shared memory*, denoted by $SM_{i,j,k}$, represents a region of memory space for the data communication between bus transactions $BT_{i,j}$ and $BT_{i,k}$. When the bus transaction $BT_{i,j}$ is executing, the bus connected to the shared memory space is reserved for bus transaction $BT_{i,j}$ to access (i.e. read or write) the shared memory space. *Bridge* connects two buses such as AMBA AHB bus so that a processing element on one bus can access the memory on another bus. We assume that the bridges are programmable and can be set as open or close during the run-time. If a bridge is open, the two buses connected by the bridge can be simultaneously reserved by different PE in the same time. On the other hand, the two buses are linked together, they will be regard as one bus and simultaneously reserved by the same PE.

Communication architecture is based on multi-bus architecture. Every bus connects at least one processing element and exactly one memory. The different buses which are connected by bridges can be reserved by different processing elements concurrently. The more buses brings more concurrency and higher cost.

The bus architecture synthesis problem is defined as following.

Definition 1. *Given a set of bus transaction graphs* $\mathbf{G} = \{BTG_1, BTG_2, ...BTG_N\}$ *and their common deadline* D*. The problem is to determine a communication architecture with the minimal number of buses subject so that the timing constraint is met.*

In the following, we use an example to illustrate the problem defined in Definition 1. Figure 2 shows the set of bus transaction graphs: BTG_0 and BTG_1. There are six bus transactions. The color and label of each vertex represent its requesting PE and bus transaction interval. For instance, $BT_{0,0}$ and $BT_{1,1}$ are requested by processing element PE_0 for 3 time units. In addition, bus transaction $BT_{1,1}$ and $BT_{1,2}$ can start only after bus transaction $BT_{1,0}$ completes, due to the precedence constraint. Figure 3 shows a communication architecture for the bus transaction graphs shown in Figure 2. This architecture uses two buses to connect three processing elements. The processing elements PE_0 and PE_2 are allocated on bus B_0 and PE_1 is allocated on bus B_1. The shared memory $SM_{0,0,1}$ and $SM_{1,0,2}$ is allocated on memory M_0. The shared memory $SM_{1,0,1}$ and $SM_{0,1,2}$ is allocated on memory M_1.

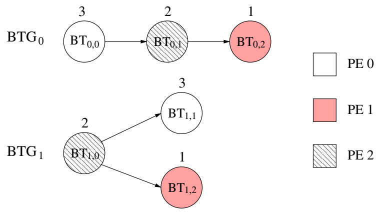


Fig. 2. An example of bus transaction graphs

Given the communication architecture, we can find the bus transaction schedules. Figure 4(a) illustrates the schedule without shared memory. Figure 4(b) illustrates the schedule with shared memory. The blocks in the schedules represent the bus reservations. There are two kinds of bus reservations which caused by local memory access and shared memory access. For example, block $BT_{0,0}$ is a local memory access, so bus 0 is reserved for the temporary files I/O when bus transaction $BT_{0,0}$ is executing in the time interval 0 to 3. $SM_{1,0,1}$ is a shared memory access between $BT_{1,0}$ and $BT_{1,1}$, so bus 1 is reserved for reading input data from the shared memory when bus transaction $BT_{1,1}$ is executing in the time interval 5 to 8. Similarly, bus 1 is reserved for writing output data to the shared memory when bus transaction $BT_{0,1}$ is executing in the time interval 3 to 5.

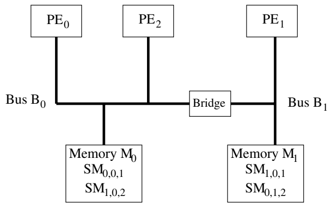
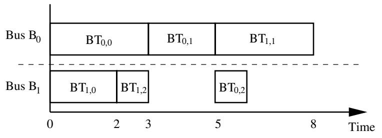
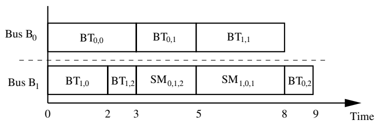


Fig. 3. An example communication architecture for BTG_0 and BTG_1

Our problem is to find a cost effective architecture with a bus transaction schedule which meets the timing constraint. We will show that the sub-problem



(a) Without Shared Memory



(b) With Shared Memory

Fig. 4. Bus Transaction Schedule Examples

to schedule the bus transaction graphs on a given architecture is a NP-complete problem. In our model, each bus transaction graph is a connected graph in DAG and released at time 0. There is a common deadline for all bus transaction graphs. In a given architecture, each bus transaction is scheduled to the requested PE's local bus, so the allocation of bus transactions are given. If all the bus transaction graphs are chains, the sub-problem is to schedule the bus transaction graphs which are chains on a given architecture. The special case of the scheduling problem is the **JOB SHOP SCHEDULING**[12] which is NP-complete. The special case of the sub-problem is a NP-complete problem, so the hardness of the problem we want to solve in this paper is also NP-complete at least.

3 On-Chip Bus Synthesis Algorithms Design

In this section, we present the heuristic to find near optimal bus architecture, *Greedy Bus Architecture Synthesis Algorithm (GBASA.)* The GBASA algorithm is an iterative algorithm and consists of three major steps. GBASA algorithm starts with the bus architecture in which each processing element is connected to one dedicated bus to conduct the feasibility analysis. If there is no feasible schedule for this architecture, the algorithm stops because the timing constraint will never be met. When the timing constraint can be met, the algorithm continues to reduce the number of buses so as to reduce the cost. The algorithm stops when the timing constraint can barely be met.

Figure 5 shows the flow of the proposed algorithm. In the first step, the algorithm synthesizes a most expensive architecture in which every processing element has its dedicated bus. The second step revises the initial architecture by scheduling the bus transactions. If the timing constraint cannot be met, the algorithm terminates and returns no feasible architecture. Otherwise, the third step chooses a pair of buses and merging them into one bus to reduce the cost. In order to shorten the finish time of the bus transaction graphs, the pair of buses which cause less effect

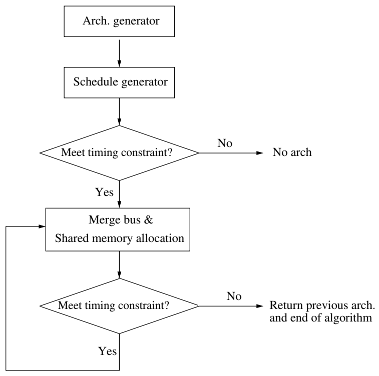


Fig. 5. Flow of GBASA Algorithm

to the finish time of the schedule is merged. After merging the bus pair, the algorithm returns to the second step to revise the architecture. If the timing constraint can be met, the third step tries to reduce cost again. On the contrary, the algorithm returns the last feasible architecture and terminates. In the following, we present the algorithm step by step.

Initial Communication Architecture Generation. The algorithm first adds processing elements to the initial communication architecture, and one memory and one dedicated bus for every processing element. Then, the algorithm checks every edge in the bus transaction graphs for examining the communications between processing elements. If there are bus transactions between any two processing elements, the algorithm adds a bridge between two dedicated buses. After this initiation, we will have a communication architecture with high concurrency and cost. It is because we use as many memories and buses as we can. Again, take Figure 2 as the input of the algorithm. The algorithm *Initial Architecture Generator* generates the initial architecture which has three PEs with its own bus and memory. After initializing PE, bus and memory, we add a bridge for a pair of buses, if the two PEs on the two buses have to communicate to each other. The output of the algorithm is shown in Figure 6.

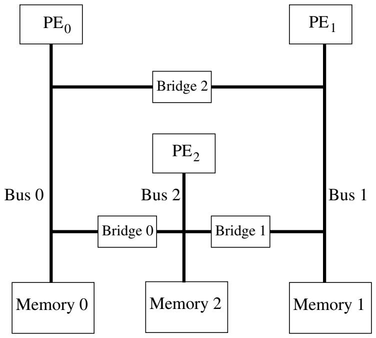


Fig. 6. The initial architecture

Architecture Evaluation. In the second step, the algorithm first conducts the feasibility test and considers the share memory allocation later.

To efficiently generate a schedule which has a short makespan, we use a heuristic instead of exhaustive search. Observing the given task set, we know that each bus transaction has its own release time[1], deadline, and is non-preemptive. The scheduling policy is to schedule the bus transaction which has longer residual execution time. The residual execution time of a bus transaction is the summation of all the successors' execution times. The rationale is that longer residual execution time implies that there are more bus transactions to be completed before the common deadline. We use the value to determine the importance of bus transactions. The time complexity of *Longest Successors' Execution Time First Algorithm* is $O(n^2)$. The algorithm is similar to the Least Slack Time First algorithm [13]. The difference is that each bus transaction does not have equal release time but has common deadline. When the requesting PEs for two consecutive bus transactions are located on different buses and need to exchange data, share memory is the most common mechanism to do so. However, more than one bus may be occupied to access a remote share memory. Hence, which memory

[1] Only the first bus transaction for every BTG, $BT_{i,1}$ for $i \geq 0$, has the same release time.

on one of the two buses is used for a share memory determine the makespan and cause deadline miss. After a feasible schedule is found in the above step, the algorithm determines the use of share memory. Three cases are considered:

– When both the schedule of the two buses have available time intervals for remote memory access, the algorithm selects the one with shorter bus transaction interval.
– When only one of the bus have available time interval for remote memory access, a remote memory access is inserted to the bus.
– When none of the two buses has available time intervals for remote memory access, a remote memory access is inserted to the bus for shorter bus transaction interval.

Inserting remote memory access in the first two cases does not prolong the makespan. However, the third case prolongs the makespan. If the timing constraint cannot be met, the architecture is not feasible and the algorithm terminates.

Communication Architecture Cost Down. When the timing constraint is still met, the GBASA algorithm continues to reduce the number of buses in the design. The algorithm selects one of the bridges if there is any to remove. Two metrics, overlap time and precedence number, are used. The *overlap time* of a bridge is the sum of time intervals during which the two buses are occupied simultaneously. The *precedence number* of a bridge is the number of bus transactions to be removed due to the remote memory access between the bus pair. First, we choose a set of bridges which have the same shortest overlap time. In the second step, we choose a bridge which has the largest precedence number. The rationale is that we merge the two buses which have less bus contention. By merging the bus pair, we can also reduce the additional bus transaction caused by shared memory access. By using this algorithm, we merge the bus pair bus B_0 and B_2. Figure 7 shows the new architecture after bridge remove. The new architecture will be evaluated by Step 2 for further cost down.

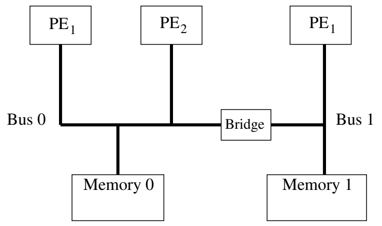
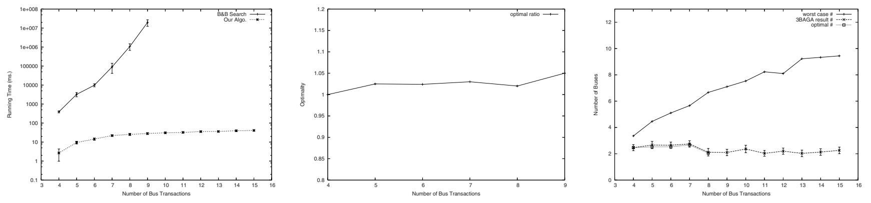


Fig. 7. The example architecture after one iteration

4 Performance Evaluations

We evaluate the performance of the GBASA algorithm by extensive simulations and compare its performance with the exponential branch and bound algorithm. Two metrics, *optimality* and *timing overhead time*, are measured to evaluate the performance. *Optimality* is the ratio of the number of buses selected by the GBASA algorithm to that selected by branch and bound algorithm. *Timing overhead* is the amount of time that the two algorithms take to complete their designs.



(a) Timing Overhead (b) Optimality ratio (c) Bus elimination with maximum 15 PEs

Fig. 8. Evaluation Results

The software environment of the experiments is *Ubuntu Linux Distribution* and *GNU GCC 4.0*. The hardware environment is a machine equipped *Intel Pentium III* 800 MHz and 128MB RAM. We evaluate our algorithm in different types of bus transaction graphs including DAG, tree and chain in order to simulate different types of applications. The transaction time are uniformly distributed in the range of 10 to 70. We compare the result of the GBASA algorithm to the worst case architecture and the optimal case architecture.

Figure 8(a) shows the timing overhead of the GBASA algorithm and B&B algorithm. The confidence interval of each data point in the figure is no less than 95%. The x-axis is the total number of bus transactions. The y-axis is the running time in milliseconds. As shown in the figure, the timing overhead of B&B search-based algorithm exponentially increases with the number of bus transactions. On the other hand, the timing overhead of GBASA algorithm slowly increases. Figure 8(b) shows the optimality of the GBASA algorithm. The figure shows that the GBASA algorithm performs as well as the branch and bound algorithm. Figure 8(c) shows how many redundant buses the GBASA algorithm eliminates, comparing to the worst design. A worst design is the one in which every processing element has its dedicated bus and memory. Figure 8(c) shows the results when there are 15 PEs. The results show that the GBASA algorithm performs as well as an optimal results and eliminates up to 80% of the buses from the worst design.

5 Conclusion and Future Work

In this paper, we presented a solution for on-chip bus synthesis for a system-on-a-chip (SoC) such that the communication cost if minimized subject to real-time performance constraints. We developed a three stages heuristic, GBASA algorithm, to synthesize on-chip bus design and shorten the makespan simultaneously. Performance evaluation results show that the GBASA algorithm reduces the run-time overhead and derive near-optimal solutions. For the future work, we can extend the communication architecture to the multi-layer bus architecture. It is because the multi-layer bus architecture can also provide higher concurrency. Another research direction is the routing path of the on-chip bus. The complex of SoC is still growing up. There will be more complex connection between hardware components. The routing path of on-chip bus will affect the size of chip, energy dissipation and the propagation delay.

References

1. Ryu, K.K., Mooney, V.J.: Automated bus generation for multiprocessor soc design. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, pp. 1531–1549. IEEE Computer Society Press, Los Alamitos (2004)
2. Wolf, W.H.: An architectureal co-synthesis algorithm for distributed, embedded computing systems. IEEE Transactions on Very Large Scale Integration Systems 5, 218–229 (1997)
3. Yen, T.Y., Wolf, W.: Performance estimation for real-time distributed embedded systems. IEEE Transactions on Parallel and Distributed Systems 9, 1125–1136 (1998)
4. Zhang, Y., Ye, W., Irwin, M.J.: An alternative architecture for on-chip global interconnect: Segmented bus power modeling. In: The Thirty-Second Asilomar Conference on Signals, Systems and Computers, November 1–4 1998, vol. 2, pp. 1062–1065 (1998)
5. Liveris, N.D., Banerjee, P.: Power aware interface synthesis for bus-based soc designs. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Feburary 16–20, 2004, vol. 2, pp. 864–869 (2004)
6. Shirvaikar, M., Estevez, L.: Digital camera with jpeg, mpeg4, mp3 and 802.11 features. In: Workshop Presentation, Embedded Systems Conference, San Francisco, USA (2002)
7. Rim, M., Jain, R., Leone, R.D.: Optimal allocation and binding in high-level synthesis. In: Proceedings. 29th ACM/IEEE Design Automation Conference, 1992, June 8–12, 1992, pp. 120–123. IEEE Computer Society Press, Los Alamitos (1992)
8. Kim, S., Im, C., Ha, S.: Schedule-aware performance estimation of communication architecture for efficient design space exploration. In: First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (October 1–3, 2003)
9. Kim, S., Im, C., Ha, S.: Efficient exploration of on-chip bus architectures and memory allocation. In: CODES+ISSS '04. Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 248–253. ACM Press, New York (2004)
10. Lahiri, K., Raghunathan, A., Dey, S.: Design space exploration for optimizing on-chip communication architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23, 952–961 (2004)
11. Pandey, S., Glesner, M., Muhlhauser, M.: Performance aware on-chip communication syhthesis and optimization for shared multi-bus based architecture. In: ACM 17th Symposium on Integrated Circuits and Systems Design, ACM Press, New York (2005)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Company, New York (1979)
13. Dertouzos, M.L., Mok, A.K.: Multiprocessor online scheduling of hard-real-time tasks. IEEE Transactions on Software Engineering 15(12), 1497–1506 (1989)