Multi-layer bus minimization for SoC

Ya-Shu Chen a,*, Hsin-Liang Tsai b, Shi-Wu Lo c

a Department of Electrical Engineering, National Taiwan University of Science and Technology, No. 43, Keelung Rd. Sec. 4, Taipei 106, Taiwan, ROC
b Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
c Department of Computer Science and Information Engineering, National Chung-Cheng University, Taiwan

A R T I C L E   I N F O

Article history:
Received 13 October 2008
Received in revised form 23 July 2009
Accepted 23 July 2009
Available online xxxx

Keywords:
Multi-layer bus
System-on-chip
HW/SW codesign
Scheduling

A B S T R A C T

The deployment of multiple processing elements such as a microprocessor or a Digital Signal Processor in embedded systems often results in significant communication overheads. The challenge lies in resolving the communication cost minimization problem, while simultaneously satisfying the timing constraints of job executions. In this paper, we explore bus-layer minimization problems by first identifying factors that contribute to the NP-hardness of these problems. Existing proposed algorithms and NP-hard problems are then identified and elucidated. A simulated annealing algorithm is proposed and compared with heuristics-based algorithms to provide further insights for system designers. Lastly, a series of extensive simulations is carried out and a case study is presented to show comparisons among different approaches and workloads.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

Due to increasing market demand, many embedded systems will eventually run multiple applications on a system-on-chip platform. In particular, multiple processing elements (PE) (e.g., one or more microprocessors/DSP’s) are needed in such platforms, to meet the processing demands of a wide range of applications, from smart phones and mobile computing devices to HDTV. On such platforms, there is a need for computational jobs running on multiple processing elements to collaborate with each other to accomplish a task such as video encoding/decoding. This collaboration is typically achieved through significant amounts of data exchange between the processing elements, and thus, data communication can become a performance bottleneck for these systems. To resolve this issue, multi-bus or multi-layer architectures (like those based on the Advanced Micro-controller Bus Architecture (AMBA) (AMBA, 1999), have been proposed by various vendors. However, empirical designs and implementations of a bus or layer architecture are likely to result in resources being over-allocated. To resolve this problem, our work adopts the notion that an approach based on rigorous theoretical foundations needs to be taken, for instance, the minimization of the multi-layer bus for a system-on-chip (SoC) system.

In the past decade, successful research results have been achieved by researchers that have proposed to minimize data communication costs under various timing constraints over multi-bus architectures (Chou et al., 1995; Gupta and Micheli, 1993; Kambe et al., 2001; Lien et al., 2007; Ogawa et al., 2003; Seceleanu et al., 2005; Sheliga and Sha, 1995; Srinivasan et al., 2005). In particular, some of these approaches (Gupta and Micheli, 1993; Chou et al., 1995; Kambe et al., 2001; Ogawa et al., 2003) have tried to minimize the number of buses through either using simulations or empirical study with system-level EDA tools, while those in (Lien et al., 2007; Seceleanu et al., 2005, 1995; Srinivasan et al., 2005; van Meeuwen et al., 2001) have focused their study on bus partitioning by applying integer linear programming or heuristics-based scheduling algorithms.

Although multi-bus architectures provide effective solutions for minimizing communication costs, they suffer from inflexibility due to the fixed partitioning of PE’s at the hardware design stage (Seceleanu et al., 2005; Yang and Zaky, 1988). In order to increase the bus capacity and provide more flexibility in resource allocation, the multi-layer bus architecture (ARM, 2004) and a multi-channel Network-on-Chip (NoC) (Sanghun Lee and Lee, 2004) architecture have been proposed. However, for those techniques, the drawback is that increasing the number of bus layers or channels would imply an increase in the overhead of the cost or area, the power consumption and the complexity of SoC systems design (AHB, 2001; ARM, 2004; Caldari et al., 2003). For example, the gate count would grow exponentially in the multi-layer bus architecture, when the number of bus layers and the number of PE’s have been increased (AHB, 2001). Unfortunately, little work has been done to study the tradeoff between the resource utilization and the system performance of a multi-layer bus or a multi-channel NoC system.
This work is motivated by the need for applying a design methodology for bus-layer minimization problems in the multi-layer bus architecture. In contrast to existing simulation approaches (Ogawa et al., 2003) and empirical studies, we explore bus-layer minimization problems concerning different design parameters and different graph topologies. By identifying the factors that contribute to the NP-hardness of bus-layer minimization problems, the system designer could change program models or software implementations to resolve the communication problem with minimal cost. The main contribution of this paper is the recommendation of a suitable scheduler and the reconstruction guide for a communication specification of a system, such as a surveillance camera, a portable media player or a transaction terminal, so that bus-layer minimization problems can be resolved within tolerable time and cost. In this paper, a transaction terminal for EFTPOS payment is presented as a demonstration example and used to provide practical considerations for real-world system designs.

In this paper, we first present existing bus-layer minimization problems with efficient algorithms, such as those with unit execution time of a bus transaction (i.e., a data transmission activity between two PE’s). We then explore factors that contribute to the NP-hardness nature of many bus-layer minimization problems. Following this, a simulated annealing (SA) algorithm is presented to be used as comparison with heuristics-based algorithms to provide insight into system designs. A series of extensive simulation experiments has also been performed using different graph topologies, deadline settings, bus translation populations, etc.

The rest of this paper is organized as follows: Section 2 defines terminologies and formulates problems. Section 3 summarizes problem formulations with efficient solutions. Section 4 presents NP-hard problems and a simulated annealing (SA) approach. Section 5 presents the experimental study of different approaches. Section 6 concludes this paper.

2. Problem definition

In this paper, we are primarily concerned with the communication cost minimization problem present in the multi-layer bus architecture. Now, let us consider a real system such as a transaction terminal as an example. This transaction terminal consists of a 3-layer Advanced High-Performance Bus (AHB) and an Advanced Peripheral Bus (APB) under AMBA as shown in Fig. 1. The AHB supports, an MPU (ARM core), a Digital Signal Processor (DSP), and Direct Memory Access (DMA) devices. This bus is also a bridge to the lower bandwidth APB, where most of the peripheral devices, such as the UARTs and Timers, in the system are located. The topology of the multi-layer bus consists of a network of shared and dedicated communication channels connected to various hardware components. As shown in Fig. 1, the DSP, ARM core and DMA could access the Ethernet, printer and audio devices simultaneously under this 3-layer bus architecture.

This transaction terminal provides the EFTPOS payment system. There are various security algorithms, a visual display for instructional messages, a stereo codec MP3 for the voice indication, an Ethernet connection for online transactions, and a thermal printer to print account details. Fig. 2a represents the three task graphs pertaining to account print, message display, and voice indication on a transaction terminal. In the task graphs, each rectangle represents one function block and each arrow represents the data flow of a task. As shown in Fig. 2a, the steps involved in the voice indication task are to access data from NAND flash and memory, perform decoding functions with the DSP, and send voice data to the audio device in sequence.

Each task graph represents an application on the system, and the transaction on each task graph is denoted as a bus transaction graph. As shown in Fig. 2b, the bus transaction graph represents the data flow on the workload of Fig. 2a by a set of graphs $G = \{BTG_1, BTG_2, BTG_3\}$. A bus transaction graph $BTG_i$ consists of a set of bus transactions. Each bus transaction $BTi_j$ denotes a data transmission between a source processing element and a destination processing element. It is represented by a vertex of the bus transaction graph $BTG_i$. For example, $BT_{i2}$ in the figure is a data transaction which is sent from memory to the DSP of the voice indication task with execution time $e_{BT_{i2}} = 0.017$ ms. Each directed edge between the vertices denotes a precedence constraint between bus transactions. Moreover, to satisfy the timing constraint of the corresponding application, each bus transaction graph is given a deadline to complete all transactions of its graph. In this paper, we assume that each bus transaction graph forms a directed acyclic graph (DAG). Each bus transaction is non-preemptive, and it can request only one bus layer for data transmission every time.

A bus transaction $BTi_j$ is termed an immediate predecessor of another bus transaction $BTi_k$ if there is a directed edge from $BTi_j$ to $BTi_k$. $BTi_k$ is an immediate successor of $BTi_j$. $BTi_k$ cannot start until the completion of $BTi_j$. $BTi_j$ is termed a predecessor of another bus transaction $BTi_p$ if there is a path with one or more directed edges between $BTi_j$ and $BTi_p$. $BTi_j$ is a successor of $BTi_p$. As shown in Fig. 2b, $BT_{i2}$ is an immediate predecessor of $BT_{i4}$, and $BT_{i4}$ is a successor of $BT_{i1}$. The execution time of a bus transaction $BTi_j$, denoted as $e_{BTi_j}$, is the amount of time required to complete $BTi_j$. We assume that $e_{BTi_j} > 0$ for all transactions in this paper. The release time of $BTi_j$, denoted by $r_{BTi_j}$, is the time when the bus transaction becomes available for execution (under consideration of the precedence constraints). The start time of $BTi_j$, denoted as $s_{BTi_j}$, is the time when the bus transaction is scheduled for execution. Note that $s_{BTi_j} \geq r_{BTi_j}$. The response time of $BTi_j$, denoted as $Resp_{BTi_j}$, is the amount of time between $r_{BTi_j}$ and the completion time of $BTi_j$. The deadline of $BTi_j$, denoted as $d_{BTi_j}$, is the time by which $BTi_j$ must complete its execution. Let $p_{BTi_j}$ indicate the priority of $BTi_j$, where a smaller value denotes a higher priority. Additionally, we assume that the priority of a bus transaction, $BTi_j$, remains unchanged from $r_{BTi_j}$ to $d_{BTi_j}$. This is because if the priority of a bus transaction can be dynamically changed during that period, the hardware complexity of both the processing element and the dynamic-priority arbiter is dramatically increased.

The multi-layer bus minimization problem arises when it is necessary to derive the minimal number of bus layers required to meet the cost and performance constraints of SoC systems. The multi-layer bus minimization problem is defined as follows:

**Problem 1. Multi-layer bus minimization (MBM):** Given a set of bus transaction graphs $G = \{BTG_1, BTG_2, \ldots, BTG_n\}$ and a deadline for each graph, the problem is to minimize the number of bus layers.
required on a SoC system such that all the deadlines of the graphs in $G$ are satisfied.

MBM is a difficult problem, because the bus transaction scheduling problem in a multi-layer bus is NP-complete in the strong sense, even for all the graphs in $G$ having a common release time and a common deadline on a 2-layer bus system. It can be noted that for a bus transaction graph $G$, a schedule of $G$ is feasible only when all bus transactions meet the deadlines and follow the precedence constraints.

**Problem 2.** MBM with Common Release Time and Deadline (MBMRD): Given a set of bus transaction graphs $G$, a 2-layer bus on the system, and that all the graphs in $G$ have a common release time and a common deadline $D$, the MBMRD problem is to find a scheduling algorithm to minimize the schedule length such that all graphs meet the common deadline.

**Theorem 1.** MBMRD problem is NP-complete in the strong sense.

**Proof.** This theorem is demonstrated by reducing any instance of the 3-PARTITION problem to an instance of the MBMRD problem. The 3-PARTITION problem is defined as follows: Given a multiset $S$ of $3m$ positive integers, the total sum of all integers being $mB$ and each integer $i$, $B/4 < i < B/2$, can be partitioned into $m$ subsets $S_1, S_2, \ldots, S_m$ such that the sum of the integers in each subset is $B$. Each subset $S_i$ is forced to consist of exactly 3 elements by the constraint $B/4 < i < B/2$. The 3-PARTITION problem is NP-complete in the strong sense (Garey and Johnson, 1979).
For the multiset $S$ which is an instance of the 3-PARTITION problem, this study constructs a set of bus transaction graphs $G$ for the instance of the MBMRD problem from $S$ as follows: Each integer in $S$ forms a bus transaction $BT$ in $G$, and the execution time of each bus transaction is equal to the value of each integer. Consequently, there are $3m$ bus transaction graphs (i.e., $BTG_{1},...,BTG_{3m}$) in $G$ with independent bus transactions, and the execution time of each bus transaction is within $B/4$ to $B/2$ as shown in Fig. 3a (i.e., $BT_{1,1},...,BT_{3m,1}$). Then, a bus transaction graph $BTG_{3m+1}$ with $3m$ bus transactions is added into $G$. The precedence of bus transactions of $BTG_{3m+1}$ is shown in Fig. 3a (i.e., $BTG_{1,1},...,BTG_{3m+1}$). The execution time of each bus transaction $BTG_{3m+1,j}$ is $B$, where $j = 1,4,7,...,(3m - 2)$, and the execution time of any other bus transaction in $BTG_{3m+1}$ is 1. The total execution time of the $BTG_{3m+1}$ is $mB + 2m$, and the total execution time of other bus transaction graphs in $G$ is $mB$ (i.e., $BT_{1,1},...,BT_{3m,1}$) which is the total sum of all integers in $S$ in the 3-PARTITION problem.

Assume that there is an algorithm that can solve any problem instance of the MBMRD. Specifically, given $G$ as mentioned above, a 2-layer bus on the system, and that all the graphs in $G$ have a common release time 0 with a common deadline $D = mB + m$, the algorithm finds a schedule such that all graphs meet the common deadline. If there is a feasible schedule for $G$, the corresponding instance of the 3-PARTITION problem with $S$ and $B$ is solved (e.g., one possible schedule is as shown in Fig. 3b). This is because the common deadline of $G$ is $mB + m$, whereby each bus transaction in $BTG_{3m+1}$ with execution time $B$ must be scheduled just after two bus transactions in $BTG_{3m+1}$ with each execution time of 1, and the two corresponding immediate predecessors must be scheduled in different bus layers. Therefore, the other $3m$ bus transaction graphs can only be scheduled in $m$ slots with duration $B$ in the idle layer (e.g., Layer 2 as shown in Fig. 3b). In other words, if there is a feasible schedule, $3m$ bus transaction graphs can be partitioned into $m$ bus transaction graph subsets such that the sum of the execution time of the bus transactions in each subset is $B$. Thus, the corresponding instance of 3-PARTITION with $S$ and $B$ is solved. The 3-PARTITION problem is NP-complete in the strong sense according to the proof in (Garey and Johnson, 1979), and thus MBMRD is also NP-complete in the strong sense.

**Corollary 1.** MBM problem is NP-complete in the strong sense.

**Proof.** This corollary can be proved following the proof of Theorem 1. □

According to Theorem 1, there is no polynomial time algorithm to find a feasible schedule on the multi-layer bus architecture. Corollary 1 also shows that there is no optimal method to find the minimal number of bus layers of the MBM problem even with pseudo-polynomial time. In later sections, the complexity of the subproblems of the MBM problem is explored in two parts: P-time solvable subproblems and NP-complete subproblems. We then present the systematic analysis methodology and propose a near-optimal strategy based on simulated annealing (SA) to resolve the MBM problem.

### 3. P-time solvable problems

In this section, we discuss the polynomial solvability of the reduced multi-layer bus minimization (MBM) problem, to provide designers with ideas for redesigning the workload or for searching...
the suboptimal solution by the use of heuristics. The MBM problem can be relaxed into two subproblems: (1) The first subproblem is to determine the number of bus layers that should be allocated, provided that feasible schedules exist. A feasible schedule of a set of bus transaction graphs $G$ exists if all bus transactions on the schedule could meet deadlines without violating precedence constraints. (2) The second subproblem is how to determine if there is a feasible schedule, when the set of bus transaction graphs $G$ and number of bus layers are known.

As the MBM problem is NP-complete in the strong sense, this problem can be explored in two steps by providing additional information (e.g., a specific scheduler or cost budget). In particular, the MBM problem is reduced to the first subproblem, when the schedule algorithm for system is determined or the scheduling policy of the arbiter is fixed. Similarly, the MBM problem is reduced to the second subproblem, when the number of layers is bounded by a cost budget.

We now propose optimal algorithms for these two subproblems under some special cases, and show a pseudo-polynomial time algorithm for the MBM problem without considering precedence constraints. Designers could extend the algorithms, by reconstructing the workload with heuristics for suboptimal solutions using the provided information (e.g., a specific scheduler or cost budget). The properties of the MBM problem for developing near-optimal solutions are explored in the next section.

To address the first subproblem, we propose the Maximum Overlapped Bus Transactions algorithm with time complexity $O(n^2)$ to determine the number of bus layers that should be allocated, provided that a feasible schedule exists, where $n$ is the number of bus transactions. The basic idea of this algorithm is that the required number of bus layers is equal to the maximum number of bus transactions that have to be executed simultaneously within the timing constraints. The pseudo code of the Maximum Overlapped Bus Transactions algorithm is shown in Algorithm 1. Initially, the number of bus layers is set to 1 and the current time is set to 0 (Steps 1–2). At each iteration, we find the ready bus transaction $BT_i$ that has the earliest start time and no uncompleted predecessors. Then, we set the current time $t$ to $s_{BT_i}$, and assign $BT_i$ to a free bus layer $BUS_{i}$ (Steps 3–17). The number of bus layers is increased when a bus transaction is ready but no free bus layers can be assigned (Steps 7–11).

**Corollary 2.** When a schedule of bus transactions of a set of bus transaction graphs $G$ is given, Algorithm **Maximum Overlapped Bus Transactions** finds the optimal solution for the MBM problem.

**Proof.** Since the schedule of bus transactions in $G$ is given, the latest start time of each bus transaction is known. Therefore, the minimal number of bus layers in a multi-layer bus system is the number of bus transactions which have to be executed simultaneously within the timing constraints.

From the above discussions, it can be noted that the first subproblem of the MBM can be solved in polynomial time, and the optimality of the solution is provided by a feasible schedule of $G$.

The schedule feasibility of the second subproblem, when $G$ and the number of bus layers are given can be determined as follows: Assume that each bus transaction can only be assigned on a single bus layer for data transmission. For a given number of bus layers, we found that an optimal schedule of all bus transactions in $G$ could be derived by applying the level strategy (Hu, 1961), when the execution times of all bus transactions are equal and each bus transaction graph is a tree. The strategy is as follows: First, each bus transaction is assigned a label equal to the level of the transaction on the bus transaction graph. Specifically, the level of a bus transaction with immediate successors is one plus the maximum level of its immediate successors on the graph, whereas the level of a bus transaction with no immediate successor is one. The label of each bus transaction corresponds to the priority of the bus transaction, where a larger value denotes a higher priority. After each bus transaction is assigned a label, all bus transactions are then scheduled by priority under the precedence constraint, when there is a free bus layer.

**Corollary 3.** When all bus transactions have unit execution times and each bus transaction graph in a given set of bus transaction graphs $G$ is a tree, the level strategy (Hu, 1961) finds the optimal schedule of $G$ under the given number of bus layers, if any feasible solution exists.

**Proof.** Each bus transaction can be viewed as a process. The correctness of the corollary directly follows the properties of the level strategy (Hu, 1961).

From the study of this subproblem, we discovered that the MBM problem is difficult because it is hard to find an optimal schedule when the bus transactions in $G$ have precedence constraints. Thus, let us consider the situation when there is no precedence constraint and all bus transactions share the same deadline. For real system applications, the execution times of bus transactions in $G$ might be the same when these bus transactions request an identical destination processing element. Assume that there are $k$ types of processing elements on the system, and subsequently, $k$ different execution times of bus transactions in $G$. Under this assumption, we can have a pseudo-polynomial time dynamic programming algorithm for finding an optimal solution for the MBM problem.

**Corollary 4.** The MBM problem could be solved in pseudo-polynomial time, when there are no precedence constraints of all bus transactions, $k$ different execution times of bus transactions, and a common deadline of all bus transaction graphs in a given set of bus transaction graphs $G$.

**Proof.** Assume that there are $n$ bus transactions without precedence constraints in the given set of bus transaction graphs $G$, and that there is a common deadline for all bus transaction graphs. By our definition, the bus transactions in $G$ could be represented by a $k$-tuple $G = (n_1, n_2, \ldots, n_k)$, where $n_i$ is the number of bus transactions with $i$th type execution time, and $\sum n_i = n$. $G$ could be partitioned into several subsets as $G = (p_1, p_2, \ldots, p_k)$, where $0 \leq p_i \leq n_i$. Some subset of $G$ could be derived, such that all

---

**Please cite this article in press as:** Chen, Y.-S., et al. Multi-layer bus minimization for SoC. J. Syst. Software (2009), doi:10.1016/j.jss.2009.07.041
bus transactions in the subsets could meet the common deadline in one bus layer. The subsets are denoted by \( Q \). A dynamic programming solution with a time complexity of \( O(m^2) \) to resolve the MBM problem without the precedence constraint is as follows:

\[
m = \text{Layer}(n_1, n_2, \ldots, n_k)
\]

For a given \( G = (n_1, n_2, \ldots, n_k) \), \( \sum_{j=1}^{k} n_j = n \)

Let:

\[
G' = \{(p_1, p_2, \ldots, p_k) | 0 \leq p_i \leq n_j, j = 1, \ldots, k\}
\]

\[
Q = \{(q_1, q_2, \ldots, q_k) \in G' | \text{Layer}(q_1, q_2, \ldots, q_k) = 1\}
\]

\[
\text{Layer}(p_1, p_2, \ldots, p_k) = 1 + \min_{q \in Q} \text{Layer}(p_1 - q_1, p_2 - q_2, \ldots, p_k - q_k)
\]

where \( m \) is the minimal number of bus layers. \( \square \)

When the bus transaction is assigned to a layer, the source processing element and the destination processing element of the bus transaction have to be connected to the corresponding layer. Thus, the above algorithms might result in all processing elements being connected to all layers. The connection minimization problem in each layer is also important and difficult. Nevertheless, the connection minimization problem is beyond the scope of this paper, and the reader is referred to (van Meeuwen et al., 2001) for further information. Moreover, to enable practical implementation on a multi-layer bus system, and to maintain backward compatibility, the assignments of processing element connections to layers are primarily decided by the speeds or the functions of processing elements. The difficulty of the MBM problem with this issue will be explored in the next section.

From our exploration of the subproblems, we can conclude that there are optimal algorithms with polynomial time that can solve the MBM problem with the precondition that, only when there is no precedence constraint or the execution time of each bus transactions is equal. In particular, to derive a feasible solution for MBM, a designer could extend the aforementioned algorithms, by removing the precedence constraints of bus transactions of applications with heuristics, or re-partitioning the data size such that the execution time of each bus transaction is equal. For example, consider the transaction terminal for EFTPOS payment as described in Section 2. If the bus arbiter in this system (such as a Time Division Multiple Access bus arbiter) is chosen by a prior system (e.g., a transaction terminal for credit-card payment), then the required number of bus layers could be determined by applying Algorithm 1. Conversely, the number of bus layers in this system might be determined by the cost budget or an existing multi-layer bus development board, such as Versatile (ARM, 2006) or AT91CAP (Atmel, 2009). In that case, scheduler selection can be resolved by reconstructing bus transaction graphs (e.g., repartitioning functions into different hardware components), according to Corollary 3 and 4.

4. NP-complete problems

In this section, we show that other subproblems of Multi-layer Bus Minimization (MBM) are still NP-complete in the strong sense when applied practicality. Based on the results obtained, we further explore the complexity and properties of the MBM problem. We then present a simulated annealing (SA) algorithm to solve the MBM problem and compare it with heuristics-based algorithms in later sections.

4.1. Problem analysis

In the design flow of an embedded system, processing elements (PE) may be assigned to a dedicated bus layer for a specific application based on prior experiments or system design. For example, consider the transaction terminal as described in Section 2, the LCD controller in this system is typically assigned to the second bus layer as shown in Fig. 1. In this manner, there are gate count or area savings because the number of components such as multiplexers and decoders in the interconnect matrix is reduced (ARM, 2004). However, the MBM problem remains complex even when each bus transaction has been partitioned into a dedicated bus layer.

**Corollary 5.** Let each bus transaction in a given set of bus transaction graphs \( G \) be partitioned into a dedicated bus layer before scheduling. The multi-layer bus minimization with dedicated bus (MBMDB) problem is NP-complete in the strong sense.

**Proof.** This theorem is demonstrated by reducing any instance of the 3-PARTITION problem to an instance of the MBMDB problem. For the multiset \( S \) which is an instance of the 3-PARTITION problem, this study constructs a set of bus transaction graphs \( G \) for the instance of the MBMDB problem from \( S \) as follows: Each integer in \( S \) forms a bus transaction \( BT \) in \( G \), and the execution time of each bus transaction is equal to the value of each integer. Consequently, there are \( 3m \) bus transaction graphs in \( G \) with independent bus transactions, and the execution time of each bus transaction is inclusively between \( B/4 \) and \( B/2 \). Then, we add a bus transaction graph \( BT_{Gm+1} \) with \( 2m \) bus transactions into \( G \) with precedence constraint, and \( BT_{Gm+1} \) forms a chain as shown in Fig. 4(a). The execution time of each bus transaction \( BT_{Gm+1,j} \), where \( j = 1, 3, 5, \ldots, 2m-1 \), in the chain is \( B \) and, the execution time of other bus transactions in the chain is 1.

![Fig. 4. NP-complete proof of the multi-layer bus minimization with dedicated bus (MBMDB) problem.](image-url)

Suppose that there is an algorithm that can solve any problem instance of the MBM MDB. Hence, given G with a common deadline $D = mb + m$ as aforementioned, we assumed that there is a 2-layer bus on the system, all bus transactions with execution time $B$ are partitioned into Layer 1 and that the other bus transactions are partitioned into Layer 2. Under these assumptions, the algorithm finds a schedule such that all graphs meet the common deadline. If there is a feasible schedule for $G$, only the corresponding instance of the 3-PARTITION with $S$ and $B$ is solved; one possible schedule is shown in Fig. 4b. Following this, each bus transaction in $BTG_{3m-1}$ with execution time $B$ must be scheduled just after the immediate predecessor in $BTG_{3m-1}$ with execution time 1 because the common deadline of $G$ is $mb + m$. Therefore, the other 3m bus transaction graphs can only be scheduled in $m$ slots with duration $B$ in Layer 2. In other words, if there is a feasible schedule, 3m bus transaction graphs can be partitioned into $m$ bus transaction graph subsets such that the sum of the execution times in each subset is $B$. Thus, the corresponding instance of the 3-PARTITION problem with $S$ and $B$ is solved. The 3-PARTITION problem is NP-complete in the strong sense by the proof shown in (Garey and Johnson, 1979), and hence MBMDB is also NP-complete in the strong sense.

Now, consider the other practical implementation issue on an event driven system. In such a system, the task will be executed by some special event. Therefore, the release time of each bus transaction graph might not be the same. When the given release times and deadlines are not common for all bus transaction graphs, the MBM problem is still NP-complete.

**Corollary 6.** Let each bus transaction graph in a given set of bus transaction graphs $G$ have a respective release time and a deadline. The Multi-layer Bus Minimization with Respective Release Times and Deadlines (MBMRTD) problem is NP-complete in the strong sense even when the number of bus layers is 1.

**Proof.** This corollary could be proven by transformation from the Sequencing with Release Times and Deadlines (SRTD) problem. The SRTD problem is defined as follows: Given a set of tasks $T$ where, for each task $t$ in $T$, a length $l(t) \in \mathbb{Z}^+$, a release time $r(t) \in \mathbb{Z}^+$, and a deadline $d(t) \in \mathbb{Z}^+$, is there one processor schedule for $T$ that satisfies the release time constraints and meets all the deadlines?

We can view each task as a bus transaction so that, the SRTD problem becomes a MBMRTD problem. The SRTD problem is NP-complete in the strong sense with reference to the proof in (Garey and Johnson, 1979), and thus MBMRTD is also NP-complete in the strong sense.

Now, consider the data transmission issue in a real system. In an embedded system, each bus transaction sends data from a source processing element to a destination processing element. As a result, the Processing Element Contention issue occurs, and the system might fail (i.e., when one bus transaction requests to send to a destination processing element which is receiving data from other bus transactions.) Therefore, for a real system, processing element contention needs to be prevented. However, the MBM problem with Processing Element Contention Prevention is still NP-complete in the strong sense, even when the execution time of each bus transaction is equal.

**Corollary 7.** The multi-layer bus minimization with processing element contention prevention (MBMCP) problem is NP-complete in the strong sense even when the execution time of each bus transaction is equal.

**Proof.** This corollary is demonstrated by reducing any instance of the Resource Constrained Scheduling problem to an instance of the MBMCP problem. The Resource Constrained Scheduling problem is defined as follows: Given a set of tasks, each having length $l(t) = 1$, number $m \in \mathbb{Z}^+$ of processors, number $r \in \mathbb{Z}^+$ of resources, resource bounds $B_i, 1 \leq i \leq r$, resource requirement $R_i(t), 0 \leq R_i(t) \leq B_i$, for each task $t$ and resource $i$, and an overall deadline $D \in \mathbb{Z}^+$, is there an $m$-processor schedule $\sigma$ for $T$ such that all tasks meet the overall deadline $D$ and obey the resource constraints? The Resource Constrained Scheduling problem is NP-complete in the strong sense (Garey and Johnson, 1979).

For the set of tasks $T$ that is an instance of the Resource Constrained Scheduling (RCS) problem, this study constructs a set of bus transaction graphs $G$ for the instance of the MBMCP problem from $T$ as follows: Each task in $T$ is taken as a bus transaction graph in $G$, and the execution time of each bus transaction is equal to the execution length $l(t)$ of each task. The number of resources refers to the number of processing elements, while the resource requirement of each task $R_i(t)$ is the processing time of each bus transaction requesting a processing element. Apparently, there is an $m$-layer bus schedule for MBMCP if and only if there is an $m$-processor schedule for RCS that meets the common deadline $D$ and obeys the resource constraints. The RCS problem is NP-complete in the strong sense by the proof shown in (Garey and Johnson, 1979), and hence MBMCP is also NP-complete in the strong sense.

The strong NP-hardness results presented in this section suggest that it is difficult to design efficient algorithms to optimally solve the multi-layer bus minimization problem. In the next section, we present a strategy based on simulated annealing (SA) to resolve the MBM problem and this strategy is compared with heuristics-based algorithms to provide further insight into system design.

4.2. A simulated annealing approach

The Simulated Annealing (SA) algorithm is presented for two purposes: (1) to provide a basic solution for a designer dealing with the MBM problem, and (2) to evaluate the effect of different workloads in the analysis of system bottlenecks. For the second purpose, extensive experiments are performed using the SA algorithm and elaborated in a later section. In this section, the SA algorithm is presented as a solution for minimizing the number of bus layers in a system. We then present the neighbor state definition, the energy function, and the refinement strategies. It needs to be noted that the framework of SA follows the design methodology presented in (Michalewicz and Fogel, 2000).

4.2.1. An initial state, state neighborhood and a feasibility test

In this work, we set the initial state for the SA as the value derived by the MAXIMUM OVERLAPPED BUS TRANSACTIONS SCHEDULING algorithm in Section 3. The release time of each bus transaction on a set of bus transaction graphs $G$ is taken as the start time of each bus transaction in the algorithm. The rationale behind the initial state setup is to speed up the SA process, instead of randomly generating an initial state.

At the derivation of a good system design in the multi-layer bus architecture, it is possible to have different bus transaction priority assignments and partitions. A state is a priority assignment and partition for all bus transactions. Two states are in the neighborhood of each other if and only if one can be derived from the other by one of the following methods: (1) the merging of partitions for two bus transactions; (2) the splitting of partitions for two bus transactions; (3) the swapping of the priorities or partitions for two bus transactions; or (4) the moving of the bus transaction to the other partition.

At each iteration of the SA algorithm, the feasibility test needs to be performed on the current priority assignment and partition of
bus transactions. The steps of the feasibility test are as follows: Firstly, we set the deadlines of all bus transactions based on the well-known deadline revision method (Mok, 1984). It can be noted that the deadline setting of bus transactions satisfies the precedence constraints for the set of bus transaction graphs $G$ and follows the “As-Late-As-Possible” policy. Then, the start time and completion time of each bus transaction in the same partition can be derived by the priorities of bus transactions, because the bus transactions under consideration are non-preemptive. The partition of each solution represents a bus layer in a multi-layer bus system, and so the current time of each partition is then advanced to the completion time (i.e., $t_{BUS_k} + e_{BT_i,j}$) of the bus transaction. Each bus transaction in each partition is then checked. If no transaction misses its deadline, the test finishes and reports a success. The time complexity of this algorithm is $O(n)$, where $n$ is the number of bus transactions.

4.2.2. The energy function

The energy function (i.e., the penalty function or the cost function) can be defined by two major factors in SA: (1) the deadline satisfaction of bus transactions, and (2) the slack time of each partition. Given a solution, the feasibility of a bus transaction could be derived in a similar fashion as the feasibility test. The energy functions $E_{BT_i,j}$, $E_{BUS_k}$ of $BT_i,j$ and $BUS_k$, respectively, with a given solution are defined as follows:

$$
E_{BT_i,j} = -e_{BT_i,j} \quad \text{if } BT_i,j \text{ meets its deadline}
$$

$$
E_{BT_i,j} = f(e_{BT_i,j}) \quad \text{if } BT_i,j \text{ misses its deadline}
$$

$$
E_{BUS_k} = \text{Hyper} - \text{period of } G + \sum_{BT_i,j \in BUS_k} E_{BT_i,j}
$$

where $f(x)$ is any increasing function of $x$ (e.g., $f(x) = x^2$).

The smaller the energy function value of a bus transaction is, the better the schedule of the bus transaction. The energy function serves to impose a high penalty on a bus transaction if the bus transaction misses its deadline. The slack time of each partition (i.e., bus layer) also serves to exaggerate any potential penalty for a schedule. The value $E_{total}$ of the energy function of a solution is the sum of the energy function values of each partition (i.e., bus layer) under consideration, as shown in the following equation:

$$
E_{total} = \sum_{k=1}^{m} E_{BUS_k}
$$

where $m$ is the number of partitions (i.e., bus layers) in the current solution.

4.2.3. Refinement strategies

The four refinement strategies in SA (i.e., Merge($S$), Split($S$), Swap($S$), and Move($S$)) are used to derive a new state by reassigning bus transactions priorities and partitions. Each refinement strategy reassigns the priorities and partitions of one or two bus transactions once and selects bus transactions randomly. In addition, when there are two bus transactions with the same priority on the same partition, all bus transactions with priorities not higher than the priority of the bus transaction with a new assignment are readjusted by an increment of one.

Merge($S$) stands for the “merging” of two bus transactions into one partition, while the priorities of the two bus transactions remain as they are. The rationale is to seek a local optimum in the reduction of the number of partitions (i.e., bus layers). To seek a global optimal solution, a Split($S$) is used for “splitting” two bus transactions on the same partition into different partitions by adding a new partition, while the priority of the bus transaction which is in the new partition is given the highest priority of the original partition. Note that the priority of the bus transaction that is in the original partition stays the same.

Swap($S$) stands for “swapping” of the priorities of two bus transactions. When two bus transactions are on different partitions, the swapping strategy also swaps the partition of these two bus transactions. The swap strategy serves to randomize the priority and partition assignments for the bus transactions. For improving the feasibility opportunity of the bus transaction, Move($S$) is used for “moving” one bus transaction to a randomly selected partition, and the priority of the bus transaction is assigned as the highest priority of the selected partition.

5. Performance evaluation

5.1. Experimental setup and performance metrics

The purpose of this section is to evaluate the proposed SA algorithm and provide insights to the redesigning of the workloads. The exhaustive search has been simulated and used as a comparison. The optimal multi-layer bus systems, in which all bus transactions meet deadlines and the number of bus layers is minimized, can be obtained through an exhaustive search.

Other evaluated algorithms include the neighborhood search (NS), random search (RS), the First Fit with Earliest Deadline First (EDF) algorithm (Masmur et al., 2007) and First Fit with Critical Path Method (CPM) algorithm (McCreary et al., 1996). The neighborhood search (NS) method and random search (RS) method use the same initial state and refinement strategies as our SA algorithm. The NS method finds the local optima and does not allow to progress to a worse solution, i.e., a higher energy value for the new schedule of bus transactions, and the RS method finds the global optimal solution by randomly picking a solution (Michalewicz and Fogel, 2000). The First Fit heuristic assigns each bus transaction to the first available bus layer that can guarantee their correct transmission, i.e., without missing the deadline. The bus layers are verified in order of increasing index. If there is no allocated bus layer that guarantees the feasibility of the bus transaction, a new bus layer will be allocated. This process is repeated until every bus transaction is assigned to a corresponding bus layer.

Four performance metrics are considered in this paper, which include: the minimal total number of bus layers ratio, the running time, the average number of bus layers per system and the normalized bus layer. The minimal total number of bus layers ratio (MTNBR), is defined as the ratio of the minimal total number of bus layers in a multi-layer bus system with a given set of bus transaction graphs under SA to that under the exhaustive search. $MTNB_{SA}$ and $MTNB_{Exhaustive}$ denotes the minimal total number of bus layers in a multi-layer bus system with a given set of bus transaction graphs under SA and under exhaustive search, respectively. The minimal total number of bus layers ratio (MTNBR) is $MTNB_{SA}/MTNB_{Exhaustive}$. The running time is the execution time of an evaluated algorithm on a notebook running Windows XP with a 1.73 GHz Intel Pentium M processor and 1 GB RAM. The SA algorithm and the exhaustive search are evaluated in terms of their running time. The average number of bus layers per system (ANBPS) (i.e., the average number of bus layers produced by each experiment result of the evaluated algorithm) is used to show the convergence speed of the SA algorithm with 100 and 500 bus transactions. Let $X$ and $Y$ denote the amount of bus layers of the algorithm under performance evaluation and that of $EBT_i$, respectively. The normalized bus layer of the evaluated algorithm is defined as $X/Y$.

To generate workload in the experiments, “Task Graph for Free” (TGFF) (Dick et al., 1998) is used to create descriptions of a task graph instance, which includes attributes for processors,
communication resources, tasks, and inter-task communication. Currently, the number of bus layers in a SoC system can be up to a maximum of eight and the average number of bus transactions was 100 in the current system (AHB, 2001). To evaluate the scalability of the algorithms, 100–1000 bus transaction graphs are generated for each set of experiments, and these results (e.g., the minimal total number of bus layers ratio, the running time and the number of bus layers per system) are separately averaged. SA is evaluated for a set of bus transaction graphs with the number of bus transactions ranging from 100 to 1000, and each graph has 1–50 out-degrees. In the experiments, the execution time of each bus transaction being executed on a processing element is derived by applying a uniform distribution function from 10 to 400 cycles (Eles et al., 1998; Yen and Wolf, 1998). The deadline of each bus transaction graph is set to 2–5 times the execution time of all bus transactions in this graph, and the release time of each bus transaction graph is set as zero.

5.2. Experimental results

In this section, we present the results from the experimentation on different workloads by comparing our SA algorithm with (1) exhaustive, random search and neighborhood search in terms of search time and (2) well-known EDF and CPM algorithms in terms of the number of bus layers. Practical implementation issues have also been considered and thus, we also conduct experiments for the MBM problem with processing element contention prevention.

5.2.1. Exhaustive, neighborhood and random search

Fig. 5 illustrates the running time and minimal total number of bus layers ratio (MTNBR) of the proposed SA algorithm and exhaustive search. From Fig. 5a, it can be noted that the running time of the SA algorithm is relatively shorter than that of the exhaustive search, and the running time of the exhaustive search grew exponentially with the number of bus transactions. Fig. 5b illustrates the minimal total number of bus layers ratio of the SA algorithm as compared to the exhaustive search. As shown in Fig. 5b, the minimal total number of bus layers of the SA algorithm is the same as that of the exhaustive search, when SA runs 100 iterations. When the SA algorithm runs less iterations, the result is slightly worse than the final solution of exhaustive search (i.e., the optimal solution), and the astringency of SA is shown in later figures.

Fig. 6 shows the comparison between our SA algorithm, the random search (RS) method and the neighborhood search (NS) method. The horizontal axis represents the number of iterations in each algorithm, while the vertical axis represents the average number of bus layers per system (ANBPS). The RS method produces the optimal solution, if there is sufficient running time to search whole solutions randomly. As shown in Fig. 6a, the final solution of the SA algorithm is close to the final solution of the RS method and better than that of the NS method after a prolonged run time. This is because the NS method has the tendency to be trapped at the local optima and our SA algorithm can deviate from the local optima by allowing undesired solutions. It can be noted that when the running time is not sufficient, the solution of NS might be better than RS as shown in Fig. 6b. It is because the RS method spends the most running time searching for whole solutions randomly, and the NS method spends less running time due to its use of the greedy policy that ensures it improves the solution at earlier iterations. The comparisons of the convergence speed of the SA algorithm, RS method and NS method under different workloads are also shown in Fig. 6. The average number of bus layers decreases with the number of iterations in the SA algorithm, and the average number of bus layers increases with the number of bus transactions in the given workload. We observe that a larger number of bus transactions required more time to converge.

5.2.2. EDF and CPM

A real system example of a transaction terminal for EFTPOS payment as shown in Fig. 2 was tested. The obtained experimental results showed that both EDF and CPM algorithms need two bus layers for this terminal, whereas SA only needs one bus layer for the same workload. The reason is because both EDF and CPM tend to schedule the ready bus transactions with the shortest period first.

To determine more properties for SA, we also performed evaluation using extensive synthetic bus transaction graph sets with results shown in Fig. 7.

Fig. 7 illustrates the difference between SA, EDF and CPM on partial order graphs for nodes without precedence constraints and chains. The horizontal axis represents the number of graphs, while the vertical axis represents the normalized bus layer. The numbers of graphs are 10, 50 and 100 whereas the average numbers of requested bus layers are 4, 18, and 38, respectively. As shown in Fig. 7, the performance of CPM is similar to EDF, and this is because bus transactions of such graphs could be easily ordered sequentially by deadline revision (Mok, 1984). In addition, the normalized bus layer of SA is decreased when the number of graphs is increased.

Fig. 8a and b show the comparison between our SA, EDF and CPM on directed acyclic graphs with arbitrary and common deadlines, respectively. The horizontal axis represents the number of out-degrees of each graph while, the vertical axis represents the normalized bus layer. It can be noted that if the number of out-degrees are 10, 30 and 50, the average numbers of requested bus layers are 5, 30, and 48, respectively.

Comparing the results shown in Figs. 7 and 8, we observe that the normalized bus layer of SA and CPM decreases when the number of out-degrees increases. The reason is because the parallelism of bus transactions increases when the number of out-degrees of a bus transaction graph increases. We also note that the out-degrees
of a node without the precedence constraint and of a chain are 0 and 1, respectively. In particular, as shown in Fig. 8b, SA significantly outperforms EDF as the deadline of each graph is the same. The chain and directed acyclic graphs of SA with different depths have also been evaluated, and the results show that the depth of the graphs had little effects on the normalized bus layer of SA.

Fig. 9 illustrates the comparison of results for the average running time between different algorithms. The horizontal axis represents the running time, while the vertical axis represents the number of bus transactions. In comparison, the EDF and CPM spend less running time as the greedy strategy is only considered at each stage, while for the SA algorithm, the running time increases with the number of bus transactions. All methods had a longer running time with increasing workloads.

5.2.3. Processing element contention prevention

In this section, we further extend all algorithms to solve the MBM problem with processing element contention. In this problem, any two bus transactions could not send to the same processing element simultaneously. As explained in Section 4, this problem is also NP-complete. This implies that the function execution order of each processing element might need to be changed. In other words, an optimal solution of the system would need to consider the execution and communication problem at the same time.

In this paper, we have focused on the communication problem and thus, the scheduling policy on each processing element execution is first-in-first-out in our experiments. Each evaluated
algorithm is extended by adding a processing element contention checking function. In other words, a bus transaction \( BT_{ij} \) is scheduled only when all of \( BT_{ij}/s \) predecessors are completed and that the destination processing element of \( BT_{ij} \) is free. Specifically, a destination processing element is free only when there is no current data transmission by a bus transaction. We tested the revised algorithms on the length of the deadline of each bus transaction graph and the number of processing element types under the given 10 chains with 100 bus transactions.

As shown in Table 1, the deadline of each bus transaction graph \( BT_{G} \) in \( G \) is set to 2–5 times of the execution time of all bus transactions in this graph. The experimental results demonstrate that the bus layers of all algorithms decrease when the deadlines of the graphs increase. Moreover, when the timing constraint became stricter, some cases failed when greedy algorithms, such as EDF and CPM are used. This has been explained in section 5.2.2.

Table 2 illustrates the results obtained when the number of processing element (PE) types in \( G \) is set from 11 to 15. The results show that the bus layers of all algorithms have not decreased or increased when the number of processing element types increased. The reason is because the execution time of each bus transaction and precedence constraints vary, and hence, there is no semantic relationship between the number of processing elements and the processing element content. Experiments have also been conducted to evaluate the transaction terminal as shown in Fig. 2 with 7 types of processing elements. Results from experiments show that EDF and CPM algorithms could not find a feasible schedule for the terminal, and that SA only needs one bus layer for the same workload.

### 6. Conclusion

In this paper, we have proven that the multi-layer bus minimization problem with precedence constraints to be NP-complete in the strong sense, when the timing constraints of applications must be satisfied. We have also elaborated on the complexity of this problem with more constraints and practical implementation considerations while exploring factors that contribute to the NP-hardness nature of many bus-layer minimization problems. We have also presented a simulated annealing (SA) algorithm and compared it with heuristics-based algorithms to provide further insight into system design. The experimental results highlighted the properties of the evaluated algorithms under different workloads and practical implementation issues. A case study for a transaction terminal has also been presented and used to provide practical considerations for real system designs. For more complicated systems, such as surveillance cameras, designers can still apply these concepts as shown in Section 2) to model the communication problem on the multiple processing elements and multi-layer bus platform. In particular, the designer can determine the minimal number of bus layers, choose a suitable scheduler, and/or reconstruct the communication specification for this system in accordance to the above stated corollaries, theorems and algorithms (as shown in Sections 3 and 4).

For future work, we aim to extend the analysis to propose an integrated scheduling methodology that manages both bus transactions and tasks execution on each processing element.

### Acknowledgement

This work is supported by a grant from the NSC program NSC 97-2221-E-011-044-MY2.

### References


1 A surveillance camera designer could model all applications, with timing constraints under given hardware components, as task graphs and bus transaction graphs.


Ya-Shu Chen joined Department of Electrical Engineering, National Taiwan University of Science and Technology, at August 2007. She currently serves as an Assistant Professor. Ya-Shu Chen earned her B.S. degree in computer information and science at National Chiao-Tung University in 2001. Then, she studied in Department of Computer Science and Information Engineering, National Taiwan University, and was supervised by Prof. Tei-Wei Kuo. She successfully defended her master thesis and doctoral dissertation at 2003 and 2007, respectively. Her research interest includes operating systems, embedded systems, and hardware/software co-design.

Hsin-Liang Tsai joined XAC Automation Corporation, at June 2005. He currently serves as a Section Manager. Hsin-Liang Tsai earned his B.S. degree in Electronic Engineering at Southern Taiwan University in 1999. After working for five years, he studied in Department of Computer Science and Information Engineering, National Taiwan University, and was supervised by Prof. Tei-Wei Kuo. He successfully defended his master thesis at 2008. His research interest includes real-time systems, embedded systems, and hardware/software co-design.

Shi-Wu Lo received the B.S.C.S, M.S.C.S and Ph.D degrees from and Yuan-Ze University, National Chung-Cheng University and National Taiwan University, respectively. He is currently an Assistant Professor at the Department of Computer Science and Information Engineering, National Chung-Cheng University. Lo has done various research work on system design and development, including virtual platform, flash memory management and QoS support SMT processor system design.