

Bounding DMA Interference on Hard-Real-Time Embedded Systems*

TAI-YI HUANG, CHIH-CHIEH CHOU AND PO-YUAN CHEN

Department of Computer Science

National Tsing Hua University

Hsinchu, 300 Taiwan

E-mail: {tyhuang; ccchou; pychen}@cs.nthu.edu.tw

A DMA controller that operates in the cycle-stealing mode transfers data by stealing bus cycles from the CPU. The concurrent contention for the I/O bus by a CPU task and a cycle-stealing DMA I/O task retards their progress and extends their execution times. In this paper we first describe a method for bounding the worst-case execution time (WCET) of a CPU task when cycle-stealing DMA I/O is present. We next use the dynamic-programming technique to develop a method for bounding the WCET of a cycle-stealing DMA I/O task executing concurrently with a set of CPU tasks. We conducted exhaustive simulations on a widely-used embedded processor. The experimental results demonstrate that our methods tightly bound the WCETs of CPU tasks and of cycle-stealing DMA I/O tasks.

Keywords: hard-real-time systems, worst-case execution time, cycle-stealing DMA I/O, concurrent execution, embedded systems

1. INTRODUCTION

In a hard-real-time embedded system, both CPU tasks and I/O tasks are required to complete executions by their deadline. A task that executes longer than its allocated computation time may lead to missed deadlines and the failure of the whole system. The schedulability analysis of such a system requires that the worst-case execution time (WCET) of each task be known in advance to ensure the completion of each task by its allocated computation time [14, 19, 20, 28, 30]. To tightly bound the WCET, the interference of concurrently executing CPU tasks and I/O tasks must be considered.

This paper addresses the problems of bounding the WCETs of concurrently executing CPU tasks and cycle-stealing DMA I/O tasks in a hard-real-time embedded system. We model each CPU task as a sequence of instructions (*i.e.*, code without undetermined loop bounds and recursive function calls) which is quite commonly found in the synthesized code of hard-real-time embedded systems [10, 11, 27, 29]. A DMA controller (DMAC) transfers data between the main memory and I/O devices with minimal CPU involvement. A DMAC may operate either in the burst mode or in the cycle-stealing mode. A DMAC that operates in the burst mode gains the control of the I/O bus once it is free and retains its ownership until all data transfers in the DMA I/O task complete. In contrast, a DMAC that operates in the cycle-stealing mode transfers data by “stealing”

Received June 21, 2004; accepted November 1, 2004.

Communicated by David H. C. Du.

* This work was supported in part by the Ministry of Economic Affairs of Taiwan, under grant No. MOEA 95-EC-17-A-01-S1-038 and 94-EC-17-A-04-S1-044.

bus cycles from an executing CPU task. A cycle-stealing DMA I/O task allows a CPU task to execute concurrently. The contention for the I/O bus by the cycle-stealing DMA I/O task and the CPU task retards their progress and extends their execution times.

This paper first analyzes the delay caused by cycle-stealing DMA I/O activities on a concurrently executing CPU instruction. Based on this analysis, we develop a method that tightly bounds the WCET of a CPU task. We next proceed to the problem of bounding the WCET of a cycle-stealing DMA I/O task. We define the execution time of a DMA I/O task to be the interval from the instant when the DMAC is ready to transfer the first unit of data to the instant when the CPU receives from the DMAC an interrupt signal when the transfer of the last unit of data is complete. We present here a method for bounding the WCET of a cycle-stealing DMA I/O task executing concurrently with a set of CPU tasks on a single-processor embedded system. We use the dynamic-programming technique in the development of this method. The running-time complexity of this method is $O(ZU) + O(K^2Z^2)$, where Z is the number of units of data to be transferred by the DMA I/O task, K is the number of CPU tasks, and U is the sum of the number of instructions in CPU tasks.

To demonstrate the effectiveness of our method on bounding the WCET of a CPU task, we compare our WCET prediction with the one obtained by the traditional pessimistic approach. Given a CPU task and a cycle-stealing DMA I/O task, which are ready at the same time, the traditional pessimistic approach estimates the WCET of the CPU task to be the sum of the execution times of both tasks when each executes alone. We measure the performance of our method in terms of the amount of reduction from the most pessimistic WCET prediction. Among the several commonly-used programs tested, our method achieves up to 39% improvement in the accuracy of the WCET prediction.

We demonstrate the correctness of our method on bounding the WCET of a cycle-stealing DMA I/O task through exhaustive simulations. Given a cycle-stealing DMA I/O task and a set of CPU tasks, we simulated all possible combinations of release times and concurrent executions of these preemptive CPU tasks and the cycle-stealing DMA I/O task. We compared the maximum execution time of the DMA I/O task recorded in the simulation experiment with the WCET prediction obtained by our method. The experimental results show that our method tightly bounds the WCET of a cycle-stealing DMA I/O task.

The rest of the paper is structured as follows. Section 2 describes our machine model. Section 3 analyzes the delay caused by cycle-stealing DMA I/O and presents the method for bounding the WCET of a CPU task. Section 4 describes the method for bounding the WCET of a cycle-stealing DMA I/O task. The experimental results are discussed in section 5. Section 6 describes related work. Finally, section 7 concludes this paper.

2. THE MACHINE MODEL

We adopt here the commonly-used single-processor machine model shown in Fig. 1. In this model the DMAC operates in the cycle-stealing mode and shares the same I/O bus with the CPU. The bus controller allows only one bus master at any time. As a result, either the CPU or the DMAC, but not both, can hold the bus and transfer data at the same time.

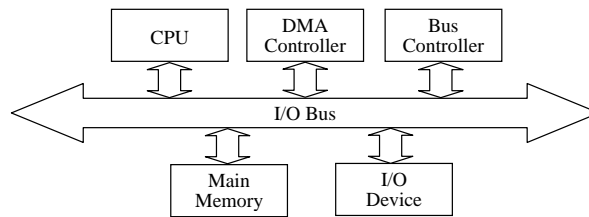


Fig. 1. The architecture of the machine model.

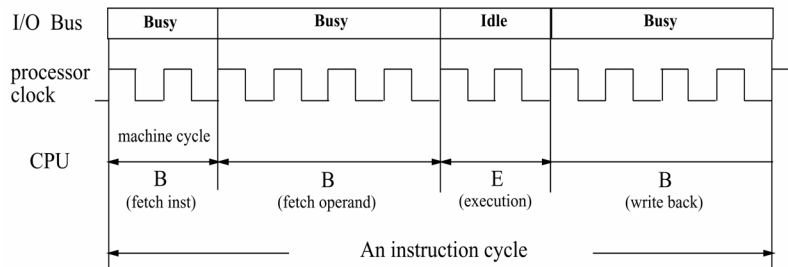


Fig. 2. The instruction cycle of ADD 1, (A0).

We assume that signal transmission in the bus is instantaneous. Our analytical method presented in this paper works only on a processor when the cache memory and the instruction pipeline are disabled. Although this assumption may seem impractical with the current microprocessor technology, the authors argue that, given the extreme complexity of the problem being addressed, our method is the first solution to bound the WCET of cycle-stealing DMA I/O tasks.

The execution of an instruction in this machine model is as shown in Fig. 2. An *instruction cycle* consists of a sequence of operations to fetch and execute an instruction. The sequence takes one or more machine cycles. A machine cycle requires one or more processor clock cycles to execute. The beginning of each machine cycle is triggered by the processor clock. For example, the instruction cycle of **ADD 1, (A0)**, shown in Fig. 2, is composed of four machine cycles: a memory-read (bus-access) cycle to fetch the instruction, a memory-read (bus-access) cycle to fetch an operand, an execution (no-bus-access) cycle to carry out the addition, followed by a memory-write (bus-access) cycle to write back the data. Each machine cycle in this example in turn takes 2, 4, 2, and 4 processor clock cycles to execute. We classify all machine cycles into two categories: B (bus-access) cycles and E (execution) cycles. A B-cycle is a machine cycle during which the CPU uses the I/O bus. In contrast, the CPU does not use the bus when it is in an E-cycle. In general, there may be several consecutive E-cycles in an instruction cycle.

For the sake of concreteness, we assume that the bus contention between the CPU and the DMAC is regulated according to the VMEbus [33] protocol. This protocol is sufficiently general such that our analysis presented in this paper may be easily applied to many other commonly-used bus protocols. To access the bus, the DMAC first sends a bus request. If the bus is already used by the CPU, the DMAC waits until the bus becomes free. When the bus is free, there is a short delay, called the *bus master transfer time* (BMT), while the DMAC gains the control of the bus. The DMAC can transfer data

when it becomes the bus master. At the end of each transfer of a unit of data, if there is no bus request from a higher priority device (e.g., the CPU), the DMAC may continue to hold the bus and transfer data. Otherwise, the DMAC must release the bus, and after another BMT delay the higher priority device gains the control of the bus and becomes the bus master.

Fig. 3 illustrates the concurrent execution of DMA I/O and a sequence of machine cycles $B_i \rightarrow E_1 \rightarrow E_2 \rightarrow \dots \rightarrow E_k \rightarrow B_{i+1}$. The DMAC gains the bus when the CPU enters E_1 cycle from B_i cycle. It keeps transferring data during the interval from E_1 cycle to E_k cycle. The CPU requests the bus at the end of each data transfer; the DMAC releases the bus at any pending bus request at the end of each data transfer; the DMAC releases the bus at the end of m -th transfer, and the CPU gains the bus after another BMT delay. The execution of B_{i+1} cycle is delayed for $(b + BMT)$, where b is the delay between when the CPU requests the bus and when the request is checked and the DMAC releases the bus.

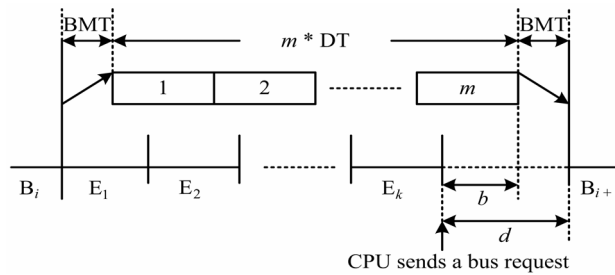


Fig. 3. The concurrent execution of DMA I/O and a sequence of E-cycles.

We assume that the transfer of each unit of data by the DMAC takes the same amount of time and denote this time by DT . Let T be the total execution time of the k consecutive E-cycles, and m be the maximum units of data the DMAC can transfer during this sequence of E-cycles. Based on the facts that $0 \leq b < DT$ and $T + b = m \times D + BMT$, we have

$$(m - 1) < \frac{T - BMT}{DT} \leq m.$$

We can compute m by the equation

$$m = \left\lceil \frac{T - BMT}{DT} \right\rceil. \tag{1}$$

The worst-case delay suffered by the CPU execution of the sequence of machine cycles is

$$m \times DT + 2 \times BMT - T.$$

Because each machine cycle is triggered by the processor clock, the first B-cycle B_{i+1} after the sequence of k E-cycles cannot start until the next processor clock cycle. As a result, the exact worst-case delay suffered by the CPU execution is equal to

$$d = \left\lceil \frac{m \times DT + 2 \times BMT - T}{T_c} \right\rceil \times T_c, \tag{2}$$

where T_c is the period of a clock cycle.

3. BOUNDING THE WCET OF A CPU TASK

Let A_C denote a CPU task, which in turn is a sequence of k CPU instructions $I_1 \rightarrow I_2 \rightarrow \dots \rightarrow I_k$. Because on a simple architecture each instruction begins with a B-cycle to fetch the instruction from the memory, no DMA data transfer can take place across two instructions. Consequently, the effects of cycle-stealing on each instruction can be analyzed independently, without considering the other instructions. The WCET of the CPU task A_C when it executes concurrently with DMA I/O is therefore bounded by the sum of the WCET of each instruction executing concurrently with DMA I/O.

To bound the WCET of an instruction I_i that executes concurrently with DMA I/O, we first use Eq. (2) to calculate the worst-case delay suffered by each sequence of E-cycles. We obtain the WCET of the instruction, denoted by $W(I_i)$, by summing the execution time of the instruction when it executes alone and the worst-case delay of all E-cycles sequences in the instruction. Similarly, by Eq. (1), we obtain the maximum units of data the DMAC can transfer during the execution of the instruction. We denote this value by $M(I_i)$. Finally, we obtain the WCET of A_C , denoted by $W(A_C)$, by the equation

$$W(A_C) = \sum_{i=1}^k W(I_i), \tag{3}$$

and the maximum units of data the DMAC can transfer during the execution of A_C , denoted by $M(A_C)$, by the equation

$$M(A_C) = \sum_{i=1}^k M(I_i). \tag{4}$$

The computation of $W(A_C)$ by Eq. (3) requires, as inputs, two parameters of the I/O bus, BMT and DT. The other information needed by the equation includes how many machine cycles each instruction is composed of, the function of each machine cycle, and the execution time of each machine cycle. We can obtain this information from the reference manual provided by the manufacturer of the processor.

4. BOUNDING THE WCET OF A DMA I/O TASK

Because a DMA I/O task proceeds by stealing bus cycles from executing instructions, its execution time depends on the sequence of instructions executing concurrently with it. We generalize the problem of bounding the WCET of a DMA I/O task with a workload that consists of a DMA I/O task and K independent CPU tasks. The DMA I/O task, denoted by A_D , transfers Z units of data. Each of the K CPU tasks, denoted by A_1, A_2, \dots, A_K , is a sequence of CPU instructions. Each CPU task has an arbitrary release

time, and these K CPU tasks are scheduled preemptively. In contrast, the DMA I/O task A_D is nonpreemptable. A_D is initialized by a task other than the K CPU tasks. Consequently, A_D may execute concurrently with any of the K CPU tasks. The method presented here works under any scheduling algorithm.

In the following we first describe three properties revealed by a sequence of instructions that execute concurrently with the DMA I/O task A_D . Based on these properties we develop a recursive formula to compute the WCET of A_D . The recursive formula gives the basis of a dynamic-programming method which can be used to bound the WCET of A_D . To simplify the discussion, we assume that the CPU is never idle during the execution of A_D . We will remove this assumption at the end of this section by modeling an idle period as an instruction of a special CPU task.

4.1 The Properties of a Concurrent Instruction Sequence

Let S denote a sequence of instructions $I_a \rightarrow \dots \rightarrow I_{a+j}$ executing concurrently with the DMA I/O task A_D . Because interrupts are processed between instruction cycles, A_D and the first instruction I_a begin at the same time. Similarly, A_D and the last instruction I_{a+j} end at the same time. Consequently, the WCET of the sequence S , denoted by $W(S)$, is bounded by the sum of the WCET of each instruction when it executes concurrently with DMA I/O. That is

$$W(S) = W(I_a) + \dots + W(I_{a+j}).$$

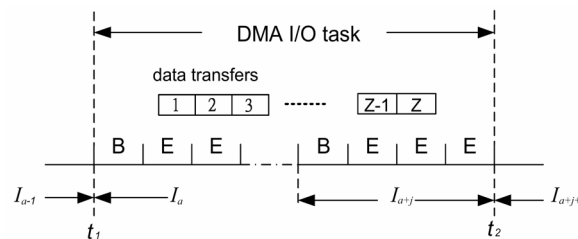


Fig. 4. The execution time of a DMA I/O task.

Example 1: Let A_D execute concurrently with a sequence of instructions $I_a \rightarrow \dots \rightarrow I_{a+j}$ as shown in Fig. 4. The CPU signals the DMAC to start its data transfer at time t_1 and starts the execution of the first instruction I_a at the same time. The DMAC transfers the first unit of data when the CPU enters the first E-cycle. The DMAC signals the CPU the completion of the last unit of data during the execution of the last instruction I_{a+j} . Because interrupt signals are processed between instruction cycles, the CPU is notified the completion of A_D at t_2 , when the last instruction I_{a+j} completes its execution. The execution time of A_D is therefore equal to $(t_2 - t_1)$, that is bounded by $W(I_a) + \dots + W(I_{a+j})$. \square

Property 1 The DMA I/O task A_D and the sequence S begin and end at the same time. The WCET of S is bounded by the sum of the WCET of each instruction when it executes concurrently with DMA I/O.

The DMAC must transfer the last unit of data during the execution of the last instruction I_{a+j} . Because interrupts are only processed between instruction cycles, some of the E-cycles in I_{a+j} may not be utilized by the DMAC as shown by the example in Fig. 4. In contrast, because the DMA I/O task is nonpreemptable, the DMAC must fully utilize all the E-cycles in the rest of the instructions to transfer data. Again, let $M(I_i)$ be the maximum units of data transferred by the DMAC during the execution of the instruction I_i . The sequence of instructions $I_a \rightarrow \dots \rightarrow I_{a+j}$ must satisfy

$$\sum_{i=1}^{a+j-1} M(I_i) < Z \leq \sum_{i=1}^{a+j} M(I_i).$$

Property 2 The DMAC must fully utilize all the E-cycles in every instruction of S except I_{a+j} . In addition, the last unit of data must be transferred during the execution of I_{a+j} .

Because these K CPU tasks are scheduled preemptively, the sequence S may contain instructions from any of the K CPU tasks. Among the instructions in $I_a \rightarrow \dots \rightarrow I_{a+j}$, let S_i denote the set of instructions from the CPU task A_i . S_i is either an empty set or a subsequence of contiguous instructions of the task A_i .

Property 3 Among the instructions of S , the set of instructions from the same CPU task must be a subsequence of contiguous instructions of the CPU task.

4.2 The Recursive Formula

Let Y denote the set of all possible sequences of instructions that may execute concurrently with the DMA I/O task A_D . According to Property 1, $W(S)$, the WCET of a sequence S , is simply the sum of the $W(I)$ for every instruction $I \in S$. Therefore, we can obtain the WCET of A_D , denoted by $W(A_D)$, as the maximum $W(S)$ for every $S \in Y$; that is

$$W(A_D) = \max_{\forall S \in Y} W(S).$$

4.2.1 The derivation

Let us divide Y into K disjoint subsets Y_1, Y_2, \dots, Y_K in such a way that the subset Y_α consists of all the sequences where the last instruction of each sequence is from the task A_α . Let $W_{(K,Z,\alpha)}$ denote the maximum $W(S)$ for every $S \in Y_\alpha$. We can redefine $W(A_D)$ as

$$W(A_D) = \max_{1 \leq \alpha \leq K} W_{(K,Z,\alpha)}. \tag{5}$$

Let us further divide Y_α into a number of disjoint subsets. Let $Y_\alpha^{(m_1, m_2, \dots, m_k)}$ denote a subset of sequences in Y_α such that each sequence S in this subset has the following property: the DMAC transfers m_i units of data during the executions of the instructions from the task A_i , $i = 1$ to K . Let $W_\alpha^{(m_1, m_2, \dots, m_k)}$ denote the maximum $W(S)$ for every $S \in Y_\alpha^{(m_1, m_2, \dots, m_k)}$. We can define $W_{(K,Z,\alpha)}$ as

$$W_{(K,Z,\alpha)} = \max\{W_\alpha^{(m_1, m_2, \dots, m_K)}\}. \quad (6)$$

Because the DMA I/O task A_D transfers Z units of data in total, we have

$$m_1 + m_2 + \dots + m_K = Z. \quad (7)$$

In addition, because the last instruction of each sequence $S \in Y_\alpha^{(m_1, m_2, \dots, m_K)}$ is from the task A_α , we have $m_\alpha > 0$ according to Property 2. And, $m_i \geq 0$ for any $i \neq \alpha$. That is

$$0 < m_\alpha, \text{ and } 0 \leq m_i \text{ for } i \neq \alpha. \quad (8)$$

To compute $W_\alpha^{(m_1, m_2, \dots, m_K)}$, we first define $f_i(m_i)$ and $p_i(m_i)$. Let $I_a \rightarrow \dots \rightarrow I_{a+j}$ be a subsequence of contiguous instructions of the task A_i such that

$$m_i = \sum_{l=a}^{a+j} M(I_l). \quad (9)$$

Let $F_i^{m_i}$ denote the set of all possible subsequences of A_i that satisfy Eq. (9). Again, we use $W(S)$ to denote the maximum execution time of a subsequence S when it executes concurrently with DMA I/O. We define $f_i(m_i)$ to be the maximum $W(S)$ for every $S \in F_i^{m_i}$. That is

$$f_i(m_i) = \max_{\forall S \in F_i^{m_i}} W(S). \quad (10)$$

Similarly, let $I_a \rightarrow \dots \rightarrow I_{a+j}$ be a subsequence of contiguous instructions of the task A_i such that

$$\sum_{l=a}^{a+j-1} M(I_l) < m_i \leq \sum_{l=a}^{a+j} M(I_l). \quad (11)$$

Let $P_i^{m_i}$ denote the set of all possible subsequences of A_i that satisfy Eq. (11). We define $p_i(m_i)$ as

$$p_i(m_i) = \max_{\forall S \in P_i^{m_i}} W(S). \quad (12)$$

Let us get back to a sequence $S \in Y_\alpha^{(m_1, m_2, \dots, m_K)}$. According to Property 3, the sequence S is in fact the concatenation of the subsequence S_i of each task A_i such that the DMAC transfers m_i units of data during the execution of the subsequence S_i , $i = 1$ to K . In addition, because no DMA transfer can take place across two subsequences, $W(S)$ is equal to the sum of $W(S_i)$, $i = 1$ to K . Consequently, we can use $f_i(m_i)$ and $p_i(m_i)$ to define $W_\alpha^{(m_1, m_2, \dots, m_K)}$ as

$$W_\alpha^{(m_1, m_2, \dots, m_K)} = e_1(m_1) + e_2(m_2) + \dots + e_K(m_K) \quad (13)$$

where

$$e_i(m_i) = \begin{cases} p_i(m_i) & \text{if } i = \alpha, \\ f_i(m_i) & \text{if } i = \beta. \end{cases} \quad (14)$$

By replacing Eq. (8) with the settings of $p_i(0)$ to $-\infty$, $i = 1$ to K , we can generalize the definition of $W_{(K,Z,\alpha)}$ given in Eqs. (6) to (8), (13), and (14) to the following form

$$W_{(K,Z,\alpha)} = \max\{e_1(m_1) + e_2(m_2) + \dots + e_K(m_K)\}$$

where $e_i(m_i)$ is given by Eq. (14) and the max function is over all m_1, m_2, \dots, m_K such that

- (1) $m_1 + m_2 + \dots + m_K = Z$, and
- (2) $0 \leq m_i \leq Z$, $i = 1, 2, \dots, K$.

By considering m_K separately, we can further rewrite the above formula as

$$W_{(K,Z,\alpha)} = \max_{0 \leq g \leq Z} \{ \max\{e_1(m_1) + e_2(m_2) + \dots + e_{K-1}(m_{K-1})\} + e_K(g) \}$$

where the inner max function is over all m_1, m_2, \dots, m_{K-1} such that

- (1) $m_1 + m_2 + \dots + m_{K-1} = Z - g$, and
- (2) $0 \leq m_i \leq Z - g$, $i = 1, 2, \dots, K - 1$.

Since the inner term in the above formula is exactly $W_{(K-1,Z-g,\alpha)}$, we simplify it to

$$W_{(K,Z,\alpha)} = \max_{0 \leq g \leq Z} \{W_{(K-1,Z-g,\alpha)} + e_K(g)\}.$$

After considering the terminative condition of this recursive formula, we obtain the definition of $W_{(K,Z,\alpha)}$ below

$$W_{(K,Z,\alpha)} = \begin{cases} e_1(Z) & \text{if } K = 1, \\ \max_{0 \leq g \leq Z} \{W_{(K-1,Z-g,\alpha)} + e_K(g)\} & \text{if } K > 1. \end{cases} \quad (15)$$

Again, $e_i(m_i)$ is given by Eq. (14). Finally, Eqs. (5) and (15) together give a recursive formula for computing the WCET of the DMA I/O task A_D .

4.2.2 Table construction

The computation of Eq. (15) requires frequent accesses to both $f_i(m_i)$ and $p_i(m_i)$. To avoid computing the same $f_i(m_i)$ and $p_i(m_i)$ repeatedly, we pre-compute each $f_i(m_i)$ and $p_i(m_i)$, and store the results in the tables $f[i, m_i]$ and $p[i, m_i]$, respectively, for $i = 1$ to K and $m_i = 0$ to Z . We rewrite Eq. (14) as below to retrieve pre-computed results from these two tables.

$$e_i(m_i) = \begin{cases} p[i, m_i] & \text{if } i = \alpha, \\ f[i, m_i] & \text{if } i \neq \alpha. \end{cases} \quad (16)$$

Fig. 5 lists the procedure for constructing the tables $f[\alpha, z]$ and $p[\alpha, z]$ of a CPU task A_α . Here we let U_α denote the number of instructions in A_α . Initially, we set $f[\alpha, 0]$ to 0, $f[\alpha, z]$ to $-\infty$, $z = 1$ to Z . In addition, we set $p[\alpha, z]$ to $-\infty$, $z = 0$ to Z . We update the table $f[\alpha, z]$ each time we locate a subsequence in A_α that belongs to F_α^z and whose WCET is larger than the current value. Similarly, we update the table $p[\alpha, z]$ each time we locate a subsequence in A_α that belongs to P_α^z and has a larger WCET. If at the end of the procedure an entry $f[\alpha, z]$ (or $p[\alpha, z]$) still has the value of $-\infty$, this fact implies that it is impossible to find in the task A_α a subsequence of instructions that belongs to F_α^z (or P_α^z).

Input: the CPU task A_α , a sequence of U_α instructions.
Output: the entries $f[\alpha, z]$ and $p[\alpha, z]$, $z = 0, 1, \dots, Z$.

Procedure:

```

for  $z = 0$  to  $Z$  do
  for  $j = 1$  to  $U_\alpha$  do {
    1. find a longest subsequence that starts with the  $j$ -th instruction and belongs to  $F_\alpha^z$ ;
    2. if (such a subsequence exists) and (its WCET is larger than  $f[\alpha, z]$ ) then
       - set  $f[\alpha, z]$  to the WCET of the subsequence;
    3. find a longest subsequence that starts with the  $j$ -th instruction and belongs to  $P_\alpha^z$ ;
    4. if (such a subsequence exists) and (its WCET is larger than  $p[\alpha, z]$ ) then
       - set  $p[\alpha, z]$  to the WCET of the subsequence;
  }

```

Fig. 5. The procedure that computes $f[\alpha, z]$ and $p[\alpha, z]$ for the task A_α

4.2.3 Running-time complexity

Instead of searching through the sequence of instructions repeatedly, the steps 1 and 3 of the procedure shown in Fig. 5 can be carried out in constant time by utilizing the information calculated in a previous iteration of the loop. Specifically, the subsequences that start with the $(j - 1)$ -th instruction can be used to locate the subsequences that start with the j -th instruction. Consequently, the running-time complexity of the procedure shown in Fig. 5 can be optimized to $O(ZU_\alpha)$. To construct the whole tables of $f[k, z]$ and $p[k, z]$, we apply this procedure to each of the K CPU tasks. The time complexity is $\sum_{k=1}^K O(ZU_k) = O(ZU)$, where U is the sum of the number of instructions of these K CPU tasks.

The procedure shown in Fig. 6 uses the tables $f[\alpha, z]$ and $p[\alpha, z]$ together with Eqs. (15) and (16) to compute $W_{(K,Z,\omega)}$. We implement Eq. (5) by the for-loop below. Initially, we set $W(A_D)$ to 0. At the end of the loop, $W(A_D)$ returns the WCET of the DMA I/O task A_D .

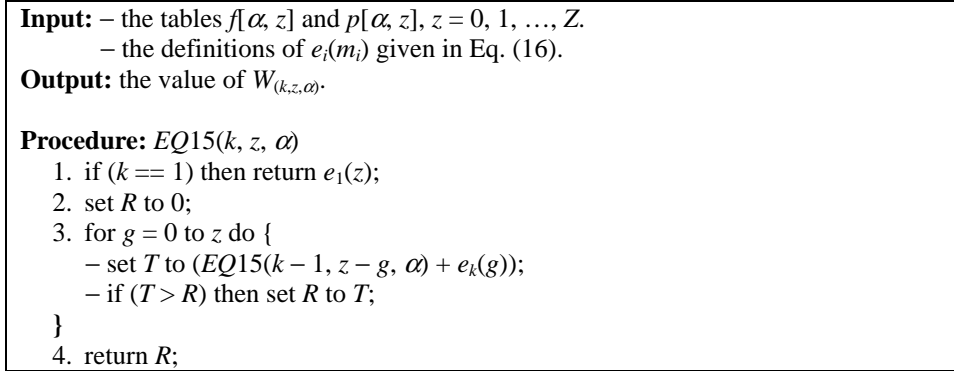


Fig. 6. The procedure that implements Eq. (15).

```

For  $\alpha = 1$  to  $K$  do {
     $W_{(K,Z,\alpha)} = EQ15(K, Z, \alpha)$ ;
    If  $(W_{(K,Z,\alpha)} > W(A_D))$  that set  $W(A_D)$  to  $W_{(K,Z,\alpha)}$ ;
}
    
```

The time complexity for computing $W_{(K,Z,\alpha)}$ with the procedure shown in Fig. 6 is

$$O(W_{(K,Z,\alpha)}) = O((Z+1)^i \times O(W_{(K-i,Z,\alpha)})) = O((Z+1)^{K-1}) = O(Z^K).$$

Finally, the time complexity of computing $W(A_D)$ with the recursive formula is $O(ZU) + O(KZ^K)$. In other words, the time complexity of the recursive formula grows exponentially as the number of CPU tasks grow.

4.3 A Dynamic-Programming Method

To avoid redundant computation, we implement Eq. (15) by the procedure shown in Fig. 7. The time complexity of this dynamic-programming method is $O(KZ^2)$, and the time complexity of computing $W(A_D)$ by this procedure is $O(K^2Z^2)$. Because the time complexity of building the tables $f[k, z]$ and $p[k, z]$ is $O(ZU)$, the time complexity of computing $W(A_D)$ is $O(ZU) + O(K^2Z^2)$, where Z is the number of units of data to be transferred by A_D , K is the number of CPU tasks that may execute concurrently with A_D , and U is the sum of the number of instructions of these K CPU tasks.

Another advantage of implementing Eq. (15) by the dynamic-programming method is that the table $W[k, z, \alpha]$ built for the purpose of bounding the WCET of A_D can be used to bound the WCET of other DMA I/O tasks. For example, to compute the WCET of another DMA I/O task $A_{D'}$ which transfer Z' units of data, $Z' < Z$, by Eq. (15) we need to compute first $W[K, Z', \alpha]$. Because $W[K, Z', \alpha]$ had already been computed in the process of computing the WCET of A_D , we can use the results stored in the table directly to compute the WCET of $A_{D'}$. Suppose that there are γ DMA I/O tasks that can execute concurrently with these K CPU tasks, and each DMA I/O task transfers Z_i units of data, $i = 1, 2, \dots, \gamma$. The time complexity of bounding the WCETs of these DMA I/O tasks is

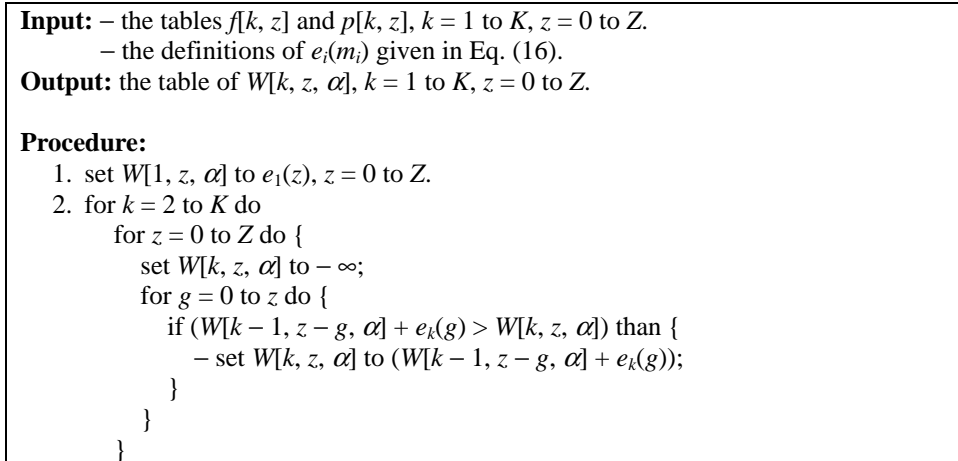


Fig. 7. A dynamic-programming method for Eq. (15).

$$O(Z_{\max}U) + O(K^2Z_{\max}^2)$$

where Z_{\max} is the maximum value of Z_1, Z_2, \dots, Z_r .

The discussion thus far assumes the CPU to be never idle. We now remove this assumption. Suppose that there is an idle period during the execution of A_D . Let m denote the number of units of data the DMAC transfers during this period. We model this idle period as an instruction I_l of a special CPU task A_{K+1} , called the *background* task. Because the DMAC takes at most $(2 \times BMT + DT)$ to transfer a unit of data, the execution time of this period is bounded by $m \times (2 \times BMT + DT)$. That is

$$M(I_l) = m, \text{ and } W(I_l) = m \times (2 \times BMT + DT). \quad (17)$$

Furthermore, with Eq. (17), the WCET of an instruction I_p in front of the idle period can still be bounded by $W(I_p)$ as described in section 3, whether the instruction I_p ends with a B-cycle or an E-cycle. Let S denote a mixed sequence of instructions and idle periods that executes concurrently with A_D . Let S' denote the new sequence of instructions after replacing each idle period in S with an instruction of the background task A_{K+1} . The new sequence S' holds the three properties discussed in section 4.1. Consequently, by adding the background task A_{K+1} to the set of the K CPU tasks that can execute concurrently with A_D and setting

$$f[K+1, z] = p[K+1, z] = z \times (2 \times BMT + DT), \quad z = 0, 1, \dots, Z,$$

the dynamic-programming method given in Fig. 7 still bounds the WCET of A_D at the time complexity of $O(ZU) + O(K^2Z^2)$ when CPU idle periods are allowed.

5. EXPERIMENTAL RESULTS

We demonstrate the effectiveness of our methods through exhaustive simulations on a widely-used embedded processor. Table 1 lists the eight CPU tasks, each of which is a random execution trace of a commonly-used program. We compiled each program into an MC68030 assembly program and executed on an MC68030 simulator with randomly-generated input data to obtain the execution trace. Column 3 of Table 1 lists the number of instructions in each CPU task.

Table 1. The CPU task set.

Program	Brief Description	Instructions
QuickSort	Recursive QuickSort	23,026
BubbleSort	Sequential BubbleSort	65,726
FFT	Fast Fourier Transform	249,107
Spline	Cubic Spline Function	209,837
Gaussian	Gaussian Elimination	47,242
Mtxmul	Matrix Multiplication	36,789
Correlate	Track-Correlate Function	26,543
Mtxmu12	Loop-Unrolled Version of Mtxmul	9,391

We obtained from the Motorola 68030 manual [2] the timing information of each instruction in the traces. The clock frequency of the microprocessor was 20 MHz: the period of a clock cycle T_c was 50 ns. We assume a 0-wait memory was used in this experiment, and each DMA transfer of a unit of data took two clock cycles. Hence, we set DT to 100 ns. Finally, BMT was 5 ns.

5.1 CPU Tasks

For each CPU task A_C listed in Table 1, we first used Eq. (3) to compute $W(A_C)$. We next used Eq. (4) to compute $M(A_C)$. We then compared our WCET prediction with the one obtained by the traditional pessimistic approach. Given the CPU task A_C and a DMA I/O task that transfers $M(A_C)$ units of data, which are ready at the same time, the pessimistic approach estimates the WCET of A_C to be equal to the sum of the execution time of A_C when it executes alone and the execution time of the DMA I/O task when it is done alone. We denote this traditional pessimistic WCET prediction by $W_t(A_C)$. We use the percentage of reduction from $W_t(A_C)$

$$R = \frac{W_t(A_C) - W(A_C)}{W_t(A_C)}$$

to measure the performance of our method.

We also investigated the relationship between the performance of our method and the computational requirement of a CPU task. We classify all instructions here into two

categories: long instructions and short instructions. An instruction is a long one if during its execution; the CPU does not need the bus for 8 processor clock cycles or more. In contrast, during the execution of a short instruction, the CPU never allows any I/O device to have the bus for such a long period. For example, the instructions MULU.W D1, D2 and DIVU.W D2, D0 are long instructions, and MOVE.L (A3)+, D0 and ADD.L, D0, D1 are short instructions. Column 2 of Table 2 gives the percentage of long instructions in each CPU task.

Table 2. The simulation results for CPU tasks.

Program	Long Instructions %	R in %
QuickSort	0%	8%
BubbleSort	0%	10%
FFT	2%	19%
Spline	3%	21%
Gaussian	5%	25%
Mtxmul	11%	35%
Correlate	17%	38%
Mtxmu12	22%	39%

Column 3 of Table 2 gives the reduction percentage on each CPU task. Because the delay caused by cycle-stealing on each instruction is bounded by Eq. (2), the overhead of each DMA transfer in a long instruction is less than that in a short instruction. In addition, more DMA data transfers can be carried out in a long instruction than in a short instruction. Therefore, our method produces a larger percentage of reduction on a CPU task with a higher percentage of long instructions. Among the tested CPU tasks, **Mtxmu12** is obtained by unrolling the whole innermost loop of **Mtxmul**. The loop-unrolling procedure significantly increases the percentage of long instructions in the trace. As a result, our method produces a higher percentage of reduction on the loop-unrolled version: a 39% reduction from the most pessimistic WCET prediction is achieved.

5.2 DMA I/O Tasks

We demonstrate the correctness of the dynamic-programming method through exhaustive simulations. To make exhaustive simulation feasible, we executed the programs listed in Table 1 with a much smaller data set to obtain the CPU task set listed in Table 3. Column 3 gives the number of instructions in each simplified CPU task. We first used the dynamic-programming method to compute the WCET of a DMA I/O task A_D when it executes concurrently with the eight CPU tasks. We next simulated the concurrent execution of these CPU tasks and A_D under the round-robin scheduling algorithm and the fixed priority assignment algorithm, and recorded the execution time of A_D . CPU tasks were simulated for all possible combinations of release times, and in the case of fixed priority assignment, all possible combinations of priority assignments were simulated. We allowed scheduling points to occur only every 100 instructions. We use $W_r(A_D)$ and $W_p(A_D)$ to denote the maximum execution times of A_D found by the simulation when the CPU

Table 3. The simplified CPU task set.

Program	Brief Description	Instructions
QuickSort	Recursive QuickSort	3,124
BubbleSort	Sequential BubbleSort	2,763
FFT	Fast Fourier Transform	3,662
Spline	Cubic Spline Function	2,101
Gaussian	Gaussian Elimination	1,436
Mtxmul	Matrix Multiplication	1,170
Correlate	Track-Correlate Function	814
Mtxmu12	Loop-Unrolled Version of Mtxmul	884

Table 4. The simulation results for DMA I/O tasks.

	The length of the I/O task			
	250	500	750	1000
$W(A_D)/W_r(A_D)$	1.060	1.029	1.017	1.014
$W(A_D)/W_p(A_D)$	1.063	1.028	1.013	1.006

tasks are scheduled by the round-robin and fixed priority assignment scheduling algorithms, respectively. We compared our WCET prediction $W(A_D)$ against $W_r(A_D)$ and $W_p(A_D)$ to show the correctness of the dynamic-programming method.

Table 4 shows the experimental results for DMA I/O tasks that transfer different units of data. Rows 2 and 3 of Column 2 give the values of $W(A_D)/W_r(A_D)$ and $W(A_D)/W_p(A_D)$, respectively, when the DMA I/O task A_D transfers 250 units of data. We also simulated the concurrent execution of the CPU task set and three other DMA I/O tasks which transfer 500, 750, 1000 units of data, and the results are shown in Columns 3, 4, and 5. As explained in section 4.3, our dynamic-programming method only computes the WCET of the DMA I/O task that transfers 1000 units of data. The WCETs of the other three DMA I/O tasks are obtained in a table-driven manner.

For every of the eight cases investigated in this experiment, our WCET prediction $W(A_D)$ is always larger than the maximum execution time of the DMA I/O task recorded in the exhaustive simulations. Our method overestimates the WCET for at most 6.3% when the CPU tasks are scheduled by the fixed priority assignment algorithm and the DMA I/O task transfers 250 units of data. The percentage of overestimation is smaller with a longer DMA I/O task. This behavior results from the overestimation of our method on the last instruction of the sequence that executes concurrently with the DMA I/O task. Obviously, the overestimation will have a smaller effect on the WCET prediction of a longer DMA I/O task. Finally, our method still produces 0.6% and 1.4% overestimation on the WCET of the DMA I/O task that transfers 1000 units of data at the round-robin and the fixed priority assignment scheduling algorithms, respectively. It is caused by the 100-instruction scheduling distance. This limit considerably trims down the set of possible instruction sequences. We are confident that, by allowing scheduling points to occur on every instruction, the overestimation by our method will be practically negligible.

6. RELATED WORK

Most of the previous studies focused on bounding the WCET of CPU tasks [1, 3-8, 12, 15-18, 21-26, 29, 31, 32]. Muller *et al.* [23] developed a static cache simulation to bound the WCET of CPU tasks executed on a contemporary machine with the instruction cache. Li and Malik [16] presented the implicit path-enumeration method to convert the problem of bounding the WCET into one of solving a set of ILP constraints. Li *et al.* [17] later extended their approach to include the timing analysis of both direct-mapped and set-associative caches. Lim *et al.* [18] proposed a timing analysis technique for modern multiple-issue machines such as superscalar processors. Kim *et al.* [15] presented quantitative analysis results on the impacts of various architecture features on the accuracy of WCET predictions. Theiling *et al.* [31, 32] adopted abstract interpretation to analyze the performance of modern architectures and integrated with the implicit path-enumeration method to bound the WCET.

All of the above methods invariably assume that a CPU task to be analyzed executes without any interference of concurrently executing I/O tasks in the system. Huang *et al.* [13] first attempts to bound the WCET of a CPU task when cycle-stealing DMA I/O is concurrently executing. Hahn *et al.* [9] bounded the worst-case DMA response time in a fixed-priority bus arbitration protocol. However, these paper did not address at all on how to bound the WCET of a concurrently executing cycle-stealing DMA I/O task. To our knowledge, our work is the first one that attempts to bound the worst-case interference between concurrently executing CPU tasks and cycle-stealing DMA I/O tasks.

7. CONCLUDING REMARKS

Cycle-stealing DMA I/O operations have often been disabled in hard-real-time embedded systems. In this paper we first presented an analysis for bounding the delay. Based on this analysis, we developed a method for bounding the WCET of a CPU task. Simulation results demonstrate that our method produces much tighter WCET predictions than the traditional pessimistic method, especially when the CPU task contains a large percentage of computation-intensive instructions.

We also derived a recursive formula for bounding the WCET of a cycle-stealing DMA I/O task executing concurrently with a set of CPU tasks with arbitrary release times and priority assignments. We reduced the running-time complexity of the recursive formula with a dynamic-programming technique. Consequently, the WCET prediction table constructed by a full evaluation of the dynamic-programming method can be used to bound the WCETs of all cycle-stealing DMA I/O tasks executing concurrently with the same set of CPU tasks. Our method of bounding the WCET of a cycle-stealing DMA I/O task is applicable on an embedded processor where each instruction begins with a B-cycle. This paper successfully provides the first solution for the real-time community to fully utilize the bandwidth of the I/O bus in a hard-real-time embedded system by allowing the concurrent execution of CPU tasks and cycle-stealing DMA I/O tasks.

REFERENCES

1. A. Colin and I. Puaut, "Worst case execution time analysis for a processor with branch prediction," *Journal of Real-Time Systems*, Vol. 18, 2000, pp. 249-274.
2. Motorola, *MC68030 Enhanced 32-bit Microprocessor: User's Manual*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
3. J. Engblom, A. Ermedahl, M. Sjoedin, J. Gubstafsson, and H. Hansson, "Worst-case execution-time analysis for embedded real-time systems," *Journal of Software Tools for Technology Transfer*, Vol. 4, 2001, pp. 437-455.
4. J. Engblom and A. Ermedahl, "Pipeline timing analysis using a trace-driven simulator," in *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, 1999, pp. 88-95.
5. J. Engblom and A. Ermedahl, "Modeling complex flows for worst-case execution time analysis," in *Proceedings of the 21st Real-Time System Symposium*, 2000, pp. 163-174.
6. C. Ferdinand, F. Martin, and R. Wilhelm. "Applying compiler techniques to cache behavior prediction," in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems*, 1997, pp. 37-46.
7. C. Ferdinand and R. Wilhelm, "Efficient and precise cache behavior prediction for real-time systems," *Journal of Real-Time Systems*, Vol. 17, 1999, pp. 131-181.
8. R. Gupta and P. Gopinath, "Correlation analysis techniques for refining execution time estimates of real-time applications," in *Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software*, 1994, pp. 54-58.
9. J. Hahn, R. Ha, S. L. Min, and J. W. S. Liu, "Analysis of worst case DMA response time in a fixed-priority bus arbitration protocol," *Journal of Real-Time Systems*, Vol. 23, 2002, pp. 209-238.
10. H. Hansson, H. Lawson, O. Bridal, C. Eriksson, S. Larsson, H. Lönn, and M. Strömberg, "BASEMENT: an architecture and methodology for distributed automotive real-time systems," *IEEE Transactions on Computers*, Vol. 46, 1997, pp. 1016-1027.
11. D. Harel and A. Naamad, "The STATEMATE semantics of statecharts," *ACM Transactions on Software Engineering Method*, Vol. 5, 1996, pp. 293-333.
12. C. Healy, R. Arnold, F. Mueller, D. Whalley, and M. Harmon, "Bounding pipeline and instruction cache performance," *IEEE Transactions on Computers*, Vol. 48, 1999, pp. 53-70.
13. T. Y. Huang, J. W. S. Liu, and D. Hull, "A method for bounding the effect of DMA I/O interference on program execution time," in *Proceedings of the 17th Real-Time System Symposium*, 1996, pp. 275-285.
14. K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *Proceedings of the 12th Real-Time System Symposium*, 1991, pp. 129-139.
15. S. K. Kim, R. Ha, and S. L. Min, "Analysis of the impacts of overestimation sources on the accuracy of worst case timing analysis," in *Proceedings of the 20th Real-Time System Symposium*, 1999, pp. 22-31.
16. Y. T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, 1995, pp. 456-561.

17. Y. T. S. Li, S. Malik, and A. Wolfe, "Cache modeling for real-time software: beyond direct mapped instruction caches," in *Proceedings of the 17th Real-Time System Symposium*, 1996, pp. 254-263.
18. S. S. Lim, J. H. Han, J. Kim, and S. L. Min, "A worst case timing analysis technique for multiple-issue machines," in *Proceedings of the 19th Real-Time System Symposium*, 1998, pp. 334-345.
19. C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard realtime environment," *Journal of the ACM*, Vol. 10, 1973, pp. 46-61.
20. J. W. S. Liu, W. K. Shih, K. J. Lin, R. Bettati, and J. Y. Chung, "Imprecise computations," *IEEE Proceedings*, Vol. 82, 1994, pp. 174-182.
21. T. Lundqvist and P. Stenström, "An integrated path and timing analysis method based on cycle-level symbolic execution," *Journal of Real-Time Systems*, Vol. 17, 1999, pp. 183-207.
22. T. Lundqvist and P. Stenström, "Timing anomalies in dynamically scheduled microprocessors," in *Proceedings of the 20th Real-Time System Symposium*, 1999, pp. 12-21.
23. F. Mueller, D. Whalley, and M. Harmon, "Predicting instruction cache behavior," in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems*, 1994.
24. G. Ottosson and M. Sjödin, "Worst-case execution time analysis for modern hardware architectures," in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1997.
25. C. Y. Park and A. C. Shaw, "Experiments with a program timing tool based on source-level timing schema," *IEEE Computer*, 1991, pp. 48-57.
26. P. Puschner and C. Koza, "Calculating the maximum execution time of real-time programs," *Journal of Real-Time Systems*, Vol. 1, 1989, pp. 159-176.
27. J. Richert, "Integration of mechatronic design tools with CAMEL, exemplified by vehicle convoy control design," in *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*, 1996, pp. 516-523.
28. L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, Vol. 39, 1990, pp. 1175-1185.
29. F. Stappert and P. Altenbernd, "Complete worst-case execution time analysis of straightline hard real-time programs," *Journal of Systems Architecture*, Vol. 46, 2000, pp. 339-355.
30. J. Sun, M. Gardner, and J. W. S. Liu, "Bounding completion times of jobs with arbitrary release times, variable execution times, and resource sharing," *IEEE Transactions on Software Engineering*, Vol. 23, 1997, pp. 603-615.
31. H. Theiling and C. Ferdinand, "Combining abstract interpretation and ILP for microarchitecture modelling and program path analysis," in *Proceedings of the 19th Real-Time System Symposium*, 1998, pp. 144-153.
32. H. Theiling, C. Ferdinand, and R. Wilhelm, "Fast and precise WCET prediction by separated cache and path analyses," *Journal of Real-Time Systems*, Vol. 18, 2000, pp. 157-179.
33. *The VMEbus Specification*, Motorola, 1985.



Tai-Yi Huang (黃泰一) received the B.S. degree in Computer Science and Information Engineering from National Taiwan University in 1991. He received both the M.S. and Ph.D. degrees from University of Illinois at Urbana-Champaign in Computer Science in 1994 and 1996, respectively. From 1996 to 2001, he was a software design engineer in Windows OS Kernel Performance Group, Microsoft Inc. Since February 2002, he has been an assistant professor with the Computer Science Department at National Tsing Hua University, Taiwan. He is currently the executive secretary of the Embedded Software Consortium, Ministry of Education, Taiwan. His research interests include low-power embedded systems, real-time operating systems, and high-performance clustered storages. He is a member of the IEEE and the ACM.



Chih-Chieh Chou (周智杰) received the B.S. degree in Computer Science from National Chung Cheng University, Taiwan and the M.S. degree in Computer Science from National Tsing Hua University, Taiwan. He is currently a software engineer at Trend Micro Enterprise, Taiwan. His projects concentrate on network security appliances.



Po-Yuan Chen (陳柏元) received the B.Sc and M.Sc. degrees from Computer Science Department, National Tsing Hua University, Taiwan, in 2002 and 2004, respectively. He is currently a Ph.D. student in Computer Science Department, National Tsing Hua University, Taiwan. His research interests include operating systems, real-time systems, hardware/software codesign, digital signal processor design, and very large scale integrated circuit design.