Schedulability Analysis of Preemptive and Nonpreemptive EDF on Partial Runtime-Reconfigurable FPGAs

NAN GUAN and QINGXU DENG
Northeastern University, China
ZONGHUA GU
Hong Kong University of Science and Technology, China
WENYAO XU
Zhejiang University, China
and
GE YU
Northeastern University, China

Field Programmable Gate Arrays (FPGAs) are very popular in today's embedded systems design, and Partial Runtime-Reconfigurable (PRTR) FPGAs allow HW tasks to be placed and removed dynamically at runtime. Hardware task scheduling on PRTR FPGAs brings many challenging issues to traditional real-time scheduling theory, which have not been adequately addressed by the research community compared to software task scheduling on CPUs. In this article, we consider the schedulability analysis problem of HW task scheduling on PRPR FPGAs. We derive utilization bounds for several variants of global preemptive/nonpreemptive EDF scheduling, and compare the performance of different utilization bound tests.

This work was partially sponsored by the National Natural Science Foundation of China under Grant No. 60773220, Hong Kong RGC CERG Grant No. 613506, the National High Technology Research and Development Program of China (863 Program) under Grant No. 2007AA01Z181, the Cultivation Fund of the Key Scientific and Technical Innovation Project of Ministry of Education of China under Grant No. 706016, the National Basic Research Program of China (973 Program) Grant No. 2006CB303000, and the National Natural Science Foundation of China under Grant No. 660503036.
Authors' addresses: N. Guan, Q. Deng, G. Yu, Institute of Computer Software, Northeastern University, Shenyang, China, 110004; email: guannan0609@hotmail.com, {dengqx, yuge}@mail.neu.edu.cn; Z. Gu, Department of Computer Science and Engineering, Hong Kong University of Science and Technology, China; email: zgu@cse.ust.hk; W. Xu, College of Electronic Engineering, Zhejiang University, Hangzhou, China, 310027; email: wenyao.xu@gmail.com.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Real-time scheduling, reconfigurable devices, FPGA

ACM Reference Format:

Guan, N., Deng, Q., Gu, Z., Xu, W., and Yu, G. 2008. Schedulability analysis of preemptive and nonpreemptive EDF on partial runtime-reconfigurable FPGAs. ACM Trans. Des. Autom. Electron. Syst. 13, 4, Article 56 (September 2008), 43 pages, DOI = 10.1145/1391962.1391964 http://doi.acm.org/10.1145/1391962.1391964

1. INTRODUCTION

Field Reconfigurable Gate Arrays (FPGAs) are very popular in today's embedded systems design due to their low-cost, high-performance and reconfigurability. FPGAs are inherently parallel, that is, two or more tasks can execute on a FPGA concurrently as long as they can both fit on it. *Partial Runtime-Reconfigurable* (PRTR) FPGAs, such as the Virtex family of FPGAs from Xilinx, allow part of the FPGA area to be reconfigured while the remainder continues to operate without interruption. In other words, HW tasks can be placed and removed dynamically at runtime. This is a very important and useful feature, since a FPGA is just used as an expensive and power-hungry ASIC without runtime reconfiguration. The task scheduler and placer must find an empty area to place a new task, and recycle the occupied area when a task is finished while making sure all task deadlines are met. In addition to the usual attributes such as computation time and deadline, each HW task has an additional attribute of area size that it occupies on the FPGA. Figure 1 shows the typical architecture consisting of a FPGA, a configuration controller, memory, and some other I/O devices. Besides the embedded software and data sections, the external memory stores the configurations for the FPGA.

Current commercial FPGA technology, for example, Xilinx Virtex-4, supports both 1D reconfiguration, where each task occupies a contiguous set of columns, and 2D reconfiguration, where each task occupies a rectangular area. Real-time scheduling for 1D reconfigurable FPGAs shares many similarities with global scheduling on identical multiprocessors [Carpenter et al. 2004], where all processors in the system have identical processing speed, and different task invocation instances may run on different processors. Similarly, a task can be relocated to a different position on the FPGA at runtime, with the associated reconfiguration overhead. But HW task scheduling on FPGA is a more general and difficult problem than multiprocessor scheduling, since each HW task may occupy a different area size on the FPGA while a SW task always occupies one and only one CPU. In fact, we can view multiprocessor scheduling as a special case of HW task scheduling on a 1D reconfigurable FPGA where all tasks have width equal to 1.

Similar to CPU scheduling, we can identify several approaches to FPGA scheduling:

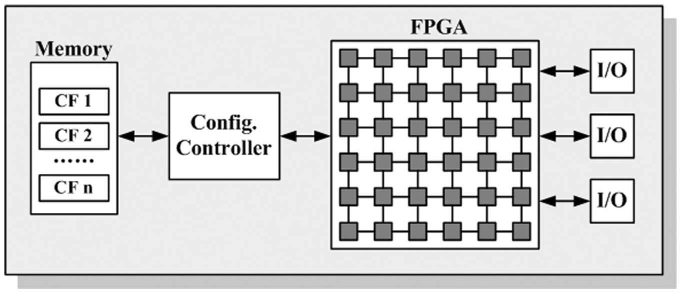


Fig. 1. Typical architecture of FPGA-based systems.

(1) For soft real-time tasks with unknown arrival times and execution times, online scheduling with optimization goals such as minimizing task rejection ratio while guaranteeing all tasks to meet their deadlines [Steiger et al. 2003] or deadline miss ratio [Lu et al. 2002] if no task is rejected.
(2) For hard real-time periodic tasks, static offline scheduling in the time interval with length equal to the hyper-period (least common multiple of all task periods).
(3) For hard real-time periodic tasks, priority-driven scheduling with well-known algorithms such as Rate Monotonic (RM) or Earliest Deadline First (EDF).

We focus on the third approach in this article. This article is an extension of our previous conference paper [Guan et al. 2007], which addressed preemptive EDF scheduling. The main enhancement of this article includes:

(1) A new pseudo-polynomial schedulability test condition for preemptive EDF scheduling.
(2) Additional analysis techniques for nonpreemptive EDF scheduling.
(3) Implementation of a prototype system for preemptive multitasking on FPGA.

Same as multiprocessor scheduling, there are two paradigms for FPGA scheduling: *partitioned* and *global* scheduling. Partitioned scheduling for FPGA has been studied by Danne and Platzner [2006b], where the FPGA is divided into several areas, and tasks are divided into several groups, each assigned to one area. Each task occupies one area while it is running regardless of its actual size. One key advantage of partitioned scheduling is its simplicity of analysis: the schedulability test of each area can be treated as a single processor problem. But partitioned scheduling may lead to poor resource utilization. For example, the taskset shown in Table I, is unschedulable using partitioned scheduling in Danne and Platzner [2006b], while it can be easily scheduled using global scheduling. On the other hand, schedulability analysis for global scheduling is more challenging and interesting, which is the topic of this paper.

Unlike CPU scheduling, where task context switch overhead is often small enough to be ignored, FPGA reconfiguration carries a significant overhead in the range of milliseconds that is proportional to the size of area being reconfigured. Each task invocation consists of two distinct stages: *reconfiguration* and

Table I. A Taskset with Low Utilization Yet Unschedulable with Partitioned EDF. δ is a Very Small Number

Task	C	D	T	A
τ_1	δ	10	10	$A(H) - 1$
τ_2	$10 - \delta$	10	10	1
τ_3	$5 + \delta$	10	10	1
τ_4	5	10	10	1

computation. We do not consider *configuration prefetch* [Li and Hauck 2002], a technique for hiding the large reconfiguration delay by allowing a task's configuration to be loaded on the FPGA sometime before the start of its actual computation. Instead, we assume there is no gap between a task's reconfiguration stage and execution stage. This allows us to add a task's reconfiguration time to its execution time as its overall execution time [Gu et al. 2007]. Since reconfiguration overhead on FPGAs is quite high, it is preferable to use scheduling algorithms that minimize the number of context-switches. As EDF generally leads to few task preemptions than static priority scheduling algorithms [Buttazzo 2005], we consider two variants of preemptive EDF scheduling algorithms in this paper. In addition, we also consider nonpreemptive EDF scheduling since it leads to fewer number of context switches than preemptive EDF scheduling.

For clarity of presentation in this article, we assume that the entire FPGA area is uniformly reconfigurable without any fixed area, and each task can be flexibly placed anywhere on the reconfigurable area as long as there is enough empty space to contain it. In reality, only part of the FPGA area is reconfigurable while the rest has fixed configuration. We use $A(H)$ to denote size of a FPGA H in terms of number of columns, which can be considered as size of the reconfigurable area if part of the FPGA is not reconfigurable. Inter-task communication for 1D reconfigurable FPGAs can be achieved with a logical shared memory that spans the entire width of the FPGA [Banerjee et al. 2007], while it is more difficult for 2D reconfigurable FPGAs, which we do not consider in this paper.

This article is structured as follows: we introduce related work in Section 2, and present the detailed theorem derivation process in Section 3. We present the terminology used in Section 3.1; the work-conserving concept for FPGA scheduling in Section 3.2, which forms the foundation for theorem derivations in later parts of this paper; utilization bound tests for preemptive and nonpreemptive EDF scheduling in Sections 3.3 and 3.4, respectively. In Section 4 we consider the placement strategy and the reconfiguration overhead issue. In Section 5, we present our HW prototype for preemptive multitasking system on a Virtex-4 FPGA. We present performance evaluation results in Section 6, and conclusions in Section 7.

2. RELATED WORK

2.1 Schedulability Analysis for Multiprocessors and FPGAs

For single-processor scheduling, there are mainly two approaches to schedulability analysis: utilization bound tests and response time analysis. Take fixed-priority Rate Monotonic (RM) scheduling for example. The well-known Liu and Layland utilization bound test [Liu and Layland 1973] states that a taskset with N tasks is schedulable if the total utilization does not exceed $N(2^{1/N} - 1)$. This is a sufficient but not necessary condition, and rejects some tasksets that are schedulable. Lehoczky et al. [1989] presented a polynomial-time algorithm for calculating a task's Worst-Case Response Time (WCRT) by performing processor demand analysis when the task and all other higher-priority tasks are initially released at time 0, the critical instant. A task is schedulable if its WCRT is less than its deadline, and the taskset is schedulable if all tasks are schedulable. This is a necessary and sufficient condition for schedulability.

For multiprocessor scheduling, an analogous algorithm for WCRT calculation does not exist since there may not be a critical instant anymore, that is, it is generally unknown what task release phase offsets cause the WCRT. Therefore, we are forced to rely on pessimistic utilization bound tests for schedulability analysis. As an example, Baker [2006a] presented a utilization bound for determining schedulability of a periodic taskset with fixed-priority scheduling on a multiprocessor platform, which rejects a significant fraction of tasksets that are actually schedulable. In order to gauge the tightness of this bound, Baker obtained a coarse upper bound on the fraction of the tasksets that might be schedulable by simulating system execution when all tasks are initially released at time 0. He stated that "this coarse bound is used because there is no known computationally feasible algorithm for determining with certainty whether or not each taskset is schedulable." Indeed, we will need to exhaustively simulate all possible task release offsets in order to determine schedulability of a taskset on a multiprocessor platform. Schedulability analysis for FPGAs faces the same problem, for example, Danne and Platzner [2006a] presented a utilization bound for schedulability analysis of global EDF scheduling on FPGAs, which also rejects a lot of feasible tasksets. This pessimism is the price we have to pay for making hard real-time guarantees.

For multiprocessor scheduling using preemptive EDF, several authors have presented utilization bound tests. Goossens et al. [2003] presented a utilization bound test, referred to as **GFB** in this paper, assuming that tasks have relative deadlines equal to the period. Baker [2003] presented another utilization bound test, referred to as **BAK1** in this paper, that can handle relative deadlines less than or equal to the period. Baker [2005a] extended **BAK1** to include tasks with post-period deadlines, and showed that EDF-US[1/2], which gives higher priority to tasks with utilizations above 1/2, is optimal. Bertogna et al. [2005] presented an improved test, referred to as **BCL** in this paper, and showed that **GFB** and **BAK1** are incomparable to each other, and each test can accept tasksets that the other test rejects. For tasksets with different timing characteristics, they have different performance in terms of acceptance ratio. Baker [2006b] further showed that all three tests, **GFB**, **BCL** and **BAK1**, are incomparable to each other, that is, no one consistently outperforms the others. **GFB** performs better than **BCL** if the taskset only consists of tasks with low time utilization (time-light tasks), while **BCL** performs better if there are tasks with high time utilization (time-heavy tasks) [Baker 2003]. Baker [2005c] further improved upon **BCL**, and presented another utilization bound test, referred to as **BAK2** in this paper, which combines **BCL** with the busy-interval analysis of **BAK1** to obtain a tighter bound than either method could achieve alone. Recently, Baruah [2007] developed a pseudo-polynomial run-time test, referred to as **BAR**, which can efficiently account for the "carry-in." It is shown that **BAR** can outperform the previous tests when the number of tasks is significantly greater than the number of processors, or when parameters of different tasks differ by several orders of magnitude. Since **BAR** has higher complexity, the following steps are suggested to determine schedulability of a taskset: first apply the polynomial-time tests, and then apply the pseudo-polynomial **BAR** test only if the system is determined to be unschedulable by any of these tests.

There are two possible variants of global preemptive EDF scheduling for FPGAs, as discussed in Danne and Platzner [2006a]. Let $A(H)$ denote the total number of columns of the FPGA, and A_i denote the area occupied by the task instance J_i.

Definition 2.1 (*EDF-FkF*). Let Q be the queue of all active task instances sorted by nondecreasing deadlines (sorted by release times if deadlines are the same). Let J_i denote the ith task instance in Q. The scheduling algorithm *EDF-First-k-Fit* (EDF-FkF) selects at any time the first k task instances R of Q for execution, with the largest k for which $\sum_{J_i \in R} A_i \leq A(H)$ holds.

Definition 2.2 (*EDF-NF*). Let Q be the queue of all active task instances sorted by nondecreasing deadlines (sorted by release times if deadlines are the same). Let J_i denote the ith task instance in Q. The scheduling algorithm *EDF-Next-Fit* (EDF-NF) determines the set of running tasks R with the following algorithm: start with an empty set R and visit all active task instances $J_i \in Q$ in order of non-decreasing deadlines. Add J_i to R if and only if $\sum_{J_k \in R \cup J_i} A_k \leq A(H)$.

Perhaps EDF-NF is a misnomer because it does not process tasks in strict deadline order. Danne and Platzner [2006a] showed that EDF-NF is superior to EDF-FkF in the sense that if a taskset Γ is schedulable using EDF-FkF, then it is also schedulable using EDF-NF. Intuitively, EDF-FkF must process tasks in strict deadline order, while EDF-NF can process tasks out of deadline order by skipping any tasks that cannot fit on the FPGA and processing a task with a longer deadline but can fit on the FPGA. Therefore, EDF-FkF may leave some HW resources idle if there are ready task instances that can fit on the FPGA but are blocked by a task instance J_k that cannot fit on the FPGA, while EDF-NF can exploit these idle resource by skipping J_k and place the task instances behind it in the queue. Our definitions are a little different from Danne and Platzner [2006a] by adding the constraint that if two task instances have the same deadline, then the one with earlier release time has the higher priority. This is a necessary condition for EDF-NF to be always superior to EDF-FkF, and it agrees with the practical implementation of priority queues.

For EDF-based global scheduling on FPGAs, Danne and Platzner [2006a] presented a utilization bound test (referred to as **DP**) based on Goossens et al. [2003] (**GFB**). Danne and Platzner [2006b] also discussed partitioned scheduling for FPGAs, where each task is restricted to executing on a given partition of the FPGA, and task execution on each partition is serialized, so the problem is reduced to task allocation followed by single-processor schedulability analysis. We focus on global EDF scheduling in this article, which is a more challenging and interesting problem.

Compared with preemptive EDF, multiprocessor scheduling with non-preemptive EDF has drawn much less attention from the research community, since the response time performance of nonpreemptive EDF is generally worse than preemptive EDF. Baruah [2006] presented a utilization bound test for global nonpreemptive EDF scheduling on identical multiprocessor. His method is similar to **GFB**, but takes into account blocking time caused by non-preemption. Similar to preemptive EDF scheduling, we can define two variants of global non-preemptive EDF for FPGAs: NP-EDF-FkF and NP-EDF-NF. In this article, we only consider schedulability analysis of NP-EDF-FkF.

2.2 HW Multitasking on FPGAs

HW is inherently parallel, and we can have multiple HW tasks executing in true concurrency on a FPGA (as opposed to interleaving concurrency on a CPU), even if we do not consider dynamic reconfiguration. It is desirable to provide a high-level API to hide the complexities of HW multitasking from the application programmer. A number of operating systems for FPGAs have been developed for this purpose, for example, Hybrid Threads [Agron et al. 2006], where each HW task is configured at a fixed location on the FPGA, and no dynamic reconfiguration is allowed; that is, time-multiplexing of different HW tasks at the same FPGA location is not allowed. Therefore, a taskset is schedulable on the FPGA if all tasks can fit on it, and each task's execution time is less than its deadline. This approach eliminates most of the complexities associated with real-time scheduling, but may be inefficient if some HW tasks with low utilization occupies too much HW space. Some operating systems for dynamically reconfigurable FPGAs have also been developed, for example, ReconOS [Lubbers and Platzner 2007] and the work of Steiger et al. [2004]. These operating systems manage online HW task queuing, dispatching and placement on the FPGA, and typically do not allow task preemption.

In this section, we focus on related work on implementing preemptive multitasking on FPGAs,[1] which is more relevant to this paper. To implement preemptive multitasking on a FPGA, we need to be able to suspend the execution of an ongoing task, save its context, and restore the context of another task that was previously interrupted. The state information of a HW task consists of values of its state registers. There are mainly two ways of doing this, as discussed in Sections 2.2.1 and 2.2.2.

2.2.1 *Bitstream Readback via Configuration Port Access (CPA).* Xilinx-II Pro and later products provide a *Internal Configuration Access Port* (ICAP), a parallel port for the on-chip processor hardcore to configure the task frames. One approach [Kalte and Porrmann 2005] of saving and restoring a task's context is to read back the configuration bitstream through ICAP, parse and

[1]Non-preemptive multitasking is already well-supported by current commercial products.

filter it to extract values of the task state registers and save them using a bitstream manipulation tool such as JBits [Guccione et al. 2000], PARBIT [Hortaa et al. 2002], JPG [Raghavan and Sutton 2002], JBitsCopy [Dyer et al. 2002] and BitLinker [Silva and Ferreira 2006]. The state information is typically a small fraction ($<$10%) of the size of the entire configuration bitstream. When the task needs to be resumed later, the state information is merged with the configuration bitstream and downloaded to the FPGA through its configuration port. This approach often incurs excessive delays due to readback and filtering of the configuration bitstream. The time it takes to read back a task's bitstream is proportional to its size in terms of the number of FPGA columns it occupies. Here are some representative examples of this approach:

—Claus et al. [2007] developed a framework for reducing reconfiguration delays by using the combitgen tool to generate efficient bitstreams, along with a new ICAP controller connected directly to the high-speed PLB (Processor Local Bus) as a master and equipped with DMA (Direct Memory Access). The configuration speed can be 20 times faster compared to the OPBHWICAP approach from Xilinx, where the ICAP controller is a slave attachment on the low-speed OPB (On-Chip Peripheral Bus). The reconfiguration flow is based on EAPR (Early Access Partial Reconfiguration) from Xilinx.
—Kalte and Porrmann [2006] developed the REPLICA2Pro (Relocation per online Configuration Alteration in Virtex-2/-Pro) filter for performing HW task relocations at runtime. It allows relocation of pre-synthesized HW tasks along the horizontal communication infrastructure by manipulating the task's bitstream data during the bitstream download process. The filter parses the bitstream during the download process and replaces the column addresses within the bitstream according to the desired location of the HW task.
—Danne et al. [2006] developed an all-HW runtime system for server-based preemptive multitasking on the CELOXICA RC203 board, which consists of a Xilinx Virtex-II 3000 FPGA, a CPLD, a flash memory card and two banks of SRAM. CPLD acts as a reconfiguration controller that reads/writes configuration bitstreams from/to the flash memory card on request of the logic implemented in the FPGA. Since partial reconfiguration is not supported, the entire reconfigurable area is configured every time, including the user tasks and the runtime system. The Save System Context (SSC) phase saves the context of user tasks and runtime system to the external SRAM, and Restore System Context (RSC) phase restores the context from the external SRAM. Although the Virtex II 3000 could be reconfigured within 20 ms via the SelectMap interface, the speed of the SmartMedia flash memory card limits the reconfiguration time on the RC203 board to 180 ms.

2.2.2 *Task Specific Access Structures (TSAS).* A more efficient approach is to use *Task Specific Access Structures* (TSAS) [Kalte and Porrmann 2005] to avoid reading back and filtering the bitstream by adding some HW resources to add read/write interfaces to task state registers, so that they can be accessed directly by the configuration circuitry during context saving and restoring. There are several ways of implementing this interface, for example, as a scan chain or an addressable RAM structure on the FPGA. When the task needs to be resumed later, the configuration bitstream is downloaded and the state register values are restored. State saving and restoring are all on-chip. This approach is much faster than the bitstream readback approach, since task registers can be read/written directly, while the CPA approach involves reading back, parse and filter the entire bitstream of the HW task. The disadvantage is that additional HW resources must be allocated to implement the register read/write interface. One way to reduce the number of registers is to implement a shutdown process for each task [Kalte and Porrmann 2005], which we do not consider in this paper.

The majority of related work falls into the CPA category, but several authors have developed techniques for the TSAS category:

—Koch et al. [2007] developed an efficient technique for extracting and restoring the state of a HW module with low delay and HW resource overhead, which can be used both for preemptive scheduling and for HW checkpointing. Several HW mechanisms are developed, including memory-mapped state access (MM), scan chain based state access (SC), and shadow based scan state access (SHC).

—Jovanovic et al. [2007] developed a HW task preemption mechanism based on scan-path register structures. The main advantage of the proposed method is that it allows context saving and restoring of a HW task without freezing other tasks during preemption phases.

3. DERIVATION OF UTILIZATION BOUND TESTS

3.1 Problem Definition and Terminology

We consider a 1D reconfigurable FPGA H with $A(H)$ columns and a taskset Γ consisting of N periodic or sporadic tasks to be scheduled on H. Each task $\tau_k = (C_k, D_k, T_k, A_k), k \in 1, \dots, N$ is characterized by its execution time C_k, period or minimum inter-arrival time T_k , a relative deadline D_k and an area size A_k, which represents the number of contiguous columns that τ_k occupies. Without losing generality, we set $C_k < D_k$ and $C_k < T_k$. A taskset Γ has *pre-period deadlines* if for each task $\tau_k \in \Gamma$, the relative deadline is not larger than its period ($D_k \leq T_k$); a taskset has *post-period deadlines* if there is some $\tau_k \in \Gamma$, the relative deadline is larger than its period ($D_k > T_k$). A *task* τ_k consists of a sequence of *task instances* J_k^j (J_k), each characterized by its release time r_k^j (r_k) and finish time f_k^j (f_k), and d_k^j (d_k) denote its absolute deadline.

We define two notions of *workload* done by a task to measure its progress:

—The *time workload* $W_i^T(t-\delta, t)$ done by task τ_i over a time interval $[t-\delta, t)$ is the sum of the lengths of all subintervals during which a task instance J_i^j executes. The *total time workload* $W^T(t-\delta, t)$ done in a time interval $[t-\delta, t)$ is the sum of time work of all task instances in the interval.

—The *time-by-area workload* $W_i^S(t-\delta, t)$ done by task τ_i over a time interval $[t-\delta, t)$ is the product of the time work of the interval and the area of the task: $W_i^S(t-\delta, t) = W_i^T(t-\delta, t) \times A_i$. The *total time-by-area workload* $W^S(t-\delta, t)$

done in a time interval $[t - \delta, t)$ is the sum of time-by-area work of all task instances in the interval.

For multiprocessor scheduling, each task occupies one processor when it is executing, so we evaluate the work done by a task instance only based on its execution time, which corresponds to the *time workload* concept. But for FPGA scheduling, each task can occupy a different area size, so we define *time-by-area workload* to evaluate the work done by a task instance based on its area size in addition to its execution time.

As in Danne and Platzner [2006a], we define two utilization metrics: the *time utilization* of a task τ_i is defined as

$$U^T(\tau_i) = C_i/T_i \tag{1}$$

and for the complete taskset Γ as

$$U^T(\Gamma) = \sum_{\tau_i \in \Gamma} U^T(\tau_i). \tag{2}$$

The *time-by-area utilization* of a task τ_i is defined as

$$U^S(\tau_i) = U^T(\tau_i)A_i \tag{3}$$

and for the complete taskset Γ as

$$U^S(\Gamma) = \sum_{\tau_i \in \Gamma} U^S(\tau_i). \tag{4}$$

Similarly, we define two density metrics: the *time density* of a task τ_i is defined as

$$\delta(\tau_i) = C_i/D_i \tag{5}$$

and for the complete taskset Γ as

$$\delta^T(\Gamma) = \sum_{\tau_i \in \Gamma} \delta^T(\tau_i). \tag{6}$$

The *time-by-area density* of a task τ_i is defined as

$$\delta^S(\tau_i) = \delta^T(\tau_i)A_i \tag{7}$$

and for the complete taskset Γ as

$$\delta^S(\Gamma) = \sum_{\tau_i \in \Gamma} \delta^S(\tau_i). \tag{8}$$

For nonpreemptive scheduling, we introduce two additional utilization concepts.

The *time blocked utilization* of a task τ_i is defined as

$$V^T(\tau_i) = \begin{cases} C_i/(D_i - C_{max}) & \text{if } D_i > C_{max} \\ \infty & \text{if } D_i \leq C_{max} \end{cases} \tag{9}$$

and for the complete taskset Γ as

$$V^T(\Gamma) = \sum_{\tau_i \in \Gamma} V^T(\tau_i). \tag{10}$$

The *time-by-area blocked utilization* of a task τ_i is defined as

$$V^S(\tau_i) = V^T(\tau_i)A_i \tag{11}$$

and for the complete taskset Γ as

$$V^S(\Gamma) = \sum_{\tau_i \in \Gamma} V^S(\tau_i). \tag{12}$$

Loosely speaking, $V^T(\tau_i)$ and $V^S(\tau_i)$ play the roles of $\delta^T(\tau_i)$ and $\delta^S(\tau_i)$ of task τ_i, respectively, when τ_i is subject to blocking for an amount of time of the maximum computation time of all tasks C_{max}. Our definition of $V^T(\tau_i)$ is a little different from the one in Baruah [2006], where $V^T(\tau_i) = C_i/(T_i - C_{max})$. The reason that we replace T_i by D_i is to consider $D_i \leq T_i$ for tasksets with pre-period deadlines.

The *interference* $I_k(t-\delta, t)$ suffered by a task τ_k over a time interval $[t-\delta, t)$ is the sum of the lengths of all the sub-intervals in $[t-\delta, t)$ during which a task τ_k is preempted. The *interference contribution* $I_{i,k}(t-\delta, t)$ of a task τ_i to $I_k(t-\delta, t)$ is the amount of interference caused by τ_i to τ_k.

The *block busy interval* is any time interval during which the idle area of the FPGA is less than or equal to $A(H) - A_{max} + 1$, where A_{max} is the largest area size of all tasks in the taskset Γ.

The *block busy time* $B(t-\delta, t)$ of a time interval $[t-\delta, t)$ is the sum of the length of all block busy intervals in $[t-\delta, t)$. The *block busy time* $B_i(t-\delta, t)$ *of task* τ_i is the total amount of time during which τ_i is executing in the block busy time $B(t-\delta, t)$.

The τ_k*-busy interval* is the interval during which τ_k always has active instances executing or waiting to execute. For each task τ_k, a *unique maximal* τ_k*-busy interval* exists, since at the start of the system it is not τ_k-busy.

3.2 Work-Conserving Concept for FPGA

If a multiprocessor CPU scheduling algorithm is *work-conserving*, it means that it never leaves any processor idle when there are any task instances in the ready queue. For example, global EDF scheduling is *work-conserving* on identical multiprocessor systems [Goossens et al. 2003]. This fact is the basis for derivation of the utilization bound tests of global EDF scheduling.

Unlike multiprocessor CPU scheduling, it is possible for parts of the FPGA area to be idle when there are task instances in the ready queue, because the idle area may not be large enough to fit any of the task instances in the ready queue. Therefore, we need an extended notion of work-conserving algorithms. Danne and Platzner [2006a] defined the concept of α*-work-conserving* scheduling algorithms, which guarantee that at least $\alpha \times A(H)$ area of the FPGA is occupied (busy) when there are task instances in the ready queue. Next, we present two definitions of α-work-conserving algorithms, and the correct α values for EDF-FkF and EDF-NF.

Definition 3.1 (*Global-α-work-conserving*). A scheduling algorithm is *global-α-work-conserving* if at least $\alpha \times A(H)$ area of the FPGA H with total area $A(H)$ is occupied whenever there are task instances in the ready queue.

LEMMA 3.2. *EDF-FkF, EDF-NF and NP-EDF-FkF are all global-α-work-conserving algorithms, and $\alpha = 1 - (A_{max} - 1)/A(H)$, where A_{max} is the largest area of all tasks.*

PROOF. Assume more than $A_{max} - 1$ of the FPGA area is idle in an overload situation. Since the area (number of columns) is an integer value, the idle area is equal to or larger than A_{max}. Using any one of EDF-FkF, EDF-NF or NP-EDF-FkF, the next task instance in the waiting queue can start to execute immediately since its area size is not larger than A_{max}. Hence the assumption must be wrong. □

Our definition of global-α-work-conserving is similar to the α-work-conserving concept in Danne and Platzner [2006a], in which a task τ_i's area A_i is assumed to be a real number instead of an integer for the purpose of generality, and α is determined to be $1 - A_{max}/A(H)$. We believe it is more reasonable to assume A_i is an integer, since it refers to the number of columns that τ_i occupies. In this case, α should be $1 - (A_{max} - 1)/A(H)$. Intuitively, if an area of size A_{max} is idle, then it is still possible to fit another task on the FPGA; but if an area of size $(A_{max} - 1)$ is idle, then it may not be possible to fit another task. Therefore, in an overload situation, that is, when the task queue is not empty, at least $A(H) - (A_{max} - 1)$ area of the FPGA must be occupied, hence $\alpha = 1 - (A_{max} - 1)/A(H)$.

Danne and Platzner [2006a] derived a schedulability condition for periodic tasksets based on **GBF**. Since **GBF** has been generalized to the case of sporadic tasksets with constrained deadlines ($D_i \leq T_i$), and with the integer task area assumption, we can easily generalize Danne and Platzner [2006a]'s test condition.

THEOREM 3.3. ***(DP)*** *Any periodic taskset Γ can be feasibly scheduled by EDF-FkF on a FPGA H with area A(H) having $A(H) \geq A_{max}$ if:*

$$\forall T_k \in T : \delta^S(\Gamma) \leq (A(H) - A_{max} + 1) \times (1 - \delta^T(T_k)) + \delta^S(T_k) \tag{13}$$

where A_{max} is the largest area of all tasks in Γ respectively.

Next, we define another notion of work-conserving algorithm in order to obtain tighter utilization bounds:

Definition 3.4 (*Interval-α-work-conserving*). A scheduling algorithm is *interval-α-work-conserving* during a time interval $[a, b)$ if at least $\alpha \times A(H)$ area size of the FPGA H with total area size $A(H)$ is occupied during $[a, b)$.

A global-α-work-conserving algorithm guarantees a lower bound of time-by-area utilization whenever there are task instances in the ready queue, but an interval-α-work-conserving algorithm only guarantees a lower bound of time-by-area utilization during certain time intervals. We define this concept in order to obtain a tighter α bound for EDF-NF:

LEMMA 3.5. *EDF-NF is an interval α-work-conserving algorithm with*

$$\alpha = 1 - (A_k - 1)/A(H) \tag{14}$$

in any time interval during which the task instance J_k is in the ready queue.

In EDF-NF, when a task instance cannot fit on the idle area of the FPGA, we can first allocate some other task instance with a longer deadline and smaller area that can fit on the FPGA. Therefore, at least $(A(H) - (A_k - 1))$ area of the FPGA must be occupied when a task instance J_k with area A_k is in the ready queue. But in EDF-FkF, we must allocate task instances in strict deadline order, therefore a task instance with a large area can block other task instances behind it in the wait queue from being allocated, so we must be pessimistic and use A_{max} instead of A_k.

3.3 Utilization Bound Test for Preemptive EDF

In this section, we derive utilization bound tests for preemptive EDF. We first derive a utilization bound test (**GDG-1**) with complexity $O(n^2)$ for EDF-NF and tasksets with preperiod deadlines, then a test (**GDG-2**) with complexity $O(n^3)$ for EDF-FkF and tasksets with post-period, and at last a test (**GDG-3**) with pseudo-polynomial complexity for EDF-NF and tasksets with pre-period. deadlines.

3.3.1 *EDF-NF for Tasksets with Pre-Period Deadlines.* The derivation is based on the utilization bound test of multiprocessor scheduling [Bertogna et al. 2005] (referred to as **BCL**). **BCL** is derived by analyzing the upper bound of the interference time suffered by a given task during its execution. For a task τ_k to meet its deadline, the total interference suffered by it must not be larger than its slack $D_k - C_k$.

If the interference that τ_i causes to τ_k in the time interval $[r_k^j, d_k^j)$ is greater than $D_k - C_k$, then it is sufficient to only consider the portion $D_k - C_k$ in the response time calculation of task τ_k. This is because if τ_k can finish its work during $[r_k^j, d_k^j)$, then the portion of τ_i's workload that exceeds $D_k - C_k$ must be executed in parallel with τ_k and does not contribute to τ_k's interference.

Here are the key steps of the derivation: suppose a instance J_k^j of task τ_k misses its deadline d_k^j, then we can find the lower bound of the interference I_k that the task instance must suffer in the interval $[r_k^j, d_k^j)$ to cause the deadline miss. Since the precise interference in any interval is impossible to obtain, we can derive an upper bound of the interference using the workload in the interval.

The worst-case interference suffered by task τ_k is $I_k^* = \max_j(I_k(r_k^j, d_k^j)) = I_k(r_k^{j*}, d_k^{j*})$, where j^* is the task instance in which the total interference is maximal. We also define $I_{i,k}^* = I_{i,k}(r_k^{j*}, d_k^{j*})$.

Lemma 3.6. *The taskset is schedulable if the following condition is true for each τ_k:*

$$\sum_{i \neq k} A_i \min(I_{i,k}^*, D_k - C_k) \leq (A(H) - A_k + 1)(D_k - C_k). \tag{15}$$

Proof. We use proof by contradiction. Suppose the taskset is not schedulable, then there must exist a task instance J_k^j that misses its deadline at time t.

The total interference on the task should be larger than the slack time of τ_k, so we have:

$$I_k(t-\delta, t) > D_k - C_k. \tag{16}$$

Let $x = D_k - C_k$, $\xi = \sum_{i:I_{i,k}\geq x} A_i$ and $\varpi(i,x) = \sum_i A_i \min(I_{i,k}(t-\delta,t), x)$, so we have

$$\varpi(i,x) = \xi x + \sum_{i:I_{i,k}<x} A_i I_{i,k}(t-\delta,t). \tag{17}$$

Let $A_{bnd} = (A(H) - A_k + 1)$. Based on the value of ξ, one of the two cases must be true:

—Case 1: $\xi > A_{bnd}$. Obviously, it follows that

$$\varpi(i,x) > (A(H) - A_k + 1)(D_k - C_k), \tag{18}$$

which causes a contradiction with the lemma.

—Case 2: $\xi \leq A_{bnd}$. The time-by-area work done by all tasks in $\tau_k's$ interference interval of $[t-\delta,t)$ is $\sum_{i=1}^{N} A_i \times I_{i,k}(t-\delta,t)$. From Lemma 3.5, we know that for EDF-NF, the occupied area cannot be less than $A(H) - A_k + 1$ in any given time point when τ_k is in the ready queue. Therefore, the time-by-area work done by all the tasks in $\tau_k's$ interference is no less than $(A(H) - A_k + 1)I_k(t-\delta,t)$, i.e.

$$\sum_{i:I_{i,k}\geq x} A_i I_{i,k}(t-\delta,t) + \sum_{i:I_{i,k}<x} A_i I_{i,k}(t-\delta,t) \geq (A(H) - A_k + 1)I_k(t-\delta,t). \tag{19}$$

By Equation (17) and (19), we have

$$\varpi(i,x) \geq \xi x + A_{bnd} I_k(t-\delta,t) - \sum_{i:I_{i,k}\geq x} A_i I_{i,k}(t-\delta,t) \tag{20}$$

Because $I_k(t-\delta,t) > I_{i,k}(t-\delta,t)$, so we have

$$\varpi(i,x) > \xi x + A_{bnd} I_k(t-\delta,t) - \xi I_k(t-\delta,t) \tag{21}$$

Because $\xi \leq A_{bnd}$ and $x \leq I_k(t-\delta,t)$, we have

$$\varpi(i,x) > A_{bnd} * x \tag{22}$$

i.e.

$$\sum_{i=1}^{N} A_i \min(I_{i,k}(t-\delta,t), x) > (A(H) - A_k + 1)(D_k - C_k) \tag{23}$$

which also contradicts the lemma, so the assumption cannot hold. □

Figure 2 illustrates Lemma 3.6. The white rectangles represent execution of the task instance J_k that we want to check, and the gray rectangles represent execution of other task instances. If we can guarantee the whole time-by-area work done by these task instances to be less than the total size of the shadowed area, then the interference certainly cannot cause J_k to meet its deadline. If the interference contribution of some task instance is larger than $D_k - C_k$, we only need to consider the $(D_k - C_k)$ part, since the rest of it must be executed in parallel with J_k and does not contribute to J_k's interference.

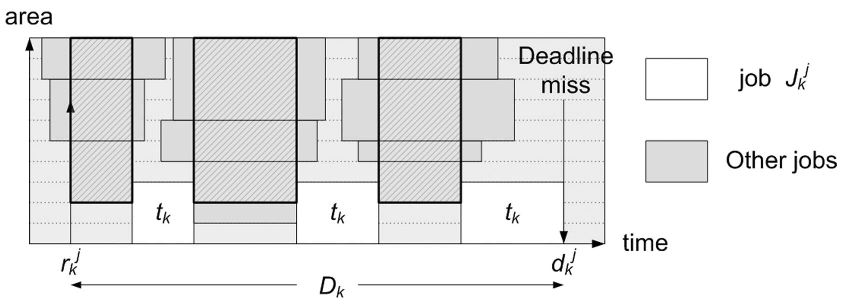


Fig. 2. Illustration of Lemma 3.6.

To apply the schedulability test of Lemma 3.6, the most straightforward approach is to compute the interference $I_{i,k}(r_k^j, d_k^j)$ for each task τ_i in every interval $[r_k^j, d_k^j]$ until the end of the hyper-period. This is infeasible without running a simulation of the system, so we derive an analytical upper bound on the interference. Since the interference $I_{i,k}(r_k^j, d_k^j)$ cannot be larger than the time work $W_i^T(r_k^j, d_k^j)$, the upper bound of $W_i^T(r_k^j, d_k^j)$ is also the upper bound of $I_{i,k}(r_k^j, d_k^j)$.

For multiprocessor scheduling, the worst case for the time workload is when the deadlines of task instance J_k^j and its interfering task τ_i are aligned, because in this case the number of instances of τ_i that interfere with τ_k is maximized [Baker 2003]. This conclusion also holds for FPGA scheduling, since the interference that J_k^j suffers from task τ_i in some given time interval is only related to their execution times, not to their area size.

LEMMA 3.7. *An upper bound on the time workload of τ_i in the interval $\left[r_k^j, d_k^j\right)$ is:*

$$W_i\left(r_k^j, d_k^j\right) \le N_i C_i + \min(C_i, \max(D_k - N_i T_i, 0))$$

in which $N_i = (\lfloor (D_k - D_i)/T_i \rfloor + 1)$.

The proof of Lemma 3.7 comes from Bertogna et al. [2005] and Baker [2003].

PROOF. The time workload $W_i(r_k^j, d_k^j)$ done by τ_i in an interval $[r_k^j, d_k^j)$ may include the following components:

(1) A portion of the execution times of the task instances that are released before r_k^j but unable to complete by that time, called the *carry-in*, as defined in Baker [2003].
(2) The full execution time C_i of the task instances released on or after r_k^j and completed by time t.
(3) A portion of the execution time of at most one task instance released at some time $d_k^j - \delta, 0 < \delta \le d_k^j - r_k^j$ but is unable to complete by time d_k^j.

We consider only the worst case mentioned above. In this situation, the time workload $W_i(r_k^j, d_k^j) \leq N_i C_i + \varepsilon_i(r_k^j, d_k^j)$, where $\varepsilon_i(r_k^j, d_k^j)$ is the carry-in of task τ_k in interval $[r_k^j, d_k^j)$. $N_i = (\lfloor (D_k - D_i)/T_i \rfloor + 1)$ is the maximum number of instances of τ_i that can be completely contained in $[r_k^j, d_k^j)$ in this situation.

Now we look for the upper bound of the carry-in. Obviously, We have $\varepsilon_i(r_k^j, d_k^j) \leq C_i$. When $T_i > D_i$ and $D_k - N_i T_i > 0$, we can see that the carry-in will never be larger than $D_k - N_i T_i$ in the worst-case situation mentioned above. □

We can then prove Theorem 3.8 based on Lemma 3.6 and Lemma 3.7.

THEOREM 3.8. ***(GDG-1)*** *A taskset Γ is schedulable using EDF-NF if, for each $\tau_k \in \Gamma$*

$$\sum_{i \neq k} A_i \min(\beta_i, D_k - C_k) < (A(H) - A_k + 1)(D_k - C_k) \tag{24}$$

where $\beta_i = N_i C_i + \min(C_i, \max(D_k - N_i T_i, 0))$

3.3.2 *EDF-FkF for Tasksets with Post-Period Deadlines.* From Lemma 3.2, we know that EDF-FkF is global-α-work-conserving, where $\alpha = (1 - (A_{max} - 1))/A(H))$. As a result, the lower bound of time-by-area utilization in the analysis interval $[t - \delta, t)$ for task τ_k is not related to the area size A_k of τ_k. This fact offers us an opportunity to take advantage of Baker's *problem window* extension [Baker 2003] to get a tighter bound of the carry-in. Furthermore, Baker's method can deal with tasksets with post-period deadlines as well as those with pre-period deadlines. Next, we will use a similar idea to derive the schedulability test of EDF-FkF for tasksets with arbitrary deadlines.

LEMMA 3.9. *If t is the time of τ_k's first deadline miss and $[t - \delta, t)$ is the corresponding maximal τ_k-busy interval then:*

$$I_k(t - \delta, t) > \delta - (\delta + T_k - D_k)C_k/T_k. \tag{25}$$

To bound the carry-in time contribution by τ_i, the maximal τ_k-busy interval is extended downward, i.e., keeping the right endpoint fixed and moving the left endpoint earlier, as far as possible while still maintaining a lower bound on block busy time as in Lemma 3.9.

Definition 3.10. *τ_k^λ-busy interval.* An interval $[t - \delta, t)$ is τ_k^λ-busy for a given constant $\lambda \geq C_k/T_k$ if $B(t - \delta, t) > \delta - \lambda(\delta + T_k - D_k)$. An interval $[t - \delta, t)$ is a maximal τ_k^λ-busy interval if it is τ_k^λ-busy and there is no $\delta' > \delta$ such that $[t - \delta', t)$ is also τ_k^λ-busy.

LEMMA 3.11. *If t is the time of the first deadline miss of τ_k^λ and $\lambda \geq C_k/T_k$, then there is a unique maximal τ_k^λ-busy interval $[t - \overline{\delta}, t)$, and $\overline{\delta} \geq D_k$.*

We call the unique interval $[t - \overline{\delta}, t)$ guaranteed by Lemma 3.11 the *maximal τ_k^λ-busy interval*, denoted by $[t - \overline{\delta}, t)$. The next step is to find an upper bound on the time workload $W_i^T(t - \overline{\delta}, t)$ done by each task τ_i in a τ_k^λ-busy interval $[t - \overline{\delta}, t)$.

LEMMA 3.12. *If t is the time of the first deadline miss of task τ_k, and $\lambda \leq C_k/T_k$ and $[t-\overline{\delta}, t)$ is the corresponding τ_k^λ-busy interval, then for any task τ_i such that $i \neq k$*

$$\frac{W_i^T(t-\overline{\delta}, t)}{\overline{\delta}} \leq \beta_k^i(i), \tag{26}$$

where

$$\beta_k^\lambda(i) = \begin{cases} \max\left(\frac{C_i}{T_i}, \frac{C_i}{T_i}\left(1-\frac{D_i}{D_k}\right)+\frac{C_i}{D_k}\right) & \text{if } \frac{C_i}{T_i} \leq \lambda \\ \frac{C_i}{T_i} & \text{if } \frac{C_i}{T_i} > \lambda \wedge \lambda \geq \frac{C_i}{D_i} \\ \frac{C_i}{T_i} + \frac{C_i - \lambda D_i}{D_k} & \text{if } \frac{C_i}{T_i} > \lambda \wedge \lambda < \frac{C_i}{D_i} \end{cases} \tag{27}$$

The proof of Lemma 3.9, 3.11 and 3.12 can be found in Baker [2005c].

LEMMA 3.13. *If the interval $[t-\delta, t)$ is a block busy interval, then*

$$\forall i (A(H) - A_{max} + 1) B(t-\delta, t) \leq \sum_{i=1}^{N} A_i B_i(t-\delta, t), \tag{28}$$

where A_{max} is the largest area of all tasks in Γ.

PROOF. The time-by-area work done by all the tasks in a τ_k-busy interval $[t-\delta, t)$ is $\sum_{i=1}^{N} A_i B_i(t-\delta, t)$. By the concept of block busy interval, the occupied area cannot be less than $A(H) - A_{max} + 1$ in any given time point in $[t-\delta, t)$. Therefore, the time-by-area work done by all the tasks in a $\tau_k's$-busy interval is no less than $(A(H) - A_{max} + 1)B(t-\delta, t)$. □

LEMMA 3.14. *If the interval $[t-\delta, t)$ is block busy interval and $B(t-\delta, t) > x$, then*

$$\sum_{i=1}^{N} \min(B_i(t-\delta, t), x) > (A(H) - A_{max} + 1)x, \tag{29}$$

where A_{max} is the largest area of all tasks in Γ.

PROOF. Let $\xi = \sum_{i:B_i \geq x} A_i$, $A_{bnd} = A(H) - A_{max} + 1$. Let $\varpi(i, x) = \sum_i A_i \min(B_i(t-\delta, t), x)$, and $\varpi(i, x) = \xi x + \sum_{i:B_i<x} A_i B_i(t-\delta, t)$. According to the comparison of ξ and A_{bnd}, one of the two cases must be true:

(1) $\xi > A_{bnd}$. Obviously, it follows that $\varpi(i, x) > (A(H) - A_{max} + 1)x$.

(2) $\xi \leq A_{bnd}$. According to Lemma 3.13, we have

$$\varpi(i, x) \geq \xi x + A_{bnd} B(t-\delta, t) - \sum_{i:Bi \geq x} A_i B_i(t-\delta, t). \tag{30}$$

Because $B(t-\delta, t) \geq B_i(t-\delta, t)$, we have

$$\varpi(i, x) \geq \xi x + A_{bnd} B(t-\delta, t) - \sum_{i:B_i \geq x} A_i B(t-\delta, t), \tag{31}$$

that is,

$$\varpi(i, x) \geq \xi x + A_{bnd} B(t-\delta, t) - \xi B(t-\delta, t). \tag{32}$$

Because $\xi \leq A_{bnd}$ and $B(t-\delta, t) > x$, we have

$$\varpi(i, x) > A_{bnd}x, \tag{33}$$

that is,

$$\sum_{i=1}^{N} \min(B_i(t-\delta, t), x) > (A(H) - A_{max} + 1)x. \tag{34}$$

The lemma is proved. □

Lemma 3.15. *If the interval $[t-\overline{\delta}, t)$ is τ_k-busy, then we have:*

$$W^S(t-\overline{\delta}, t) > (A(H) - A_{max} + 1 - A_{min})B(t-\overline{\delta}, t) + A_{min}\overline{\delta}, \tag{35}$$

where A_{max} is the largest area of all tasks in Γ.

Proof. Let $A_{bnd} = A(H) - A_{max} + 1$. The interval $[t-\overline{\delta}, t)$ can be divided into two parts: the blocked part with length $B(t-\overline{\delta}, t)$, and the unblocked part with length $\overline{\delta} - B(t-\overline{\delta}, t)$. The time-by-area work done in $[t-\overline{\delta}, t)$ is sum of the work done in these two parts.

By the definition of blocking time, the work done in the blocked part is never less than $(A(H) - A_{max} + 1)B(t-\overline{\delta}, t)$. Since $[t-\overline{\delta}, t)$ is a τ_k-busy interval, the FPGA cannot be idle at any time, so there must be at least one task instance executing in the unblocked part. So the work done in the unblocked part is never less than $A_{min}(\overline{\delta} - B(t-\overline{\delta}, t))$.

It follows that

$$W^S(t-\overline{\delta}, t) > A_{bnd}B(t-\overline{\delta}, t) + A_{min}(\overline{\delta} - B(t-\overline{\delta}, t)), \tag{36}$$

so we have

$$W^S(t-\overline{\delta}, t) > (A(H) - A_{max} + 1 - A_{min})B(t-\overline{\delta}, t) + A_{min}\overline{\delta}. \tag{37}$$

The lemma is proved. □

Now we present Theorem 3.16 and its proof:

Theorem 3.16. ***(GDG-2)*** *Let $\beta_k^{\lambda}(i)$ be as defined as in Lemma 3.12 and let $\lambda_k = \lambda \max(1, T_k/D_k)$. A taskset Γ is schedulable using EDF-FkF if, for every task τ_k, there exists $\lambda \geq C_k/T_k$ such that one or more of the following conditions are satisfied:*

$$1) \sum_{i=1}^{N} A_i \min\left(\beta_k^{\lambda}(i), 1 - \lambda_k\right) < A_{bnd}(1 - \lambda_k) \tag{38}$$

$$2) \sum_{i=1}^{N} A_i \min\left(\beta_k^{\lambda}(i), 1\right) \leq (A_{bnd} - A_{min})(1 - \lambda_k) + A_{min} \tag{39}$$

where $A_{bnd} = A(H) - A_{max} + 1$, A_{max} and A_{min} is the largest and smallest area of all tasks in Γ, respectively.

Proof. We use proof by contradiction. Suppose the taskset Γ with a release time assignment r is not schedulable, then there must be some task τ_k that misses its deadline for the first time at time t. Let $[t - \overline{\delta}, t)$ be the τ_k^λ-busy interval guaranteed by Lemma 3.11. By the definition of τ_k^λ-busy,

$$\frac{B(t-\overline{\delta},t)}{\overline{\delta}} > 1 - \lambda + \lambda\frac{T_k - D_k}{\overline{\delta}}. \tag{40}$$

There are two cases:

(1) If $T_k \le D_k$, then the value of the expression on the right-hand side of the inequality above is nondecreasing with respect to $\overline{\delta}$, and since $\overline{\delta} \ge D_k$,

$$\frac{B(t-\overline{\delta},t)}{\overline{\delta}} \ge 1 - \lambda + \lambda\frac{T_k - D_k}{D_k} = 1 - \lambda\frac{T_k}{D_k}. \tag{41}$$

(2) If $T_k > D_k$, then the value of the expression on the right-hand side of Equation (40) is decreasing with respect to $\overline{\delta}$, and so

$$\frac{B(t-\overline{\delta},t)}{\overline{\delta}} \ge 1 - \lambda. \tag{42}$$

Since $\lambda_k = \lambda \max(1, \frac{T_k}{D_k})$, Equations (41) and (42) can be combined into

$$\frac{B(t-\overline{\delta},t)}{\overline{\delta}} \ge 1 - \lambda_k. \tag{43}$$

Since $[t - \overline{\delta}, t)$ is τ_k-busy, from Lemma 3.15,

$$W^S(t-\overline{\delta},t) > (A_{bnd} - A_{min})B(t-\overline{\delta},t) + A_{min}\overline{\delta}. \tag{44}$$

Since $W_i^T(t-\overline{\delta},t) \le \overline{\delta}$, we have

$$\sum_{i=1}^{N} A_i \min\left(W_i^T(t-\overline{\delta},t), \overline{\delta}\right) = W^S(t-\overline{\delta},t). \tag{45}$$

It follows from Equations (43), (44), and (45):

$$\sum_{i=1}^{N} A_i \min\left(\frac{W_i^T(t-\overline{\delta},t)}{\overline{\delta}}, 1\right) > (A_{bnd} - A_{min})(1-\lambda_k) + A_{min}. \tag{46}$$

It follows from Lemma 3.12 that

$$\sum_{i=1}^{N} A_i \min(\beta_k^\lambda, 1) > (A_{bnd} - A_{min})(1-\lambda_k) + A_{min}. \tag{47}$$

Therefore condition 2) of the theorem must be false. Next, we show that condition 1) must also be false.

By Lemma 3.14 with $x = (1-\lambda_k)\overline{\delta}$ and using Equation (43), it must hold that:

$$\sum_{i=1}^{N} A_i \min\left(\frac{B_i(t-\overline{\delta},t)}{\overline{\delta}}, 1-\lambda_k\right) > A_{bnd}(1-\lambda_k). \tag{48}$$

Combining Lemma 3.12 and Equation (48), and since $B_i(t-\overline{\delta},t) \le W_i(t-\overline{\delta},t)$, we have

$$\sum_{i=1}^{N} A_i \min\left(\beta_k^\lambda(i), 1-\lambda_k\right) > A_{bnd}(1-\lambda_k). \tag{49}$$

This contradicts condition 1) of the theorem.

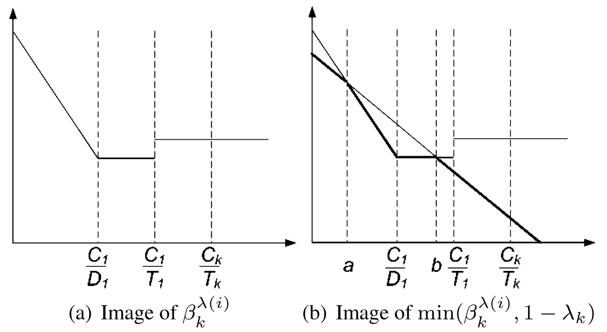


Fig. 3. Images of $\beta_k^{\lambda(i)}$ and $\min(\beta_k^{\lambda(i)}, 1-\lambda_k)$.

So the assumption cannot hold, and the taskset Γ must be schedulable. □

It seems that we should check every possible value of λ with **GDG-2**, which is not feasible in practice. Next, we will show how to implement **GDG-2** by checking a limited number of values of λ.

First, we consider Condition (38). By its definition, we know $\beta_k^{\lambda(i)}$ is a linear and monotonic function respect to λ in each segment of its domain. Figure 3(a) is one possible graph of $\beta_k^{\lambda(i)}$. Now we apply the $\min(\beta_k^{\lambda(i)}, 1-\lambda_k)$, and the result function is shown in Figure 3(b) (the bold line). We observe that the slope of the function may change at the points with $\beta_k^{\lambda(i)} = 1-\lambda_k$ (a and b), besides the original endpoints of each segment according to $\beta_k^{\lambda(i)}$'s definition. So $\min(\beta_k^{\lambda(i)}, 1-\lambda_k)$ is still a piecewise linear and monotonic function respect to λ. Since the sum of a set of linear and monotonic functions is also linear and monotonic, the left side of Condition (38) is a piecewise linear and monotonic function. To compare it with the right side of Condition (38), which is also linear and monotonic with respect to λ, we only need to consider the endpoints of the segments and the points at which $\beta_k^{\lambda(i)} = 1-\lambda_k$. And since $\beta_k^{\lambda(i)}$ is a constant with the first two cases of its definition, we only need consider the following points for λ:

—$\lambda = C_i/T_i,\ i = 1, ..., N$
—$\lambda = C_i/D_i,\ i = 1, ..., N$ if $D_i > T_i$
—$\lambda \models (C_i/T_i + (C_i - \lambda D_i)/D_k = 1-\lambda_k \wedge \lambda < C_i/T_i \wedge \lambda < C_i/D_i),\ i = 1, \ldots, N$

The consideration of Condition (39) is similar to (38), and we only need to consider same points as shown above. So the utilization bound test **GDG-2** is of complexity $O(N^3)$.

3.3.3 *A Pseudo-Polynomial Schedulability Test for EDF-NF.* In this section, we introduce a pseudo-polynomial schedulability test of EDF-NF for a taskset with post-period deadlines. This approach is based on the technique for multiprocessor systems by Baruah [2007], which only needs to consider carry-in from $m-1$ tasks, where m is the number of processors, while all the previous tests must account for carry-in from all tasks.

Consider any legal sequence of job requests of task system Γ that is scheduled with EDF-NF and misses a deadline. Suppose a job of task τ_k is the first one to

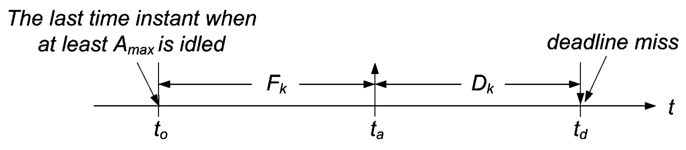


Fig. 4. Illustration of the notations.

miss a deadline, and this deadline miss occurs at time instant t_d, as shown in Figure 4. Let t_a denote this instance's arrival time: $t_a = t_d - D_k$. Let t_o denote the latest time instant $\leq t_a$ at which at least A_{max} area of the FPGA is idle. F_k is defined as $t_a - t_0$. For τ_k to miss its deadline, it must execute for strictly less than C_k time units during $[t_a, t_d)$, that is, the FPGA executes jobs other than τ_k's jobs for strictly more than $(D_k - C_k)$ during $[t_a, t_d)$. Let Z_k denote a collection of intervals, not necessarily contiguous, of cumulative length $(D_k - C_k)$ over $[t_a, t_d)$, during which the FPGA is executing task instances other than τ_k's instance. Let $I^T(\tau_i)$ denote the contribution of τ_i to the time work done during $[t_o, t_a) \bigcup Z_k$, and $I^S(\tau_i) = I^T(\tau_i)A_i$ denote the contribution of τ_i to the time-by-area work done during $[t_o, t_a) \bigcup Z_k$.

Since EDF-NF is interval-(1-$A_k - 1/A(H)$)-work-conserving during Z_k, at least $A(H) - A_k + 1$ area must be occupied at any time instant in Z_k. By the definition of t_o, at least $A(H) - A_{max} + 1$ area must be occupied at any time instant in $[t_o, t_a)$. The following condition must be satisfied if a deadline miss occurs:

$$\sum_{\tau_i \in \Gamma} I^S(\tau_i) > (A(H) - A_{max} + 1) \times F_k + (A(H) - A_k + 1) \times (D_k - C_k) \tag{50}$$

where $F_k = t_a - t_0$.

Equation (50) shows the necessary condition for a deadline miss to occur with EDF-NF scheduling. Conversely, in order for all deadlines of τ_k to be met, it is sufficient that Equation (50) is violated for all values of F_k. Lemma 3.17 follows immediately:

LEMMA 3.17. *Taskset Γ is schedulable with EDF-NF upon a FPGA with area A(H), if for all tasks τ_k and all $F_k \geq 0$:*

$$\sum_{\tau_i \in \Gamma} I^S(\tau_i) \leq (A(H) - A_{max} + 1) \times F_k + (A(H) - A_k + 1) \times (D_k - C_k) \tag{51}$$

Baruah [2007] has shown how to compute the upper bound of the $\sum_{\tau_i \in \Gamma} I^T(\tau_i)$ in the context of multiprocessor systems. We will compute $\sum_{\tau_i \in \Gamma} I^S(\tau_i)$ in the context of a FPGA system by following Baruah [2007]'s idea.

A task instance that arrives before t_o and has not completed execution by t_o is called a *carry-in instance*. The set of tasks with no carry-in instances is denoted as Γ_1 and their contribution in $\sum_{\tau_i \in \Gamma} I^S(\tau_i)$ is denoted as $I_1^S(\tau_i)$, while the set of tasks with carry-in instances is denoted as Γ_2 and their contributionin

$\sum_{\tau_i \in \Gamma} I^S(\tau_i)$ is denoted as $I_2^S(\tau_i)$. So we have

$$\sum_{\tau_i \in \Gamma} I^S(\tau_i) = \sum_{\tau_i \in \Gamma_1} I_1^S(\tau_i) + \sum_{\tau_i \in \Gamma_2} I_2^S(\tau_i). \tag{52}$$

Baruah [2007] showed that $I_1^T(\tau_i)$ is computed as follows:

$$I_1^T(\tau_i) = \begin{cases} \min(\mathrm{DBF}(\tau_i, F_k + D_k), F_k + D_k - C_k) & \text{if } i \neq k \\ \min(\mathrm{DBF}(\tau_i, F_k + D_k) - C_k, F_k) & \text{if } i = k, \end{cases} \tag{53}$$

in which $\mathrm{DBF}(\tau_i, t)$ is **demand bound function**. For any interval length t, the demand bound function $\mathrm{DBF}(\tau_i, t)$ of a sporadic task τ_i bounds the maximum cumulative execution requirement by instances of τ_i that both arrive in, and have deadlines within, any interval of length t. It has been shown [Baruah et al. 1990] that:

$$\mathrm{DBF}(\tau_i, t) = \max\left(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i\right). \tag{54}$$

And $I_2(\tau_i)$ is computed as follows:

$$I_2^T(\tau_i) = \begin{cases} \min(\mathrm{DBF'}(\tau_i, F_k + D_k), F_k + D_k - C_k) & \text{if } i \neq k \\ \min(\mathrm{DBF'}(\tau_i, F_k + D_k) - C_k, F_k) & \text{if } i = k, \end{cases} \tag{55}$$

in which $\mathrm{DBF'}(\tau_i, t)$ denotes the amount of work that can be contributed by τ_i over a continuous interval of length t, if some job of τ_i has its deadline at the very end of the interval and each job of τ_i executes during the C_i units immediately preceding its deadline, and it is shown that:

$$\mathrm{DBF'}(\tau_i, t) = \left\lfloor \frac{t}{T_i} \right\rfloor \times C_i + \min(C_i, t \mod T_i). \tag{56}$$

Let $I_{dif}^T(\tau_i)$ denote the difference between $I_2^T(\tau_i)$ and $I_1^T(\tau_i)$, and $I_{dif}^S(\tau_i)$ denote the difference between $I_2^S(\tau_i)$ and $I_1^S(\tau_i)$:

$$I_{dif}^S(\tau_i) = I_2^S(\tau_i) - I_1^S(\tau_i) = \left(I_2^T(\tau_i) - I_1^T(\tau_i)\right) A_i = I_{dif}^T(\tau_i) A_i. \tag{57}$$

And we have:

$$\sum_{\tau_i \in \Gamma} I^S(\tau_i) = \sum_{\tau_i \in \Gamma} I_1^S(\tau_i) + \sum_{\tau_i \in \Gamma_2} I_{dif}^S(\tau_i). \tag{58}$$

By definition of t_o, at most $(A(H) - A_{max} + 1)$ area of the FPGA is occupied at t_o. To identify the taskset with total area not exceeding $(A(H) - A_{max} + 1)$ and having largest $\sum_{\tau_i \in \Gamma_2} I_{dif}^S(\tau_i)$ is similar to bin-packing, which is known to be NP-hard. Algorithm 1 shows an efficient approach to obtaining an approximate upper bound B_{dif} for $\sum_{\tau_i \in \Gamma_2} I_{dif}^S(\tau_i)$, based on the following observation: By $I_{dif}^S(\tau_i) = I_{dif}^T(\tau_i) A_i$, we know that tasks with larger $I_{dif}^T(\tau_i)$ will lead to larger $I_{dif}^S(\tau_i)$ with same hardware resource area, so we sort the tasks in decreasing order of the value of $I_{dif}^T(\tau_i)$, as shown in Figure 5.

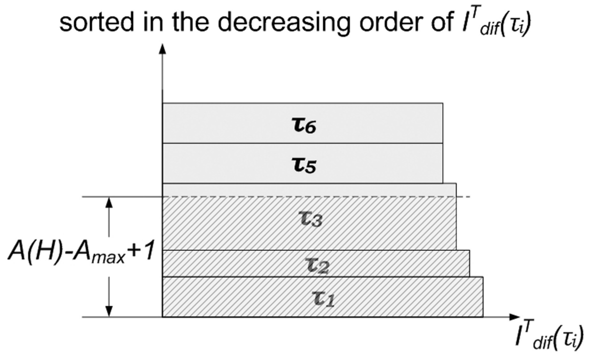


Fig. 5. Illustration of the algorithm in Algorithm 1.

Algorithm 1 Computing B_{dif}

```
$B_{dif} = 0$
$a = A(H) - A_{max} + 1$
TaskQueue Q = all tasks sorted in the decreasing order of $C_i$
$\tau_i$ = the first task in $Q$
WHILE (true)
    IF ($a > \tau_i.A$)
        $a = a - \tau_i.A$
        $B_{dif} = B_{dif} + \tau_i.I^S_{dif}$
        $\tau_i$ = next task in $Q$
    ELSE
        $B_{dif} = B_{dif} + \tau_i.I^T_{dif} * a$
        return $B_{dif}$
    ENDIF
ENDWHILE
```

Actually, τ_3 can not be active along with τ_1 and τ_2, but we can guarantee the $\sum_{\tau_i \in \Gamma_2} I^S_{dif}(\tau_i)$ will never exceed the total area of the shadowed part. So we have:

$$B_{dif} = I^S_{dif}(\tau_1) + I^S_{dif}(\tau_2) + I^T_{dif}(\tau_3)(A(H) - A_{max} + 1 - A_1 - A_2)$$

By solving the bin-packing problem, we know the exact maximum $\sum_{\tau_i \in \Gamma_2} I^S_{dif}(\tau_i)$ is $(I^S_{dif}(\tau_1) + I^S_{dif}(\tau_4) + I^S_{dif}(\tau_5))$, which is smaller than the area of the shadowed part B_{dif}.

By Lemma 3.17 and the computation of the upper bound of $\sum_{\tau_i \in \Gamma} I^S(\tau_i)$, we have the schedulability test condition for EDF-NF:

THEOREM 3.18. ***(GDG-3)*** *Taskset Γ is schedulable with EDF-NF upon a FPGA with area A(H), if for all tasks τ_k and all $F_k \geq 0$:*

$$\sum_{\tau_i \in \Gamma} I^S_1(\tau_i) + B_{dif} \leq (A(H) - A_{max} + 1) \times F_k + (A(H) - A_k + 1)) \times (D_k - C_k) \tag{59}$$

in which $I_1^S(\tau_i)$ is defined in Equation (53) and B_{dif} is obtained by the algorithm in Figure 1.

For given τ_k and F_k, it is easy to see that Condition (59) can be evaluated in time that is linear to task number n:

—Compute $I_1^T(\tau_i)$, $I_2^T(\tau_i)$ and $I_{dif}^T(\tau_i)$ for each τ_i: total time is $O(n)$.

—Sorting tasks in decreasing order of $I_{dif}^T(\tau_i)$ by *Radix Sort* [Knuth 1973] and computing B_{dif} is also in linear time.

Next, we will show how many values of F_k must be tested in order to make sure that Condition (59) is satisfied for all $F_k \geq 0$.

THEOREM 3.19. *For the taskset Γ with $U^S(\Gamma) < A(H) - A_{max} + 1$, if Condition (59) is false for any F_k, then it must be false for some F_k satisfying the condition below:*

$$F_k \leq \frac{W_\triangle + D_k U^S(\Gamma) + \sum_{\tau_i} C_i A_i - (A(H) - A_k + 1)(D_k - C_k)}{A(H) - A_{max} + 1 - U^S(\Gamma)}, \tag{60}$$

in which $W_\triangle$ is computed with the Algorithm 2.

PROOF. $W_\triangle$ is the upper bound of the sum of $C_i * A_i$ of all tasks that can be simultaneous active with HW resource $A(H) - A_{max} + 1$. It can be computed with algorithm in Figure 2, which is similar to the calculation of B_{dif}.

It is known that $I_1^T(\tau_i) \leq \text{DBF}(\tau_i, F_k + D_k)$ and $I_2^T(\tau_i) \leq \text{DBF}(\tau_i, F_k + D_k) + C_i$ [Baruah 2007], so it directly follows that:

$$I_1^S(\tau_i) \leq \text{DBF}(\tau_i, F_k + D_k) A_i \tag{61}$$

and

$$I_2^S(\tau_i) \leq (\text{DBF}(\tau_i, F_k + D_k) + C_i) A_i. \tag{62}$$

Algorithm 2 Computing $W_\triangle$

```
$W_\triangle = 0$
$a = A(H) - A_{max} + 1$
TaskQueue Q = all tasks sorted in the decreasing order of $C_i$
$\tau_i$ = the first task in $Q$
WHILE (true)
    IF ($a > \tau_i.A$)
        $a = a - \tau_i.A$
        $W_\triangle = W_\triangle + \tau_i.C * \tau_i.A$
        $\tau_i$ = next task in $Q$
    ELSE
        $W_\triangle = W_\triangle + \tau_i.C * a$
        return $W_\triangle$
    ENDIF
ENDWHILE
```

So it can be obtained that

$$\sum_{\tau_i \in \Gamma} I^S(\tau_i) \leq W_\triangle + \sum_{\tau_i \in \Gamma} \mathrm{DBF}(\tau_i, F_k + D_k)A_i. \tag{63}$$

Since we assume Condition (59) is violated, we have:

$$W_\triangle + \sum_{\tau_i \in \Gamma} \mathrm{DBF}(\tau_i, F_k + D_k)A_i > (A(H) - A_{max} + 1) \times F_k + (A(H) - A_k + 1)) \times (D_k - C_k) \tag{64}$$

is the necessary condition for the deadline miss to occur.

We can bound the $\sum_{\tau_i \in \Gamma} \mathrm{DBF}(\tau_i, F_k + D_k)A_i$ by:

$$\begin{aligned} \sum_{\tau_i \in \Gamma} \mathrm{DBF}(\Gamma, F_k{+}D_k)A_i &\leq \sum_{\tau_i \in \Gamma} \left(\left\lfloor \frac{T_i}{F_k{+}D_k} \right\rfloor + 1 \right) C_i A_i \\ &\leq (F_k + D_k)U^S(\Gamma) + \sum_{\tau_i \in \Gamma} C_i A_i. \end{aligned} \tag{65}$$

By Equation (64) and (65) and $U^S(\Gamma) < A(H) - A_{max} + 1$ we have

$$F_k < \frac{W_\triangle + D_k \cdot U^S(\Gamma) + \sum_{\tau_i} C_i A_i - (A(H) - A_k + 1)(D_k - C_k)}{A(H) - A_{max} + 1 - U^S(\Gamma)}. \tag{66}$$

□

So for the taskset with $U^S(\tau_i) < A(H) - A_{max} + 1$, Condition (59) can be tested in time pseudo-polynomial to the task parameters. For taskset with $U^S(\tau_i) \geq A(H) - A_{max} + 1$, **GDG-3** 3.18 is not applicable.

3.4 Utilization Bound Test for Nonpreemptive EDF

In this section, we derive a utilization bound test for NP-EDF-FkF for tasksets with pre-period deadlines. The derivation is based on an existing scheduling test for nonpreemptive EDF on multiprocessors [Baruah 2006]:

THEOREM 3.20. *A taskset with pre-period deadlines Γ can be feasibly scheduled by EDF$_{np}$ on an identical multiprocessor of m ($m > 1$) unit capacity processors, if:*

$$V^T(\Gamma) \leq m - (m-1) \times V^T_{max}(\tau), \tag{67}$$

where $V^T_{max}(\tau)$ is the maximal $V^T(\tau)$ of all tasks in Γ.

The proof follows the resource augmentation approach proposed in Phillips et al. [1997]. Note that there is an important restriction of $m > 1$ for this theorem, since the item $(m-1)$ was used to multiply both sides of an inequality in the proof.

Next, we generalize Theorem 3.20 to single processor scheduling ($m = 1$), which will turn out to be useful later.

THEOREM 3.21. *A taskset with pre-period deadlines Γ can be feasibly scheduled by EDF$_{np}$ on a single processor with unit capacity if*

$$V^T(\Gamma) \leq 1. \tag{68}$$

The proof of Theorem 3.21 is similar to that in Baruah [2006] and omitted here.

Now we can derive the utilization bound test for NP-EDF-FkF and tasksets with pre-period deadlines on HRDs.

THEOREM 3.22. ***(GDG-NP)****. Any taskset Γ can be feasibly scheduled by NP-EDF-FkF on an FPGA H with area $A(H) \geq 2A_{max}$ or $A_{max} + A_{min} - 1 \geq A(H) \geq A_{max}$, if for $\forall \tau_i \in \Gamma$:*

$$V^S(\Gamma) \leq (A(H) - A_{max} + 1) \cdot (1 - V^T(\tau_i)) + V^S(\tau_i) \tag{69}$$

where A_{max} and A_{min} denote the largest and smallest area of all tasks in Γ, respectively.

The proof of Theorem 3.22 consists of two parts for different value ranges of $A(H)$, presented in Sections 3.4.1 (Lemma 3.23) and 3.4.2 (Lemma 3.30), respectively.

3.4.1 *Case 1: $A_{max} + A_{min} - 1 \geq A(H) \geq A_{max}$*

LEMMA 3.23. *(Case 1 of Theorem 3.22). Any taskset Γ can be feasibly scheduled by EDF-FkF on an FPGA H with area $A_{max} + A_{min} - 1 \geq A(H) \geq A_{max}$, if for $\forall \tau_i \in \Gamma$:*

$$V^S(\Gamma) \leq (A(H) - A_{max} + 1) \cdot (1 - V^T(\tau_i)) + V^S(\tau_i). \tag{70}$$

where A_{max} is the largest area of all tasks in Γ.

PROOF. From the condition $A_{max} + A_{min} - 1 \geq A(H) \geq A_{max}$, we know that there will never be two or more HW tasks executing on the FPGA simultaneously, so we can treat the HW taskset as a SW taskset with the same execution time on a single processor.

Let τ_k be the task with the smallest area size A_{min}, then we have:

$$V^S(\Gamma) \leq (A(H) - A_{max} + 1) \cdot (1 - V^T(\tau_k)) + A_{min} V^T(\tau_k) \tag{71}$$

Since $A_{max} + A_{min} > A(H)$, we have:

$$V^S(\Gamma) \leq A_{min} \tag{72}$$

Since $V^S(\Gamma) = \sum_{\tau_i \in \Gamma} V^T(\tau_i) A_i$ and $\sum_{\tau_i \in \Gamma} V^T(\tau_i) A_i \geq \sum_{\tau_i \in \Gamma} V^T(\tau_i) A_{min}$, we have $V^T(\Gamma) \leq 1$.

By Theorem 3.21, taskset Γ is schedulable. □

3.4.2 *Case 2: $A(H) \geq 2A_{max}$.* For this case, the proof follows the resource augmentation approach [Phillips et al. 1997] and is closely related to the nonpreemptive EDF utilization bound on multiprocessors in [Baruah 2006]. We proceed in three steps:

—Construct a theoretically feasible nonpreemptive blocked multi-FPGA machine π.

—Calculate the required condition such that, for all $t \geq 0$, the NP-EDF-FkF algorithm will never do less work by time $t + C_{max}$ than the OPT algorithm on the theoretical machine constructed in the last step by time t.

—Prove that NP-EDF-FkF produces a feasible schedule for Γ on H.

In Danne and Platzner [2006a], a reference feasible multi-FPGA model is used to simulate the case of executing every task on its corresponding FPGAs simultaneously, in which each FPGA has the same area size as its corresponding task's area size, and has the speed of $U^T(\tau_i)$. An algorithm OPT assigning each task to its corresponding FPGA can guarantee all the task instances to meet their deadlines. Here are the two definitions from Danne and Platzner [2006a]:

Definition 3.24 (*FPGA*). An FPGA H is a processing device with area $A(H)$ and speed $S(H)$. It can execute a set of task instances R simultaneously, iff $\sum_{j_i \in R} A(J_i) \leq A(H)$. If a task instance J_i is in execution on H for t units of time, it completes $S(H) \cdot t$ units of its computation time C_i. The computing capacity of H is defined as $Cap(H) = A(H) \cdot S(H)$.

Definition 3.25 (*Multi-FPGA*). A multi-FPGA π is a set of FPGAs $H_1, H_2, \ldots$, each with its own area $A(H_j)$ and speed $S(H_j)$. At each point of time, each FPGA H_j can execute its individual set R_j of task instances, iff $\sum_{J_i \in R_j} A(J_i) \leq A(H_j)$ for all $H_j \in \pi$. The computing capacity of π is defined as $Cap(\pi) = \sum_{H_j \in \pi} Cap(H_j)$.

Since we are concerned with nonpreemptive scheduling, we will construct a new reference feasible multi-FPGA model, named *nonpreemptive blocked feasible multi-FPGA*, which takes into account tasks' blocking time due to non-preemption.

Definition 3.26 (*Non-preemptive blocked feasible multi-FPGA*). For a given periodic taskset Γ, we define a nonpreemptive blocked feasible multi-FPGA π with capacity $Cap(\pi) = V^S(\Gamma)$ such that for any task $\tau_i \in \Gamma$ there is an FPGA $H_j \in \pi$ with $A(H_j) = A_j$ and $S(H_j) = V^T(\tau_i)$. Algorithm OPT assigns each task τ_i to its corresponding FPGA H_j.

LEMMA 3.27. *For any taskset Γ running on its corresponding nonpreemptive blocked feasible multi-FPGA, algorithm OPT defined in Definition 3.26 can guarantee that each task instance J_i^j completes its work by the time instance $r_j + D_i - C_{max}$, where r_j is the release time of task instance J_i^j and D_i is the relative deadline of J_i^j.*

The proof of Lemma 3.27 directly follows from the definition of $V^T(\tau_i)$ and Definition 3.4.2.

In Danne and Platzner [2006a], a function W is defined to capture the amount of work done on a given task instance or task instance set by some algorithm on some machine:

Definition 3.28. *Work-done function.* A task instance with computation time C_i and area A_i represents $C_i \cdot A_i$ work. If the job has been executed for t time units on an $S(H)$ speed *FPGA*, the work that has been done on this task instance is $t \cdot S(H) \cdot A_i$. Let I denote any set of task instances and π any hardware platform. For any algorithm *alg* and time instance $t \geq 0$, $W(alg, \pi, I, t)$ denotes the amount of work done on task instances of I over the interval $[0, t)$, when I is scheduled by *alg* on π.

Now we will show that on a specific single FPGA H, the NP-EDF-FkF algorithm will never do less work by time $t + C_{max}$ than the OPT algorithm on the reference platform π defined in Definition 3.26 by time t, for all $t \geq 0$.

LEMMA 3.29. *Let Γ be a periodic taskset with at least two tasks. Let I be the related set of task instances produced by Γ. Let π and OPT be the nonpreemptive blocked feasible multi-FPGA machine and the scheduling algorithm according to Definition 3.26. Further, let H be a single $FPGA$ with speed $S(H) = 1$. If for $\forall \tau_i \in \Gamma$, the following two conditions are both true:*

$$V^S(\Gamma) \leq (A(H) - A_{max} + 1) \cdot (1 - V^T(\tau_i)) + V^S(\tau_i) \tag{73}$$

$$A(H) - A_{max} + 1 > A_i \tag{74}$$

for any $t \geq 0$, the work done on I by algorithm NP-EDF-FkF on FPGA H during $[0, t + C_{max})$ is never less than the work done on I by algorithm OPT on the nonpreemptive blocked feasible multi-FPGA π:

$$W(NP - EDF - FkF, H, I, t + C_{max}) \geq W(OPT, \pi, I, t) \tag{75}$$

PROOF. We use proof by contradiction. We assume that Inequality (75) is violated, and prove that this contradicts the assumption that Γ satisfies Condition (73).

Let t_0 denote the earliest value of t at which Inequality (75) is violated. Since the total amount of work done on all task instances in I over $[0, t_0 + C_{max})$ by NP-EDF-FkF is strictly less than the total amount of work done on all task instances in I over $[0, t_0)$ in OPT, there must exist at least one task instance J_i^j that has received less service by time $t_0 + C_{max}$ in NP-EDF-FkF than by time t_0 in OPT.

$$W\big(NP - EDF - FkF, H, J_i^j, t_0 + C_{max}\big) < W\big(OPT, \pi, J_i^j, t_0\big). \tag{76}$$

Let $r_i^j < t_0$ denote the release time of task instance J_i^k. By our choice of t_0 as the first time instant at which Inequality 75 is violated, it must be the case that:

$$W\big(NP - EDF - FkF, H, J_i^j, r_i^j + C_{max}\big) \geq W\big(OPT, \pi, J_i^j, r_i^j\big). \tag{77}$$

Therefore, the amount of work done on I in OPT over $[r_i^j, t_0)$ is strictly greater than the amount of work done on task instance in I in NP-EDF-FkF over the interval $[r_i^j + C_{max}, t_0 + C_{max})$.

The single FPGA H is either *overloaded*, when the ready queue is not empty, or *underloaded*, when the ready queue is empty. In a given time interval $[r_i^j + C_{max}, t_0 + C_{max})$, let x be the amount of time when H is overloaded, and y the amount of time when H is underloaded such that:

$$x + y = (t_0 + C_{max}) - \big(r_i^j + C_{max}\big) = t_0 - r_i^j. \tag{78}$$

We make the following observations concerning work of task instance T_i^j done by OPT during time interval $[r_i^j, t_0)$ and by NP-EDF-FkF during interval $[r_i^j + C_{max}, t_0^j + C_{max})$,

—During $[r_i^j + C_{max}, t_0 + C_{max})$, algorithm NP-EDF-FkF executes J_i^j on H for at least y time units, i.e., it does at least $y \cdot A_i$ work on J_i^j.

—During $[r_i^j, t_0)$, algorithm OPT executes J_i^j for $t_0 - r_i^j$ time units, i.e. OPT does $(t_0 - r_i^j) \cdot S(H_i) \cdot A_i$ work on J_i^j.

—By our assumption Inequality (76), OPT performs more work on J_i^j than NP-EDF-FkF, and by Equation (78):

$$(x + y) \cdot S(H_i) > y \tag{79}$$

Now we consider the work on the entire set of task instances I done by OPT during time interval $[r_i^j, t_0)$ and by NP-EDF-FkF during interval $[r_i^j + C_{max}, t_0^j + C_{max})$. We make these observations:

—Since NP-EDF-FkF is $(1 - A_{max} - 1/A(H))$-work-conserving, at least $A(H) - A_{max} + 1$ area of the FPGA is utilized during the overload portion. By our assumption Inequality (76), J_i^j must not have finished until $t_0 + C_{max}$, so during the underload portion, at least J_i^j must be executing. So the overall work done with algorithm NP-EDF-FkF on H during $[r_i^j + C_{max}, t_0 + C_{max})$ is at least $(A(H) - A_{max} + 1) \cdot x + A_i \cdot y$.

—The overall work done by algorithm OPT on I is at most $(t_0 - r_i^j) \cdot Cap(\pi)$, that is, $(x + y) \cdot Cap(\pi)$, which corresponds to full utilization of all FPGAs.

—As shown above, the amount of work done on I in OPT over $[r_j^i, t_0)$ is strictly greater than the amount of work done on I in NP-EDF-FkF over the interval $[r_j^i + C_{max}, t_0 + C_{max})$:

$$(x + y) \cdot Cap(\pi) > x \cdot (A(H) - A_{max} + 1) + y \cdot A_i \tag{80}$$

Now we show that Inequalities (79) and (80) contradict Conditions 73 and 74.

Since $A(H) \geq 2A_{max}$, we have $A(H) - A_{max} + 1 - A_i > 0$. Multiply Equation (79) by $A(H) - A_{max} + 1 - A_i$ and add it to Equation (80). Let A_{bnd} denote $(A(H) - A_{max} + 1)$. From (74), we have:

$$(x + y)(S(H_i)(A_{bnd} - A_i) + Cap(\pi)) > (x + y)A_{bnd} \tag{81}$$

By Definition 3.26 we have $S(H_i) = V^T(\tau_i)$ and $Cap(\pi) = V^S(\Gamma)$, so we have

$$(V^T(\tau_i)(A_{bnd} - A_i) + V^S(\Gamma)) > A_{bnd} \tag{82}$$

i.e.

$$V^S(\Gamma) > (A(H) - A_{max} - 1) \cdot (1 - V^T(\tau_i)) + V^S(\tau_i) \tag{83}$$

which contradicts Condition (73). Hence, the assumption must be wrong, and the lemma is proved. □

Now we have the conclusion of the second case of Theorem 3.22:

LEMMA 3.30. *(Case 2 of Theorem 3.22). Any taskset Γ can be feasibly scheduled with EDF-FkF on an FPGA H with area $A(H) \geq 2A_{max}$, if for $\forall \tau_i \in \Gamma$:*

$$V^S(\Gamma) \leq (A(H) - A_{max} + 1) \cdot (1 - V^T(\tau_i)) + V^S(\tau_i) \tag{84}$$

where A_{max} is the largest area of all tasks in Γ.

The proof of Lemma 3.30 is similar to that of Theorem 1 in Baruah [2006].

PROOF. We prove the lemma by induction on the number of task instances. We sort the jobs in the nondecreasing deadline order.

The base case is an empty set of task instances: all deadlines are met trivially.

For the inductive step, we prove that for each integer $k \geq 1$, if the task instances with the first $k - 1$ deadlines have completed by their deadlines by NP-EDF-FkF, then the k'th-earliest deadline task instance (J_k) completes by its deadline by NP-EDF-FkF.

Let d_k denote the deadline of the kth-earliest deadline task instance.

—By Lemma 3.27, OPT completes J_k by time-instant $r_k + D_k - C_{max}$, where r_k and D_k denote the release time and relative deadline of J_k. So J_k is completed by OPT by the time-instant $d_k - C_{max}$.

—By Lemma 3.29, NP-EDF-FkF performs at least as much work on J_k by time instant d_k as OPT does by time-instant $d_k - C_{max}$. But since OPT completes all these task instances by time-instant $d_k - C_{max}$, it must be the case that NP-EDF-FkF also completes all these task instances by time-instant d_k. □

To summarize, there are two key differences between the derivations of non-preemptive EDF (**GDG-NP**) and preemptive EDF (**DP**):

—When we construct the reference multi-FPGA platform, we assign a different speed for each referenced single-FPGA, that is, $V^T(\tau_i)$ instead of $U^T(\tau_i)$.

—Preemptive EDF never "falls behind" the OPT, which means that by any given time t, preemptive EDF never does less work than OPT. But nonpreemptive EDF can "fall behind" OPT by a bounded amount, which means that for any given time t, work done by nonpreemptive EDF by time $t + C_{max}$ is never less than work done by OPT by time t. This is because each task instance may suffer a maximum interference time of C_{max} due to the blocking nature of non-preemptive scheduling.

4. TASK PLACEMENT STRATEGY AND RECONFIGURATION OVERHEAD

We adopt the task placement policy proposed in Danne and Platzner [2006a], referred to as PLC1, for preemptive EDF scheduling. If we pick one side of the FPGA as the bottom and the other side as the top, then tasks are stacked on the FPGA from bottom to top in priority order. Whenever a task finishes, all tasks above it are shifted downwards by a relocation process; when a task starts or resumes from preemption, it is placed on the FPGA according to its priority. Therefore, each time a task is added or removed from the FPGA, the entire reconfigurable area may have to be reconfigured in the worst case. We define the *reconfiguration factor* rf such that the time it takes to reconfigure the entire FPGA is $rf * A(H)$. Due to our assumption described in Section 1 that there can be no gaps between a task's reconfiguration stage and computation stage, we can account for the reconfiguration overhead by adding it to a task's computation time.

Danne and Platzner [2006a] also derived the worst-case time overhead due to reconfiguration in the preemptive EDF scheduling with this scheme, and showed that the time overhead can be accounted for by increasing the computation time of each task to:

$$C_i^p = C_i + (1 + 2 \times N_i + O_i) \times rf \times A(H), \tag{85}$$

in which

$$N_i = \sum_{\tau_j \in \Gamma} \left\lfloor \frac{T_i}{T_j} \right\rfloor - 1 \tag{86}$$

$$O_i = \max_{\forall R_l \subseteq \Gamma - \tau_i} |R_l| : \sum_{\tau_k \in R_l} A_k \leq A(H) - A_i. \tag{87}$$

We propose a different task placement policy for nonpreemptive EDF as follows, referred to as *PLC2*:

—Tasks are stacked on the device from bottom to top sorted with their start times, so that a new task is stacked on top of all currently running tasks.

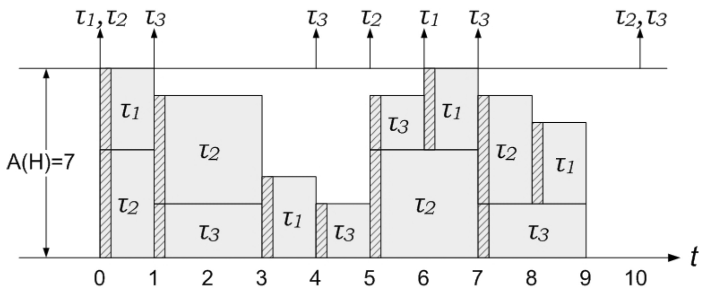
—When a task terminates, all tasks on top of it are shifted downwards, and tasks below it are not affected.

PLC2 performs task shifting when a task finishes, while PCL1 performs task shifting both when a task starts and when it finishes. Figure 6 compares the schedules of the taskset in Table II using either EDF-FkF+PLC1 or NP-EDF-FkF+PLC2. The number of time instants when reconfiguration occurs is 8 using EDF-FkF+PLC1 and 5 using NP-EDF-FkF+PLC2.

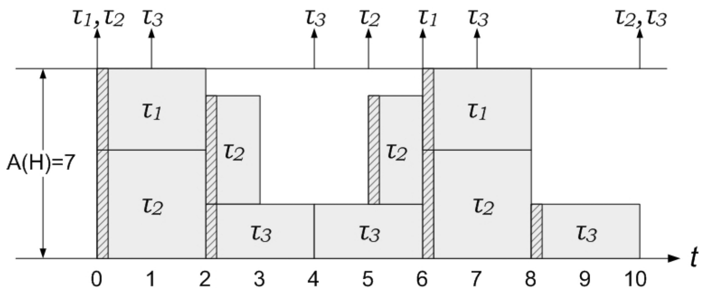
We can reduce the reconfiguration overhead in the test condition for NP-EDF-FkF+PLC2 than EDF-FkF/EDF-NF+PLC1: Since tasks are stacked in the order of their start times, a task is only affected by tasks that are already running when it starts, the worst-case number of which is O_i as discussed in Danne and Platzner [2006a]. So for NP-EDF-FkF+PLC2, we can account for the reconfiguration overhead by increasing the computation time of each task:

$$C_i^{np} = C_i + O_i * rf * A(H) \tag{88}$$

Both PLC1 and PLC2 use the full device reconfiguration time to derive the bounds, since one task start or termination event can potentially lead to several task preemptions, resumes or shifts. As most FPGAs only have a single reconfiguration port, all tasks' reconfiguration stages must be serialized. To account for the worst case, we must make the pessimistic assumption that the whole device undergoes reconfiguration.



(a) EDF-FkF + PLC1



(b) NP-EDF-FkF + PLC2

Fig. 6. Scheduling of the taskset in Table II with EDF-FkF+PLC1 and NP-EDF-FkF+PLC2.

Table II. A Taskset with Low Utilization but Nonschedulable with Partitioned EDF

Task	C	D	T	A
τ_1	2	6	6	3
τ_2	3	5	5	4
τ_3	2	3	3	2

5. HW PROTOTYPE FOR PREEMPTIVE MULTITASKING ON FPGA

Preemptive multitasking on FPGAs is somewhat controversial, since many people believe the reconfiguration delay is too large for preemptive multitasking to be practical. However, with the rapid advancement in HW capabilities, the reconfiguration delay may no longer be a serious issue. For example, Virtex-II Pro provides an ICAP with an 8-bit wide interface working at 50MHz; Virtex-4 provides an ICAP with a 32-bit wide interface working at 100MHz. This means that the theoretical upper limit of reconfiguration throughput has been increased from 50KB/ms to 400KB/ms from Virtex-II Pro to Virtex-4. Considering that most HW tasks have bitstream size in the range of a few hundred KBs, the configuration delay can be potentially lower than 1ms.

Table III lists Xilinx Virtex-4 FX series device parameters obtained from the Virtex-4 configuration guide [Xilinx 2007]. The Total Reconfiguration Time and Reconfiguration Factor are obtained using the theoretical upper limit of reconfiguration throughput of 400KB/ms, and assuming that the entire FPGA area is reconfigurable. In reality, only part of the FPGA area is reconfigurable

Table III. Xilinx Virtex-4 FX Series Device Parameters

Device	#Columns	#Rows	Config. Mem.(KB)	Total Reconfig. Time(ms)	Reconfig. Factor
XC4VFX12	64	24	590.400	1.48	0.023
XC4VFX20	64	36	900.032	2.25	0.035
XC4VFX40	96	52	1588.544	3.97	0.041
XC4VFX60	128	52	2660.064	6.65	0.052
XC4VFX100	160	68	4127.880	10.32	0.064
XC4VFX140	192	84	5876.816	14.69	0.077

while the rest has fixed configuration, so the total reconfiguration time would be less. The total reconfiguration time is the worst-case context-switch delay for both nonpreemptive and preemptive EDF, and its effect in schedulability tests is discussed in Section 4. As we will show in Figure 11, the schedulability test acceptance ratio for all the tests considered in this paper deteriorates rapidly with increase of the reconfiguration delay, so the reconfiguration time places severe constraints on the task execution rates. If we assume the task period to be 100-200ms, then it is only realistic to use preemptive or non-preemptive EDF scheduling on small devices, that is, XC4VFX12, XC4VFX20, XC4VFX40, or large devices with smaller reconfigurable areas. Of course, applications with larger task periods can tolerate larger reconfiguration times, and the designer needs to carefully consider the reconfiguration time when choosing a suitable HW device for a given application. With emerging of new techniques for speeding up reconfiguration, for example, multiport reconfiguration, higher task execution rates can be expected to be achievable in the future.

We have implemented a simple prototype system for preemptive multitasking on a Xilinx Virtex-4 FPGA (XC4VFX12) based on the TSAS (Task Specific Access Structures) approach.[2] Of the 64 columns on the FPGA, 24 columns are used for the reconfigurable area, and 40 columns are used as fixed configuration for the HW overheads of our prototype system (ICAP controller, TSA, IFIP, memory drive device and other overheads). Since this is a very preliminary prototype, we did not perform any optimizations to further reduce the fixed configuration area, but the HW overhead can be much smaller in a realistic system. As discussed in Section 2.2, this approach has lower delay than the CPA approach, which is the main reason for choosing it, since the reconfiguration delay can have a large negative impact on schedulability tests. The TSAS approach has another benefit that we do not need to know the detailed bitstream format, while the CPA approach using JBits requires it. Unlike Virtex-II Pro, detailed documentation of the bitstream format for Virtex-4 is not publicly available, so it is inconvenient to use the CPA approach for Virtex-4. Xilinx supports two basic styles of partial reconfiguration: module-based and difference-based. Module-based partial reconfiguration uses modular design concepts to reconfigure large

[2]We would like to emphasize that we are not claiming original research contribution with this simple HW prototype, since its implementation is not very complicated using Xilinx Virtex-4 and related SW tools. Our main purpose is to develop a proof of concept system and obtain some performance numbers to support the premise of this paper, i.e., preemptive multitasking on FPGAs is indeed feasible with todays technology, not to develop a full-fledged OS for FPGA, or novel HW reconfiguration mechanisms.

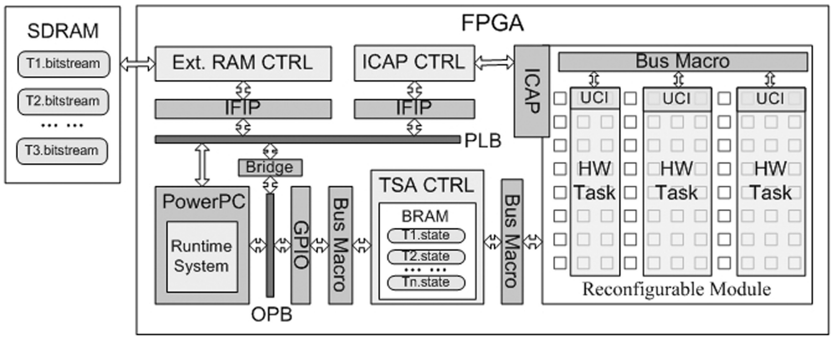


Fig. 7. HW architecture of our prototype system.

blocks of logic. The distinct portions of the design to be reconfigured are known as reconfigurable modules. Difference-based partial reconfiguration is a technique for making small changes in an FPGA design, so that if we would like to let task τ_i preempt task τ_j, and these two tasks have similar bitstreams, then we can offline compute and store a difference bitstream that can be downloaded to the FPGA, which is often much smaller than the full bitstream. We can see that the difference-based approach has a much smaller bitstream file size, and consequently faster reconfiguration time. Our experience shows that the difference bitstream is typically much smaller than the original bitstreams, even when the two HW tasks do not look similar at all. However, the difference-based approach requires multiple bitstream files to be stored for all possible ordered task pairs, for example, if we have 3 tasks τ_1, τ_2 and τ_3, and any task can preempt any other task, then we need to store 6 difference bitstreams: Diff(τ_1, τ_2), Diff(τ_2, τ_1), Diff(τ_1, τ_3), Diff(τ_3, τ_1), Diff(τ_2, τ_3), Diff(τ_3, τ_2). In general, if we have n tasks, then we need to store $n*(n-1)$ difference bitstreams. This is acceptable to us, since our main goal is to minimize reconfiguration delay.

For simplicity of implementation, we impose the restriction that all HW tasks be pin compatible, that is, the TSAS Controller should have a uniform interface to connect to the registers of different HW tasks.

Our HW prototype addresses the following issues:

—**On-chip runtime system**. We would like the OS to run on the PowerPC core on-chip instead of an off-chip processor for performance reasons.
—**Context saving/restoring of HW tasks**. Similar to the processor-based preemptive task system, the HW task context should be saved when the task is preempted and restored when it is resumed.
—**Relocation of HW tasks**. We need to be able to suspend a HW task and resume it at a different location to support the relocation policies PLC1 and PLC2.
—**High-speed reconfiguration**. This is especially important for preemptive scheduling, which may incur many task reconfigurations.

Our HW prototype is implemented on a Xilinx ML403 development board, which contains a Virtex-4 XC4V12F FPGA. The system architecture is shown

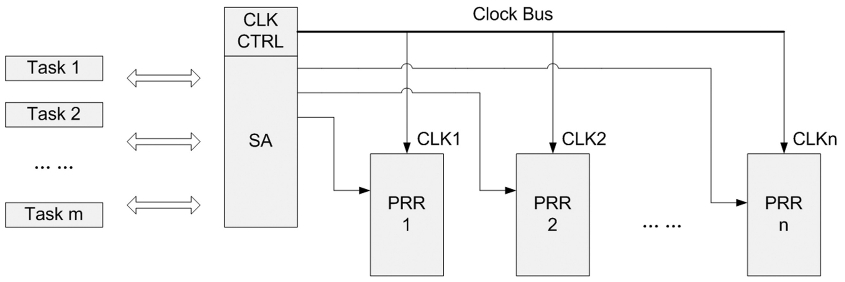


Fig. 8. Block Diagram for TSAS CTRL.

in Figure 7. The major functional modules include:

—Runtime System. The runtime system on the PowerPC core manages the task-related data structures and makes decisions on task scheduling and placement.
—TSAS CTRL. A HW module that helps to save and restore the execution context of HW tasks by accessing the task state registers directly through the Bus Macro and storing their values in its internal BRAM.
—ICAP CTRL. A HW module that controls the ICAP. It is connected to the Processor Local Bus (PLB), using an in-house PLB IP Interface (IPIF). The throughput of PLB-based ICAP access is about 20 times faster than the OPB-HWICAP approach provided by Xilinx [Kalte and Porrmann 2005].
—Ext. RAM CTRL. A HW module that controls the external RAM used to restore the HWtask bitstreams. It is also connected to the PLB via IPIF. The configuration bitstream is downloaded from the external RAM to the ICAP directly under the control of the PowerPC core. (Note that we never read back the bitstream from the ICAP in the TSAS approach.)

We manually designed the TSAS CTRL with Verilog, and used Xilinx ISE to generate the final implementation in the form of a bitstream that is downloaded and configured on the FPGA. Figure 8 shows the system block diagram. TSAS CTRL includes two parts: one is the clock controller(CLK CTRL), which freezes the clocks in the Partial Reconfigurable Regions (PRR) that are overwritten during the preemption phase, but do not disturb the normal operation of tasks in other PRRs. The other part is State Access (SA), which can access all the state registers of tasks in all PRRs through direct datapaths. Suppose task τ_1 is running on the FPGA, and we download another task τ_2 to preempt it. When the TSAS CTRL receives a command from the runtime system on the PowerPC to switch in task τ_2, it first stops the clock of τ_1's PRR, then reads τ_1's state registers and stores them in its internal BRAM (Block RAM). The bitstream of task τ_2 is loaded from the external RAM to ICAP through PLB, and then state information of τ_2 is restored in their state registers by the TSAS Controller from the BRAM. Finally, TSAS CTRL restarts the clock, and τ_2 starts to run. The state machine in Figure 9 captures the behavior of TSAS CTRL, where S_Taskid refers to the task to be suspended, and R_Taskid refers to the task to be resumed. About half of the FPGA resource is used to implement the these modules and the other part is used to run HW tasks.

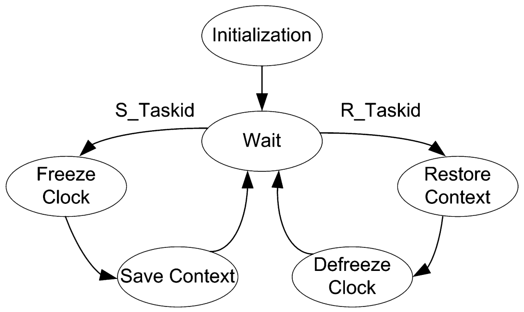


Fig. 9. State machine for TSAS CTRL.

We ran some experiments with our prototype system to measure its reconfiguration delay. We first place a task τ_1 on the FPGA, then download another task τ_2 to preempt it. Then we have these 3 steps:

—Save the state registers of τ_1 to the BRAM with a speed of v_s KB/ms.

—Download τ_2's bitstream from RAM to ICAP with a speed of v_d KB/ms.

—Restore τ_2's state registers from the BRAM with a speed of v_r KB/ms.

The total delay is:[3]

$$st_{old}/v_s + bs_{new}/v_d + st_{new}/v_r. \tag{89}$$

Our measurement results are $v_s = v_r = 33.3KB/ms$ and $v_d = 378KB/ms$. To preempt a task whose bitstream size is 200 KB and state information size is 0.1KB and configure a task with the same parameters, the total delay is:

$$0.1/33.3 + 200/378 + 0.1/33.3 \approx 0.535ms. \tag{90}$$

Of course, the delay is related to task τ_2's bitstream size (or Diff(τ_1, τ_2) if difference-based reconfiguration is used), and will be different for different applications. Since most HW tasks are computation-intensive kernels [Jovanovic et al. 2007], that is, the part of application code that is not very large but executed very often, and it is reasonable to assume the bitstream size to be in the range of few hundred KBs.

We also measured the time it takes to reconfigure the entire FPGA, which is related to the context switch overhead as discussed in Section 4. With the module-based approach, the reconfiguration bitstream consists of the entire configuration memory (590.4KB) if we assume the entire FPGA area to be reconfigurable. We do not consider the time for save and restore of state registers, which is typically negligible since typical task state size is very small. Note that even though our HW prototype has a reconfigurable area of only 24 columns out of the total 64 columns, this does not affect our time measurements since the reconfiguration time is equal to the bitstream download time, regardless of whether the bitstream is actually used for reconfiguration. (In this case,

[3]Note that if we had use the bitstream readback via CPA, there would be an additional term for reading back and filtering τ_1's bitstream.

Table IV. Comparison of Different Utilization Bound Tests

Test	EDF-FkF	EDF-NF	NP-EDF-FkF	Post-Period Deadline	Complexity
DP	$\surd$				$O(N)$
GDG-1		$\surd$			$O(N^2)$
GDG-2	$\surd$			$\surd$	$O(N^3)$
GDG-3		$\surd$			pseudo-polynomial
GDG-NP			$\surd$		$O(N)$

only (24/64)*590.4 = 221.4KB of the 590.4KB is actually used for reconfiguration.) The measured best-case reconfiguration time is 1.51ms, the worst case is 1.82ms and the average-case is 1.65ms, quite close to the theoretical optimal in Table III. With the difference-based approach, the bitstream size and reconfiguration time are dependent on the degree of similarity between the current task and the incoming new task. In our experiments, the reconfiguration time of the difference-based approach is roughly $1/3 \sim 3/4$ of that of the module-based approach.

6. PERFORMANCE EVALUATION

For schedulability analysis of preemptive EDF on FPGAs, the only known related work is Danne and Platzner [2006a] for EDF-FkF. There is no existing work on nonpreemptive EDF scheduling on FPGAs. Table IV shows the comparison between Danne's test (**DP**) and ours (**GDG-1**, **GDG-2**, **GDG-3** and **GDG-NP**). **GDG-1** and **GDG-3** are applicable to EDF-NF, (we can apply **GDG-1** and **GDG-3** to EDF-FkF by replacing the item $A(H) - A_k + 1$ by $A(H) - A_{max} + 1$ since EDF-FkF is global-$(1 - A_{max} - 1/A(H))$-work-conserving.), while **DP** and **GDG-2** are applicable to EDF-FkF. Since EDF-NF is a more optimal scheduling algorithm than EDF-FkF, that is, if a taskset is schedulable with EDF-FkF, then it is also schedulable with EDF-NF, any schedulability bound test for EDF-FkF can also serve as a correct (but pessimistic) test for EDF-NF. Hence, the tests **DP** and **GDG-2** (for EDF-FkF) are also applicable to EDF-NF. Since only **GDG-2** works for tasksets with post-period deadlines, D_i is replaced by T_i implicitly in **DP**, **GDG-1**, **GDG-3** and **GDG-NP** if any task in the taskset has deadline longer than its period.

We use randomly generated synthetic tasksets to evaluate the performance of different utilization bound tests. We use an extension of the method used in Baker [2005b] to generate tasksets: for each group of experiments, we set the FPGA size to be $(A(H))$, and set the range of task parameters as: task period (T_i), deadline-to-period ratio $(k1 = D_i/T_i)$, task utilization (execution time-to-period ratio) $(k2 = C_i/T_i)$, task size (A_i), and reconfiguration factor (rf). An initial set of 3 tasks was randomly generated, with parameters randomly chosen from the set range. All the schedulability tests were applied to that set. Then another randomly-generated task was added to the previous set, and all the schedulability tests were run on the new set. This process of adding tasks was repeated until the total system utilization exceeded $A(H)$. The whole procedure was then repeated, starting with a new initial set of 3 tasks. Unlike Baker [2005b], which starts with $m + 1$ tasks, where m is the number of processors, we choose a (somewhat arbitrary) small number of 3, since unlike

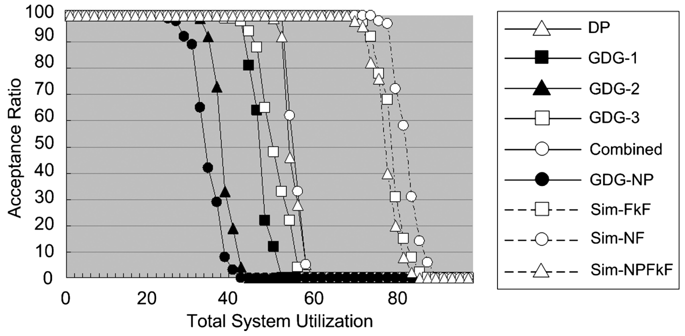


Fig. 10. Comparison of acceptance ratios of the utilization bound tests and simulation.

CPU tasks, FPGA tasks may have different sizes. This method of generating random tasksets produces a fairly uniform distribution of total system utilizations [Baker 2005b]. Due to this method of generating tasksets, the number of tasks in each taskset can be different for different tasksets.

Figure 10 shows comparison of acceptance ratios of the utilization bound tests and simulation. Dotted lines denote the simulations of EDF-FkF, EDF-NF and NP-EDF-FkF, and solid lines denote the utilization bounds. The line labeled "Combined" denotes the acceptance ratio obtained by combining **DP**, **GDG-1**, **GDG-2** and **GDG-3**; that is, the taskset is determined to be nonschedulable only if all tests fail. The taskset parameters are: $A(H) = 100$, $T_i \in [10, 20]$, $k1 = 1$), $k2 \in [0.1, 0.3]$, $A_i \in [1, 30]$ and $rf = 0$. 10,000 tasksets are used in this group of experiments. Since it is not computationally feasible to try all possible task release offsets exhaustively in simulation of periodic/sporadic task systems, all task release offsets are set to be zero as in Baker [2006a], that is, all tasks are released at the same time, and simulation is run for the hyper-period of all task periods. Simulation results obtained under this assumption may sometimes determine a taskset to be schedulable even though it is not, but they can serve as a coarse upper bound of the acceptance ratio. Since simulation is very time-consuming and does not provide any correctness guarantees and its comparison with schedulability tests does not reveal much insight, in order to obtain a large enough sample space (1,000,000 tasksets in each group of experiments), we will concentrate on the analytical utilization bounds, and not perform simulations for the rest of the experiments. This is also the approach commonly used for performance evaluation in research work on multiprocessor scheduling [Cirinei and Baker 2007].

Figure 11 shows performance comparisons with different reconfiguration factors rf. The other parameter settings are: $A(H) = 100$, $T_i \in [100, 200]$, $k1 \in [0.8, 1]$), $k2 \in [0.01, 0.3]$ and $A_i \in [1, 30]$. When rf is assumed to be zero, the utilization bound tests for preemptive scheduling outperform those for nonpreemptive scheduling, as expected. With the increase of rf, the performance of tests for preemptive scheduling deteriorates faster than tests for non-preemptive scheduling, since the additional overheads added to the execution time for preemptive scheduling in the schedulability tests are much larger than those added for non-preemptive scheduling.

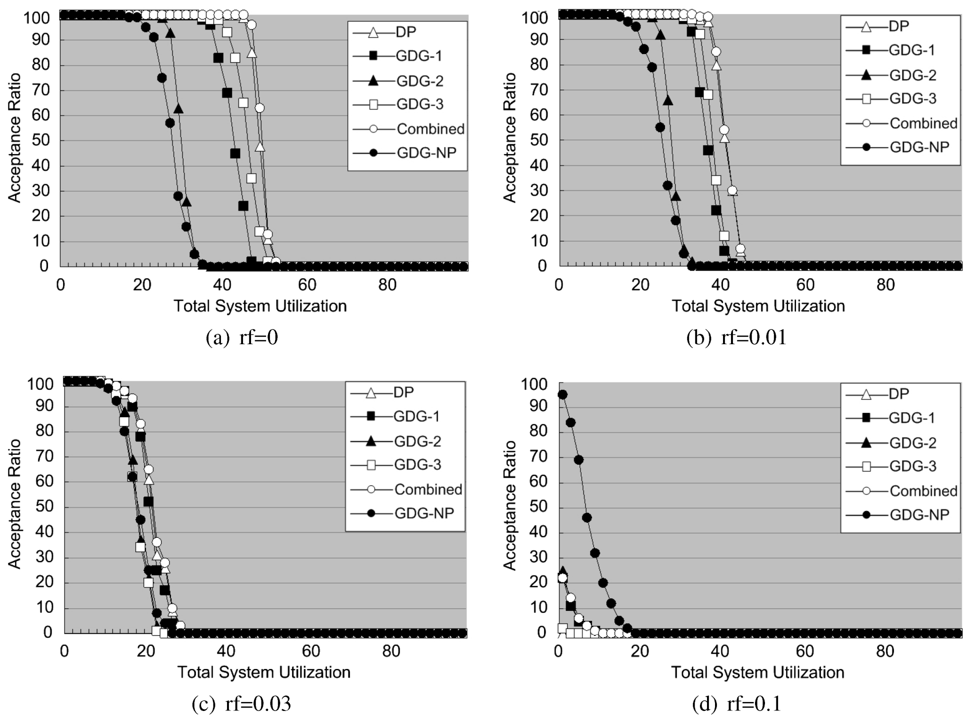
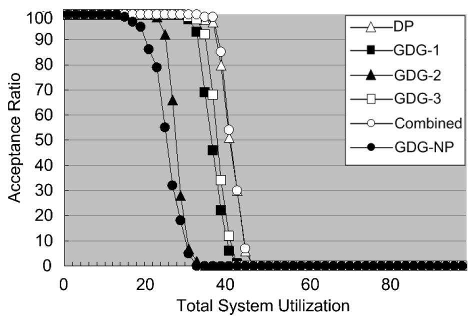


Fig. 11. Performance comparisons with different reconfiguration factors.

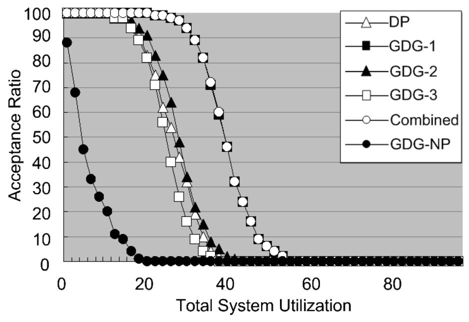
Figure 12 shows performance comparisons for tasksets with different ranges of the ratio of execution time vs. period and different area. The other settings are: $A(H) = 100$, $T_i \in [100, 200]$, $k1 \in [0.8, 1]$) and $rf = 0.01$. **DP** has best performance for time-light spatial-light tasksets (tasksets with both low time utilization and low area utilization), **GDG-1** has best performance for time-heavy tasksets, and **GDG-3** has best performance for time-light spatial-heavy tasksets. **GDG-NP** has much worse performance for time-heavy taskset than for time-light tasksets, since **GDG-NP** always adds the maximum execution time of all tasks C_{max} to each task's interference time in order to take into account blocking time due to non-preemptive scheduling.

Figure 13 shows performance comparisons for tasksets with pre and post-period deadlines. We can see that performance of **GDG-2** improves as the deadline-to-period ratio increases while other tests keep unchanged.

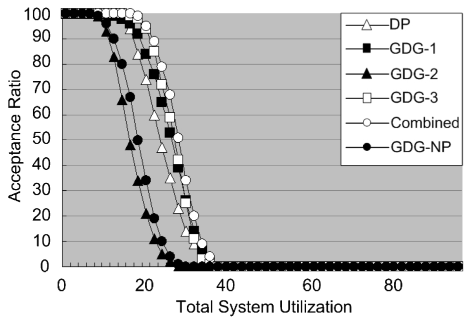
From the performance evaluation results, we can see that nonpreemptive scheduling can tolerate larger reconfiguration overheads than preemptive scheduling. The utilization bound tests for the preemptive EDF are indeed incomparable to each other, that is, no utilization bound test consistently out-performs the others, and different tests may exhibit better performance for tasksets with different characteristics. In order to reduce the degree of pessimism, all tests should be applied together for any given taskset, and the taskset is determined to be nonschedulable only if all tests fail.



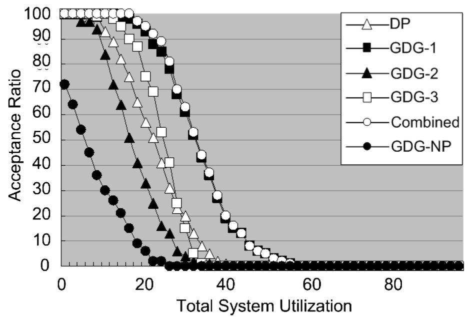
(a) $1 \leq A_i \leq 30 \wedge 0.01 \leq k2 \leq 0.3$



(b) $1 \leq A_i \leq 30 \wedge 0.01 \leq k2 \leq 0.8$

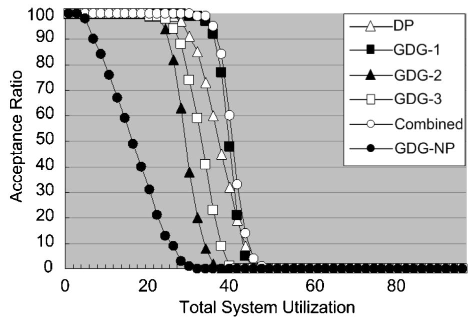


(c) $1 \leq A_i \leq 80 \wedge 0.01 \leq k2 \leq 0.3$

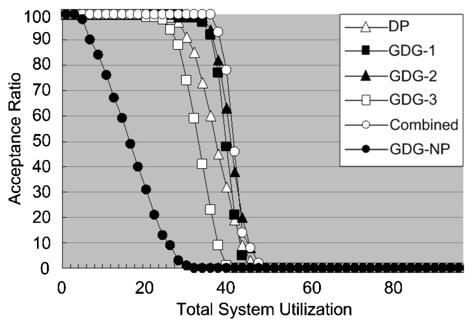


(d) $1 \leq A_i \leq 80 \wedge 0.01 \leq k2 \leq 0.8$

Fig. 12. Performance comparisons with different task sizes and utilizations.



(a) $0.8 \leq k1 \leq 1$



(b) $1 \leq k1 \leq 2$

Fig. 13. Performance comparisons with different deadline-to-period ratios $k1 = D_i/T_i$.

Similar to multiprocessor scheduling, these utilization bound tests allow the designers to get a safe but pessimistic estimate of system schedulability. If a more accurate schedulability decision is desired, then the designer must resort to running simulation experiments, which are less pessimistic but also not safe; that is, simulation may determine a taskset to be schedulable while it is in fact not schedulable in the worst case.

7. CONCLUSIONS

PRTR FPGAs are becoming increasingly widely used in the mainstream FPGA community, so it is important to address real-time multitasking and scheduling analysis. In this paper, we have presented utilization bound tests for preemptive and nonpreemptive EDF scheduling of HW tasks on PRTR FPGAs by extending previous work on utilization bound tests for multiprocessor CPU scheduling. Performance evaluation validates the feasibility of EDF scheduling on PRTR FPGAs, but it may be limited to the smaller FPGA devices if task periods are relatively small, due to excessive reconfiguration time for larger devices.

REFERENCES

Agron, J., Peck, W., Anderson, E., Andrews, D. L., Komp, E., Sass, R., Baijot, F., and Stevens, J. 2006. Run-time services for hybrid cpu/fpga systems on chip. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 3–12.

Baker, T. P. 2003. Multiprocessor edf and deadline monotonic schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 120–129.

Baker, T. P. 2005a. An Analysis of EDF Schedulability on a Multiprocessor. *IEEE Trans. Para. Distr. Syst. 16,* 8, 760–768.

Baker, T. P. 2005b. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Tech. rep. TR-050601, Florida State University.

Baker, T. P. 2005c. Further improved schedulability analysis of edf on multiprocessor platforms. Tech. Rep. TR-051001, Florida State University.

Baker, T. P. 2006a. An analysis of fixed-priority schedulability on a multiprocessor. *Real-Time Syst. 32,* 1-2, 49–71.

Baker, T. P. 2006b. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proceedings of the International Conference on Real-Time and Network Systems (RTSN)*. 119–130.

Banerjee, S., Bozorgzadeh, E., Dutt, N., and Noguera, J. 2007. Selective band width and resource management in scheduling for dynamically reconfigurable architectures. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*. 771–776.

Baruah, S. 2007. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*.

Baruah, S., Mok, A., and Rosier, L. 1990. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium (RTSS)*. 182–190.

Baruah, S. K. 2006. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Syst. 32,* 1-2, 9–20.

Bertogna, M., Cirinei, M., and Lipari, G. 2005. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 209–218.

Buttazzo, G. C. 2005. Rate monotonic vs. EDF: Judgment day. *Real-Time Syst. 29,* 1, 5–26.

Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., and Baruah, S. 2004. *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms*. Chapman and Hall, 30-1–30-19.

Cirinei, M. and Baker, T. P. 2007. Edzl scheduling analysis. In *Proceedings of the European Conference on Real-Time Systems (ECRTS)*. IEEE, 9–18.

Claus, C., Muller, F. H., Zeppenfeld, J., and Stechele, W. 2007. A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW)*.

Danne, K., Muhlenbernd, R., and Platzner, M. 2006. Executing hardware tasks on dynamically reconfigurable devices under real-time conditions. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*.

Danne, K. and Platzner, M. 2006a. An EDF schedulability test for periodic tasks on reconfigurable hardware devices. In *Proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. 93–102.

Danne, K. and Platzner, M. 2006b. Partitioned scheduling of periodic real-time tasks onto reconfigurable hardware. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW)*.

Dyer, M., Plessl, C., and Platzner, M. 2002. Partially reconfigurable cores for xilinx virtex. In *Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL)*. Springer-Verlag, 292–301.

Goossens, J., Funk, S., and Baruah, S. K. 2003. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Syst. 25,* 2-3, 187–205.

Gu, Z., Yuan, M., and He, X. 2007. Optimal static task scheduling on reconfigurable hardware devices using model-checking. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

Guan, N., Gu, Z., Deng, Q., Liu, W., and Yu, G. 2007. Improved schedulability analysis of edf scheduling on reconfigurable hardware devices. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems (WDPRTS)*.

Guccione, S., Levi, D., and Sundararajan, P. 2000. Jbits: A java-based interface for reconfigurable computing. In *Proceedings of the Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*.

Hortaa, E., Lockwood, J., and Kofuji, S. 2002. Using parbit to implement partial run-time reconfigurable systems. In *Proceedings of the the International Conference on Field Programmable Logic and Applications (FPL)*.

Jovanovic, S., Tanougast, C., and Weber, S. 2007. A hardware preemptive multitasking mechanism based on scan-path register structure for fpga-based reconfigurable systems. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 358–364.

Kalte, H. and Porrmann, M. 2005. Context saving and restoring for multitasking in reconfigurable systems. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. 223–228.

Kalte, H. and Porrmann, M. 2006. Replica2pro: task relocation by bitstream manipulation in virtex-ii/pro fpgas. In *Proceedings of the ACM International Conference on Computing Frontiers*. 403–412.

Knuth, D. 1973. *The Art of Computer Programming; Volume 3/Sorting and Searching.* Addison Wesley.

Koch, D., Haubelt, C., and Teich, J. 2007. Efficient hardware checkpointing: concepts, overhead analysis, and implementation. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 188–196.

Lehoczky, J. P., Sha, L., and Ding, Y. 1989. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*.

Li, Z. and Hauck, S. 2002. Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 187–195.

Liu, C. L. and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20,* 1, 46–61.

Lu, C., Stankovic, J. A., Son, S. H., and Tao, G. 2002. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst. 23,* 1-2, 85–126.

Lubbers, E. and Platzner, M. 2007. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*.

Phillips, C. A., Stein, C., Torng, E., and Wein, J. 1997. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*. 140–149.

Raghavan, A. K. and Sutton, P. 2002. Jpgc a partial bitstream generation tool to support partial reconfiguration in virtex fpgas. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), Reconfigurable Architecture Workshop (RAW)*.

Silva, M. L. and Ferreira, J. C. 2006. Support for partial run-time reconfiguration of platform fpgas. *J. Syst. Architec. 52,* 12, 709–726.

Steiger, C., Walder, H., and Platzner, M. 2004. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Trans. Comput. 53,* 11, 1393–1407.

Steiger, C., Walder, H., Platzner, M., and Thiele, L. 2003. Online scheduling and placement of real-time tasks to partially reconfigurable devices. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 224–235.

Xilinx. 2007. Virtex-4 configuration guide.http://www.xilinx.com.

Received March 2007; revised October 2007, March 2008; accepted March 2008