Partitioned Scheduling of Periodic Real-Time Tasks onto Reconfigurable Hardware

Klaus Danne and Marco Platzner

Paderborn University
Dept. of Computer Science
{danne, platzner}@upb.de

Abstract

Reconfigurable hardware devices, such as FPGAs, are increasingly used in embedded systems. To utilize these devices for real-time work loads, scheduling techniques are required that generate predictable task timings.

In this paper, we present a partitioning-EDF (earliest deadline first) approach to find such schedules. The FPGA area is partitioned along one dimension into slots. The tasks are partitioned into groups. Then, each group is scheduled to exactly one slot using the EDF rule. We show that the problem of finding an optimal partitioning is related to the well-known 2-dimensional level bin-packing problem. We extend a previously reported ILP model to solve our partitioning problem to optimality. By a simulation study we demonstrate that the partitioning-EDF approach is able to find feasible schedules for most task sets with a system utilization of up to 70%. Additionally, we allow a task to be realized in alternative implementations. A simulation study reveals that the scheduling performance increases considerably if three instead of one task variants are considered. Finally, we model and study the impact of the device reconfiguration time on the scheduling performance.

1 Introduction and Related Work

Reconfigurable hardware devices, such as the field-programmable gate array (FPGA), are general-purpose devices that can be programmed after fabrication. The major resource offered by an FPGA is an array of configurable logic blocks which allows the realization of arbitrary digital circuits. FPGAs are increasingly used in embedded systems as they can deliver a unique trade-off between processor-based solutions and application-specific circuits with respect to performance, cost, and energy efficiency.

SRAM-based FPGA variants can be re-programmed arbitrarily often, even at runtime. Runtime reconfiguration [4] opens up the way to multitasking digital circuits on FPGAs. To manage the reconfigurable resources and the execution of circuits, also denoted as hardware tasks, a runtime system is required. A preemptive runtime system is able to stop a hardware task, save its context, and later on restore the context and resume the task. Concepts and implementations of preemptive runtime systems for FPGAs have been described in, e.g., [5, 13, 11].

The algorithmic challenge in developing a runtime system is to design effective and efficient algorithms for the scheduling and placement of hardware tasks. Such problems have been studied in, e.g., [1, 15, 8, 13, 10, 3, 9] and also under real time constaints in, e.g., [7, 14]. Most authors assume a 2-dimensional area model where tasks are modeled as relocatable rectangles which can be placed anywhere on the FPGA device by partial reconfiguration.

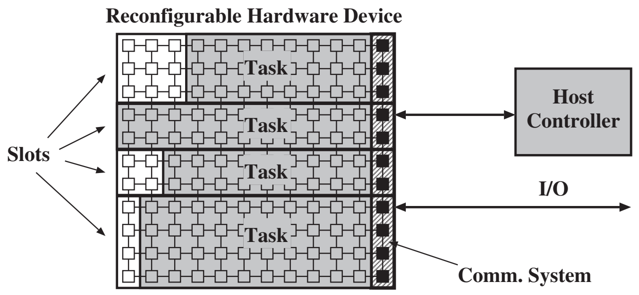

Figure 1. One-dimensional Execution Model

In this work, we develop and investigate a method for scheduling periodic real-time tasks onto a partially reconfigured device using a one-dimensional area model as shown in Fig. 1. The device resource is partitioned into slots that can accommodate hardware tasks. At any time, at most one hardware task is allocated to a slot. Compared to two-dimensional area models, the one-dimensional model potentially leads to a lower device utilization. However, there are two key advantages. First, the one-dimensional model is supported by commercial devices that are able to reconfigure the device in vertical frames, e.g., Xilinx Virtex. Second, task communication and I/O can be realized by a communication system running across the slot structure [17, 16], either within the device or externally [2].

The contributions of this paper are twofold. First, we adapt the EDF (earliest deadline first) partitioning approach known from multiprocessor scheduling to the one-dimensional, slot-based FPGA execution model. Second, we allow for variants of tasks differing in area and speed characteristics, e.g., fast but resource-consuming implementations and slower but resource-sparing versions. This is a common design option for reconfigurable hardware cores. As a result, we can offer a scheduling strategy together with a feasibility test for periodic task sets on reconfigurable hardware systems.

Section 2 presents the resource and execution models used in this work and defines basic terms and metrics. In Section 3, we propose an EDF partitioned scheduling approach respecting implementation variants for tasks, pose several scheduling-related questions, and establish the relationship between our scheduling problem and bin-packing problems. Section 4 introduces the binary integer linear program (BIP) to find the optimal partitioning. The performance of our scheduling approach and the benefits of having implementation variants for tasks are evaluated in Section 5. In Section 6, we study and evaluate the impact of the reconfiguration time overhead on the scheduling performance. Section 7 concludes the paper and points to future work.

2 The Scheduling Problem

2.1 Task and Resource Models

We consider the scheduling of a set of n periodic tasks $\mathcal{F} = \{\Gamma_1, \Gamma_2, \ldots \Gamma_n\}$ onto a reconfigurable device H. The instances $\Gamma_{i,j}$ of task Γ_i are released with period P_i. Each task instance is associated with an absolute deadline $d_{i,j}$. We assume that the relative deadline of Γ_i equals its period.

Each task Γ_i exists in one or more implementation variants $\Gamma_i = \{T_i^1, T_i^2, \ldots\}$. Each variant $T_i^k \in \Gamma_i$ is specified by a worst-case computation time C_i^k and the amount of required reconfigurable logic resources A_i^k.

Table 1 shows an example task set with four periodic tasks $\mathcal{F}^* = \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$. The first two tasks have two implementation variants.

2.2 Execution Model

The considered reconfigurable hardware device H offers a certain amount of computational resources $A(H)$, i.e., the configurable logic blocks of an FPGA. These resources are also referred to as the *area* of the device. In order to perform the requested execution of a task instance $\Gamma_{i,j}$, one of the task's implementation variants T_i^k has to be loaded (configured) onto the device and executed for C_i^k time units. During this execution, an amount A_i^k of the device area is occupied.

Γ_i	T_i^k	P_i	C_i^k	A_i^k	$U^T(T_i^k)$	$U^S(T_i^k)$
Γ_1	T_1^1	12	3	6	1/4	3/2
	T_1^2		6	3	1/2	3/2
Γ_2	T_2^1	4	2	4	1/2	2
	T_2^2		1	8	1/4	2
Γ_3	T_3^1	6	5	3	5/6	5/2
Γ_4	T_4^1	12	2	2	1/6	1/3

Table 1. Example task set with variants $\mathcal{F}^*$

2.3 Feasible Schedule

Generally spoken, we would like to schedule and execute a set of periodic tasks on the reconfigurable device such that each task instance meets its deadline. Let Γ denote the unified set of all implementations of all tasks, i.e., $\Gamma = \{\Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_n\}$. A schedule for a set of periodic tasks $\mathcal{F}$ is given by function $R : \mathbb{R}^+ \to \mathbf{P}(\Gamma)$. $R(t)$ denotes the set of task implementations from Γ running at time t. A schedule is called *feasible*, if for each request $r_{i,j}$ of task Γ_i one of its implementations $T_i^k \in \Gamma_i$ is scheduled for execution for at least C_i^k time units within the interval given by the release time and deadline of Γ_i.

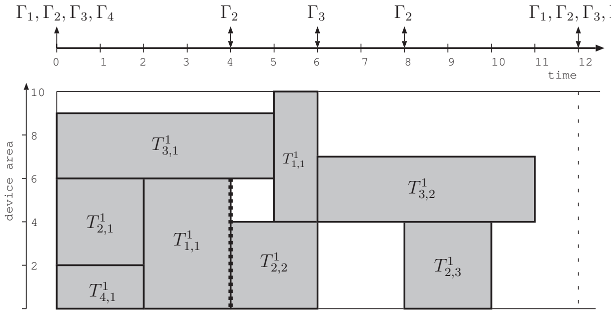

Figure 2. Preemptive (global) schedule

As an example, Fig. 2 displays a possible schedule for the task set shown in Table 1. The upper part of Fig. 2 indicates the release times and deadlines of the tasks. The lower part illustrates which instances of the tasks are executed, and how they share the device area over time. The schedule shown can easily be proven feasible, because every task instance meets its deadline for the entire *hyper-period* of the task set (which amounts to 12 time units). The hyper-period is the least common multiple of all task periods. A feasible schedule defined over the hyper-period can be repeated an infinite number of times.

We are interested in finding answers to following questions:

1. Given a task set $\digamma$, can we execute it on a reconfigurable device H with area $A(H)$, such that all deadlines are met?
2. What is the size of the smallest device on which $\digamma$ can be feasibly executed?
3. How does this feasible schedule R look like?

2.4 Utilization Metrics

We define two utilization metrics to measure the computational load generated by the task implementations of a task set. The time-utilization given by

$$U^T(T_i^k) = \frac{C_i^k}{P_i} \tag{1}$$

is the fraction of time a certain task implementation T_i^k occupies the device in order to complete its execution. The cumulative time-utilization for a set of task-variants $\tilde{\Gamma} \subseteq \Gamma$ is defined as $U^T(\tilde{\Gamma}) = \sum_{T_i^k \in \tilde{\Gamma}} U^T(T_i^k)$. Obviously, a set of task implementations $\tilde{\Gamma}$ cannot be executed sequentially if $U^T(\tilde{\Gamma}) > 1$. Since a running task usually does not occupy the entire device, we define a system-utilization metric that captures the degree by which the device is utilized by T_i^k:

$$U^S(T_i^k) = U^T(T_i^k) \cdot A_i^k \tag{2}$$

In order to measure the load generated by a task set $\digamma$, we now define the total system-utilization of a task set by summing up the system utilization factors of all tasks. If a task has several alternative implementations, we account the one with the minimal system-utilization:

$$U^S(\digamma) = \sum_{\Gamma_i \in \digamma} \min_{T_i^k \in \Gamma_i} \left(U^S(T_i^k) \right) \tag{3}$$

Thus, $U^S(\digamma)$ is the minimum amount of combined area and time required by the task set. If $U^S(\digamma) > A(H) \cdot 1$, no feasible schedule exists since the system is utilized more than 100%. We use the U^S metric in our simulation study, in order to rate the performance of the proposed scheduling algorithm.

3 Partitioned Scheduling

Related work in multiprocessor scheduling differentiates between *non-partitioned* and *partitioned* scheduling. In a non-partitioned or global schedule the different instances of a periodic task can execute on different processors and preempted task instances might even be migrated to other processors before resuming execution. In contrast, in a *partitioned* schedule all instances of a task execute on the same processor. While we studied non-partitioned scheduling in [7, 6], this work applies the concept of a partitioned schedule to our reconfigurable hardware execution model:

Definition 1 (partitioned schedule). *A schedule R for a periodic task set $\digamma$ is said to be* partitioned by χ *with selection f, if the following three statements hold:*

1. *$f(i)$ is a function that selects exactly one implementation variant $T_i^{f(i)}$ for each task Γ_i. The set of all selected implementations is denoted by $\Gamma_{|f} = \{T_1^{f(1)}, T_2^{f(2)}, \ldots\}$.*
2. *$\chi = \{G_1, G_2, \ldots, G_m\}$ is a* partitioning *of $\Gamma_{|f}$. That is, χ is a set of disjunct subsets of $\Gamma_{|f}$, called* partition blocks, *such that the union of all partition blocks results in $\Gamma_{|f}$.*
3. *At any point in time, at most one task implementation of each partition block is executed on the reconfigurable device.*

Each partition block is exclusively assigned a certain area of the reconfigurable device. The area requirement of a partition block $A(G_j)$ is determined by its *largest* task; the required overall device area $A(\chi)$ is given by the cumulated area over all partitions:

$$A(G_j) = \max_{T_i^k \in G_j} (A_i^k), \quad A(\chi) = \sum_{G_j \in \chi} A(G_j) \tag{4}$$

Since each partition block presents an independent task system of its own, we can easily formulate a test whether a given partitioned schedule is feasible by applying the basic results from single-processor EDF (*earliest deadline first*) theory:

Lemma 1 (EDF feasibly partitioned schedule). *Given a periodic task set $\digamma$, let f be a selection function and R be a schedule of the selected task variants $\Gamma_{|f}$ with partitioning χ, such that the tasks of each partition $G_i \in \chi$ are scheduled separately by EDF. The schedule is feasible on the reconfigurable device H if:*

- *$A(\chi) \leq A(H)$, i.e., all partitions fit onto the device, and*
- *$\forall G_i \in \chi : U^T(G_i) \leq 1$, i.e., all partitions have a time utilization of less than 100%.*

Fig. 3 illustrates an EDF partitioned schedule on the example of the task set Γ^* shown in Table 1. In this example,

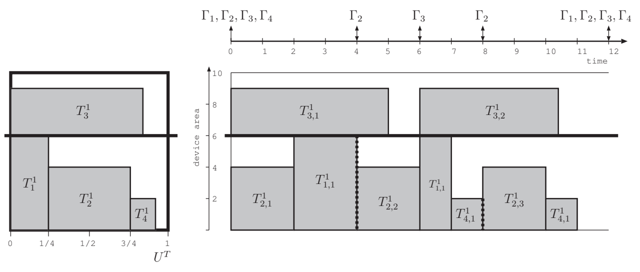

Figure 3. Example of a Partitioned EDF Schedule

the first variant for each task has been selected, i.e., $\Gamma^*_{|f} = \{T_1^1, T_2^1, T_3^1, T_4^1\}$, and two partition-blocks have been generated, e.g., $\chi = \{G_1 = \{T_1^1, T_3^1, T_4^1\}, G_2 = \{T_2^1\}\}$. The left-hand side of Fig. 3 presents the partitioning χ and illustrates the division of the device area. The according partitioned EDF schedule R is shown on the right-hand side of Fig. 3. Since both partition blocks have a time utilization of less than 100%, the schedule is feasible:

$$U^T(G_1) = 1/2 + 1/4 + 1/6 \leq 1, \quad U^T(G_2) = 5/6 \leq 1$$

Based on Lemma 1 we are able to define an optimization problem in order to answer our scheduling questions from Section 2.3. To this end, we formulate the problem of finding the least resource-consuming *EDF Feasibly Partitioned Schedule (EFPS)*:

Definition 2 (EFPS Problem). *Given a periodic task set $\digamma$, find:*

- *a function f that selects a variant for each task and*
- *a partitioning χ of the selected variants $\Gamma_{|f}$,*
- *subject to $U^T(G_i) \leq 1 : \forall G_i \in \chi$,*
- *minimize $A(\chi)$.*

Solving the EFPS problem answers directly question two of Section 2.3: The smallest device that allows to feasibly execute $\digamma$ is of size $A(H) := A(\chi)$. The according decision problem in question one of Section 2.3 is also solved, as given $\digamma$ and a device H, $\digamma$ can be feasibly scheduled if $A(H) \geq A(\chi)$. From the resulting partitioning χ, we can easily derive the scheduling function R which answers our third question.

3.1 Relation to 2-Dimensional Packing Problems

Let us assume for a moment, that each task comes in only one implementation variant, i.e., $\Gamma_i = \{T_i^1\}$ for all i. For this special case, the *EFPS* problem becomes equivalent to the *Two-Dimensional Level Strip-Packing Problem (2LSP)*[12], which is a variant of the *Two-Dimensional Strip Packing (2SP)* problem. In 2SP, a set of rectangular items is packed into a strip of given width but infinite height such that height of the required strip is minimized. The packing must be non-overlapping, orthogonal, and without rotation. In the 2LSP variation of 2SP, the rectangular items have to be packed in rows forming levels. The bottom of the first level is the bottom of the empty strip. The bottom of each other level is determined by a horizontal cut line on top of the highest item in the previous level. Within a level, items are not allowed to be packed on top of each other.

In Definition 2, the tasks variants T_i^k can be modeled as rectangular items where the time utilization factor $U^T(T_i^k)$ corresponds to the rectangle width and the area $A(T_i^k)$ corresponds to the rectangle height. The width of the strip is set to one, according to the maximal time utilization of an EDF schedule. Each slot of the reconfigurable device – and the set of tasks G_i assigned to it – corresponds to one level of the packing. Finally, the required device area corresponds to the height of the stripe.

For the *2LSP* problem, a *Binary Integer Programming (BIP)* model was proposed in [12]. When a task comes in several implementations, the packing problem has to be extended to include a set of possible rectangles for each item. In the following section we will extend the BIP model of [12] accordingly.

4 Optimal Partitioning by ILP

In this section we develop an extension of the BIP model for the 2-dimensional level strip-packing problem presented in [12]. Our extension is able to deal with implementation variants for tasks. Instead of modeling each task with a rectangle, we apply the BIP model on the *set of rectangles* corresponding to all task variants. The BIP constraints are modified, such that one and only one variant of each task is packed.

According to the model of [12], we make the following basic assumptions:

Order of task variants: We sort and renumber the set of variants Γ, according to non-increasing area. Let $\bar{\Gamma}$ denote the sorted set:

$$\Gamma = \{T_1^1, T_1^2, T_1^3, \ldots, T_2^1, T_2^2, \ldots, T_n^1, \ldots\} \quad (5)$$

$$\bar{\Gamma} = \{\bar{T}_1, \bar{T}_2, \bar{T}_3, \ldots\}, \quad (6)$$
$$A(\bar{T}_j) \geq A(\bar{T}_{j+1}), j = 1, \ldots, |\bar{\Gamma}| - 1$$

The index function $\delta(i,k)$ maps the index of a variant T_i^k in Γ to the according task variant in $\bar{\Gamma}$, i.e., $\bar{T}_{\delta(i,k)} = T_i^k$.

Order of partition blocks: The task implementation $\bar{T}_l$ having the largest area within a partition block is said to *initialize* the partition block. The index l is also used to index the partition block, i.e., if the largest task in a partition block is $\bar{T}_l$ the partition block is denoted as G_l.

It follows, that a partitioning χ has $m = |\Gamma|$ potential partition blocks $G_1 \dots G_m$ (some may be empty), and that the partition blocks are sorted by non-increasing areas, i.e., $A(G_l) \geq A(G_{l+1})$. Further, the area of a partition block is equal to the area of the task that initializes the partition block, i.e., $A(G_l) = A(\bar{T}_l)$.

For any task implementation $\bar{T}_j \in \bar{\Gamma}$, we can therefore differentiate the following cases:

- $\bar{T}_j$ is not packed in any G_l since another implementation variant of the task is packed.
- $\bar{T}_j$ initializes a partition block, i.e., $T_j \in G_j$.
- $\bar{T}_j$ is packed into a partition block with greater area, forcing the partition block index to be smaller than j.

The assumptions made are without loss of generality but reduce the search space considerably, since a task implementation T_i^k cannot be assigned to an arbitrary partition block G_l. When modelling the BIP, we use the binary decision variable $x_{l,j}$ to indicate that task variant $\bar{T}_j$ is packed into partition block G_l:

$$x_{l,j} = \begin{cases} 1 & \text{if } \bar{T}_j \in G_l \\ 0 & \text{if } \bar{T}_j \notin G_l \end{cases}$$
$$l = \{1, \dots, |\Gamma|\}, j = \{1, \dots, |\Gamma| : i \geq l\} \quad (7)$$

If $x_{l,j} = 1$ and $j = l$, the partition block G_l is said to be *initialized* and its largest task is $\bar{T}_l$. If $x_{l,l} = 0$, G_l is uninitialized and empty. The resulting ILP has three main components:

1. The cost function accumulates the area required by all non-empty partition blocks, hence minimizing the overall area $A(\chi)$.

2. A set of constraints ensures that for each task Γ_i exactly one variant T_i^k is packed into one of the partition blocks. For task Γ_i this can be expressed by the term:

$$\left(\sum_{l=1}^{\delta(i,1)} x_{l,\delta(i,1)}\right) + \left(\sum_{l=1}^{\delta(i,2)} x_{l,\delta(i,2)}\right) + \dots + \\ + \left(\sum_{l=1}^{\delta(i,k)} x_{l,\delta(i,k)}\right) + \dots + \left(\sum_{l=1}^{\delta(i,m)} x_{l,\delta(i,m)}\right) = 1 \quad (8)$$

The first term of Eq. 8 equals one, if T_i^1 is packed into some partition block, the second term checks whether variant T_i^2 is packed into some partition block, and so on. Note, that by convention a variant $T_{i,k}$ can only be packed into $G_1, \dots, G_{\delta(i,k)}$. Therefore, the upper bounds of the sum indices are given by $\delta(i,k)$.

3. Another set of constraints enforces EDF schedulability. Each non-empty partition block must have a time utilization less or equal than 1, i.e., $U^T(G_l) \leq 1$.

The final ILP can be written as:

$$\boxed{\begin{aligned} &\min \sum_{l=1}^{|\Gamma|} A(\bar{T}_l) \cdot x_{l,l} \\ &\sum_{k=1}^{|\Gamma_i|} \left(\sum_{l=1}^{\delta(i,k)} x_{l,\delta(i,k)} \right) = 1 \qquad i \in \{1, \dots, n\} \\ &\sum_{j=l}^{n} U^T(\bar{T}_j) \cdot x_{l,j} \leq 1 \cdot x_{l,l}, \qquad l \in \{1, \dots, |\Gamma|\} \end{aligned}} \quad (9)$$

Let n be the number of tasks in $\digamma$ and $|\Gamma|$ be the cumulative number of variants over all tasks. The corresponding ILP contains $|\Gamma| \cdot (|\Gamma| + 1)/2$ binary variables and $n + |\Gamma|$ constraints.

To illustrate the BIP model, we consider again the example task set of Table 1. The four tasks of $\digamma^*$ have a total of six task implementations (the set Γ) which get sorted and renumbered according to non-increasing areas. The resulting set $\bar{\Gamma}$ is given by:

$$\bar{\Gamma} = \{\bar{T}_1 := T_2^2, \bar{T}_2 := T_1^1, \bar{T}_3 := T_2^1, \bar{T}_4 := T_1^2, \\ \bar{T}_5 := T_3^1, \bar{T}_6 := T_4^1\} \quad (10)$$

Fig. 4 shows the optimal partitioning χ, which consists of the two partition blocks $G_3 = \{T_1^2, T_2^1\}$ and $G_5 = \{T_3^1, T_4^1\}$. Since $|\Gamma| = 6$, the BIP model contains $6 \cdot (6+1)/2 = 21$ binary variables. The configuration of the decision variables $x_{l,j}$ is shown in Table 2.

The required overall device area is 7 units. For comparison, if each task comes only in one variant (the first one), the optimal partitioning would require 9 units of device area (see Fig. 3). The benefit of having task variants comes at the price of a higher model complexity. To quantify the scheduling performance of the ILP considering task variants, we will present simulation results in the next section.

5 Performance Evaluation

Introducing implementation variants for tasks makes it more likely, that an EFPS exists for a reconfigurable de-

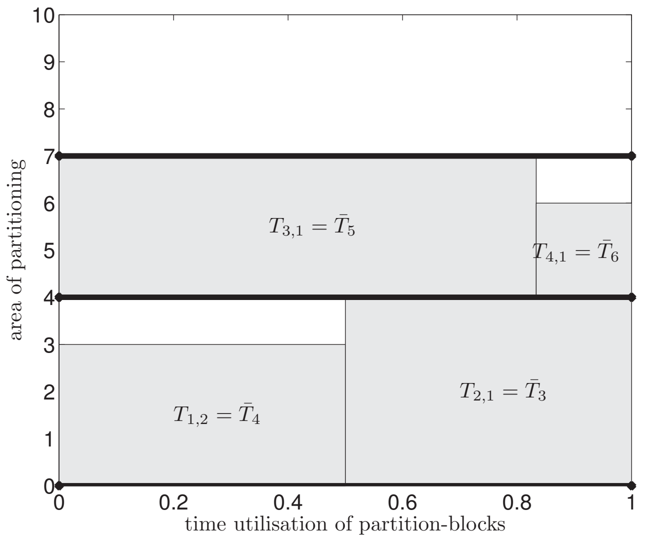

Figure 4. Optimal partitioning of Γ^*

l	$x_{l,1}$	$x_{l,2}$	$x_{l,3}$	$x_{l,4}$	$x_{l,5}$	$x_{l,6}$
1	0	0	0	0	0	0
2	-	0	0	0	0	0
3	-	-	1	1	0	0
4	-	-	-	0	0	0
5	-	-	-	-	1	1
6	-	-	-	-	-	0

Table 2. The configuration of decision variables

vice of given size. However, finding such a schedule becomes computationally more complex since the design space grows rapidly with the number of implementation variants. In order to evaluate the benefit we gain by introducing implementation variants we present a simulation study.

To create a benchmark of task sets without and with variants, we used the following procedure: First, we generated periodic task sets Γ (with only one variant per task) with various system utilizations by composing randomly generated task implementations. For each task $\Gamma_1 = \{T_i^1\}$, the area A_i^1 was chosen according to a uniform distribution in $[0.1, 0.5]$ and the computation time and period were chosen[1] such that $U^T(T_i^1)$ is uniformly distributed in $[0.1, 0.5]$. Such a setup creates task sets of up to 15 tasks. Based on these task sets, we created task sets with variants using the following methods:

3 variants per task: To the existing variant T_i^1 of each task, we added one variant T_i^2 with half the computation time but doubled area requirement, i.e., $T_i^2 = (P_i^1, \frac{1}{2}C_i^1, 2 \cdot A_i^1)$, and another variant with doubled computation time but only half the area requirement, i.e., $T_i^3 = (P_i^1, 2 \cdot C_i^1, \frac{1}{2} \cdot A_i^1)$.

up to 5 variants per task: For each task the number of variants was randomly chosen from 1 to 5. Each variant T_i^k was created choosing a form factor q from the continuous interval $[1, 4]$ and a sign s from $\{-1, +1\}$. The computation time and area were set to $C_i^k = C_i^1 \cdot q^s$ and $A_i^k = A_i^1 \cdot q^{-s}$, respectively. Note, that each variant still has the same system utilization as the initial variant: $U^S(T_i^k) = C_i^k/P_i \cdot A_i^k = C_i^1 q^s/P_i \cdot A_i^1 q^{-s} = U^S(T_i^1)$

[1]First $U^T(T_i^1)$ was chosen from $[0.1, 0.5]$, then C_i^1 was chosen from $\{1,...,30\}$ and the period was set to $P_i^1 = \text{round}\,(C_i^1/U^T(T_i^1))$.

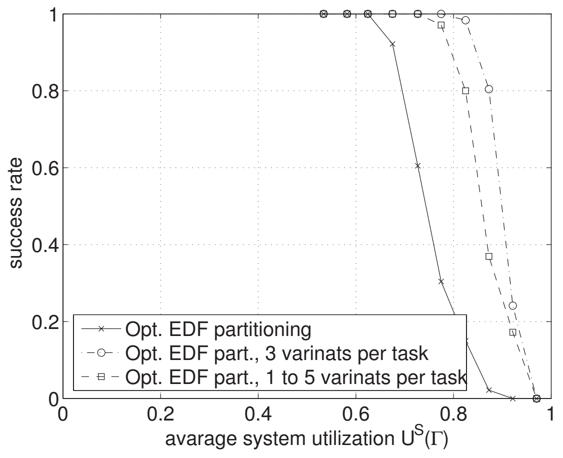

Figure 5. Relative success rate with and without implementation variants

Fig. 5 shows the relative scheduling success rates, depending on the task sets system utilization, when scheduled onto a device H of size $A(H) = 1$. For an average system utilization of 0.7, the optimally partitioned EDF schedule produced feasible solutions for about 80% of the task sets in the reference case (with only one variant per task). Introducing two additional implementation variants for each task yields an enormous advantage - even task sets with average system utilization of 0.87 could be scheduled with an 80% success rate. If some tasks have more and some tasks have less variants, we can also observe a great improvement over the reference case. 80% of the task sets with average $U^S \simeq 0.83$ could be successfully scheduled.

To solve the BIP model, we employed the *lp_solve* library version 4.0 on a 2.8 GHz Pentium 4 machine. We were able to optimally solve most problem instances with a size of $|\Gamma|$ up to 30 in less than 15 seconds. That is, we could compute the schedule for 30 tasks with only one implementation variant, or for 10 tasks with three variants each. In particular, from the 1000 task sets generated for the evaluation in Fig. 5, only 28 exceeded the 15 second time limit.

6 Accounting for Reconfiguration Overhead

Up to know, we have neglected the overheads due to context switching and running the scheduling routines in our analysis. This is common practice in microprocessor-based real time systems, but only justified when the time overhead is small compared to the tasks' computation times. In the following, we analyze the overheads for reconfigurable systems in more detail.

In an EDF schedule, a task preemption can only occur when other tasks are released. For periodic task sets we know in advance how many task releases can appear within a certain period and thus we can derive the worst-case number of preemptions N_i of an instance of a periodic task Γ_i by other tasks of $\mathcal{F}$:

$$N_i = \sum_{\Gamma_j \in \mathcal{F}} \left\lfloor \frac{P_i}{P_j} \right\rfloor - 1 \tag{11}$$

6.1 Reconfiguration Overhead in a Partitioned Schedule

In our execution model, the tasks in each partition block G_i are scheduled separately by EDF and the context switches in a partition block are realized by *partial* reconfiguration. Hence, tasks executing in other partition blocks do not need to be interrupted. Let G_l be the partition block which contains the selected implementation of Γ_i. The maximum number of preemptions of a task Γ_i is given by:

$$N_i = \sum_{T_j^k \in G_l} \left\lfloor \frac{P_i}{P_j} \right\rfloor - 1 \tag{12}$$

Equation 12 is identical to Equation 11, but the number of preemptions N_i for each task may be smaller.

Now we are ready to account for the reconfiguration overhead by increasing the computation times of all tasks in Γ. We have to add $(1 + N_i)$ times the required time for the device reconfiguration[2], once for reconfiguring the task before it starts execution and N_i times when it needs to resume its execution after preemption (Eq. 13). The time utilization of the task increases according to Eq. 14.

$$\tilde{C}_i^k = C_i^k + (1 + N_i) \cdot t_{reconf} \tag{13}$$

$$\tilde{U}^T(T_i^k) = U^T(T_i^k) + \frac{(1 + N_i) \cdot t_{reconf}}{P_i} \tag{14}$$

[2]Note, that we have to account the time t_{reconf} for a full device reconfiguration, even if we assume partial reconfiguration. Although the time to reconfigure a small portion of the device will be shorter than t_{reconf}, in the worst case all partition blocks may request a reconfiguration simultaneously.

Obviously, a partitioning χ of a task set $\mathcal{F}$ might be feasibly schedulable when neglecting the reconfiguration overhead, but may become infeasible when considering the increased time utilization of Eq. 14. To decide whether a feasible partitioning χ including reconfiguration overhead exists, is a difficult problem: The optimal partitioning χ of tasks depends on the tasks' time utilization $\tilde{U}^T$, but the actual time utilization of a task depends on the partitioning, i.e., on the tasks in the same partition block. One might think about packing rectangles that change their widths during the packing process, depending on the other rectangles currently packed to the same level. We cannot reasonably account for this cross dependency in our ILP model.

Making the simplifying assumption that the overheads due to preemption are similar for all partition blocks, the following two steps seems to be a reasonable heuristic:

1. Given the task set $\mathcal{F}$ and device area $A(H)$, compute the partitioning χ that minimizes the maximal time utilization over all partition blocks:

$$\min \left(\max_{G_l \in \chi} \left(U^T(G_l) \right) \right) \tag{15}$$

2. Account for the reconfiguration overhead of each task according to Eq. 14. If the updated time-utilization factors for each partition block $\tilde{U}^T(G_l)$ are still below one, the partitioned schedule is still feasible.

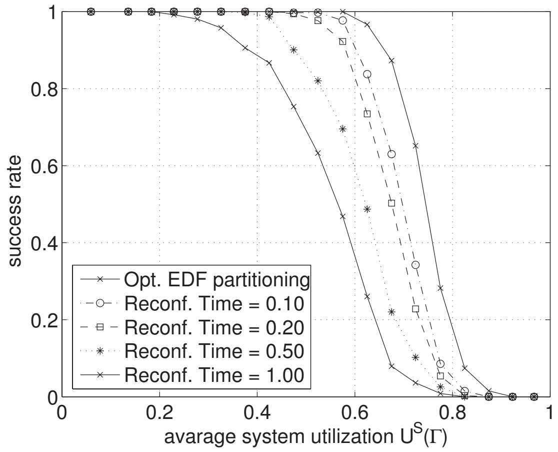

Figure 6. Impact of device reconfiguration time

Fig. 6 shows the relative scheduling success rates for the task sets with one variant per task, including reconfiguration overheads. The device reconfiguration time is varied between $0.1, 0.2, 0.5$ and 1 times the computation time of the shortest task.

The results show that the effect of device reconfiguration can almost be neglected, when the runtime of the shortest tasks is about one magnitude higher than the device reconfiguration time. Even at reconfiguration times in the range of the runtime of the shortest task, a considerable number of task sets could be feasibly scheduled.

7 Conclusion

In this paper, we considered the problem of scheduling periodic real-time tasks to a reconfigurable hardware device operated in a one-dimensional area model. The problem differs from previous work in that we apply a partitioned scheduling approach and respect arbitrary implementation variants for hardware tasks. We showed the relation of the considered problem to two-dimensional packing problems, namely an extension of the *2-dimensional-level-strip-packing* problem. Based on a previously reported ILP model for such a packing problem, we developed an extended ILP model to compute the optimal partitioned schedule for a given task set. In a simulation study, we evaluated the performance of the scheduling approach. The main characteristics of our approach and the resulting findings are:

- The one-dimensional area model can be realized with todays reconfigurable devices and tools.
- The partitioned-EDF scheduling approach achieves an acceptable device utilization.
- The device utilization increases considerably if tasks come in several implementation variants.
- The optimal partitioning for medium-sized task sets (up to 30 task variants) can be computed in reasonable time.
- The reconfiguration time overhead can be included into the scheduling method.

We intend to follow several lines for future work. We will work on heuristics and approximation algorithms in order to find partitioned schedules for larger task sets. Further, we plan to compare the partitioning approaches to other scheduling techniques, such as global EDF scheduling [7, 6]. A more detailed modeling of the reconfiguration port and context save and restore mechanisms could possibly lead to improved scheduling guarantees. Finally, we seek to implement our scheduler in a runtime system prototype.

References

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Design and Test of Computers*, pages 68–83, Mar. 2000.

[2] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, J. Teich, S. P. Fekete, and J. van der Veen. The erlangen slot machine: A highly flexible FPGA-based reconfigurable platform. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005.

[3] G. Brebner and O. Diessel. Chip-based Reconfigurable Task Management. In *Field-Programmable Logic and Applications (FPL'01)*, pages 182–191. Springer-Verlag, Berlin, Germany, 2001.

[4] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.

[5] K. Danne. Memory management to support multitasking on FPGA based systems. In *Proc. of the International Conference on Reconfigurable Computing and FPGAs (ReConFig04)*. SMCC, Sept. 2004.

[6] K. Danne and M. Platzner. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware. In *Proc. of the Intern. Conf. on Field Programmable Logic and Applications (FPL05)*, Tampere, Finland, IEEE Aug. 2005.

[7] K. Danne and M. Platzner. Periodic real-time scheduling for FPGA computers. In V. Turau and C. Weyer, editors, *The Third IEEE Intern. Workshop on Intelligent Solutions in Embedded Systems (WISES'05) at Hamburg University of Technology*. IEEE 2005.

[8] O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt. Dynamic scheduling of tasks on partially reconfigurable FPGAs. *IEEE Proc. – Computers and Digital Techniques*, May 2000.

[9] H. ElGindy, M. Middendorf, B. Scheuermann, and H. Schmeck. An evolutionary approach to dynamic task scheduling on FPGAs. In Hartenstein and Grnbacher, editors, *Field-Programmable Logic and Applications (FPL'00)*. Springer-Verlag, Berlin, September 2000.

[10] J. S. N. Jean, K. Tomko, V. Yavagal, J. Shah, and R. Cook. Dynamic reconfiguration to support concurrent applications. *IEEE Trans. on Computers*, 48(6):591–602, June 1999.

[11] H. Kalte and M. Porrmann. Context saving and restoring for multitasking in reconfigurable systems. In *15th Intern. Conf. on Field Programmable Logic and Applications*, Tampere, Finland, IEEE Aug. 2005.

[12] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 202.

[13] H. Simmler, L. Levinson, and R. Manner. Multitasking on FPGA coprocessors. In *FPL*, pages 121–130, 2000.

[14] C. Steiger, H. Walder, and M. Platzner. Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-time Tasks. *IEEE Transactions on Computers*, 53(11):1392–1407, November 2004.

[15] J. Teich, S. Fekete, and J. Schepers. Optimization of dynamic hardware reconfigurations. *The J. of Supercomputing*, 19(1):57–75, May 2000.

[16] M. Ullmann, M. Hübner, B. Grimm, and J. Becker. An FPGA run-time system for dynamical on-demand reconfiguration. In *IPDPS*, 2004.

[17] H. Walder and M. Platzner. A runtime environment for reconfigurable operating systems. In *In Proceeding 14th International Conference on Field-Programmable Logic and Applications (FPL)*, 2004.