# Practical Design of a Computation and Energy Efficient Hardware Task Scheduler in Embedded Reconfigurable Computing Systems

Tyrone Tai-On Kwok and Yu-Kwong Kwok
Department of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam Road, Hong Kong
Corresponding Author: Yu-Kwong Kwok (email: *ykwok@hku.hk*)

## Abstract

*By utilizing massively parallel circuit design in FPGAs, the overall system efficiency, in terms of computation efficiency and energy efficiency, can be greatly enhanced by offloading some computation-intensive tasks which are originally executed in the instruction set processor to the FPGA fabric. In essence, a hardware task scheduler is needed. However, most of the work in the literature considers scheduling algorithms which are unable or difficult to be implemented using the design flows in current development platform. Moreover, little of the work takes energy consumption into consideration. In this paper, we present the design of a hardware task scheduler which takes energy consumption into consideration, and can be readily implemented using current design flows.*

## 1. Introduction

A typical embedded reconfigurable computing system is composed of an instruction set processor and some FPGA fabric. By utilizing massively parallel circuit design in FPGAs, computation-intensive tasks in software applications, which are originally executed in the instruction set processor, can be carried out by hardware and hence increase the overall system efficiency. Moreover, the FPGA fabric can be dynamically reconfigured for different tasks at run-time.

The above-mentioned platform is an ideal platform for systems that need high performance demands and need to adapt to the changing workload requirements. For example, it suits ideally for a pervasive computing system where it needs to execute numerous context-aware tasks. Moreover, it is also suitable for systems where energy efficiency is a key concern, since executing computation-intensive tasks in hardware can be more energy-efficient than executing in the processor. All these request for a reconfigurable computing system that can be operated in a multitasking manner. Specifically, a hardware task scheduler is needed. However, most of the work in the literature considers scheduling algorithms which are unable or difficult to be implemented using the design flows in current development platform [4], such as 2D resource models [3]. The reason is that current development platform only allows the FPGA area to be partitioned into a number of reconfigurable modules having the same height, as detailed in [4]. On the other hand, little of the work takes energy consumption into consideration. In this paper, we present the design of a hardware task scheduler that can utilize the flexibility of FPGAs, provide a balance between computation efficiency and energy efficiency, and more importantly, can be readily implemented using current design flows. The rest of this paper is organized as follows. Section 2 describes the scheduling model. Sections 3 and 4 present our proposed hardware task scheduling algorithm and its evaluation, respectively. Finally, we conclude in Section 5.

This research was supported by a HKU CRCG Small Project Research Grant under project number 10205130.

## 2. Scheduling Model

### 2.1. Reconfigurable Architecture

In our study, we consider *only* reconfigurable architecture that is feasible for implementation using current technology. Figure 1 shows the target reconfigurable architecture. As can be seen, the FPGA device is divided into two areas, static area and dynamic area. The dynamic area implements some reconfigurable modules (RMs) for executing some hardware tasks, while the static area implements a static module which servers as a bridge between the host CPU and the RMs. The function of the static area is fixed after the whole FPGA device is configured.

We suppose that the dynamic area comprises $W \times H$ Configurable Logic Blocks (CLBs), where $H$ is the height of the reconfigurable device and $W$, the width, is variant to the size of the dynamic area. On the other hand, we assume that the size of the smallest RM is of $W_{MIN} \times H$. In practical implementation, the height of each RM spans the full height of the device [4], and $W_{MIN}$ depends on the hardware task which requires the smallest number of CLBs. Then, the maximum number of RMs is:

$$N_{MAX} = \frac{W}{W_{MIN}} \qquad (1)$$

Here, we assume that $W$ is a multiple of $W_{MIN}$. After the whole FPGA device is reconfigured, the dynamic area can be divided into different number of RMs of different widths (we call this different partitioning strategy of the dynamic area), so that different hardware tasks of different CLB requirements can be executed simultaneously in the dynamic area. In addition, each RM can be partially reconfigured, without affecting the rest of the device. This way, the system can adapt to different workload requirements.

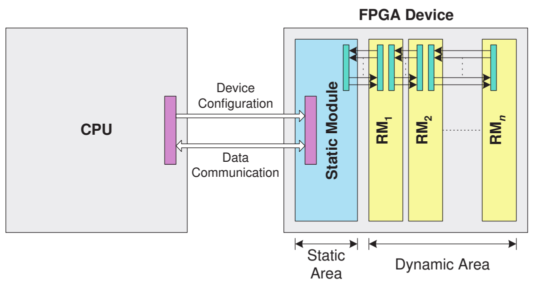


**Figure 1. Target reconfigurable architecture.**

## 2.2. Task Definition

We assume that the hardware tasks executing in the RMs are independent and cannot be preempted. A hardware task $T_i$ in our target reconfigurable architecture is characterized by two parameters: 1.) $f_i^{MAX}$, which specifies the maximum clock frequency, in MHz, the hardware task can be executed at; and 2.) $w_i$, which specifies the width of the reconfigurable module needed (it should be reminded that all the RMs share the same height). In addition, each hardware task has execution time $t_i$ and energy consumption $E_i$ when it is working at $f_i^{MAX}$ MHz for a predefined amount of work. $t_i$ can be found out by considering the amount of work to be done by the task. $E_i$ can be estimated by adopting the energy model developed by Choi *et al.* [2]. Moreover, when scheduled in the system, each task has an arrival time $A_i$ and completion time $C_i$. All the above-mentioned attributes and parameters of a task $T_i$ are stored in a vector $\mathbf{v_i}$. In our target system, a special type of hardware task called NOP (no-operation) is defined for each RM of different widths. We assume that NOP tasks will consume negligible energy when scheduled in the RMs. The use of NOP tasks will be elaborated in Section 3.2.

Formally, the following two functions are defined to extract the estimated execution time and energy consumption information, respectively, of each task $T_i$ executing at different frequency $f$:

$$g^{time}(\mathbf{v_i}, f) = \frac{f_i^{MAX}}{f} \cdot t_i \qquad (2)$$

$$g^{energy}(\mathbf{v_i}, f) = \frac{f}{f_i^{MAX}} \cdot E_i \qquad (3)$$

When applying the energy model developed by Choi *et al.* [2], it is shown that the energy consumption is directly proportional to the working frequency, which is the reason why Equation 3 is such defined.

## 3. Hardware Task Scheduler Design

### 3.1. The Runtime System

Figure 2 depicts the block diagram of our target runtime system. Specifically, a user's application program is partitioned into two types of tasks, namely software task to be executed by the instruction set processor and hardware task to be executed by the reconfigurable device. In this paper, we focus on developing an algorithm for scheduling the hardware tasks. The long-term goals of the runtime system are to minimize the total energy consumption and total execution time of the set of hardware tasks:

$$Minimize(\sum g^{energy}(\mathbf{v_i}, f)) \qquad (4)$$

$$Minimize(\max_i(C_i) - \min_i(A_i)) \qquad (5)$$

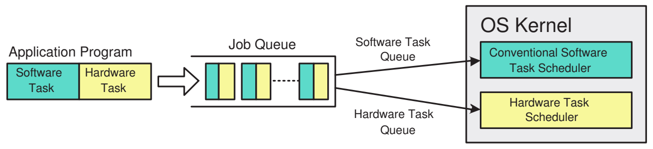


**Figure 2. Target runtime system.**

## 3.2. The Proposed Scheduling Algorithm

Algorithms 1–3 describe our proposed hardware task scheduling algorithm, namely **ECfEE** (**E**nergy-**E**fficient and **C**omputation-**E**fficient algorithm with **f**requency adaptation). By referring to Figure 2, our algorithm works on a hardware task queue, $T_{queue}$, where hardware tasks are continuously being injected into it. Our algorithm considers the first $q_{range}$ hardware tasks when it carries out the scheduling procedure. To prevent some tasks from starving, we introduce another parameter, $s_i$, to each task $T_i$.

In Step 3 of Algorithm 1, $F$ denotes the set of frequencies that *all* the RMs in the reconfigurable device may be clocked at. All RMs sharing the same working frequency is a consideration for practical implementation, as recommended by [4]. Moreover, the fact that the available frequencies are stepped by 5 is also a practical consideration. The primary reason is to reduce the memory space required for storing the configuration files of the reconfigurable device. In $SelectWorkingFrequency()$ of Algorithm 3, we choose a working frequency for the scheduled tasks such that all the tasks can finish execution all together, with the minimal average difference of execution time. The reason for doing so is that it can reduce the amount of executing NOP tasks in Steps 14–25 of Algorithm 1, so as to better use the resources of RMs, and hence better use the energy of the system.

In Steps 14–25 of Algorithm 1, after a task has finished execution, we do not choose any task that can fill the RM. There are two reasons for that. Firstly, we want to avoid the loop of scheduling NOP tasks when there is no task suitable for execution under the current partitioning strategy but we need to wait for the executing tasks to finish. Secondly, after all the tasks under the current partitioning strategy finish their execution, the system can consider another partitioning strategy so as to schedule the maximum possible number of tasks for execution.

The dominating computation of **ECfEE** is in the $SelectPS-T()$ function, where for each partitioning strategy, $q_{range}$ tasks have to be looked up so as to choose a suitable partitioning strategy. Thus, the complexity of **ECfEE** is $O(p \cdot q_{range})$ where $p$ is the maximum number of partitions in the dynamic area.

**Algorithm 1 ECfEE**

```
Schedule($T_{queue\_head}$)
 1: $q_{range} \leftarrow \alpha \cdot N_{MAX}$ /* $\alpha$ is a predefined constant */
 2: $s_{threshold} \leftarrow \beta \cdot N_{MAX}$ /* $\beta$ is a predefined constant */
 3: $F \leftarrow \{f_{MIN}, f_{MIN}+5, f_{MIN}+10, ..., f_{MAX}\}$
 4: $P \leftarrow GeneratePartitioningStrategies(N_{MAX})$
 5: $p_{current} \leftarrow null$ /* current partitioning strategy used */
 6: $T_{scheduled} \leftarrow null$ /* set of scheduled tasks */
 7: $T_{not\_scheduled} \leftarrow null$ /* considered but not scheduled */
 8: $f_{working} \leftarrow null$ /* selected working frequency */
 9: while (TRUE) do
10:   $(p_{current}, T_{scheduled}, T_{not\_scheduled}) \leftarrow$
      $SelectPS\text{-}T(T_{queue\_head}, q_{range}, s_{threshold}, P, T_{not\_scheduled})$
11:   $f_{working} \leftarrow SelectWorkingFrequency(T_{scheduled}, F)$
12:   Reconfigure the dynamic area for execution of tasks.
13:   while (not all the tasks have finished execution) do
14:     if a task has finished execution in $RM_j$ then
15:       if there is some other task still executing then
16:         if $\exists$ task $T_i \in T_{queue}[1..q_{range}]$ and $T_i$ will finish
            execution no later than any current executing task
            then
17:           Schedule $T_i$ to execute in $RM_j$.
18:           if $T_i \in T_{not\_scheduled}$ then
19:             $T_{not\_scheduled} \leftarrow T_{not\_scheduled} - T_i$
20:           end if
21:         else
22:           Schedule NOP task to execute in $RM_j$.
23:         end if
24:       end if
25:     end if
26:   end while
27: end while
```

**Algorithm 2 ECfEE**—Selection of Partitioning Strategy and Tasks for Scheduling

```
SelectPS-T($T_{queue\_head}, q_{range}, s_{threshold}, P, T_{not\_scheduled}$)
 1: if $\exists\ s_i \geq s_{threshold}$ and $T_i \in T_{not\_scheduled}$ then
 2:   $s_j \leftarrow \max(s_i | T_i \in T_{not\_scheduled})$
 3:   Select $p_{current} \in P$ which allows the task having $s_j$ and
      the maximum number of other tasks $\in T_{queue}[1..q_{range}]$
      to execute in the RMs.
 4: else
 5:   Select $p_{current} \in P$ which can accommodate the maxi-
      mum number of tasks within $T_{queue}[1..q_{range}]$.
 6: end if
 7: Denote the set of tasks which contributes to the selection of
    $p_{current}$ as $T_{scheduled}$.
 8: $T_{not\_scheduled} \leftarrow T_{not\_scheduled} \bigcup (T_{queue}[1..q_{range}] - T_{scheduled})$
 9: $T_{queue} \leftarrow T_{queue} - T_{scheduled}$
10: $s_j \leftarrow s_j + 1$, where $T_j \in T_{not\_scheduled}$
```

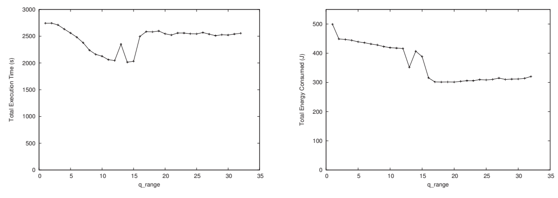
**Algorithm 3 ECfEE**—Selection of Working Frequency for the Tasks Selected for Scheduling

```
SelectWorkingFrequency($T_{scheduled}, F$)
 1: $f^{MAX}_{scheduled} \leftarrow \min(f^{MAX}_i | T_i \in T_{scheduled})$
 2: For each $f \in F$ and $f_{MIN} \leq f \leq f^{MAX}_{scheduled}$, calculate:
      • $G^{time}_{average} = \frac{1}{|T_{scheduled}|} \cdot \sum g^{time}(\mathbf{v_i}, f)$, where $T_i \in T_{scheduled}$
      • $\sum(G^{time}_{average} - g^{time}(\mathbf{v_i}, f))^2$, where $T_i \in T_{scheduled}$
 3: Denote the $f$ which gives the smallest value of $\sum(G^{time}_{average} - g^{time}(\mathbf{v_i}, f))^2$ in the previous step as $f_{working}$.
```

## 4. Simulation Results

Because currently there is no benchmark package for a reconfigurable system, similar to [3], the performance of the proposed scheduling algorithm is studied through simulation. We construct $T_{queue}$ by randomly generating a set of tasks of the following parameters: 1.) $N_{MAX} = 8$ and $W_{MIN} = 1$, and task width $w_i \in [1, 2, 4, 8]$; 2.) $F_{MIN} = 20$ and $F_{MAX} = 50$ such that $f_i^{MAX} \in [20..50]$ and $f_{working} \in [20, 25, 30, 35, 40, 45, 50]$; 3.) $t_i \in [500, 5000]$ ms; and 4.) $E_i \in [100, 1200]$ mJ, which is proportional to $w_i$.

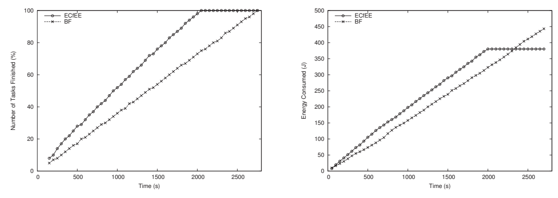
In our simulations, we set $\beta = 2$, i.e., $s_{threshold} = 16$. Figures 3(a) and 3(b) show, respectively, the total execution time and total energy consumed in executing 1000 randomly generated tasks. As can be seen, for $q_{range}$ smaller than 15, the total execution time and total energy consumed decrease with $q_{range}$. The reason is that by using a larger $q_{range}$, the scheduler can choose a better partitioning strategy to schedule more tasks for better utilization of the RMs. This also means that the tasks are likely to execute at a frequency lower than $f_i^{MAX}$ and hence energy is saved. On the other hand, for $q_{range}$ larger than 15, an increase in $q_{range}$ causes the total execution time to increase and the total energy consumed to decrease. This is due to the fact that by further increasing $q_{range}$, the scheduler can further schedule more tasks in a partitioning strategy. However, by considering more tasks, the $f_{scheduled}^{MAX}$ chosen will be of a lower value. As a result, the $f_{working}$ chosen will also be of a lower value, causing the tasks take more time to finish, but save more energy. Thus, from Figures 3(a) and 3(b), we can see that $q_{range} = 15$ is a good tradeoff between the total execution time and total energy consumed. In our simulations, the same value is observed when 2000 tasks are executed.



(a) execution time (b) energy consumed

**Figure 3. Total execution time and energy consumed vs. different values of $q_{range}$.**

The problem of placing reconfigurable modules onto the dynamic area, as discussed in this paper, is similar to the 1D bin-packing problem. Best-Fit and First-Fit are two well-known online algorithms for the 1D bin-packing problem, and they have been considered for hardware task placement [1]. In our study, we have adapted Best-Fit (**BF**) to compare the performance with our proposed scheduling algorithm. **BF** is effectively **ECfEE** with $q_{range} = 1$. To choose a suitable partitioning strategy, **BF** only considers the head task of the task queue for each partitioning strategy. Thus, the complexity of **BF** is $O(p)$ where $p$ is the maximum number of partitions in the dynamic area. In Figure 4, we compare our proposed scheduling algorithm **ECfEE** with **BF**. For the comparison, 1000 tasks are executed. From the figures, we can see that, when compared to **BF**, **ECfEE** can significantly reduce execution time and energy consumption by 26% and 14%, respectively.



(a) task completion (b) energy consumed

**Figure 4. Comparison of ECfEE and BF.**

## 5. Conclusions

By exploiting adaptive working frequency of hardware tasks, our proposed scheduling algorithm shows significant reduction in both execution time and energy consumption. This result gives some insight on the potential of changing the working frequency of hardware tasks in task scheduling.

## References

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 68–83, Jan. 2000.

[2] S. Choi, J.-W. Jang, S. Mohanty, and V. K. Prasanna, "Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures," *Proc. Engineering of Reconfigurable Systems and Algorithms (ERSA'02)*, June 2002.

[3] C. Steiger, H. Walder, and M. Platzner, "Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks," *IEEE Trans. Computers*, vol. 53, no. 11, pp. 1393–1407, Nov. 2004.

[4] Xilinx Inc., "Xilinx Application Note XAPP290: Two Flows for Partial Reconfiguration: Module-Based or Difference-Based," v1.2 edition, Sept. 2004.