Online Binding of Applications to Multiple Clock Domains in Shared FPGA-based Systems

Farzad Samie, Lars Bauer, Chih-Ming Hsieh and Jörg Henkel
Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany

***Abstract*—Modern FPGA-based platforms provide multiple clock domains and their frequencies can be changed at runtime by using PLLs and clock multiplexers. This is especially beneficial for platforms that run several applications simultaneously (e.g. modern wireless sensor nodes that are shared by multiple users), as different processing modules may be fed by different clock frequencies at different time windows. However, since the number of clock domains on a platform is limited, several processing modules need to share the same clock domain.**

In this paper, we study the problem of binding multiple applications to multiple clock domains, such that the latest finishing time of any application (i.e. the *makespan*) is minimized. We present an Integer Linear Programming (ILP) formulation and then propose a novel algorithm that (i) quickly identifies those applications that are *dominated* by others (and thus can be ignored without losing optimality) and that (ii) uses the *ascending* property of the optimal binding to reduce the search space. The experimental results show up to 17% makespan reduction compared to state-of-the-art. The overhead when executing on a low-power SmartFusion2 SoC equipped with an ARM Cortex-M3 core is on average 8.9 ms, i.e. our algorithm is suitable for runtime decisions.

I. Introduction

Current developments in embedded systems like multimedia systems or modern wireless sensor network (WSN) platforms integrate multiple independent applications on a single device [1, 2, 3]. For example, consider a powerful WSN platform in a smart building which is shared by multiple departments for different purposes [4]. The building owner may deploy a structural monitoring application to detect locations of possible damages. The security department may need to detect the presence of an intruder by deploying a security application which processes video, image or acoustic data [5]. Furthermore, in a smart conference room, a 'meeting detection' application [6] may be used to detect the start/end of a meeting and analyze if the room condition is satisfactory for attendees during the meeting. Yet another application may process the acoustic signals collected by a microphone to record the proceedings of the meeting. In this example, the embedded platform (WSN node) is shared by **multiple independent applications** which are needed to (a) run in parallel but typically not all at the same time, and (b) share one platform to keep cost and maintenance of the platform low.

The main design goal is energy efficiency, especially for battery-operated systems due to their limited energy supply. Dynamic Power Management (DPM) is one of the most commonly applied techniques for energy conservation in FPGA-based embedded systems [7, 8], especially those that operate in an alternating active/sleep pattern. As an example, consider the Fast Fourier Transformation (FFT) operation that is used for acoustic event recognition. Table I reports the execution time, current and energy consumption of an FFT at different clock frequencies. Increasing the operating frequency increases the current consumption but saves more than 50% energy consumption due to the execution time reduction. This example

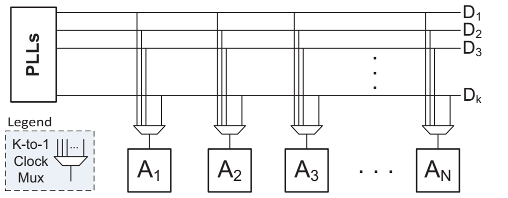

Fig. 1: Several clock domains are driven by PLLs. The hardware module of each application A_i can be connected to one clock domain.

shows that systems that follow the alternating active/sleep pattern (i.e. waking up, processing fast, going back to sleep), can reduce their energy consumption by minimizing the *makespan*, i.e. the time required to process all the given tasks. This is particularly the case for low-power flash-based FPGAs that do not need to be reconfigured first after waking up.

Typically, applications are implemented on the FPGA as independent hardware modules [9, 10]. Each hardware module is associated with its maximum operating frequency that is determined by the maximum delay of the critical path within its hardware implementation.

Commercially available FPGAs provide several clock domains [11], while the clock frequency of domains can be modified dynamically during runtime [12]. The number of clock domains available on a specific FPGA platform is fixed and limited. For instance, the Microsemi IGLOO (AGL600), Smartfusion (A2F500), and Xilinx Spartan III (XC3S200) devices which are already used as WSN nodes [13, 14, 15] have 3, 6 and 4 clock domains, respectively. When the number of applications exceeds the number of clock domains, multiple applications need to share a common clock domain as shown in Figure 1. Then, the clock frequency of a domain is determined by the frequency of its slowest application.

A particular binding of applications to domains may significantly affect the applications' execution times and thus the makespan. To determine the best binding, not only the applications' maximum frequency, but also the required clock cycles to complete their execution must be considered.

Motivational Example: Let us assume that three applications with maximum frequencies of 50, 110 and 220 MHz are implemented on an FPGA-based system and they need 100, 2200 and 1100 clock cycles to finish, respectively. If only one clock domain is available, then its clock frequency must necessarily be the lowest maximum frequency of all applications (here: 50 MHz). In this case, the makespan is

TABLE I: Energy consumption of FFT operation on reconfigurable platform at different clock frequencies

Frequency [MHz]	Time [s]	Current [mA]	Energy [mJ]
48	0.28	45.17	15.18
96	0.14	53.61	9.01
144	0.09	60.28	6.51
deep	N/A	9.88	N/A

$\max(\frac{100}{50}, \frac{2200}{50}, \frac{1100}{50})\frac{cycles}{MHz} = 44\,\mu s$. If two clock domains are available, then the makespan can be minimized by binding the first application to a 50 MHz clock domain and the others to a 110 MHz clock domain, which leads to a makespan of $\max(\frac{100}{50}, \frac{2200}{110}, \frac{1100}{110})\frac{cycles}{MHz} = 20\,\mu s$. Other mappings of applications to clock domains (e.g. $\max(\frac{100}{50}, \frac{2200}{50}, \frac{1100}{220})\frac{cycles}{MHz} = 44\,\mu s$) lead to suboptimal makespans.

The application binding needs to be accomplished online due to the facts that **(i)** some applications may not be needed over a period of time, or **(ii)** their workload (clock cycles) may change according to the runtime situation. A priori knowledge about dynamic application requirement and workload is typically not available for all applications. For instance, the meeting detection application in the smart conference room needs to process acoustic data only during a meeting, while at other times it is not used. Moreover, the facility application needs more updated information about the room condition during a meeting, which changes its workload at runtime.

The contributions of this paper are the following:

- We present an ILP formulation for the problem of binding multiple applications to clock domains and determining the frequency of each domain.
- We propose a novel algorithm to identify applications that are *dominated* by others to reduce the problem size.
- We introduce the *ascending* property of optimal binding to reduce the search space.

Paper structure: in Section II, we provide an overview of the related work. After that, we present the problem formulation in Section III and provide a detailed presentation of our novel algorithm in Section IV. Experimental results and overhead analysis are presented and discussed in Section V, while Section VI concludes the paper.

II. Related Work

Using multiple clock domains in both general-purpose and embedded microprocessor has attracted a lot of attention in the recent years, especially in *globally asynchronous and locally synchronous* (GALS) systems in order to maximize performance and energy efficiency [16]. Nevertheless, their domains are statically assigned and cannot be reconfigured [17]. In contrast to general-purpose and embedded microprocessors, the applications in FPGA-based systems are implemented as hardware modules each of which has a maximum frequency. This makes existing dynamic frequency scheduling algorithms for multi-core architectures [18] inapplicable for multi-application FPGA-based systems.

Some work has considered using multiple clock domains in reconfigurable FPGA-based systems. Shaochun [19] has suggested using a multi-clock domain mechanism to increase the throughput of streaming applications that are implemented on FPGA platforms. These applications execute a set of simultaneous processes on a large sequence of data items. The proposed mechanism uses a fast clock to speed up the bottleneck processes. However, the general problem of binding applications to clock domains has not been studied.

Dittmann et al. [20] proposed to use multiple clock domains on partially reconfigurable FPGAs to hide the reconfiguration latency. They partition the tasks into clock domains considering their execution time. They group tasks with similar clock frequencies into one domain. However, they have not taken the number of clock cycles of each task into account.

Sirowy et al. [21] studied the problem of partitioning hardware accelerators among multiple clock domains for a SoC equipped with microprocessor and FPGA. They developed a dynamic programming algorithm to determine the binding of applications to clock domains in polynomial time, but they focused on a single sequential application whose critical functions are implemented on hardware accelerators (i.e. the hardware accelerators do not run in parallel). Even though their goal is to minimize the *sum* of accelerator execution times, their approach is the closest to ours and we extended it to minimize the *makespan* and compare with it in Section V.

III. Formal Problem Formulation

We consider a total of N applications $\mathcal{A} = \{A_1, A_2, \ldots, A_N\}$, each of which is implemented by a dedicated hardware module on the FPGA. Without loss of generality, we assume that all these applications are being used (i.e. those that are unused at a certain time are not present in this set).

- Application A_i, $1 \le i \le N$, has the maximum frequency f_i and an execution time of c_i cycles. Applications are sorted into a non-decreasing order of their maximum frequency (i.e. $f_1 \le f_2 \le \cdots \le f_N$).
- Applications need to be bound to multiple clock domains. It is analogous to **partition** set $\mathcal{A}$ into K pairwise disjoint subsets denoted by $\mathcal{D} = \{D_1, \cdots, D_K\}$, such that

$$\bigcup_{1 \le j \le K} D_j = \mathcal{A}$$
$$\forall \quad i, j, i \ne j : \quad D_i \cap D_j = \emptyset \tag{1}$$

- The frequency Φ_j of clock domain D_j is determined by the slowest application bound to it (i.e. $\Phi_j = \min_{A_i \in D_j} f_i$).
- Assuming that application A_i is bound to clock domain D_j, its execution time is $t_i = \frac{c_i}{\Phi_j}$.
- The main objective is to minimize the makespan.

$$\text{minimize} \quad \overbrace{\max_{1 \le j \le K} \left\{ \frac{c_i}{\Phi_j} : A_i \in D_j \right\}}^{\text{makespan } \mathcal{T}} \tag{2}$$

IV. Proposed Solution

A. Integer Linear Programming (ILP) Formulation

We present an Integer Linear Programming (ILP) formulation of our clock domain partitioning problem.

$$\min \quad \mathcal{T} \tag{3}$$
$$\text{subject to} \quad \forall i: \quad \sum_j x_{ij} = 1 \tag{4}$$
$$\forall i, j: \quad x_{ij} \times \Phi_j \le f_i \tag{5}$$
$$\forall i, j: \quad \mathcal{T} \ge x_{ij} \times c_i \times \frac{1}{\Phi_j} \tag{6}$$

where $\mathcal{T}$ denotes the *makespan* (see Eq. (2)) and

$$x_{ij} = \begin{cases} 1 & \text{if } A_i \text{ is in clock domain } D_j \text{ (i.e. } A_i \in D_j) \\ 0 & \text{otherwise} \end{cases}$$

Though the presented formulation is non-linear, it can be easily linearized by standard techniques [22]. For instance, Eq. (6) not only consists of a rational term $({}^1/_{\Phi_j})$, it also contains a product of two variables $(x_{ij} \times {}^1/_{\Phi_j})$ which both make it non-linear, respectively. We replace the expression $^1/_{\Phi_j}$ by a variable w_j to handle the rational term. Then we linearize Eq. (6) by reformulating it to $\mathcal{T}-c_i w_j \geq -M\times(1-x_{ij})$ where M is a sufficiently large constant coefficient value representing a known upper bound for the term $c_i w_j$ [22].

B. Problem size reduction

The number of ways to partition N elements into K non-empty subsets is called '**Stirling number of the second kind**' in the field of combinatorics [23] and it is denoted by $S(N,K)$. In [23], its lower bound $L(N,K) \leq S(N,K)$ is proven to grow exponentially with N and K. In order to reduce the problem size (i.e. N) we define the following criterion:

Definition 1: An application A_x is said to *dominate* another application A_y if $f_x \leq f_y$ and $c_x \geq c_y$. Note that whenever the frequencies or execution cycles of two applications are identical, then one application always dominates the other.

Claim 1: If application A_x *dominates* A_y, then they can be bound to the same clock domain D_j without losing optimality.

Proof: Follows immediately from Def. 1. The frequency of D_j cannot be faster than $\min(f_x, f_y) = f_x$. Thus, the makespan of D_j cannot be smaller than $\max(\frac{c_x}{f_x}, \frac{c_y}{f_x}) = \frac{c_x}{f_x}$. This means that the makespan of D_j cannot be reduced by binding A_y to any other domain. Intuitively, binding A_y to any other domain D_l, can also not improve the makespan of D_l and therefore, the makespan of the system cannot be reduced by binding A_y to any other domain than D_j. This shows that A_y can be excluded from the binding (partitioning) problem, as it can be bound to the same clock domain as A_x without losing optimality. ■

The number of all combinations to be examined for finding all dominated applications is $\binom{N}{2} = N\times(N-1)/2$ which is of order $\mathcal{O}(N^2)$. We now introduce a property that can be used to reduce the number of necessary examinations.

Claim 2: None of the applications in set $\mathcal{A}' \subseteq \mathcal{A}$ can dominate another application *if and only if* there is an application sequence $(A_{s_i}), A_{s_i} \in \mathcal{A}', 1 \leq i \leq |\mathcal{A}'|$ such that the sequence is sorted into an increasing order of maximum frequency **and** an increasing order of execution clock cycles, i.e. $\forall 1 \leq i < |\mathcal{A}'| : f_{s_i} < f_{s_{i+1}}$ and $c_{s_i} < c_{s_{i+1}}$.

Proof: Suppose for the purpose of contradiction that $\mathcal{A}'$ is not sorted into strict increasing order of (i) execution clock cycles and (ii) frequencies. Then $A_{s_x}, A_{s_y} \in \mathcal{A}'$ can be found, where either (i) $f_{s_x} < f_{s_y}$ but $c_{s_x} \not< c_{s_y}$ (i.e. $c_{s_x} \geq c_{s_y}$), or (ii) $c_{s_x} < c_{s_y}$ but $f_{s_x} \not< f_{s_y}$ (i.e. $f_{s_x} \geq f_{s_y}$). According to Definition 1, in case (i) A_{s_x} dominates A_{s_y} and in case (ii) A_{s_y} dominates A_{s_x}, which is a contradiction. ■

We now present an efficient method inspired by Claim 2 that removes all applications that are dominated by others. The applications are given in a sequence sorted by their maximum frequency. Starting from the application with the lowest frequency, the procedure iterates over all N applications. If two successive applications are identified where one dominates the other, then the dominated application is removed. The time complexity is of order $\mathcal{O}(N)$. After performing the reduction procedure, the reduced set is not only sorted into an increasing order of maximum frequency, but **also** into an increasing order of execution clock cycles. Therefore, according to Claim 2, no application in the reduced set can dominate another one.

Though the problem size N can be reduced using the above-mentioned procedure, the number of possible partitions is still large in some cases. For example, if there are $N = 11$ applications to be bound to $K = 4$ clock domains, there are $136,421$ different options. Assuming that 3 applications are dominated by others, the procedure reduces the problem size to 8. However, there are still 1701 different combinations to be examined in order to find the optimum partition. This necessitates further improvements to reduce the problem complexity to make the algorithm suitable for runtime execution.

C. Search space reduction

We introduce a property of our partitioning problem which provides an opportunity to reduce the solution space.

Definition 2: For sets $\mathcal{B}, \mathcal{C} \subseteq \mathcal{A}$, we define $\mathcal{B} <_f \mathcal{C}$ if and only if for each application $A_b \in \mathcal{B}$ and each $A_c \in \mathcal{C}$ it holds that $f_b < f_c$.

Definition 3: A partition $\mathcal{D} = \{D_1, \cdots, D_K\}$ (see Eq. (1)) is called an *ascending partition* of set $\mathcal{A}$, if for each two subsets D_b and D_c $(1 \leq b < c \leq K)$ it holds $D_b <_f D_c$.

Figure 2 shows examples for an ascending partition (a) and a non-ascending partition (b) of 11 applications into 4 domains.

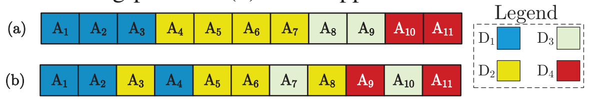
(a)
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11
(b)
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11
Legend
D1
D2
D3
D4

Fig. 2: (a) An ascending partition ($D_1 <_f D_2 <_f D_3 <_f D_4$) and (b) a non-ascending partition of 11 applications into 4 clock domains

Claim 3: An ascending partition $\mathcal{D}^*$ of applications $\mathcal{A}$ into K non-empty domains ($|\mathcal{A}| > K$) has the minimum makespan.

Proof: For the sake of contradiction, assume that there is a partition $\mathcal{D} = \{D_1, \ldots, D_K\}$ that does not fulfill the $<_f$ property, but that has a makespan $\mathcal{T}$ better than the makespan of all ascending partitions. Hence, there must be $D_i, D_j \in \mathcal{D}$,

$$D_i = \{\ldots, A_x\}, \quad D_j = \{A_y, \ldots, A_z\}, \quad 1 \leq i < j \leq K, \ f_y < f_z,$$

where $f_y < f_x < f_z$ violates the $<_f$ property. Since the applications are also sorted in increasing order of their execution cycles (Section IV-B), $c_y < c_x < c_z$. We now show that there is another partition that does not have this violation and whose makespan is less than or at least equal to $\mathcal{T}$. Generally, the makespan of clock domain D_h is $\frac{\max\{c_l : A_l \in D_h\}}{\min\{f_l : A_l \in D_h\}}$, hence, $\mathcal{T}_{D_i} = c_x/\Phi_i$ and $\mathcal{T}_{D_j} = c_z/\Phi_j$ where $\Phi_j = f_y$.

We interchange A_y and A_x, which results into two new sets

$$D_i' = D_i \cup \{A_y\} \setminus \{A_x\} \quad \text{and} \quad D_j' = D_j \cup \{A_x\} \setminus \{A_y\}.$$

Let C denote the maximum clock cycle of applications in D_i' which necessarily holds $C < c_x$. The makespan of clock domain i and j become $\mathcal{T}_{D_i'} = C/\Phi_i'$ and $\mathcal{T}_{D_j'} = c_z/\Phi_j'$, respectively, where $\Phi_j' > f_y$ and $\Phi_i' = f_y$ or Φ_i. Since $\Phi_j' > f_y$, $\mathcal{T}_{D_j} > \mathcal{T}_{D_j'}$. On the other side, Φ_i' is either f_y, which implies $\mathcal{T}_{D_j} > \mathcal{T}_{D_i'}$ or it is Φ_i, which implies $\mathcal{T}_{D_i} > \mathcal{T}_{D_i'}$, due to the fact that $C < c_x < c_z$. It concludes that $\max(\mathcal{T}_{D_i'}, \mathcal{T}_{D_j'}) < \max(\mathcal{T}_{D_i}, \mathcal{T}_{D_j})$. To summarize, the new sets have a makespan that is less than or equal to $\mathcal{T}$, which is a contradiction. ■

Claim 3 implies that for finding the optimal solution, it is sufficient to consider the ascending partitions. Let $\Gamma(N,K)$ denote the number of possible ascending partitions of a set with size N into $K \leq N$ non-empty subsets. It satisfies the following recurrence relation:

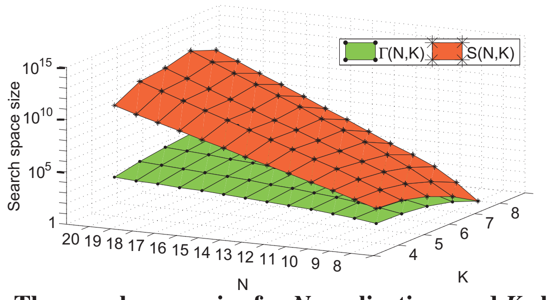

Fig. 3: The search space size for *N* applications and *K* clock domains before (*S*(*N*,*K*)) and after (Γ(*N*,*K*)) applying our reduction algorithm

$$\Gamma(N,K) = \Gamma(N{-}1,K) + \Gamma(N{-}1,K{-}1) \quad (7)$$

$$\Gamma(i,i) = 1, \quad \Gamma(i,1) = 1, \quad 1 \le i \le K \quad (8)$$

We derive a closed form expression for the number of ascending partitions, $\Gamma(N,K)$ based on the relations in Eq. (7) and (8).

$$\Gamma(N,K) = \binom{N-1}{K-1} \quad (9)$$

The inequality $\Gamma(N,K) < L(N,K)$ for $1 < K \le N$ can be proven by induction, similar to the description in [23], which, due to the lack of space, cannot be described in depth. This indicates that the number of ascending partitions is strictly less than the 'Stirling number of the second kind', i.e. $S(N,K)$. Figure 3 shows the search space size of our problem by plotting $S(N,K)$ and $\Gamma(N,K)$. While $S(N,K)$ indicates the original search space size, $\Gamma(N,K)$ represent the search space size after using our proposed reduction procedure (considering just ascending partitions). Hence, using Claim 3 reduces the search space by orders of magnitude.

D. Algorithm for fast binding of applications to clock domains

Without losing optimality, the algorithm starts with the application with the lowest clock frequency, i.e. A_1, and binds it to D_1 which sets the clock frequency of D_1 to f_1 (regardless of which other applications will be bound to it). For the next application, i.e. A_2, there are two possible options: (1) A_2 can be bound to D_1 and run at f_1, or (2) A_2 can be bound to D_2, which would determine the clock frequency of D_2 as f_2. In general, assuming that applications $A_1, \ldots, A_i$ $(1 \le i < N)$ are already bound to $k' < K$ clock domains, there are at most two options for binding A_{i+1}:

- binding it to clock domain $D_{k'}$ or
- binding it to a new clock domain $D_{k'+1}$ and setting its clock frequency to f_{i+1}.

The restriction to not consider any other clock domain (e.g. $D_{k'-1}$) for binding A_{i+1} ensures that the partition is ascending. All possible ascending partitions can be presented in a tree structure. We construct the tree in an iterative approach as shown in Algorithm 1.

Figure 4 shows a tree associated with the search space of a partitioning problem with 7 applications and 3 clock domains. Any straight line shows a clock domain which is labeled with its frequency. At the i^{th} tree level, the clock domain and operating frequency of A_i is determined. Each path from the root of the tree to any leaf node corresponds to a specific ascending partition. To find the optimal binding, the tree has to be traversed from the root to all leaves.

Algorithm 1: Tree construction for ascending partitions

```
Inputs : Number of applications $N$, number of clock domains $K$
Output : root node of constructed tree
root ← new Node();
root.AppId ← 1;
root.DomainId ← 1;    // Slowest application is bound to the first domain
getDomain(1).setFrequency($f_1$);
queue ← new Queue();
queue.push(root);
while !queue.empty() do    // Tree is not completely constructed yet
    node ← queue.pop();
    i ← node.AppId;
    j ← node.DomainId;
    if $N-i < K-j$ then    // To make sure none of domains remains unused
        left_child ← new Node();
        left_child.AppId ← i+1;
        left_child.DomainId ← j; // Is bound to the same Domain as its parent
        node.setLeftChild(left_child);
        queue.push(left_child);
    end
    if $j < K$ then    // Application can also be bound to a new domain
        right_child ← new Node();
        right_child.AppId ← i+1;
        right_child.Domain ← j+1; // The first (slowest) node on this domain
        getDomain($j+1$).setFrequency($f_{i+1}$);
        node.setRightChild(right_child);
        queue.push(right_child);
    end
end
return root;
```

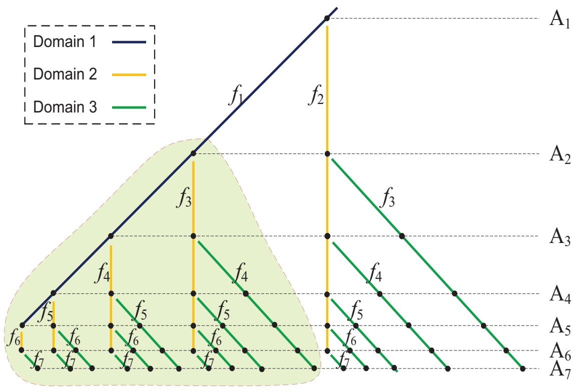

Fig. 4: The tree structure represents ascending partitions for *N*=7 applications and *K*=3 clock domains; shaded area shows the sub-tree for *L*=6 applications

For a given number of applications and clock domains, the tree structure is actually independent of specific application properties such as their maximal frequencies f_i and execution cycles c_i. Thus, the tree needs to be constructed only once. For a particular binding problem, just the corresponding information f_i, c_i of each node need to be updated. Furthermore, when less than the maximal number of applications for which the tree was constructed are required (i.e. active at the same time), then a subtree of the constructed tree can be used. Let us assume that a binding problem for $L<N$ applications shall be solved, then the subtree rooted at the leftmost node at level $N{-}L{+}1$ can be used. For example, the shaded area in Figure 4 shows the subtree for $L{=}6$ applications.

V. Evaluation & Results

A. Experimental Setup

In this section we provide experimental results and comparisons that show the effectiveness of the proposed solution. For this purpose, we employ a reconfigurable SoC platform

TABLE II: Maximum frequency of some applications

App.	RS_enc	RS_dec	CRC	AES	Huffman	ECC
max freq. [MHz]	302	161	436	177	66	97

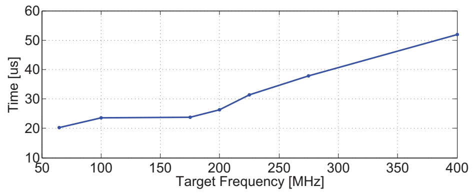

Fig. 5: The time overhead to generate target frequency using PLL on a SmartFusion2 M2S010 platform

based on a SmartFusion2 M2S010 SoC from Microsemi that integrates a non-volatile flash-based ultra-low power FPGA fabric and an ARM Cortex-M3 processor on a single chip.

We choose some typical applications which have been used in reconfigurable systems and WSNs like [5, 24, 25, 26]. These applications include AES, ECC, CRC, Huffman encoding, and Reed-Solomon (RS) error correction. Table II shows the maximum frequency of the applications implemented on our reconfigurable platform. This table is used to derive a possible range for the maximum clock frequency of typical applications.

We consider a varying number of applications ranging from 5 to 20. For each application we consider a maximum clock frequency ranging from 50 up to 500 MHz as it is typical for real applications (see Table II). The clock cycles that the applications need are chosen in the range of 10^5 to 10^6 cycles, which is typical for complex applications [5, 21]. We consider 2 to 8 clock domains, which is typical for current FPGA-based platforms [13, 14, 15].

B. Overhead Analysis

Design Overhead: Our proposed technique and algorithms do not require any change or extension on FPGA and hardware. Dynamically binding the applications to any of the K different clock domains at runtime introduces a negligible hardware overhead, i.e. the clock multiplexers (one K-to-1 mux for each application) as shown in Figure 1. Note that the clock network resources on FPGAs (i.e. global and local clock signals as well as global clock buffer) are adequate to support online clock binding for multiple applications [11].

For instance, the Smartfusion (A2F500) provides six clock domains (PLLs) that are fed to six global clock networks and three additional networks per quadrant. Altogether, there are 18 global networks (6 global + 3 per Quadrant * 4 Quadrants), which allows feeding up to 18 different applications.

Time Overhead: Once the application binding has completed, the clock frequency of each domain is known. The PLLs have to be programmed at runtime to generate the required clock frequencies, which comes at the cost of time overhead. Figure 5 shows the time that the PLLs need to settle at the target frequency. It lasts at most 52 μs till the PLLs are locked, which is quite negligible wrt. the application execution time.

C. Comparison with State-of-the-art

To compare our approach with state-of-the-art, we have re-implemented the algorithm from Sirowy et al. [21], whose approach is the closest to our problem (see Section II), and adopt it to optimize for makespan. We implemented [21] and our proposed algorithm on the Cortex-M3 micro-controller of the SmartFusion2 SoC FPGA and exercise them with identical

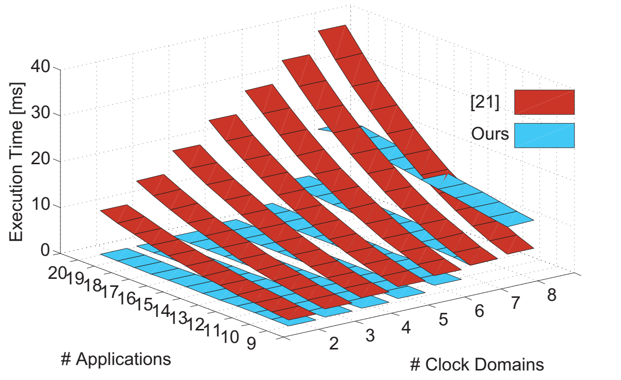

Fig. 6: The overhead to run the algorithm and find the best binding of applications to clock domains

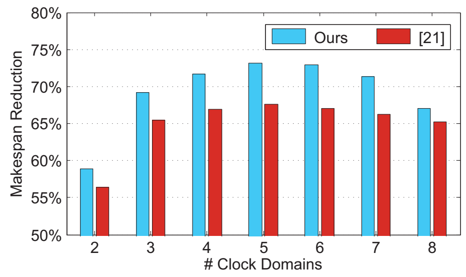

Fig. 7: The average makespan reduction for varying numbers of clock domains compared to having only one clock domain

application sets as input. Like our proposed algorithm, the modified version of [21] also determines the optimal binding, but it does not benefit from our novel reduction procedures.

We first compare the algorithm execution times in Figure 6 to evaluate the impact of our contribution. For a given number of clock domains (typically fixed for a certain FPGA-based platform), an increasing number of applications leads to an exponentially increasing algorithm execution time for [21], whereas the execution time of our algorithm only slightly increases. The good scalability and generally short execution time of our proposed algorithm allow to use it online to dynamically change the number of active applications and recalculate an optimal binding of active applications to clock domains after every change. Only when having 8 or more clock domains together with only few more applications than clock domains, [21] performs faster. But such a large number of clock domains is untypical for FPGA-based platforms and thus we did not further optimize for that case.

Now we evaluate to which degree the reduced overhead improves the makespan. Note that the overall makespan consists of (i) calculating the application to clock binding, (ii) reconfiguring the clocks, and (iii) executing the applications. Both, our algorithm and [21] calculate the optimal application to clock binding in step (i), thus they require exactly the same time for steps (ii) and (iii), but the time to execute step (i) differs. Figure 7 illustrates the average makespan improvement of our algorithm and of [21] obtained by using multiple clock domains compared to the case of just one clock domain (which then necessarily has to be the lowest frequency of the application set, i.e. f_1). Although using more clock domains provides more opportunities to decrease the makespan, it also leads to larger overhead which erases some of the original benefit. As shown in Figure 7, using more than 6 clock domains decreases the average makespan improvement gained

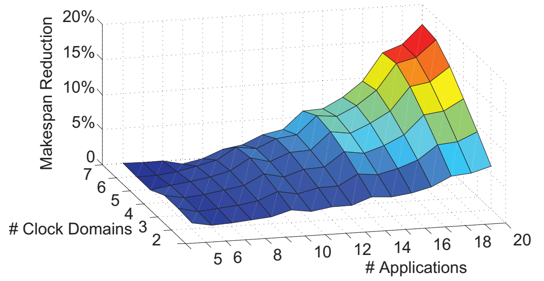

Fig. 8: The makespan reduction due to our algorithm compared to [21]

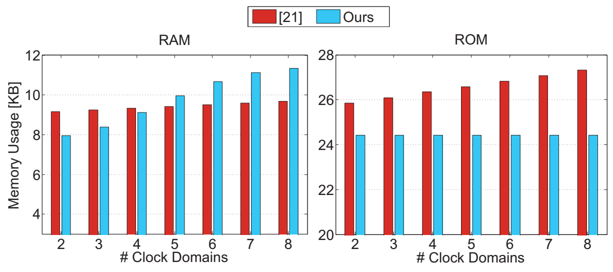

Fig. 9: Code Size and Memory Usage

by having multiple clock domains. It implies that exploiting more clock domains is not gainful.

Figure 8 shows the detailed makespan reduction achieved by our algorithm compared to [21]. Our algorithm reduces the overall makespan by up to 17% compared to state-of-the-art. The improvement of our proposed algorithm over [21] increases as the number of applications or number of clock domains increases. The reason is that the time overhead of our algorithm increases more gracefully compared to [21].

Code Size and Memory Usage: The binding algorithms are both implemented as software applications to run on the micro-controller. The software code is stored on the ROM (eNVM on our platform). At runtime, the software code is loaded into the RAM (eSRAM on our platform) to improve the performance. In addition, the stack, buffers and heap are allocated on the RAM. Figure 9 shows the memory usage for storing and executing the software code for our algorithm and [21]. Our algorithm reduces the code size by up to $2.9\,\mathrm{KB}$ and increases the RAM usage by at most $1.5\,\mathrm{KB}$ compared to [21]. As the number of clock domains increases, the code size of [21] increases due to the data structures and tables used in its dynamic programming approach. On the other side, our algorithm scales better with respect to code size.

VI. Conclusion

In this work we studied the problem of binding multiple applications to multiple clock domains in order to minimize the makespan. We presented an Integer Linear Programming formulation for the binding problem. Then, we proposed reduction algorithms to reduce the problem size and solution space without loosing optimality. Experimental results show that, by using our binding algorithm, a low-power SoC equipped with an ARM Cortex-M3 can achieve up to 17% reduction in makespan with respect to state-of-the-art. The average time overhead for running our algorithm on this platform is $8.9\,ms$ which allows using it for runtime decisions.

Acknowledgment

This work was supported in part by the German Research Foundation (DFG), Research Training Group GRK 1194.

References

[1] Chih-Ming Hsieh, et al. Hardware/software co-design for a wireless sensor network platform. In *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 1–10, 2014.

[2] Akash Kumar, et al. Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 13(3):40, 2008.

[3] Yang Yu, et al. Supporting concurrent applications in wireless sensor networks. In *Int'l Conf. on Emb. Netw. Sensor Syst.*, pp. 139–152, 2006.

[4] You Xu, et al. Near optimal multi-application allocation in shared sensor networks. In *International symposium on Mobile ad hoc networking and computing (MobiHoc)*, pp. 181–190, 2010.

[5] Fucai Liu, Zhiping Jia, and Yibin Li. A novel partial dynamic reconfiguration image sensor node for wireless multimedia sensor networks. In *International Conference on HPCC-ICESS*, pp. 1368–1374, 2012.

[6] Andrey Temko and Climent Nadeu. Acoustic event detection in meeting-room environments. *Pattern Recog. Letters*, 30(14):1281–1288, 2009.

[7] Young-Si Hwang, Sung-Kwan Ku, and Ki-Seok Chung. A predictive dynamic power management technique for embedded mobile devices. *IEEE Transactions on Consumer Electronics*, 56(2):713–719, May 2010.

[8] B. de Abreu Silva and V. Bonato. Power/performance optimization in FPGA-based asymmetric multi-core systems. In *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, pp. 473–474, Aug 2012.

[9] Harry Sidiropoulos, et al. A platform-independent runtime methodology for mapping multiple applications onto FPGAs through resource virtualization. In *International Conference on FPL*, pp. 1–4, 2013.

[10] Li-minn Ang, et al. FPGA wireless multimedia sensor node hardware platforms. In *Wireless Multimedia Sensor Networks on Reconfigurable Hardware*, pp. 69–103. Springer Berlin Heidelberg, 2013.

[11] Julien Lamoureux and Steven JE Wilton. FPGA clock network architecture: flexibility vs. area and power. In *International symposium on Field programmable gate arrays (FPGA)*, pp. 101–108, 2006.

[12] Ian Brynjolfson and Zeljko Zilic. Dynamic clock management for low power applications in FPGAs. In *Custom Integrated Circuits Conference (CICC)*, pp. 139–142, 2000.

[13] Björn Stelte. Toward development of high secure sensor network nodes using an FPGA-based architecture. In *Int'l Wireless Communications and Mobile Computing Conference (IWCMW)*, pp. 539–543, 2010.

[14] Chih-Ming Hsieh, Zhonglei Wang, and Jörg Henkel. A reconfigurable hardware accelerated platform for clustered wireless sensor networks. In *Int'l Conf. on Parallel and Distr. Systems (ICPADS)*, pp. 498–505, 2012.

[15] Yana Esteves Krasteva, et al. Embedded runtime reconfigurable nodes for wireless sensor networks applications. *IEEE Sensors Journal*, 11(9):1800–1810, 2011.

[16] John Oliver, et al. Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor. *ACM SIGARCH Computer Architecture News*, 32(2):150, 2004.

[17] Chi Bun Chan, Jingzhao Ou, and Jeffrey D Stroomer. Clock frequency exploration for circuit designs having multiple clock domains, September 13 2011. US Patent 8,020,127.

[18] Navid Toosizadeh, Safwat G Zaky, and Jianwen Zhu. Using variable clocking to reduce leakage in synchronous circuits. In *International Conference on Computer Design (ICCD)*, pp. 328–335, 2010.

[19] Shaochun Guo. Realization of replicated streaming applications on multi-clocked FPGAs. Master's thesis, KTH, School of Information and Communication Technology (ICT), 2011.

[20] Florian Dittmann and Tales Heimfarth. Clock frequency variation of partially reconfigurable systems. In *ARCS Workshops*, pp. 195–204, 2006.

[21] S. Sirowy, et al. Clock-frequency assignment for multiple clock domain systems-on-a-chip. In *Design, Automation and Test in Europe Conference (DATE)*, pp. 1–6, 2007.

[22] H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 2013.

[23] BC Rennie and AJ Dobson. On stirling numbers of the second kind. *Journal of Combinatorial Theory*, 7(2):116–121, 1969.

[24] Jorge Portilla, et al. Adaptable security in wireless sensor networks by using reconfigurable ECC hardware coprocessors. *International Journal of Distributed Sensor Networks*, 2010.

[25] Yibin Li, et al. Dynamically reconfigurable hardware with a novel scheduling strategy in energy-harvesting sensor networks. *IEEE Sensors Journal*, 13(5):2032–2038, 2013.

[26] Heiko Hinkelmann, Peter Zipf, and Manfred Glesner. A domain-specific dynamically reconfigurable hardware platform for wireless sensor networks. In *Int'l Conf. on Field-Progr. Technology*, pp. 313–316, 2007.