

A Partitioning Algorithm for Power-constrained Reconfigurable Real-Time Systems

ABSTRACT

Existing approaches in hardware-software partitioning do not consider the maximum available system power while making decisions on partitioning. Such decisions are crucial for power-aware design when the target systems consist of a conventional CPU and an FPGA to support reconfigurable real-time applications. In this paper, we present simple heuristics to partition the real-time applications at task level for scheduling between CPU and FPGA so that they meet the deadline constraints for a given total system power. Experiments were built with 16-QAM and DTMF applications to demonstrate the efficiency of these algorithms. It has been observed that partitioning time of these heuristics is only less than a second compared to about five hours in the case of random extensive search techniques. This fast partitioning mechanism would enable dynamic reconfiguration of systems for power and performance tuning. The proposed techniques also determine lower bounds on deadline and system power requirement for given application(s). Furthermore, an upper bound on hardware area requirement is obtainable for single/multi-functional applications based on valid partitions. Per-case studies, when applications adopt resource sharing, 20-30 percentage of hardware savings is noticeable at different values of deadlines.

Keywords

Partitioning, scheduling, codesign, multifunction, power-aware

1. INTRODUCTION

Today's system architects adopt heterogeneous architectures as design paradigms for high performance and configurable devices as components for cost effective solutions. The HW-SW Codesign methodology helps designers to embrace complex designs by reusing IP cores to shorten the time to market gap. A generic system consists of a CPU (software processing entity), and an FPGA (reconfigurable hardware) as its data processing resources. In HW-SW Codesign, applications are usually specified at task level

granularity. The key issues involve in Codesign are partitioning, synthesis and cosimulation prior to the final implementation. The partitioning [15] is a critical design step that entails appropriate mapping (hardware or software) and scheduling of tasks among processing resources such that the timing constraints of real-time applications are met with minimum cost (hardware area and/or power). The partitioning problem is non-trivial since its solution space increases exponentially with number of tasks and their implementation techniques. An efficient partitioning strategy is always in demand to reduce the design time while satisfying the cost and performance.

Although attentions are focused on speed and accuracy of design while partitioning and scheduling, power consumption is becoming a dominant factor in complex system designs. Therefore, the system architects need to make their designs power-aware. An efficient power-aware design could be achieved by considering task level power consumptions during the partitioning stage of the Codesign.

New research opportunities in the system architecture are evident due to synergism of codesign methodology and advent of reconfigurable devices. Reconfigurable components in a system can adopt real-time requirements of an application by instantly modifying its processing architecture. A complex system can be spatially decomposed into simple functions (tasks) and appropriately scheduled in the FPGA for fast processing. Multiple functions can be integrated for processing on a single target system by sharing the processing resources (when possible) for cost effective solutions. In a reconfigurable system, tasks could be swapped between the processing components (CPU and FPGA) to meet different performance goals. Based on the power budget of a system, more tasks could be executed on the hardware (FPGA) to enhance the processing speed. Similarly, new tasks may be admitted in a multifunctional system if the power consumptions of incoming tasks meet the available power at that instant. For effectiveness of real-time reconfigurable system, fast partitioning techniques are highly desirable when the reconfiguration is achieved on the fly.

In this paper, a simple heuristic algorithm is proposed to partition the functional blocks of an application between CPU and FPGA for processing, so as to meet the application's deadline while satisfying the total system power requirements. The algorithm has been extended to consider multifunctional system that includes sharing of com-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

mon tasks amongst applications (if any) while mapping onto resources for cost effective solutions. A working system has been built and applied to 16-QAM and DTMF applications as case studies. The experiments show the lower bounds on application deadlines for given power values. The use of these algorithms can estimate minimum power requirements for each application including an upper bound on hardware requirements. In the case of resource sharing, a significant amount of hardware saving is demonstrated. The proposed algorithm yields feasible partitioning outcomes with time complexities of $O(N^2 \log N)$ compared to the corresponding extensive search algorithms of order $O(2^N N \log N)$.

The rest of the paper is organized as follows. In Section 2 we define the problem formally. Section 3 analyzes the related work done in this area. We present the power-aware partitioning algorithm for single functional systems in Section 4 and extend it to multifunctional systems in Section 5. Discussion on the experimental results is provided in Section 6 and Section 7 concludes the work.

2. PROBLEM DEFINITION

We assume a HW/SW co-design architecture model that consists of a single programmable processor (CPU), a reconfigurable FPGA device and an underlying communication architecture (shared bus) between them. The architecture is constrained by the maximum power rating P_{max} of the non-rechargeable battery source and the total available reconfigurable hardware area Ah_{total} (total number of logic gates/CLB's/slices).

Each individual application is described as a Directed Acyclic Graph (DAG) $G = (V, E)$. Node $v \in V$ represents a task and each edge $e \in E$ represents data dependence between the connecting task nodes. If there are any loops in the graph they can be broken into required number of DAGs. Each task v is either mapped to the CPU or to the reconfigurable hardware. Each task in the DAG is associated with five parameters (T_{sv}, P_{sv} : Execution time and instantaneous power dissipation when mapped to CPU, Th_v, Ph_v : Execution time and instantaneous power dissipation when mapped to the FPGA, Ah_v : Number of CLB's used when mapped to the FPGA). The execution time of a task is assumed to non-uniform when executed on both the CPU and the FPGA resource. The instantaneous power dissipation is assumed to be uniform during the time of execution of a task on a processing element. An edge $u \rightarrow v$ in the DAG implies that task u is a predecessor of task v . The weight of the edge between two tasks represents the communication time between the tasks. The weight is determined based on the underlying communication architecture between the CPU and the FPGA and the amount of data communicated between the tasks. The weight of an edge between tasks where both tasks are mapped to the same resources is assumed to be negligible.

The problem of power-aware partitioning of multifunctional systems could be formally defined as follows: *Given an application set denoted by $\{A_1, A_2, \dots, A_n\}$, where each individual application A_j is specified as a DAG with a periodic deadline D_j , total available hardware Ah_{total} and maximum power rating of the system P_{max} , generate a partition for each application such that they meet their timing and*

maximum power rating constraints with minimum hardware cost. More than one application could be active at run time.

3. RELATED WORK

While we are not aware of any partitioning methodology that concurrently partitions and schedules an application and takes into account the maximum power rating of the system, a number of algorithms have been proposed for partitioning of single and multifunctional systems such that they meet only their timing requirements. Some of the works include iterative improvement, heuristics and mathematical programming techniques [1, 2, 3, 4, 5, 6, 7, 8]. In [6], an integrated partitioning and scheduling algorithm for hardware-software partitioning is discussed but their work is limited to applications where tasks have uniform execution times when mapped to software and negligible execution times when mapped to hardware. The authors in [7] presents two partitioning algorithms for multifunctional systems by modifying the GCLP algorithm discussed in [5]. The authors in [14] present scheduling algorithms for embedded systems which operate on energy constraints. These works verify if the applications meet its timing requirement but do not consider the power rating while making the partitioning decision. In [9], a scheduling technique for power critical systems is presented where tasks are rescheduled using the available slacks such that the applications meet its timing and power requirements. Since their approach assumes an initial mapping and schedule, it leaves little flexibility for rescheduling the tasks. In our approach, concurrent partitioning and scheduling provides more flexibility in scheduling the Real Time tasks and ensures that the applications meet its deadline and power requirements.

4. PAP : POWER AWARE PARTITIONING ALGORITHM

In this section, we present the power-aware partitioning algorithm for single functional systems.

4.1 Algorithm Foundation

Power-aware partitioning algorithm is based on iterative improvement techniques. The algorithm initially maps all the tasks in the application to software. The start times of the tasks on the CPU are determined on the basis of a priority function. The priority function used ranks the tasks in the order of decreasing depth in the task graph. One task per iteration is selected to be moved to hardware based on the tasks's *mobility* indices and a *Task Selection Routine*. The start time of the execution of the selected task η is determined. The power profile of the schedule representing the instantaneous power consumption of all tasks is computed and verified to see if it meets the maximum power rating of the system. The partitioning process is repeated until the schedule of the application meets its deadline or until all available hardware logic is exhausted.

4.2 Task Mobility

The *mobility* of a task in the application provides information about the parallelism that could be achieved by moving the task from software to hardware and executing it in parallel with other tasks. The mobility of a task determines its earliest possible start and latest possible finish times.

The mobility indices of the tasks in the application are dependent on the schedule of the application and may change at each iteration of the partitioning algorithm. A task is defined as *mobile* if its latest possible finish time is greater than its earliest possible start time and *immobile* otherwise. The following procedure summarizes the computation of the mobility of a task.

Let t_i denote the execution time of task i .

$$t_i = ts_i, \text{ task } i \text{ is mapped to SW} \quad (1)$$

$$th_i, \text{ task } i \text{ is mapped to HW} \quad (2)$$

Let E_i denote the earliest possible start time and L_i denote the latest possible finish time of task i . E_i and L_i provide the lower and upper time bound for determining the start time of the task.

$$E_i = \max_{k \in \text{pred}(i)} \eta(k) \quad (3)$$

where $\text{pred}(i)$ denotes the immediate predecessor set of task i , $\eta(k)$ denotes the finish time of task k . Similarly

$$L_i = \min_{k \in \text{succ}(i)} \{\eta(k) - t_i\} \quad (4)$$

where $\text{succ}(i)$ denotes the immediate successor set of task i , $\eta(k)$ denotes the start time of task k . The *mobility* of a task i , $\mu(i)$ is defined as follows:

$$\mu(i) = \begin{cases} 1, & L_i > E_i \\ 0, & L_i = E_i \end{cases} \quad (5)$$

4.3 Task Selection Routine

At every step of the partitioning algorithm, one task is selected to be moved from software to hardware and executed in parallel with other tasks where it is feasible. The task to be moved is selected on the basis of a priority function. The software tasks are ranked in the decreasing order of their execution times and their *mobility* indices are used to select the task to be moved to hardware. The following procedure summarizes the *Task Selection Routine*.

Procedure: Task Selection Routine

Inputs: ts_i , for all $i \in N_s$, N_s : Set of software tasks

Output: Selected Task

S.1 Rank the tasks in N_s in the order of decreasing software execution times ts_i

S.2 Compute the *mobility* $\mu(i)$ for all $i \in N_s$

S.3 If $\mu(i) = 0$ for all $i \in N_s$

task i with maximum execution time ts_i is selected to be moved to hardware
terminate procedure

Else

task $i \in N_s$ with maximum execution time ts_i is considered

If $\mu(i) = 1$

task i is selected to be moved to hardware
terminate procedure

Else If $\mu(i) = 0$

remove task i from N_s
go to S.3

4.4 Power Characteristic of the Partitioning Schedule

The embedded system is specified to perform at a maximum power rating denoted by P_{max} . The *power profile* of the schedule is defined as the instantaneous power consumption of the tasks. The Power profile (P_σ) is computed as follows:

$$P_\sigma(t) = \sum_{\substack{i \in \text{set of tasks} \\ \text{active at time instant } t}} P(i) \quad (6)$$

The schedule is called *power-valid* if the power profile of a single iteration of the application is less than or equal to the maximum power rating of system (P_{max}). If the power profile of the schedule at any time instant exceeds the maximum power rating of the system, then the power profile at that time instant is having a *power spike*. The schedule with one or more power spikes is a infeasible schedule. The power spike could be removed by either rescheduling the execution within its slack or changing the map of one or more tasks active at that time instant. The total energy of the schedule (E_σ) is computed as follows:

$$E_\sigma = \int_0^{T_\sigma} P_\sigma(t) dt, \quad T_\sigma \text{ is the finish time} \quad (7)$$

of a single iteration of the application

4.5 Time-Valid Schedule

The finish time of a single iteration of the application (T_{exec}) is defined as the time when all the tasks in the application finish their execution. It is defined as:

$$T_{exec} = \max(\eta(i) + t_i), \text{ for all } i \in N \quad (8)$$

where $\eta(i)$ is the start time, t_i is the execution time and N is the total number of tasks in the application.

The schedule of the application is called *Time-valid* if T_{exec} is less than or equal to D , where D is the application deadline.

4.6 Communication Model for Partitioning

The mapping of tasks in the application to either hardware or software is influenced by underlying communication architecture (bus, protocol) and resulting communication delays. The performance gain obtained by moving tasks to hardware could be lost if the communication overhead is too large or due to non-availability of resources (bus). Thus, the scheduling of tasks and their communications on the bus are interdependent. In the proposed scheme the partitioning algorithm should also schedule the communication between CPU and FPGA. In order to schedule the communication, we need to estimate the delay between any two communicating tasks as is presented in the next section.

4.6.1 Calculation of the communication delay

We have used a 32 bit, 33 MHz PCI bus architecture as the communication interface between the CPU and the FPGA. The communication delay $t_{comm}(u, v)$ is the weight of the edge between tasks (u) and (v) in the application specified at a task level granularity. The communication delay t_{comm} is assumed to be the time required for the transmission of one sample of data on the channel (PCI Bus). The communication delay is computed according to the method in [10] as follows:

$$t_{comm} = \frac{CC * N_{sample}}{N_{bus} * F} + AC \quad (9)$$

where: CC is 2, the communication cycles required per data element. N_{sample} is the total amount of data being transmitted between the communicating tasks. N_{bus} is 32, the channel bit width. AC is 2, the arbitration cycles required. F is 33 MHz, the channel frequency. The communication delay due to partitioning of an application(s) can be estimated using a trace-based simulation[16], resulting in non-uniform delays. The traces provide the communication schedule and amount of data transferred when an application is simulated based on some partitioning results.

4.6.2 Power overhead due to communication

The power dissipation for the communication of data is computed based on the scheme described in [11]. The power dissipation P_{bus} is estimated as follows:

$$P_{bus} = \frac{1}{2} * C_{bus} * V^2 * m * n \quad (10)$$

where: C_{bus} is assumed to $10pF$, the capacitance of the bus. V is 3.3 V, the V_{cc} voltage. m is the number of words sent per second which is computed based on the sample data. n is 32, number of bits per word. We had defined the *power profile* at any time instant as the sum of the instantaneous power consumption of all tasks active at that time instant. The *power profile* of the schedule needs to be modified to include the power dissipation on the bus. The modified *power profile* (P_σ) is defined as:

$$P_\sigma(t) = \sum P(i) + P_{bus}(t), \text{ for all } i \in \text{set of tasks} \quad (11)$$

active at time instant t

4.6.3 Scheduling the communication

In the partitioning process, the *mobility* of the task defines the earliest possible start time(lower time bound) and the latest possible finish time (upper time bound) when the task could be scheduled. These time bounds define an interval where the task could be scheduled and executed in hardware. The communications for the task need to be scheduled on the underlying architecture (bus) during this time interval. The scheduling of communication between communicating tasks i and j where i is mapped to software and j is mapped to hardware is scheduled such that:

$$\eta(i) + t_i \leq \eta(comm) + t_{comm} < L_j \quad (12)$$

$$E_j < \eta(j) + t_j \leq L_j \quad (13)$$

where $\eta(i)$ and $\eta(j)$ are the start times of tasks i and j . t_i and t_j are the execution times of tasks i and j . E_j be earliest possible start time and L_j be latest possible finish time for task j . $\eta(comm)$ is the start time of the communication between i and j . $t_{comm}(i, j)$ denote the communication time between tasks i and j . To ensure a feasible schedule for the application, the communications for a task mapped to hardware are scheduled such that:

1. There are no resource (bus) conflicts.
2. The execution of the task and its communications lie within the interval specified by its *mobility*.

4.7 PAP Algorithm

Algorithm: **PAP**

Input: Application specified as a task graph (TG), Task metrics ($ts_i, th_i, Ps_i, Ph_i, Ah_i$) for all $i \in N$, N is the

total number of tasks in TG . Communication costs t_{comm} , Maximum Power Rating P_{max} . Total Available Hardware Area Ah_{total} , Deadline D

Output : All mapped tasks, *Time-Valid* and *Power-Valid* Schedule

Procedure: While{Schedule is not *Time-Valid* }

S.1 Initially all tasks $\in N$ are mapped to software. Schedule is assumed to be *Power-Valid*

S.2 Compute t_{exec} , the finish time of a single iteration for the application

S.3 If ($t_{exec} < D$)

Schedule is *Time-valid*. Terminate Algorithm

S.4 Select task i to be mapped to hardware, $i \in N$ using the *Task Selection Routine*

S.5 Determine the start time $\eta(i)$ for task i , $i \in N$ using its *mobility* $\mu(i), E_i$ and L_i

Update the Bus Activity using the communication costs

Reschedule the task graph TG and compute t'_{exec} , the new finish time of one iteration for the application

Update the total hardware used Ah

S.6 If ($t'_{exec} < t_{exec}$) and If (schedule is *Power-Valid*) and If ($Ah < Ah_{total}$)

$t_{exec} = t'_{exec}$ (Update t_{exec})

Goto Step S.2

Else If ($Ah > Ah_{total}$)

Invalidate the selection of the current task for hardware

mapping for all future iterations

Goto Step S.4

Else If (Schedule is not *Power-Valid*)

Invalidate the selection of the current task for hardware

mapping for the next iteration

Goto Step S.4

Else If ($t'_{exec} > t_{exec}$)

Invalidate the selection of the current task for hardware

mapping for the next iteration

Goto Step S.4

4.8 Example

Figure 1 shows an example of the PAP algorithm. The initial all software implementation is shown in Figure 1(a). Task 1 (shaded gray) is selected to be moved to FPGA in first iteration. The generated schedule is power-valid but does not meet the deadline D (Figure 1(b)). In the second iteration, Task 2 (shaded gray) is selected to be moved to FPGA. A possible schedule which meets the deadline but fails to meet the maximum power rating P_{max} is shown in Figure 1(c). The execution and communication (shaded black) for Task 2 are rescheduled within the time interval defined by its mobility such that the new schedule is both time and power valid (Figure 1(d)).

5. PARTITIONING OF MULTIFUNCTIONAL SYSTEMS

In multifunctional partitioning, the set of applications active at run time are analyzed to determine the similar tasks across the set of applications. The applications which are specified at the task level of granularity are combined into

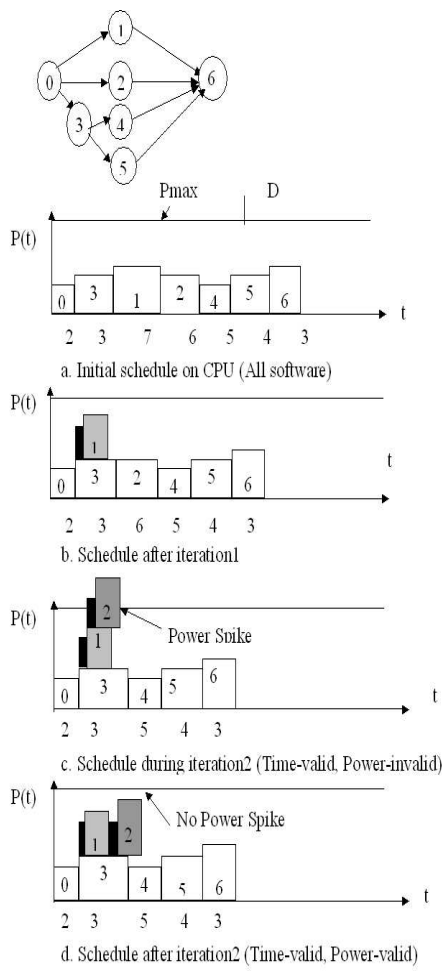


Figure 1: A example of the PAP algorithm

a single task graph. The power-aware partitioning (PAP) algorithm is then applied to the combined task graph. The partitioning algorithm uses the information about similar tasks and hardware re-usability to map and schedule the tasks of each individual application.

All the tasks belonging to the set of active applications are initially mapped to software. The tasks are referenced by their application name to distinguish similar tasks. Our approach assumes the similar tasks to be identical but could be extended to other forms of similarities between tasks across the set of applications. During each iteration, one task of each application is considered for hardware mapping. The order in which the applications are considered is based on their criticality's and is significant because the mapping information of tasks is propagated between applications in the current and successive iterations.

5.1 Criticality of Applications

At each step of the partitioning algorithm for multifunctional systems, the set of active applications $\{A_1, A_2, \dots, A_n\}$ are ordered based on their criticality's. The set of active applications are interfaced into a combined task graph (CTG). The criticality of an application (AC) is a measure of its

computational complexity and is described as:

$$AC_i = \frac{T_{CTG}}{D_i} \quad (14)$$

where: AC_i is the application criticality of A_i , $A_i \in \{A_1, A_2, \dots, A_n\}$. D_i is the deadline of A_i . T_{CTG} is the finish time of one iteration of CTG . When a task of the CTG is moved from software to hardware, the finish time of one iteration (T_{CTG}) is modified. Therefore the application criticality's need to be computed at the beginning of each iteration of the partitioning algorithm. The application with maximum criticality is considered first and a task from this application is selected to be moved from software to hardware using the *Modified Task Selection Routine*.

5.2 Modified Task Selection Routine

All tasks are initially mapped to software. During each step of the partitioning algorithm, one task of each application belonging to the set of active applications is moved from software to hardware and executed in parallel with other tasks. The priority function used to move a task from software to hardware for single functional systems has been described in Section 4.3. In the case of multifunctional systems, the information about similar tasks across the set of active applications and hardware re-usability is used to select the task to be moved from software to hardware. Thus, the *combined priority function* to move a task of one application from software to hardware uses information about its execution time, *mobility* within the application (*self priority*) and information about similar tasks and hardware availability for re-use across the set of applications (*shared priority*). The *combined priority (CP)* of a task is defined as:

$$CP(i) = SeP(i) + ShP(i) \quad (15)$$

where $SeP(i)$ and $ShP(i)$ are the self and shared priorities of task i .

Next, we present the computation of the *Self Priority* of all the software tasks of an application which is part of the combined task graph.

Procedure: Self Priority of software tasks in an Application

Inputs: $A_k, A_k \in \{A_1, A_2, \dots, A_n\}$

Output: Self Priority of all software tasks

S.1 Initialize counter $Count = 0$

S.2 Compute the *mobility* $\mu(i)$ for all $i \in N_s$. N_s is set of software tasks in A_k

S.3 Determine $N_{s1} \in N_s$, set of all software tasks with non-zero mobility and $N_{s2} \in N_s$, set of all software tasks with zero mobility.

S.5 Sort all tasks $\in N_{s1}$ in non-increasing order based on their execution times.

S.6 Sort all tasks $\in N_{s2}$ in non-increasing order based on their execution times.

S.7 Extract task i , $i \in N_{s1}$ with maximum execution time ts_i

S.7.1 Compute Self Priority as follows

$$SeP(i) = \frac{N_s - Count}{N_s}$$

S.7.2 Increment $Count$

S.7.3 Remove task i from N_{s1}

S.7.4 Goto Step S.7
 S.8 Extract task i , $i \in N_{s2}$ with maximum execution time ts_i

S.8.1 Computer Self Priority as follows:

$$SeP(i) = \frac{N_s - Count}{N_s}$$

S.8.2 Increment $Count$

S.8.3 Remove task i from N_{s2}

S.8.4 Goto Step S.8

The *self priority* $SeP(i)$ of software task i is computed based on its execution time and its mobility with respect to other tasks in the same application. By moving a software task with highest *self priority* to hardware, a greater performance gain could be achieved but it results in an increase in the total hardware area. Moving a task which is repeated across the set of applications such that the hardware implementation could be reused and results in a reduction in the total hardware cost.

The *shared priority* of a software task i is computed as the total number of times a similar task across the set of applications has been mapped to hardware Num_i . The measure is normalized across all the software tasks of the application N_s . The *shared priority* ($ShP(i)$) of a software task i is described as:

$$ShP(i) = \frac{Num_i}{\max Num_j} \quad \text{for all } j \in N_s \quad (16)$$

As described in Equation 15, each software task of an application is assigned an *combined priority* based on its *self priority* and *shared priority*. The combined priority is used to bias similar tasks for hardware mapping which could result in reduction in the total hardware used.

5.3 Partitioning of Multiple Applications

Algorithm: MPAP

Input: Application A_i , Deadline D_i for all $A_i \in \{A_1, A_2, \dots, A_n\}$

Total Available Hardware Area Ah_{total}

Maximum Power Rating of the system P_{max}

Output: Time and Power valid schedules for the set of applications.

Procedure: While the schedules for the set of applications are not time and power valid

S.1 Set of applications are combined to form a single task graph CTG . All tasks are initially mapped to software. Schedule is assumed to be *Power-Valid*.

S.2 The Application Criticality AC_i for application A_i is computed, for all $A_i \in \{A_1, A_2, \dots, A_n\}$.

S.3 The applications are ranked by the criticality's such that the one with maximum application criticality is considered first.

S.4 A task of the application is selected for hardware mapping using the modified task selection routine and using steps S.5 and S.6 of the PAP algorithm. Repeat for other applications in the ordered set $\{A_1, A_2, \dots, A_n\}$.

S.5 For all $A_i \in \{A_1, A_2, \dots, A_n\}$

If (Schedule is both time and power valid)

Remove A_i from $\{A_1, A_2, \dots, A_n\}$

End algorithm if schedules of all applications are time and power valid.

Else

Repeat from step S.2

5.4 Time complexity of the MPAP algorithm

In each iteration, for an combined task graph (CTG) containing N tasks, the tasks's *mobility* indices are computed in $O(N)$ time and the tasks are sorted in $O(N \log N)$ time. The self and combined priorities of the tasks can be computed in $O(N)$ time. Therefore the modified task selection routine takes $O(N \log N)$ time. The rescheduling during each iteration takes $O(N)$ time. Since there can be at most N iterations and the initial all software implementation on a single CPU takes $O(N^2)$, the time complexity of the MPAP algorithm is $O(N^2 \log N)$.

6. CASE STUDIES

We consider practical signal processing applications with periodic timing constraints. Two examples used are: 8 Khz 16-QAM Modem and DTMF Codec. These applications are specified in CGC domain of the Ptolemy system[12]. The tasks/blocks in the applications are generated in C language. The software time (ts_i) and software power estimates (Ps_i) for each task i are obtained on the StrongARM SA-1100 processor (operating at 206 MHz) using *JouleTrack* [13]. For the hardware implementations, VHDL RTL was developed for all the functional blocks. The target reconfigurable hardware resource in our study is the *Virtex-II* FPGA (*XCV4000*) from Xilinx. The hardware execution time (Th_i) is computed as the worst case propagation delay after Place and Route (PAR) on the FPGA using the Xilinx ISE 4.2 simulator. The hardware power dissipation (Ph_i) for each task i in the application is obtained using the Xpower tool which is part of the Xilinx ISE 4.2 simulator. The underlying bus architecture between the software processor and the FPGA is considered to be the 33 MHz PCI bus.

Next we present two experiments. In the first experiment described, the PAP algorithm is independently applied to both the applications. The partitioning and scheduling results are compared with results obtained from the extensive search. In the second experiment, the MPAP algorithm is applied to the modem and codec applications running concurrently and sharing hardware resources as a multifunctional system. The results are compared with joint implementations without sharing the resources.

6.1 Experiment and Results

We considered that the incoming data in both the applications are periodic with deadlines varying from $800\mu s$ to $100\mu s$. The PAP algorithm was applied to partition both the cases with varying power constraints ranging from 8W to 2W to see their impacts on finish time for various prescribed deadlines. For a given deadline, as the power constraints are tightened(reduced), the finish time satisfies the deadline until a point beyond which the partitions are time invalid. On reducing the deadline, at the same power value, the heuristic uses more hardware resources for partitioning to meet the deadlines. However, there is an upper bound on the use of hardware slices at that particular power value, beyond which

Table 1: Results from the PAP algorithm and Extensive Search

App. Example	Algorithms	Power (W)	Finish Time (μs) (LB)	Deadline (μs) (LB)	Part. Time (s)
16QAM(1)	PAP	5	54	100	0.7
		4.0	54	100	0.7
		3.5	116	150	0.7
		3.0	116	150	0.7
		2.5	226	250	0.7
	Ext. Search	2.5	172	250	15310
DTMF(2)	PAP	4.0	53	100	0.8
		3.5	53	100	0.8
		3.0	320	350	0.8
		2.5	480	500	0.8
	Ext. Search	2.5	456	500	22160
COMBO (1 + 2)	MPAP	4.0	107	150	1.1
		3.5	107	150	1.1
		3.45	107	150	1.1

the partitions are again time invalid. At this point, the corresponding deadline is the lowest bound achieved at that power constraint. For example, in the case of 16-QAM, at power constraint of 2.5W the minimum achievable deadline is 250 μs . The Table-1 depicts all such lower-bounds(LBs) on deadlines that are time valid and corresponding power constraints for these examples. The minimum power requirement corresponding to DTMF-PAP and MPAP are 2.5W and 3.5W respectively. On further squeezing of power or deadline will make the partitioning results invalid. However, on increasing the available power, more tasks are pushed to the FPGA for parallel implementation so that the finish time(also the deadline) is reduced. From Figure 2, it may be noticed that with available power of 4W and above the finish time does not improve indicating the point of saturation.

In order to compare the *partitioning time*, of PAP algorithm with extensive search, we considered the use of equal number of FPGA slices in addition to identical values of power and deadlines for both the cases. It has been found that the extensive search algorithm takes about four to six hours compared to less than a second in the case of PAP algorithm when simulated using an UltraSPARC5. The partitioning time of MPAP algorithm for multifunctional applications(COMBO) is found to be 1.1 seconds. However, the extensive search time of MPAP grows to the $O(N^2 N \log N)$, where N is equal to 50 and could not be computed in reasonable time.

In order to examine the efficiency of MPAP algorithm with respect to PAP, we compared the use of FPGAs in both the cases while implementing the QAM and DTMF applications. In the Figure-3, the hardware size used in terms of number of slices are shown for DTMF and QAM without sharing any resources. The third curve shows the use of hardware by MPAP algorithm when QAM and DTMF

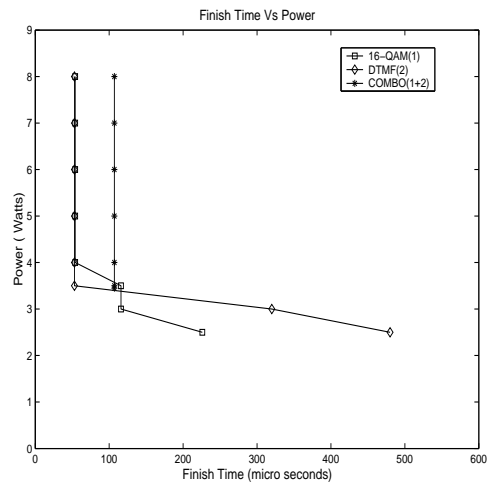


Figure 2: Lower Bound on Finish Time vs Power

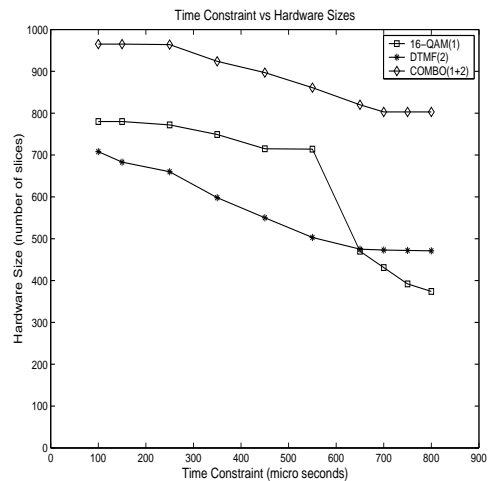


Figure 3: Variation in FPGA size for different deadlines

are combined to share the resources. On comparison, it is found that the resource sharing helps to reduce the hardware consumption on an average by 23 percentage. It may be further observed that an FPGA with 1000 slices will suffice such applications. This provides an upper bound on the hardware requirements.

7. CONCLUSION

In this paper, fast and efficient *power-constrained* partitioning heuristics are proposed for reconfigurable real-time systems. Using power constraint as input, the algorithm concurrently partitions and schedules tasks in an application to meet the deadline. The algorithm has been applied to both single and multifunction systems using two case studies. The multifunctional partitioning algorithm shares resources between functions and proves to be an area efficient solution. Since the proposed PAP/MPAP algorithm's run time is low, several utilities can be addressed when applied to dynamic real-time system design. For example, various

tasks/functions in an application can be swapped between processor and reconfigurable fabrics to meet different power-performance goals as needed. As our future research, we plan to extend the scheme to distributed embedded systems and adopt voltage scaling selectively.

8. REFERENCES

- [1] F. Vahid, J. Gong, and D. Gajski. A binary constrained search algorithm for minimizing hardware during hardware-software partitioning. In *European Design Automation Conference*, pages 214–219, 1994.
- [2] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. Hardware/software partitioning with iterative improvement heuristics. In *9th International Symposium on System Synthesis*, pages 71–76, 1996.
- [3] R. Niemann and P. Marwedel. An algorithm for hardware /software partitioning using mixed integer linear programming. In *ED&TC*, 1996.
- [4] R. K. Gupta and G. De Michelli. Hardware-software cosynthesis for digital systems. *IEEE Trans. Design and Test of Computers*, 10(3):29–41, Sept. 1993.
- [5] A. Kalavade and E. A. Lee. The extended partitioning problem: Hardware/software mapping and implementation-bin selection. In *Sixth International Workshop on Rapid Systems Prototyping*, pages 12–18, 1995.
- [6] H. Liu and D. F. Wong. Integrated partitioning and scheduling for hardware/software co-design. In *International Conference on Computer Design*, pages 609–614, 1998.
- [7] A. Kalavade and P. A. Subrahmanyam. Hardware/software partitioning for multifunction systems. *IEEE Trans. Comput.-Aided Des. Integrated Circuits & Syst.*, 17(9):819–837, Sept. 1998.
- [8] M. Potkonjak and W. Wolf. Cost optimization in ASIC implementation of periodic hard-real time systems using behavioral synthesis techniques. In *International Conference on Computer Aided Design*, pages 446–451, 1995.
- [9] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *Design Automation Conference*, pages 840–845, 2001.
- [10] P. V. Knudsen and J. Madsen. Integrating communication protocol selection with partitioning in hardware/software codesign. In *11th International Symposium on System Synthesis*, pages 111–116, 1998.
- [11] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *International Journal of Computer Simulation*, 4(2):155–182, April 1994.
- [12] *The Almagest: Manual for Ptolemy Version 0.7*. Department of EECS, University of California, Berkeley, California, 1988.
- [13] A. Sinha and A. P. Chandrakasan. Jouletrack—a web based tool for software energy profiling. In *Design Automation Conference*, pages 220–225, 2001.
- [14] C. Rusu, R. Melhem, and D. Mosse. Maximizing the System Value while Satisfying Time and Energy Constraints. In *IEEE Real-Time Systems Symposium, Austin*, December 2002.

[15] J. Axelsson. Hardware/Software Partitioning of Real-Time Systems. In *IEEE Colloquium on Partitioning in Hardware-Software codesign s Digest number 1995/032*, Digest number 1995/032.

[16] Omitted for blind review.