K. Guha et al., *Self Aware Security for Real Time Task Schedules in Reconfigurable Hardware Platforms*
https://doi.org/10.1007/978-3-030-79701-0_3

3. Scheduling Algorithms for Reconfigurable Systems

Krishnendu Guha[1] ✉, Sangeet Saha[2] and Amlan Chakrabarti[1]
(1) A. K. Choudhury School of Information Technology (AKCSIT), University of Calcutta, Kolkata, West Bengal, India
(2) School of Computer Science and Electronic Engineering (CSEE), University of Essex, Colchester, UK

✉ **Krishnendu Guha**
Email: kgchem_rs@caluniv.ac.in

3.1 Introduction

Many of today’s modern embedded systems is employing FPGAs within their architectures. The examples of such systems are: PDAs (personal digital assistants), mobile phones, automotive systems etc. [HKM+14]. FPGAs have also been used in computer vision, object tracking [JLJ+13], cryptography [BGHD13], and audio/video [TKG10] applications. Many of such applications are real-time in nature. For a complex safety critical system, FPGAs can act as an efficient backup platform for real-time task execution. In a typical situation, when the primary processor(s) fails, the FPGA can be used for processing the real-time tasks that were executing on the faulty primary processor(s).

However, execution of a set of hard real time tasks on FPGAs impose many challenges. First of all, it requires a well defined resource allocation and admission control mechanism. Moreover, such mechanism must guarantee timing constraints as well as high resource utilization by effectively placing hardware tasks on the 2D reconfigurable floor area.

If we recall, FPGAs consists of a 2-dimensional array of $W \times H$ as Configurable Logic Blocks (CLBs). A hardware task T_i is represented by a digital circuit which is rectangular in shape. Each task T_i consumes a sub-region $w_i \times h_i$ on the floor (having total area $W \times H$) of the FPGA. Given a set of tasks $T = \{T_1, T_2, \ldots, T_n\}$ to be executed, the placement problem can be defined as to find a subregion of size $w_i \times h_i$ for each task T_i such that no subregion overlaps with the FPGA boundaries or with other subregions (placed tasks). After placing a set of tasks, any vacant region which area is not enough to accommodate any new task, could be considered to be wasted due to fragmentation.

To minimize fragmentation, In [BKS00], Bazargan et al. proposed KAMER (Keeping All MERs), a mechanism which proceeds by maintaining a list of Maximal Empty Rectangles (MERs). Upon arrival of a task, the algorithm places it in the bottom-left corner of the largest available MER in a worst-fit manner. After placement, the remaining vacant area of the region is partitioned using either a vertical or horizontal split to produce two empty rectangular sub-regions. Although this method produces good placement quality, a drawback is that a wrong splitting decision could cause the rejection of an otherwise feasible task. Walder et al. [WSP03] proposed an enhanced version of the Bazargan partitioner which postpones the vertical/horizontal splitting decision until the arrival of a new task in order to overcome the possibility of a wrong decision. The authors in [EFZK08] used First Fit and Best Fit placement strategies and claimed to achieve lower fragmentation and task rejection rates compared to KAMER [BKS00] and Enhanced Bazargon [WSP03] methods. However, this work maintains the occupancy status of each CLB of an FPGA in a 1D array and therefore, is susceptible to high computational overheads. A few other placement approaches aimed towards the efficient management of empty regions include algorithms by Handa et al. [HV04] and Tabero et al. [TSMM04] (which employs a

Vertex List Set (VLS), where a given free space fragment is represented by a list of vertices). Ahmadinia et al. [ABBT04] proposed management of occupied area instead of free area as they observed that records for occupied spaces grow at a much slower rate than those for free spaces, making data management for the accommodation of new tasks simpler.

The above approaches are primarily oriented towards spatial management of the reconfigurable resource. Diessel and Elgindy [DE01], combined placement along with a temporal scheduling mechanism for non real-time preemptible task sets. Tasks are allocated from a ready queue starting from the bottom-left position using a first-fit strategy. When the allocator fails to accommodate the next pending task, it attempts to create additional space by preempting and locally repacking the currently running tasks utilizing spare logic resources created due to the partial execution of these tasks.

Spatio-temporal scheduling of tasks with pre-assigned priorities have also been considered for reconfigurable systems, where priority could be based on the temporal (deadline, laxity) or geometrical (size, aspect-ratio) properties of the tasks (or both) [WL12]. Walder and Platzner [WP03] used non-preemptive scheduling schemes like *First Come First Serve (FCFS) and Shortest Job first (SJF)* for block-partitioned 1D reconfigurable devices. In [CH05], authors present a scheduling mechanism called *Classified Stuffing (CS)*, where both geometrical and temporal parameters have been used to obtain priorities. The tasks are first inserted into a ready queue based on their temporal priorities (For non real-time tasks, these priorities are obtained using the *Shortest Remaining Processing Time (SRPT)* strategy, while for real-time tasks, the *Least Laxity First (LLF)* strategy is followed). The tasks in the ordered ready queue are then placed on the basis of a spatio-temporal parameter called *Space Utilization Rate (SUR)*, which is defined as the ratio between the area requirement and the execution requirement of a task. Using a 1D area model, leftmost available columns of the FPGA were filled with tasks with high SUR ($SUR > 1$) , while the rightmost available columns were filled with low SUR tasks ($SUR > 1$).

Along with placement, in order to achieve high resource utilisation, the scheduling should be preemptive in nature. However, there has been very little work in this regard for FPGAs. The main reason for this is the challenges involved in storing the status of a partially completed task and restoring saved states for re-execution. Along with CLBs, the FPGA floor additionally contains Hard embedded Blocks (HBs). These HBs may include pre-fabricated blocks of BRAMs, MULs, DSPs etc. [FPMM11].

When a context switching occurs, the contexts of the tasks are stored in Flip-flops and LUT-RAMs of CLBs and other HBs. This context storing is then followed by context restore event which attempts to restore task's context as it was before the preemption, so that the execution can be resumed. Authors in [JTW07, LWH02] employed *Scan Path* generation strategy to allow task context extraction/insertion on FPGAs. However, for this preemption technique, the methodology requires task specific components known as "scan-chains" to be added with each hardware task, thus it consumes huge spatial overheads. An improvement over this technique is *bit-stream read back* methodology [JTHT10]. In [JTHT10], the authors collect FPGA configuration information by reading the bitstream through ICAP (Inter Configuration Access Port) in order to realize the context switching. Hardware task preemption on heterogeneous FPGAs is also feasible and recently demonstrated in [HTK15]. The work is similar to [JTHT10, MVGR13] and additionally allow context capture and restore of BRAMs.

3.2 Challenge for Devising Real-Time Scheduling Algorithm for FPGAs

To achieve the higher resource utilization and minimize task rejections, a different dynamic scheduling and placement approach has been discussed in [Mar14]. In this article, the author assumes that a hardware task could have multiple hardware variants (varying size) such that, larger the size of a task, faster is its performance. Selection of appropriate hardware task variants is a trade-off between maximal utilization of reconfigurable resources versus the timing requirements of the tasks. A similar algorithm which considers multiple hardware task shapes has been proposed in [WBL+14]. The authors showed that resource utilizations using conventional scheduling algorithms like EDF and LLF may be significantly improved by using the flexibility of multiple shapes, against a rigid task scenario. This work neglected reconfiguration overheads and assumed tasks to be soft real-time in nature. However, reconfiguration overhead is a major constraint which may adversely affect the temporal performance of tasks if not handled appropriately. In [LMBG09], authors described the *reuse and partial reuse* approach which allows a single configured task on the reconfigurable floor to be shared and reused among multiple applications, thus reducing the number of reconfigurations required. An important observation here is that most of the above works are primarily focused towards partially reconfigurable platforms with very few articles attempting to handle the problem for fully reconfigurable FPGAs.

As the first attempt, Danne and Platzner [DP05] has considered the scheduling preemptive real-time tasks on FPGAs. They proposed EDF-Next Fit (EDF-NF), a variant of the EDF [Liu00]. Similar approaches for FPGAs may be found in [6], [SRBS10, BGSH12]. Recently, there has been a few contemporary multiprocessor scheduling strategies like ERfair [AS00], DP-Fair [LFS+10] etc. which provides higher resource utilisation for multiprocessor systems. The concept of such “fair” scheduling strategies is that given a set of *n* tasks $\{D_i, D_i, ..., D_i\}$, where T_i has an execution requirement of e_i time units, which has to be completed within a period/deadline of p_i time units. ERfair algorithm calculates *weight* of T_i as $wt_i = \frac{e_i}{p_i}$ and maintains a rate of execution for each task T_i within a scheduling time slot. Thus, ERfair suffers from huge migration and context-switching overheads. On the other hand, DP-Fair [LFS+10] enforces a maintains such rate at task period / deadline boundaries i.e. all tasks have to complete a some share of (proportional to its weight) at the end of the deadline. DP-Fair divides time into some small windows or slots, based on the task deadlines. Within that window, each task is allocated a workload. However, this constraint can only be achieved if $\sum_{i=1}^{n} e_i/p_i \leq m$, where *m* is the number of processor. But, these fair scheduling strategies cannot be directly employed for FPGAs due to their architectural constraints and reconfiguration overheads.

The main architectural constraints is the placement of tasks. In this chapter, we have devised real-time scheduling algorithms for FPGAs based on 2D partitioned area model where both the width W and height H of the FPGA is partitioned so that a fixed number equal sized rectangular tiles can be obtained. As there exists no partition inside the FPGA, these tiles can be called as Virtual Partitions or VPs. Now onwards, we will call each tile of the FPGA as VP. Each VP must be large enough to accommodate any task. The main reason for choosing this area model is that it may be noted that FPGAs are primarily used in embedded systems where all tasks that may possibly arrive for execution is known offine, although their actual arrival instance at runtime may not be known. Hence, the use of equi-sized tiles (2D slotted area model) with sufficient resources to execute any arbitrary task, is a legitimate and realistic assumption for embedded system.

As we have seen in our previous chapter that relaxations on the 2D slotted area model leads to flexible 2D area model which enables tasks to be placed at any arbitrary region of the FPGA. But this strategy becomes a complex placement problem which costs higher computational complexity. We can further argue that in general, relaxations on the 2D slotted area model, lead to two important drawbacks:

- Even though they improve average performance for a set of tasks, they often degrade scheduling predictability, especially in real-time systems. This is because, it becomes more complex in this case to deterministically account for the spatio-temporal capacity available in the worst-case, within a given time interval in future.
- These models tend to incur much higher overheads at each spatial scheduling point, making the complexity of the overall spatio-temporal scheduling problem very expensive towards online application.

Majority of the existing scheduling algorithms for FPGAs mainly focus on the problem of efficient task placement on the FPGA floor such that fragmentation is minimized. Comparatively a very few research has been conducted towards scheduling so that the deadlines of real-time tasks are effectively managed. Moreover, research towards real-time scheduling have mostly been confined to the handling of soft real-time tasks and aimed for partially reconfigurable FPGA. In comparison, there is very little researches that address the hard real-time scheduling and effective resource management in fully reconfigurable systems.

In this chapter, we will devise our scheduling strategies for fully and partially reconfigurable FPGAs by assuming a 2-dimensional FPGA and the FPGA floor is equi-partitioned into a set of VPs. It has been assumed that each VP ha sufficient resources to accommodate a task. We consider fully reconfigurable FPGAs with reconfiguration overhead denoted as O_{frg}. However, it has to be noted that for fully

reconfigurable FPGA, all VPs have to be reconfigured simultaneously. In partially reconfigurable FPGA, the reconfiguration overhead is denoted as O_{frg}. For partially reconfigurable FPGA, we can reconfigure a VP

while the other VPs can operate seamlessly. Now, it may be noted that the configuration time of an FPGA depends upon two factors, one is the initialization time which requires only one time, on power up of the system and another is bit-stream loading time, to configure the FPGA with specific hardware circuit. Here, we are not considering the initialization time, the O_{frg} & O_{frg} are the configuration time or more

specifically bit-stream loading time. Now configuration time depends on the bit-stream size, the clock frequency and the bandwidth of configuration port (Described later Section).

As the modern FPGAs support high data width, parallel configuration port and high (50 MHz) configuration clock frequency, O_{frg} differs due to variable bit-stream size as the information needed

to fully configure the floor of the small FPGA varies from larger one [Sta11]. In case of O_{frg}, the partial reconfiguration time is directly

proportional to the size of the reconfigurable region or more specifically to the bit-stream size needed to partially reconfigure the region [HSH09]. A study has been made to have a view on the different full configuration times for Xilinx FPGAs and found that it varies from around 3 ms to 30 ms depending upon the available floor size. The real-time scheduling strategies for FPGAs must incorporate these reconfiguration overhead (O_{frg} and O_{frg}) for fully and partially reconfigurable systems respectively). It has to be ensured that by considering these overheads, the timing constraints of the system do not get violated.

This chapter is organized as follows. The next section provides a brief discussion on the system model and assumptions. Detailed descriptions of our proposed scheduling strategies (for fully reconfigurable system and partially reconfigurable systems), have been presented along with illustrative examples in Sect. 3.4. Section 3.5 describes how the dynamic tasks can be handled by both the strategies. Section 3.6 represents experimental results along with analysis and discussion on the same. The chapter finally concludes in Sect. 3.7.

3.3 System Model and Assumptions

The overall system model contains FPGA, a General Purpose Processor (GPP) and memory, as shown in Fig. 3.1. The bitstream of a task (bit file or configuration file which is the executable unit of a hardware task) is stored in the memory unit. GPP is responsible for executing hardware tasks on the different tiles/VPs (Virtual Partitions) of FPGAs by loading bitstream file from the memory unit to the configuration memory of the FPGA via ICAP (Internal Configuration Access Port). The architecture of the FPGA is similar to that of the Xilinx Virtex series of FPGAs. The FPGA consists of CLBs and other Hard Blocks (HBs) like Block RAMs (BRAM), Multipliers (MULs), I/O Blocks (IOBs), Clocks (CLKs) etc.

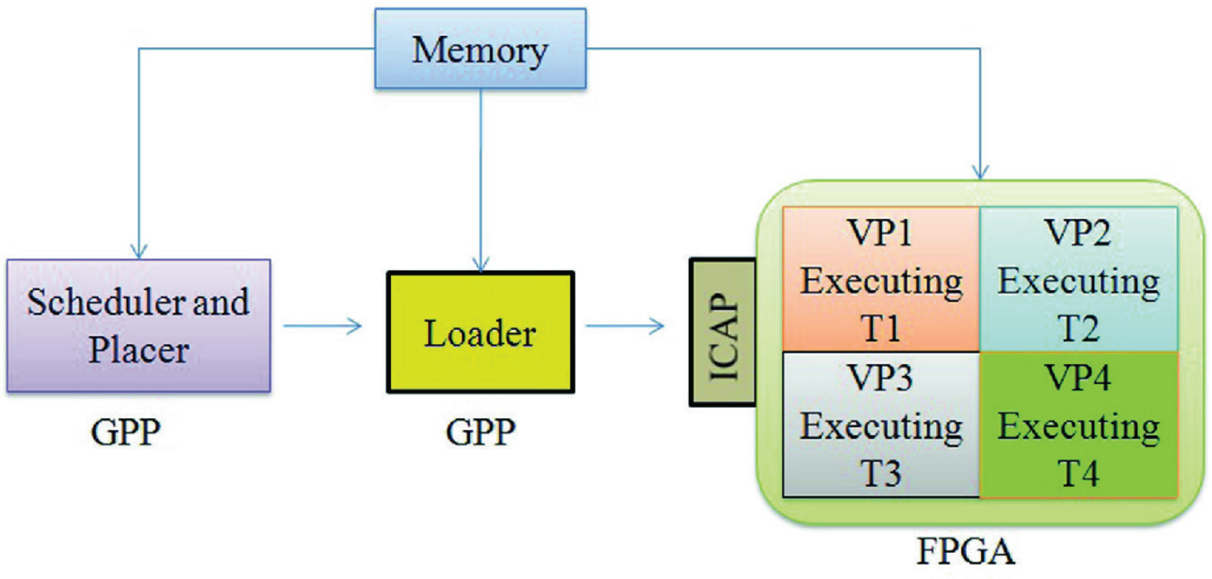

Fig. 3.1 System model

The floor of the FPGA is equi-partitioned into *M* VPs. Each VP can execute any hardware task irrespective of their resource requirements. The considered task set are preemptive and hence, it will consume context switch overheads. This overheads are determined by reconfiguration overheads of the FPGA. For partially reconfigurable FPGA, individual VPs can be reconfigured in an asynchronous fashion but for a fully reconfigurable FPGA, context switching becomes a global event because all the VPs have to be reconfigured synchronously.

In order to reconfigure a VP in FPGA, (of area say, α), we need to halt/stop the currently running task within that VP and then the task's bitstream is extracted/captured from the configuration memory through ICAP. In [JTE+12], the authors showed that actual context of a task is present within less than 8% of its total bitsream and while conducting the context switch, only this part of the bitstream requires to be extracted. Thanks to this selective read-back mechanism, with the aid of this technique, actual extraction overhead is reduced to 1/10th of the full bitstream read-back time. Once the extraction is complete, the captured task bitstream is manipulated [JTHT10] and the new bitstream is created. For the context restoration, the updated bitstream is loaded.

The time required for (say, *TR*) extraction/restoration of the region α is depends on the the size of the bitstream (*BS*) and the configuration clock frequency (*CCLK*) of the controller and its data bus width (*DBW*). TR can be calculated as [Tap13]:

$$TR = BS/(CCLK \times DBW) \tag{3.1}$$

To quantify TR for Xilinx Virtex-4 FPGA which has an area of 52×128 [Xil10], full configuration bitstream of size 2.21 MB [Xil09], $CCLK = 100$ MHz and $DBW = 32$ bit [Xil10]. From these values, the total context restoration time becomes ≈ 6 ms, while the extraction time is ($\frac{1}{10} \times 6$ ms) = 0.6 ms, which we assumed to be 1 ms. The bit manipulation time and the overhead for new bitstream formation is ≈ 6 ms. Hence, the total context switching overhead (O_{frg}) becomes 1 ms+ 5 ms+ 6 ms=12 ms (extraction + manipulation + restoration). (O_{frg}) for partially reconfigurable FPGA having *M* VPs can be calculated as: $O_{prg} = O_{frg}/M$. Thus, if Virtex-4 is partitioned into four equi-sized VPs, then $O_{prg} = \frac{12}{4} = 3$ ms.

3.4 Scheduling Strategies

Given a set of *n* tasks $\{D_i,\ D_i$, ..., $D_i\}$ to be scheduled on *M* VPs $\{VP_1$, VP_1, ..., $VP_m\}$ of an FPGA, the proposed scheduling algorithms maintain the time partitioned into slices / windows (the *r*th time-window is denoted by tw_r, difference between $(r-1)$th and *r*th task deadlines) demarcated by the task deadlines. At any time-window boundary, each task T_i has to complete a workload-quota WQ_i^r for the time-window tw_r (having length twl_r) given by: $WQ_i^r = \lceil \frac{e_i}{p_i} \times twl_r \rceil$.

The sum of the workload-quota of all tasks can be calculated as: $sum\_WQ^r = \sum_{i=1}^{n} WQ_i^r$, and the total system capacity becomes ($twl_r \times M$) over the interval twl_r, Now in order to achieve a feasible schedule both for fully and partially reconfigurable systems, following conditions needs to be satisfied:

$$sum\_WQ^r \leq twl_r \times M \tag{3.2}$$

3.4.1 Scheduling Algorithm for Full Reconfigurable Systems

Context switching in fully reconfigurable FPGAs can only be realised through synchronous configuration of VPs and each such context switch will consume a full reconfiguration overhead. The total overhead that for all VPs is given by: $O_{frg} \times M$, where O_{frg} denotes the full reconfiguration time. Now, this overhead can be compensated from the available slack (if any) at time-window duration twl_r and is given by: $twl_r \times M - sum\_WQ^r$. Hence, in order to achieve the feasible solution, the following condition needs to be hold true i.e.:

$$O_{frg} \times M \leq twl_r \times M - sum\_WQ^r \tag{3.3}$$

The maximum number of context switches/ full reconfiguration D_i that can be afforded within twl_r is:

$$C_r = \lfloor \frac{twl_r \times M - sum\_WQ^r}{O_{frg} \times M} \rfloor \tag{3.4}$$

Therefore, the scheduling algorithm checks whether the condition in Eq. 3.3 is satisfied or not and calculates the maximum number of affordable full reconfigurations (D_i) using Eq. 3.4. These reconfigurations is required to allocate te tasks to the M VPs. The time duration between two consecutive reconfiguration events is denoted as *time-frame*. Thus, within a time-window tw_r, there will be D_i time-frames and denoted as $g_1, g_2, \ldots, g_{C_r}$. These time-frames are equi-sized and the length is denoted as G_r. It should be noted that within a time-frame in a vp, tasks will be executed.

At the beginning of each time-frame in twl_r, the scheduling algorithm selects M tasks with the *highest remaining workload-quota* and executes them for the duration of a time-frame, on the M available VPs. Hence, T_i requires $\lceil \frac{WQ_i^r}{G_r} \rceil$ number of time-frames to complete its workload-quota WQ_i^r. All tasks will finis their allotted workload-quota, if te total number of time-frames required by all the tasks, ($\sum_{i=1}^{n} \lceil \frac{WQ_i^r}{G_r} \rceil$) becomes at most the total number of available time-frames ($C_r \times M$). Thus,

$$\sum_{i=1}^{n} \lceil \frac{WQ_i^r}{G_r} \rceil \leq C_r \times m \tag{3.5}$$

As a task can not execute in parallel in multiple VPs, and $C_r \times G_r$, denotes the the maximum execution time for a task within a time-window.

$$WQ_i^r \leq C_r \times G_r \tag{3.6}$$

Example 1 Let us assume five tasks D_i, D_i, ..., D_i, having weights ($\frac{e_i}{p_i}$) 14/60, 36/90, 42/60, 72/90, 54/90 and 54/90. Without loss of generality, we represented the real-time constraint as: "T_i *must have to compute* e_i *number of instructions*[1] *within* p_i *time units"*. Also, we have $M = 4$ (4 partitions), $O_{frg} = 6$ time units[2] and the first time-window

$tw_1 = 60$. The task workload-quota to be executed within the interval tw_1 are: $WQ_1 = 14$; $WQ_1 = 14$; $WQ_1 = 14$ and $WQ_4 = WQ_4 = 36$. As $sum\_WQ = 152$, $tw_1 \times m = 240$ and $OF_{rg} \times m = 24$. $C_r = \lfloor(240 - 152)/24\rfloor = 3$ (refer Eq. 3.4). Hence, we may allow three time-frames in the time-window, each of length 14 time units. As there are three time frames hence, each task will have three subtasks in order to complete its execution within a time slice. We have represented this subtask in each frame as ts_1 where *j* denotes the subtask id and it should be same as the frame id.

The condition in Eq. 3.5 is also satisfied. After a full reconfiguration from 0 to 6 time slots, the first time-frame executes with the tasks T_{31}, T_{31}, T_{31}, T_{31} within 6 to 20 time units. The 2 nd full reconfiguration occurs between 20 and 26 time units. Tasks T_{31}, T_{31}, T_{31}, T_{31} are selected and executed in the second time-frame within the time unit interval 26–40. Finally tasks T_{31}, T_{31}, T_{31}, T_{31} executed in third time-frame from interval 46 to 60. Thus, each task finishes its workload-quota within the time-window/slice as illustrated in Fig. 3.2.

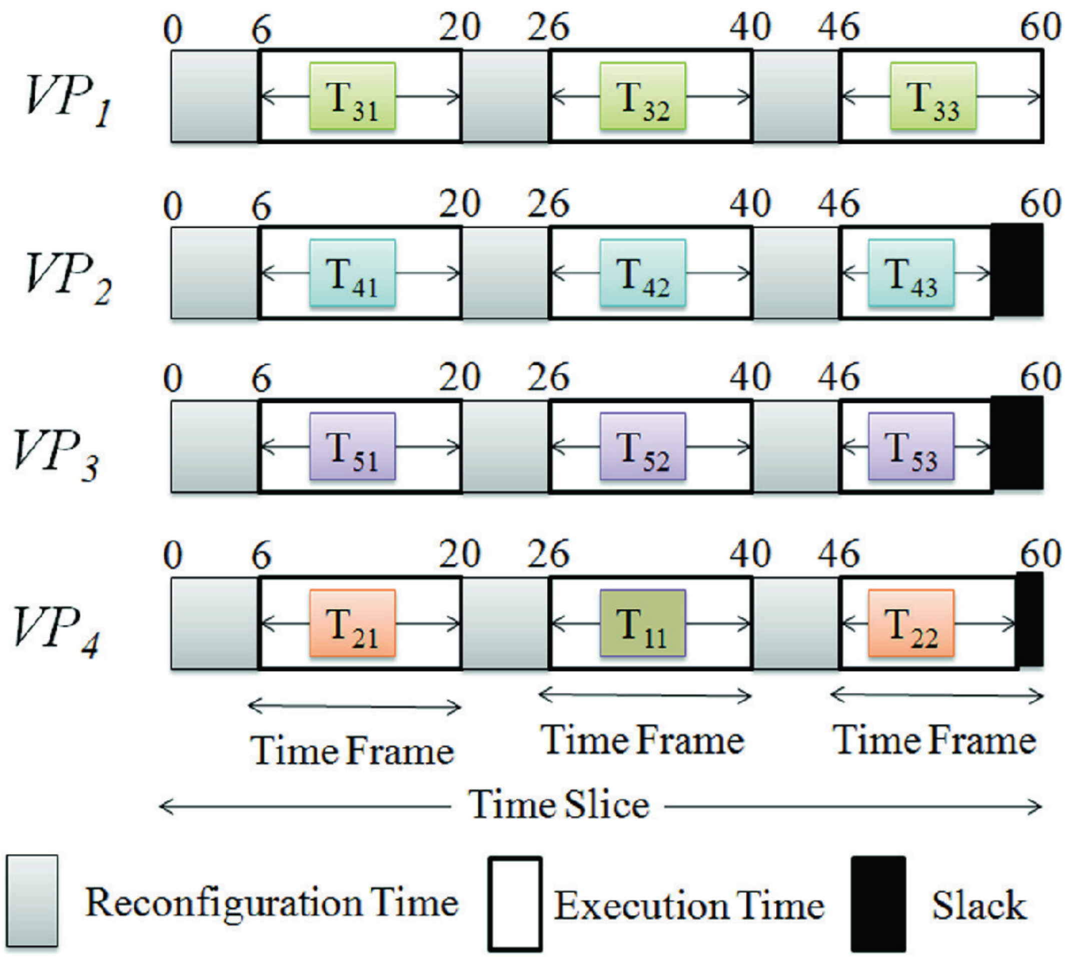

Fig. 3.2 Task scheduling for full reconfiguration system

3.4.2 Scheduling Algorithm for Partially Reconfigurable Systems

The main architectural advantage of a partially reconfigurable FPGA is that, unlike full reconfigurable FPGA, context switching (reconfiguration) is not a global event and is localized to individual VPs. In addition, O_{frg} is lower than O_{frg}.

If the condition in Eq. 3.2 is satisfied, the scheduling strategy will partition the task set into *M* disjoint subsets as follows: the tasks are allocated from the first VP, VP_1. While allocating tasks within a VP, it has to be ensured that the sum of task workload-quota along with O_{frg} should be less than length of the time-window twl_r. It should be noted that all VPs consumes O_{frg} at the beginning of the time-window; hence, the remaining capacity re_i for each VP R_i is given by: $rc_i = twl_r - O_{frg}$. A task say T_j, is allocated to a VP R_i by consuming O_{frg} and hence, re_i of R_i becomes: $rc_i = rc_i - WQ_j^r - O_{prg}$. However, Such an allocation is only allowed if $tw_1 \times m = 240$. Otherwise, T_j must be divided into two subtasks; the first task T_{j1} with a workload-quota value $WQ_{j1}^r = rc_i$, is executed in R_i while the second part with $WQ_{j2}^r = WQ_j^r - rc_i$, is executed in the next VP V_{i+1} provided $i < M$. If R_i be the last available VP ($i < M$), then scheduling will not commence in time-window tw_r as the system capacity is not sufficient.

Example 2 Let us consider the partially reconfiguarble FPGA contains 4 VPs and the same set of 6 tasks as used in our previous example however, D_i's execution requirement T_i is increased from 72 to 74. Hence, the workload-quota WQ_4^1 within the first time-window ($tw_1 = 60$ ms) becomes 49. We assumed that $O_{frg} = 6$ ms and $O_{prg} = 2$ ms. The sum of workload-quota (sum_WQ^1) is 193.

This task set can't be scheduled on fully reconfigurable FPGA. The initial capacity of each of the four VPs will be $twl_1 - O_{frg} = 54$ ms. Figure 3.3 shows the schedule generated by the scheduling algorithm for partially reconfigurable FPGA. It may be noted that while D_i, D_i and D_i is executed completely on a single VP, D_i, D_i and D_i is executed partially on two different VPs with their subtasks.

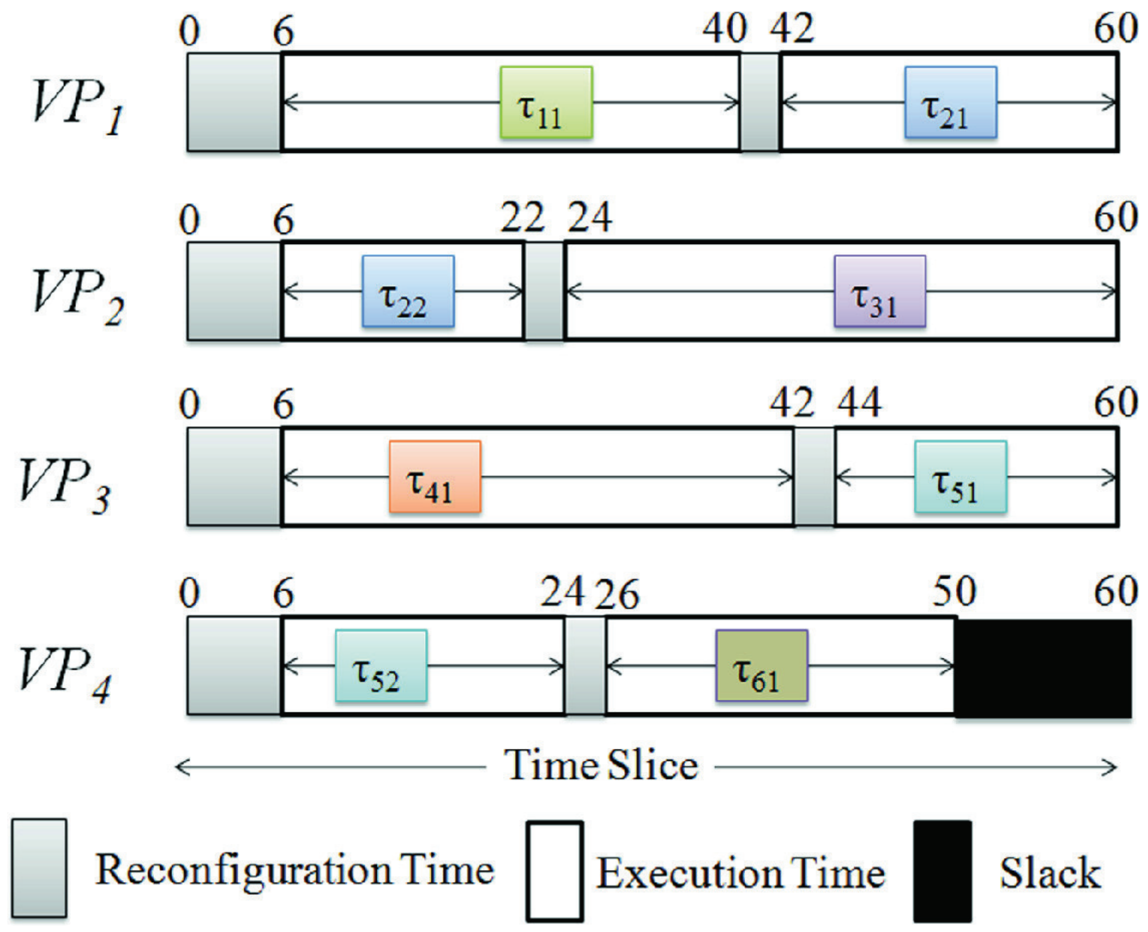

Fig. 3.3 Task scheduling for partially reocnfigurable FPGA

3.4.2.1 Avoidance of ICAP Conflict

Modern FPGAs contain a single configuration port i.e. ICAP and hence, no two VPs can be reconfigured simultaneously. This essentially means that, if we load a task bitstream in one VP via ICAP then at the same time, we cannot load any other task's bitstream as the ICAP port is busy. Our scheduling algorithm handles the situation in following way. Scheduling algorithm proceeds VP by VP, starting from the first VP. When it generates the schedule for the first VP, the time instants for reconfigurations are also obtained. Thus, scheduler has these reconfiguration instants when it generates schedule for the second VP. So, if any reconfiguration instant of the second VP overlaps with the reconfiguration instant of the first VP, then the execution of the task workload-quota is adjusted so that any overlapping can be avoided. Similarly, for the third VP, the task schedules along with reconfiguration instants for the first and second VPs are already known. Therefore, if any reconfiguration event of the third VP overlaps with those of other VPs, then the task workload-quota for the third VP is adjusted. This procedure will continue till the last VP is reached.

3.4.3 Handling Dynamic Tasks

In dynamic real-time system, task arrival and departure is a dynamic process. However, it as to be ensure that the newly arrived task will only be accepted if the inclusion of a new task will not violate the deadlines of already accepted tasks. We will now discuss how dynamic tasks can be handled in case of fully and partially reconfigurable FPGAs, respectively.

3.4.4 For Fully Reconfigurable FPGAs

Let us assume task T_i leaves the system, as a result, future time-frames dedicated for T_i's execution in the present time-window becomes free. These free time-frames can be used for the allocation of newly arrived tasks.

Let us assume task T_i arrives at a time instant say s_i in the middle of time-frame (corresponding time-window starts at time t_v), T_i has to wait till the current time-frame completes, as reconfiguration is not allowed between a time-frame. T_i may be allocated to a VP for execution in time-frames, where there exists free unallocated VPs. These time-frames will be executed at higher priority so that T_i meets the deadline in case, $d_i < (t_v + twl_r - t_u)$, where a_i is the deadline.

At the end of this allocation, the amount of execution could be be completed by the tw_r is calculated. If it is found that T_i's deadline can be met then it's remaining execution time and period will be updated.

3.4.5 For Runtime Partially Reconfigurable Systems

In case of partially reconfigurable FPGA, at each time-window task T_i is partitioned into subtasks and allocated specific sub-interval(s) on one or two VPs. For example, from Fig. 3.3, it may be observed that task D_i has been partitioned and the subtasks are allocated on V_2 and V_2, respectively.

If a task T_i departs from the system within a time-window, then the dedicated VP becomes free and can be utilised for allocating a dynamically arriving task within the time-window. However, in order to ensure that no VP remains unutilised due to certain termination of a task thus, the entire schedule (task execution sequence) after T_i's completion is preponed and executed starting from e_i, departure instance.

Figure 3.4.a shows a typical schedule in a time-window of length 60 on a VP, V_2. Let us assume that task D_i departs at time $t_d = 50$ (This is shown in Fig. 3.4b). Then the rest of the schedule is preponed and executed from time 50, as shown in Fig. 3.4c.

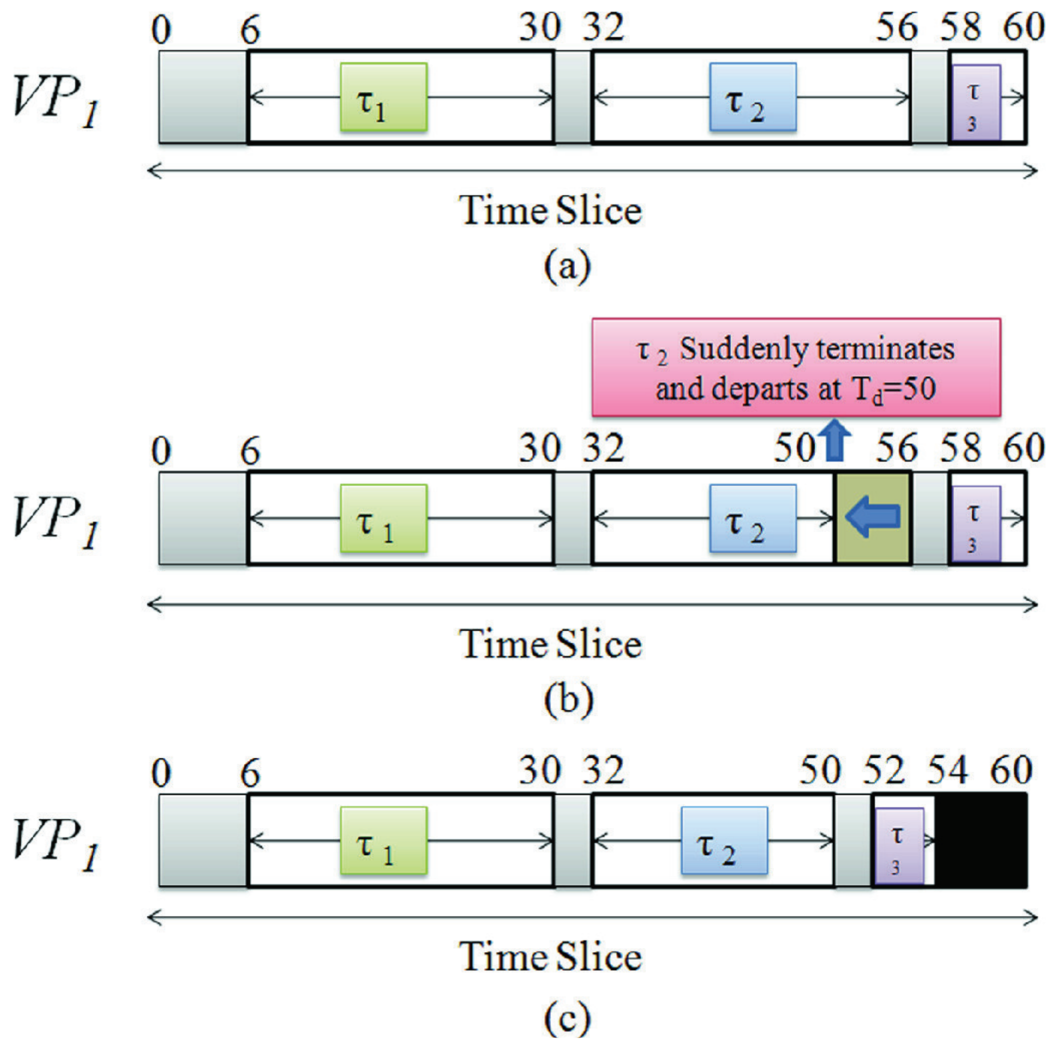

Fig. 3.4 Handling dynamic task departure

Let us consider T_i dynamically arrives at time (say s_i) in the middle of the *r*th time-window tw_r, its workload-quota (WQ_i) for the remaining part of the time-window is calculated and allocated in the free sub-intervals (if any) of one or more VPs starting with the VP having the highest remaining sub-interval. If the VP (ts_i say) with the highest remaining sub-interval is in idle state at time s_i (which means that ts_i currently has no task to execute for the rest of the time slice), then the entire workload-quota of T_i may be accommodated in ts_i.

Otherwise, the WQ_i of T_i must be partitioned and allocated to more than one time-window. While making such an allotment of free sub-intervals for T_i at time s_i, T_i is assigned the highest priority (this is done by postponing the rest of the schedules on these VPs to the end of the time-window) to account for the case in which T_i's deadline is earlier than the next time-window boundary. This task allocation mechanism has been depicted in Fig. 3.5.

Figure 3.5 shows the schedules for an arbitrary time-window on two VPs V_2 and V_2. At a time $t_a = 22$ when a new task T_{new} arrives with workload-quota $WQ_i = 18$, then the task allocation scheme is described in Fig. 3.6.

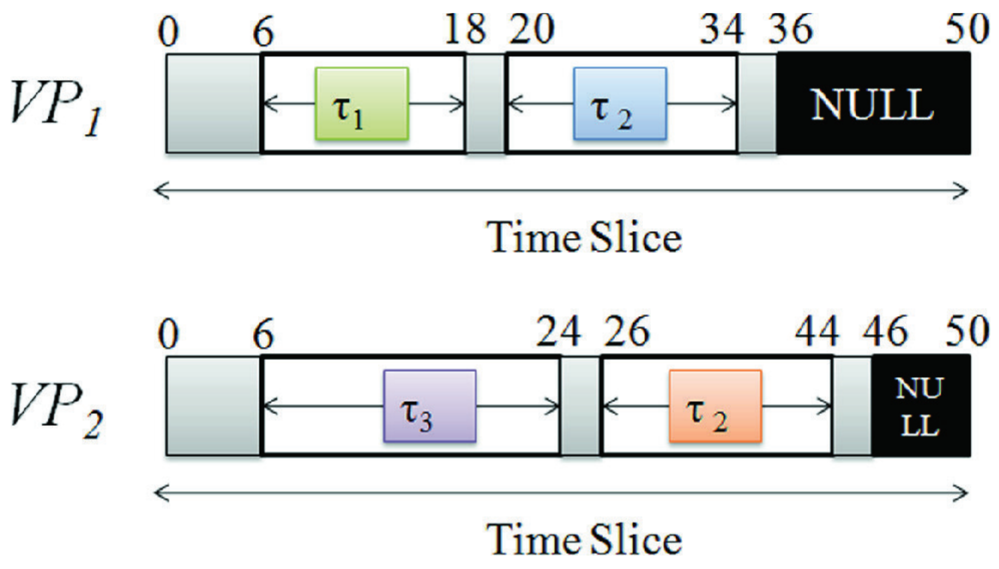

Fig. 3.5 Schedule in tile V_2 & V_2

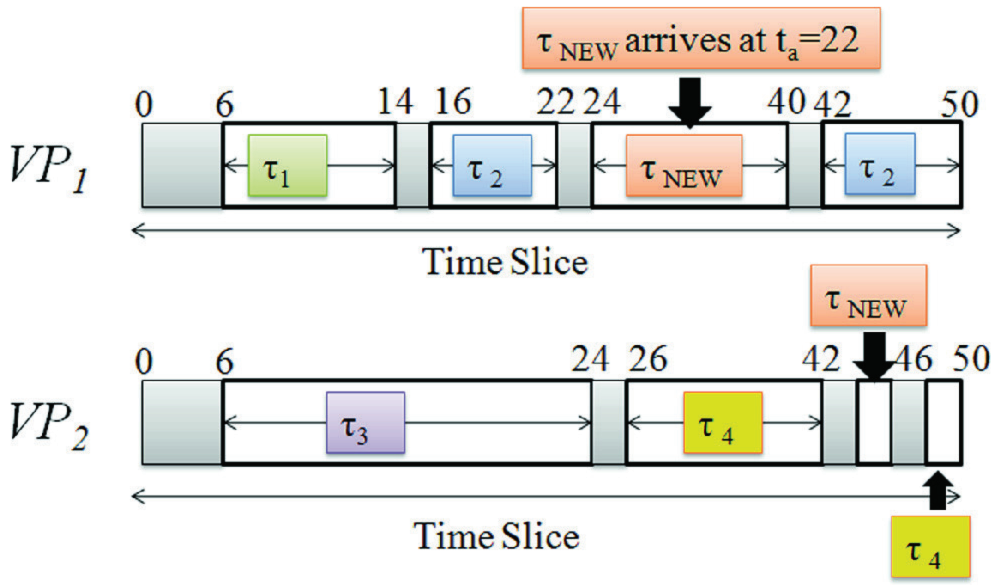

Fig. 3.6 Handling dynamic task arrival

3.5 Experiments and Results

We have evaluated the scheduling algorithms through simulation experiments followed by hardware implementations. *Task Rejection Rate (TRR)* has been used as the principal metric for the evaluation. *TRR* can be defined as the ratio between the total number of tasks rejected and total number of tasks arrived. That is,

$$TRR = (\nu/\psi) \times 100 \tag{3.7}$$

where, ν and r_i denote the total number of rejected tasks and total number of arrived tasks, respectively.

Experimental Setup: We have considered normal distributions for the task set generation.. Tasks weights ($wt_i = \frac{e_i}{p_i}$) and task execution periods (p_i) have also been taken from normal distributions. Given the task weights, we obtain *WL*, work-load by summing up the the task weights. Utilization *U* can be obtained as:

$$U = \frac{WL}{M} \times 100\,(\%) \tag{3.8}$$

where, *M* denotes the number of VPs.

Now, we will introduce various parameters for our simulation.

1. *Number of VPs M*: The FPGA contains into 2, 4, 6 and 8 VPs.
2. *Utilization U*: Task utilization values varying from $U = 50\%$ to $U = 90\%$.
3. *Average individual tasks weight* μ_{wt}: μ_{wt} have been considered in the range 0.1–0.5.

For a given the system utilisation (*U*), the average number of tasks (β) can be derived as:

$$\rho = \frac{U \times M}{100 \times \mu_{wt}} \tag{3.9}$$

The total schedule length is 200,000 time-slots. All results are generated by running 40 different instances and taking average of these 40 runs.

3.5.1 Results and Analysis

Fig. 3.7 shows the *TRR* suffered by the fully reconfigurable FPGA having eight VPs ($M = 8$) when the utilisation varies from 50 to 90%. From the figure, it can be observed that as the utilisation increases the *TRR* also increases. The main reason for this is that higher utilisation causes higher number of tasks (ref Eq. 3.9) in the system. As the number of tasks increases, the slack inside the VPs decreases and thus, the less number of tasks are scheduled. Specifically, higher *U*, the LHS of Eq. 3.5 becomes higher. Thus, the probability of failure of Eq. 3.5 increases. Another interesting observation is that as the μ_{wt} increases from 0.1 to 0.5, *TRR* decrease. The main reason behind this is that, for a fixed utilisation, higher μ_{wt} refers less number of tasks (refer Eq. 3.9) and thus, rejection rate decreases.

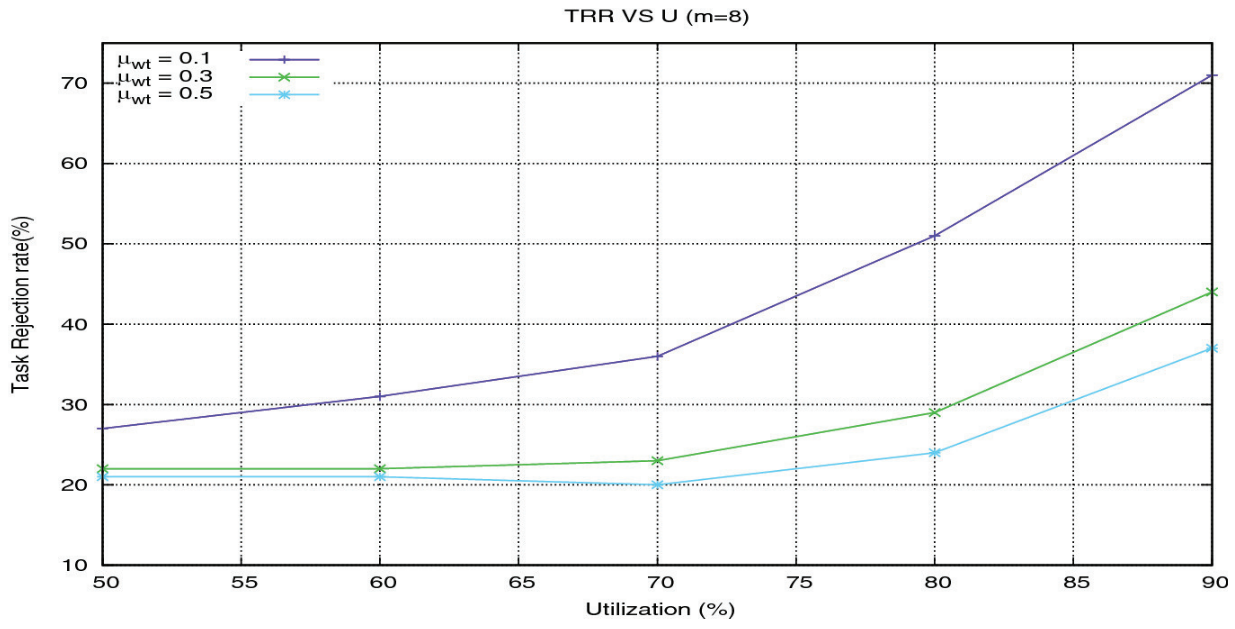

Fig. 3.7 *TRR* versus *U*; $M = 8$

Figure 3.8 shows the variation of the *TRR* with respect to number of VPs, *M*. It can be observed that as the VPs increase while maintaining the *U* fixed at 70%, the *TRR* decreases. This could be attributed to the fact that as the VPs increase, the system wide capacity also increase and thus, more number of tasks can be accommodated in the system and thus, *TRR* decreases Similarly, as the μ_{wt} increases the number of task in the system decreases and this attributes to lower rejections.

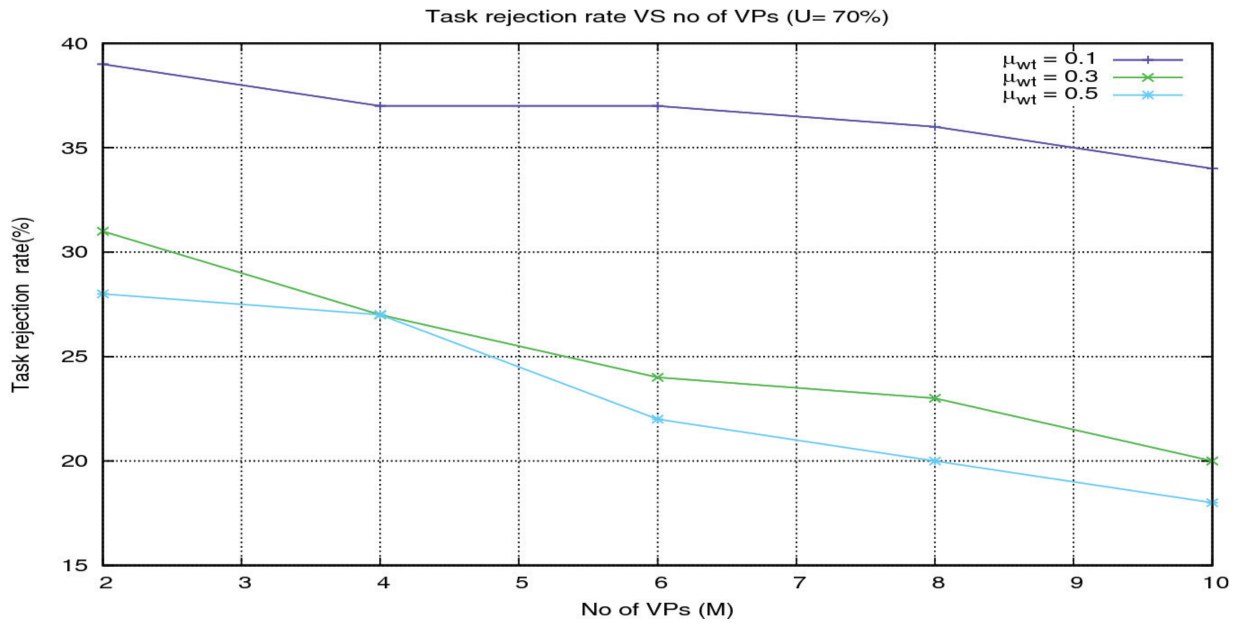

Fig. 3.8 *TRR* versus *M*; $C_r \times M$ %

Table 3.1 TRR versus O_{frg}; μ_{wt} = 0.3; $M = 8$ and $C_r \times M$%

O_{frg} **(ms)**	**12**	**18**	**24**	**30**
TRR (%)	23	32	42	55

Table 3.1 depicts how the *TRR* varies when the full reconfiguration overhead O_{frg} varies. Here, we fixed the VP at 8, utilisation *U* was fixed at 70%, and μ_{wt} is fixed at 0.3. Now, it can be observed that higher value of O_{frg} causes higher value of *TRR*. This is mainly due to the fact that when the O_{frg} increases D_i decreases (refer, Eq. 3.4) and as D_i reduces,the probability of failure of Eq. 3.5 increases, causing a higher number of tasks to be rejected.

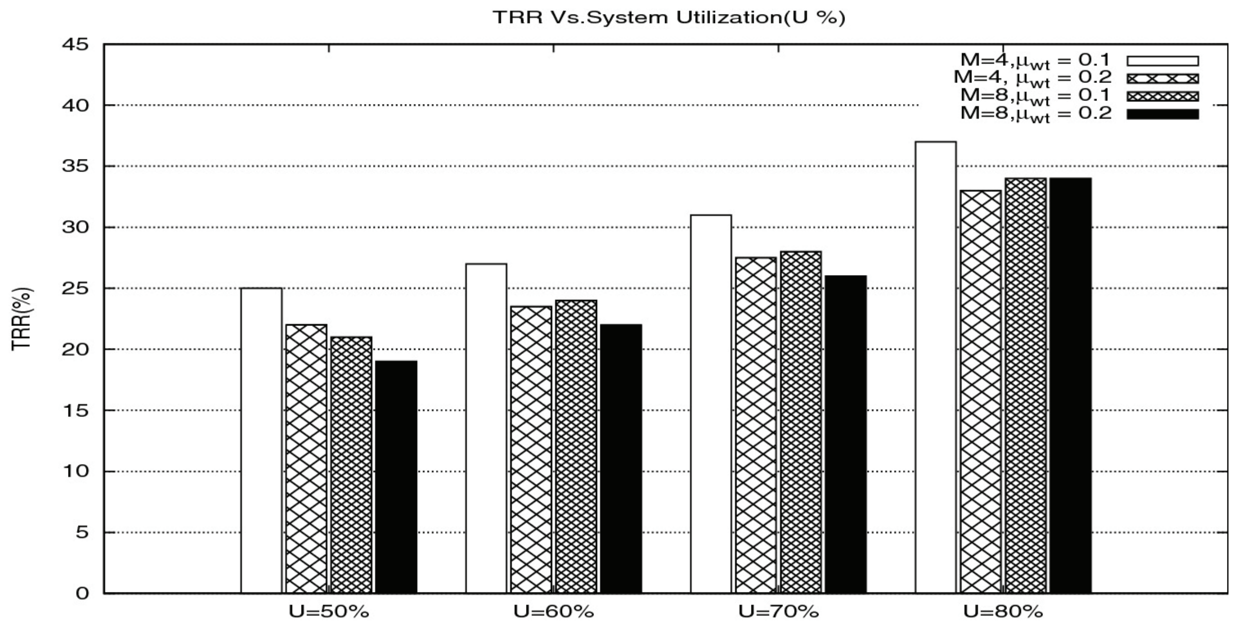

Fig. 3.9 *TRR* versus *U*

Figure 3.9 exhibits the performance of our scheduling algorithm devised for partially reconfigurable systems. We will now discuss the obtained trends from this figure and its corresponding justifications.

- *TRR increases with U*: This is because higher *U* is causing more number of tasks in the system and thus, reducing the available capacity within the tiles. Hence, rejection is higher.
- *TRR decreases with* μ_{wt}: Higher μ_{wt} refers to less number of tasks in the system and thus, rejection is also less.
- *TRR is less than fully reconfigurable systems*: This mainly due to two facts. Firstly, the context switching is no longer a global event rather it is localised to individual VPs. This asynchronous reconfiguration leverages the better utilisation of individual VPs and thus, rejection is less. Another reason is that the reconfiguration overhead for partially reconfiguarable FPGAs is less than the fully reconfigurable systems.
- *TRR decreases with the increase in M*: This observation can be attributed to the fact that, higher be the number of *M* lower be the O_{frg} (refer, Sect. 3.3). As the O_{frg} decreases rejection rate also decreases.

Table 3.2 Task rejection rate (%)

	M = 2			M = 4			M = 8		
U (%)	μ_{wt} = 0.1	μ_{wt} = 0.3	μ_{wt} = 0.5	μ_{wt} = 0.1	μ_{wt} = 0.3	μ_{wt} = 0.5	μ_{wt} = 0.1	μ_{wt} = 0.3	μ_{wt} = 0.5
50	30	28	26	25	21	19	21	18	16
60	31	30	28	27	23	22	24	21	20
70	35	33	32	31	27	26	28	25	22
80	42	40	38	37	32	30	34	33	31
90	45	43	425	42	37	335	39	37	35

Table 3.2 shows *TRR* suffered for a partially reconfigurable FPGAs. From this Table, three trends can be observed. First, as the utilisation increases, *TRR* also increases, this is mainly because, higher *U* is causing large number of tasks in the system and as a result the slack within the VPs are reducing causing higher rejections. Similarly, as μ_{wt} increases

TRR decreases. But, it has to be noted that for fixed parameters (i.e., *M*, *U* and μ_{wt}) partially reconfigurable systems suffer less rejections than

fully reconfigurable systems this mainly for two reasons i. the partial reconfiguration overhead is lower than the full reconfiguration overhead. ii. Each individual VPs can be reconfigured without interrupting other VPs, this asynchronous reconfiguration is another factor for less rejections. Another interesting observation is that *TRR* decreases as the number of VPs increase. This observation can be supported by the fact that the ($O_{prg} = \frac{O_{frg}}{M}$); as shown in Sect. 3.3)

O_{frg} decreases with increase in the number of VPs.

3.6 Hardware Prototype for Multiple Tasks Processing on FPGA

A simple prototype system for hardware task multitasking has been implemented on ZYNQ:ZC702 FPGA board based on our scheduling approach with benchmark task sets. *We would like to emphasize that with this simple HW prototype, we want to develop a proof of concept* *system that multitasking on FPGAs with "tiled" architecture is indeed feasible.* Our hardware prototype works as follows:

- The the task scheduling algorithm runs on ARM processor, located in the ZYNQ board.
- We have taken three applications from *EPFL Combinational Benchmark Suite* [AGDM15] i.e. *Context-adaptive variable-length coding (CAVLC)*, *Decoder (Dec)*, *Integer to float conversion (I2F)*. The actual execution cost of sample tasks is shown in Table 3.3.
- Each of the application/task is coded in VHDL and performance of each such task is measured through proper test-bench.
- At the beginning, hardware tasks are stored in its executable format (as *.bit*) in an external memory.
- Three RRs (Reconfigurable Regions) are created and three tasks were mapped into these RRs.
- Tasks were executed for length of pre-calculated *time-frame* (as determined by the s) till a full reconfiguration overhead.

Table 3.3 Benchmark tasks execution overhead on ZYNQ

Tile	Tasks	Execution overhead (ticks)
RR1	CAVLC	465335
RR2	Dec	211654
RR3	I2F	106723

The ZYNQ FPGA contains two types of PEs that is the FPGA fabric and dual core ARM Cortex-A9 processor. The portion which contains ARMs are termed as PS (Processing System) and the region with FPGA logic is termed as PL or Programmable logic region [CEES14]. In PL, we have created VPs (using Xilinx Vivado tool) so that the hardware tasks can be executed. In PS, ARMs are prefabricated and available in IC format.

Customization of the platform: For the evaluation of our scheduling strategies, we have customized the architecture of ZYNQ platform.

- In the PL, the VPs are created and the execution of hardware task on that VP is designed by writing the placement constraints on the UCF [CEES14] file. Here, the UCF stands for user constraint file.

- These VPs are operating with clock (FCCLK) of 50 MHz frequency.
- One of the available ARM core is used for the execution of our scheduling algorithm and the frequency of ARM core is 650 MHz.
- As the algorithm is executed in the PS side and hardware tasks are executed in the PL side. Hence, PS and PL needs to be tightly coupled. PS and PL communications are ensured through GP0 and GP1 port. PS and PL are connected with On-Chip Memory using AXI interface.

Table 3.4 shows the *TRR* obtained by varying *U*, where the number of VPs are fixed at four. From the obtained results, it can be concluded that the hardware results follow the similar trends as obtained in software simulations.

Table 3.4 TRR versus *U*

***U* %**	**30**	**40**	**50**	**60**
TRR (%)	10	16	22	28

3.7 Conclusion

Scheduling of real-time tasks on FPGAs is a challenging problem. One has to ensure that the proposed algorithm should achieve high resource utilization. Challenges are mainly due to two facts. Firstly, effectively placing hardware tasks on FPGAs such that the FPGA remains less fragmented and secondly, the reconfiguration overheads require to be judiciously handled so that deadlines are not jeopardized.

In this chapter, we have shown two scheduling methodologies for real-time task sets on fully and partially reconfigurable FPGAs. We have followed 2D slotted area model. The dynamic task handling mechanisms have also been discussed. Simulation results reveal that the proposed strategies are able to achieve higher utilisation with lower task rejection rate under various simulation scenarios. The software outcomes are also validated through real hardware implementation.

References

[ABBT04] A. Ahmadinia, C. Bobda, M. Bednara, J. Teich, A new approach for on-line placement on reconfigurable devices, in *18th International on Parallel and Distributed Processing Symposium, Proceedings* (IEEE, 2004), p. 134

[AGDM15] L. Amarú, P.-E. Gaillardon, G. De Micheli, The EPFL combinational benchmark suite, in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, number EPFL-CONF-207551 (2015)

[AS00] J.H. Anderson, A. Srinivasan, Early-release fair scheduling. ECRTS **2000**, 35–43 (2000)

[BGHD13] S. Bhasin, S. Guilley, A. Heuser, J.-L. Danger, From cryptography to hardware: analyzing and protecting embedded Xilinx Bram for cryptographic applications. J. Crypt. Eng. **3**(4), 213–225 (2013)
[Crossref]

[BGSH12] L. Bauer, A. Grudnitsky, M. Shafique, J. Henkel, Pats: a performance aware task scheduler for runtime reconfigurable processors, in *2012 IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2012), pp. 208–215

[BKS00] K. Bazargan, R. Kastner, M. Sarrafzadeh, Fast template placement for reconfigurable computing systems. IEEE Des. Test Comput. **17**(1), 68–83 (2000)
[Crossref]

[CEES14] L.H. Crockett, R.A. Elliot, M.A. Enderwitz, R.W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc* (Strathclyde Academic Media, 2014)

[CH05] Y.-H. Chen, P.-A. Hsiung, Hardware task scheduling and placement in operating systems for dynamically reconfigurable SOC. Embed. Ubiquitous Comput.-EUC **2005**, 489–498 (2005)

[DE01] O. Diessel, H. Elgindy, On dynamic task scheduling for FPGA-based systems. Int. J. Found. Comput. Sci. **12**(05), 645–669 (2001)
[MathSciNet][Crossref]

[DP05] K. Danne, M. Platzner, Periodic real-time scheduling for FPGA computers, in *Third International Workshop on Intelligent Solutions in Embedded Systems* (2005), pp. 117–127

[EFZK08] M. Esmaeildoust, M. Fazlali, A. Zakerolhosseini, M. Karimi, Fragmentation aware placement algorithm for a reconfigurable system, in *Second International Conference on Electrical Engineering, 2008. ICEE 2008* (IEEE, 2008), pp. 1–5

[FPMM11] U. Farooq, H. Parvez, H. Mehrez, Z. Marrakchi, Exploration of heterogeneous FPGA architectures. Int. J. Reconfig. Comput. **2011**, 2 (2011)
[Crossref]

[Gua+07] N. Guan et al., Improved schedulability analysis of edf scheduling on reconfigurable hardware devices, in 2007 *IEEE International Parallel and Distributed Processing Symposium* (2007)

[HKM+14]

T. Hayashi, A. Kojima, T. Miyazaki, N. Oda, K. Wakita, T. Furusawa, Application of FPGA to nuclear power plant i&c systems, in *Progress of Nuclear Safety for Symbiosis and Sustainability* (Springer, 2014), pp. 41–47

[HSH09] P.-A. Hsiung, M.D. Santambrogio, C.-H. Huang, *Reconfigurable System Design and Verification*, 1st edn. (CRC Press Inc, Boca Raton, FL, USA, 2009) [Crossref]

[HTK15] M. Happe, A. Traber, A. Keller, Preemptive hardware multitasking in reconos, in *Applied Reconfigurable Computing* (Springer, 2015), pp. 79–90

[HV04] M. Handa, R. Vemuri, An efficient algorithm for finding empty space for online FPGA placement, in *Proceedings of the 41st annual Design Automation Conference* (ACM, 2004), pp. 960–965

[JLJ+13] J. Jin, S. Lee, B. Jeon, T.T. Nguyen, J.W. Jeon, Real-time multiple object centroid tracking for gesture recognition based on FPGA, in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication* (ACM, 2013), p. 80

[JTE+12] K. Jozwik, H. Tomiyama, M. Edahiro, S. Honda, H. Takada, Comparison of preemption schemes for partially reconfigurable FPGAs. IEEE ESL **4**(2), 45–48 (2012)

[JTHT10] K. Jozwik, H. Tomiyama, S. Honda, H. Takada, A novel mechanism for effective hardware task preemption in dynamically reconfigurable systems, in *2010 International Conference on Field Programmable Logic and Applications (FPL)* (IEEE, 2010), pp. 352–355

[JTW07] S. Jovanovic, C. Tanougast, S. Weber, A hardware preemptive multitasking mechanism based on scan-path register structure for FPGA-based reconfigurable systems, in *Second NASA/ESA Conference on Adaptive Hardware and Systems, 2007. AHS 2007* (IEEE, 2007), pp. 358–364

[LFS+10] G. Levin, S. Funk, C. Sadowski, I. Pye, S. Brandt, Dp-fair: a simple model for understanding optimal multiprocessor scheduling. ECRTS **2010**, 3–13 (2010)

[Liu00] J.W.S. Liu, *Real-Time Systems*, 1st edn. Prentice Hall (2000)

[LMBG09] Y. Lu, T. Marconi, K. Bertels, G. Gaydadjiev, Online task scheduling for the fpga-based partially reconfigurable systems, in *Reconfigurable Computing: Architectures, Tools and Applications* (Springer, 2009), pp. 216–230

[LWH02] W.J. Landaker, M.J. Wirthlin, B.L. Hutchings, Multitasking hardware on the slaac1-v reconfigurable computing system, in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream* (Springer, 2002), pp. 806–815

[Mar14] T. Marconi, Online scheduling and placement of hardware tasks with multiple variants on dynamically reconfigurable field-programmable gate arrays. Comput. Electr. Eng. **40**(4), 1215–1237 (2014) [Crossref]

[MVGR13]

A. Medina Villanueva, A.G. Ross, Htr: on-chip hardware task relocation for partially reconfigurable FPGAs, in *Proceedings of the 9th International Conference on Reconfigurable Computing: Architectures, Tools, and Applications, ARC'13* (2013), pp. 185–196

[SRBS10] M.D. Santambrogio, V. Rana, I. Beretta, D. Sciuto, Operating system runtime management of partially dynamically reconfigurable embedded systems, in *2010 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)* (2010), pp. 1–10

[Sta11] E. Stavinov. *100 Power Tips for FPGA Designers*, 1st edn (CreateSpace, 2011)

[Tap13] S. Tapp, Bpi fast configuration and impact flash programming with 7 series FPGAs. *XAPP-587, Xilinx Application Notes* (2013)

[TKG10] D. Theodoropoulos, G. Kuzmanov, G. Gaydadjiev, A 3d-audio reconfigurable processor, in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on FPGAs* (2010), pp. 107–110

[TSMM04] J. Tabero, J. Septién, H. Mecha, D. Mozos, A low fragmentation heuristic for task placement in 2d rtr hw management, in *FPL* (Springer, 2004), pp. 241–250

[WBL+14] G. Wassi, M.E. Amine Benkhelifa, G. Lawday, F. Verdier, S. Garcia, Multi-shape tasks scheduling for online multitasking on FPGAs, in *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)* (IEEE, 2014), pp. 1–7

[WL12] G. Wassi-Leupi, Online scheduling for real-time multitasking on reconfigurable hardware devices (2012)

[WP03] H. Walder, M. Platzner, Online scheduling for block-partitioned reconfigurable devices, in *Proceedings of the Conference on Design, Automation and Test in Europe-Volume 1* (IEEE Computer Society, 2003), p. 10290

[WSP03] H. Walder, C. Steiger, M. Platzner, Fast online task placement on FPGAs: free space partitioning and 2d-hashing, in *Proceedings. International on Parallel and Distributed Processing Symposium, 2003* (IEEE, 2003), p. 8

[Xil09] Xilinx. Virtex-4 FPGA configuration user guide, xilinx. *June* (2009)

[Xil10] Incorporation Xilinx, Virtex-4 family overview. *Tech. Doc. DS112 (v2. 0)* (2010), pp. 1–8

Footnotes

1 Assuming 1 instruction is completed in 1 time unit using standard FPGA clock frequency.

2 Assuming a hypothetical FPGA with small floor area.