Hardware Partitioning Algorithm for Reconfigurable Operating System in Embedded Systems

Krishnamoorthy Baskaran, Wu Jigang and Thambipillai Srikanthan
Centre for High Performance Embedded Systems (CHiPES),
School of Computer Engineering,
Nanyang Technology University, Singapore.
{asbaskaran, asjgwu, astsrikan}@ntu.edu.sg

Abstract

In focusing the efficacy of reconfigurable computing, one of the important dimensions is the partitioning of the Field Programmable Gate Arrays [FPGAs]. As the amount of reconfigurable resources in a single-chip FPGA is increased tremendously, it allows the change of hardware tasks at run-time. The efficient area utilization of FPGA requires an efficient partitioning algorithm. A reconfigurable operating system is needed in run-time reconfigurable environment to manage the system resources effectively in a multitasking manner.

This paper addresses our work toward an efficient initial and run-time partitioning algorithm for block partitioning of FPGA reconfigurable resources and a new methodology for the real-time scheduling of hardware tasks in block-partitioned FPGA. The partitioned blocks in the FPGA are one-dimensional static blocks with different widths. A novel partitioning and scheduling algorithm for reconfigurable embedded devices is presented together with simulation results.

1 Introduction

Today, much of the recent research work on Field Programmable Gate Arrays (FPGA) architectures for reconfigurable computing focuses on run-time reconfiguration. The run-time reconfiguration platform based design makes use of novel technologies for improving performance, cost, energy-efficiency, and time-to-market. A run-time reconfigurable platform system would be able to decide - on the fly - which hardware task to execute on the FPGA area and what software task to run on the CPU. The use of reconfigurable system is continuously growing in the areas like wearable computing [1], mobile communication system [2], video communications [3], neural computing, and network processors.

Run-time reconfiguration is the ability to rapidly change the functionality of an FPGA during the execution of the hardware task. The reconfigurable hardware can be used to execute designs, which are larger than the available hardware resources. The current FPGAs technologies provide several millions of gates along with partial reconfiguration and readback features.

Our partitioning algorithm for run-time reconfiguration platform is based on 1-D partition and placement technique. The proposed methodology is very well suited for currently available partial reconfigurable FPGAs like the Xilinx Virtex Family[[4]. In this paper we propose an efficient partitioning algorithm for FPGAs. The partitioning process is used to determine the optimal number and widths of blocks in reconfigurable FPGA area, which can be individually reconfigured to run a hardware task.

The remainder of this paper is organized as follows. Section 2 presents a survey of related work. Section 3 gives the details of target architecture and problem model. Section 4 explains the initial and runtime-partitioning algorithm. Section 5 concludes the paper.

2 Survey of Research Activity in Related Works

In this section, we briefly survey a selection of related work in the area of runtime reconfigurable platform operating system and point out the task placement and partitioning methods. Mignolet et al. [6] introduce relocatable tasks, which can be executed either

in software or hardware, depending on the available resources and the performance required. Simmler addressed multitasking and task preemption on FPGA in [5]. In [7] [9], discussed hardware operating system functionalities like partitioning, placement and routing.

The partitioning and placement is an important issue in embedded systems. Fragmentation of the FPGA's resources is known to cause low area utilization [11, 12] in the dynamic reconfiguration systems. Bazargan et al. [10] addressed the issue of placing application mappings onto a single device for hardware execution in a reconfigurable computing system. Their work presents a placement method that can be applied to dynamically adaptive hardware. The solution offered is a hybrid between the typical first-fit in a finite space and best-fit algorithms and trades quality of placement for speed. Walder et al. [8] combined an enhanced form of Bazargan's partitioning algorithm and a placement-finding algorithm using 2D-hashing. The placement finding algorithm has linear time complexity. The update of the hash matrix is a quadratic time complexity in the worst case. Up to now, not much work has been reported to deal with partition of the FPGA's logic area.

3 Target Architecture And Problem Model

3.1 Run-time Reconfigurable Platform Architecture

The proposed reconfigurable computing target architecture is shown in Fig. 1. The ReConfigurable System-on-Chip (RCSoC) is used for this platform. It consists of one soft-core CPU, HOPES (Hardware OPErating System), external memory, and communication bus. The embedded processor is implemented in the RCSoC as a fixed (non-reconfigurable) core. HOPES manages the RTOS facilities, hardware task allocation, and the placement engine. The communication block serves to communicate among the hardware blocks, HOPES, and the embedded processor.

3.2 Technology Involves

ReConfigurable System-On-Chip (RCSoC): The RCSoC is an integration of microprocessor, memory, dedicated peripherals and embedded programmable logic area. We make use of a soft-CPU macro for this platform, because soft-CPU allows us to carry-out architectural changes to the CPU core to better support partitioning. The programmable logic area can be dynamically and partially reconfigurable.

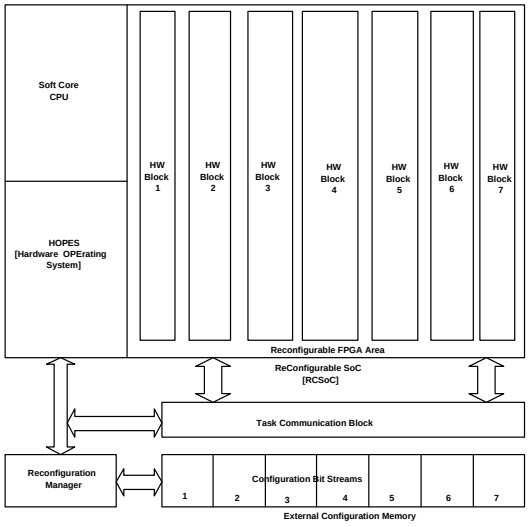

FIGURE 1: *Block Diagram of RCSoC*

Hardware OPErating System (HOPES): The reconfigurable hardware operating system is a new line of research and involves several challenges. HOPES manages the available hardware resources of the node, i.e., CPU, reconfigurable hardware unit, I/O, and memory. The main function of HOPES is to manage tasks. Generally, an application will contain a combination of tasks. While software tasks can be run on the CPU, hardware tasks execute on the reconfigurable hardware area. HOPES should keep track of available resources, meaning to find the location for incoming hardware task.

HOPES includes:

- Partitioning the reconfigurable hardware area depending upon task size
- Placement of tasks in the configurable area.
- Scheduling of the tasks
- Loading, executing and removing tasks

We propose a HOPES kernel running on a CPU within an FPGA. The rest of the resources on this RCSoC are available for custom computing circuits that can be reconfigured at run-time. The CPU has access to memory on the FPGA as well as an interface to larger off-chip memory. The CPU will have an off-chip communication interface.

3.3 The Problem Model

Partitioning, scheduling and placement of the hardware task are important issues in reconfigurable computing. A hardware task is a circuit, which is executable on the partitioned FPGA blocks. Every task

has different type of behavior, area requirements, and timing.

We introduce an efficient partitioning algorithm for runtime reconfigurable platforms to achieve better hardware resources utilization. Now we consider a sequence of tasks with different width. These tasks are assumed to be uniformly distributed in [w_{min} , w_{max}], where w_{min} and w_{max} are the lower bound and the upper bound of the task width respectively. Figure 2 illustrates different types of tasks of different widths. Let $\overline{w}$ be the size of the reconfiguration device. This section explains two types of partitioning problems, namely initial partitioning and run-time partitioning. The initial partitioning algorithm will be described next.

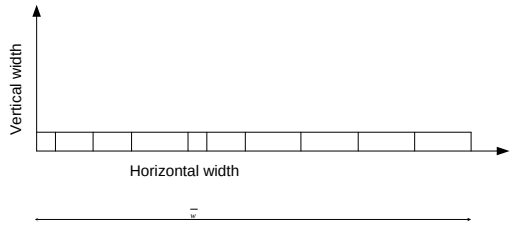

FIGURE 2: *Task Width distribution*

The following notations will be employed to describe the proposed partitioning algorithm:

w_{max}: Upper bound of task width.
w_{min}: Lower bound of task width.
$\overline{w}$: Device width.
w_i : Block width. ($w_{min} \leq w_i \leq w_{max}$)
m_i : Number of Blocks having width w_i
k : Number of distinct sized blocks, $k \geq 1$.

4 Proposed Partitioning Algorithm

4.1 Initial Partitioning Algorithm

The goal of initial partitioning of the runtime reconfigurable platforms is to achieve good partitioning of the FPGA reconfigurable area. It also lays the groundwork for the run-time partitioning algorithm. The initial partitioning problem can be stated as: Given a reconfigurable device of width $\overline{w}$, partition it (Figure 3) such that:

$$\overline{w} = \sum_{i=1}^{k} m_i w_i \tag{1}$$

$$\max_{1 \leq i,j \leq k} \{|m_i - m_j|\} \tag{2}$$

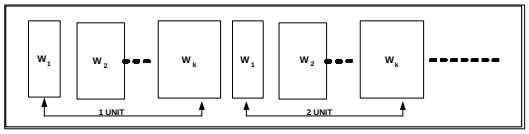

FIGURE 3: *Reconfigurable device partitioning*

We propose a heuristic algorithm for the initial partitioning problem, as it is an optimization problem. The following outlines the proposed algorithm.

1. Define k and w_i as follows: -

$$k = [\frac{w_{max}}{w_{min}}] \tag{3}$$

$$w_i = (i+1)w_{min} \tag{4}$$

$$w_k = w_{max} \tag{5}$$

This is because all task uniformly emerge in the interval [w_{min} , w_{max}]. In other words, we have k blocks of distinct size, such that

$$\begin{aligned} 2w_{min} &= w_1 < w_2 < w_3 ..... < w_{k-1} < w_k \\ &= w_{max} \qquad (6) \\ & \qquad (7) \end{aligned}$$

2. Determine m_i according to k and w_i

$$Assume \;\; s = \sum_{i=1}^{k} w_i.$$

$$Define \;\; t = [\frac{\overline{w}}{s}]$$

In other words, the device width is divided into t units each of width s and each unit is further subdivided into k distinct sized blocks.
Set initial value of each m_i to t.

3. For partitioning the remainder of the device width
i.e. $\overline{w} - ts$,
Define remaining width

$$r = \overline{w} - ts$$

If current remaining width $r > w_n$ n=k, k-1, ...1 then $m_n = m_n + 1$. After each partitioning, update the remainder width as follows: -

$$r = r - w_n \;\; n = k, k-1, ...1$$

Merge the last remaining width with a block of size w_1 .

In this section we demonstrate the partitioning algorithm through some simple, yet important, examples and also show that it is capable of yielding feasible solutions. Let us consider an FPGA device with $\overline{w}$ =100, w_{min} = 5, and w_{max}=20. Then k=20/5 = 4. The partitioned block widths are: w_1 = 8, w_2 = 12, w_3 = 16 and w_n =20 respectively.

5 Run-time Partitioning Algorithm

During the initial partitioning of an FPGA reconfigurable fabric, the minimum and maximum size of the hardware tasks are known. However during runtime, the incoming task size can be larger than the available block size. In that case no block is available for executing the incoming hardware task. So we have to efficiently combine the available free blocks to form a new block suitable for incoming hardware task. We propose a run-time partitioning algorithm for reconfigurable platforms to handle the partitioning issues. This algorithm helps to optimally merge the currently available free hardware blocks in FPGA area to obtain sufficient space for incoming task to execute.

The run-time partitioning algorithm satisfies the following conditions:

Merging Blocks:

- Only adjacent blocks can be merged
- At the time of merging, no tasks should be running in those blocks.
- After merging, the block size should be greater than or equal to the incoming task size
- Time taken for merging the blocks should be minimal
- The configuration time is proportional to the block size

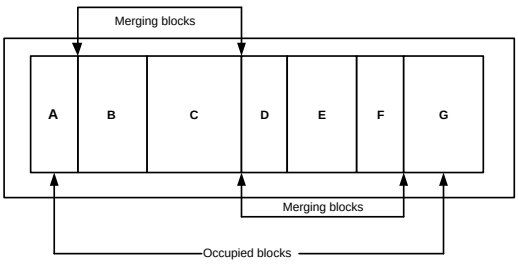

FIGURE 4: *Runtime Partitioning of the FPGA*

The above Figure 4 shows how to merge the available free blocks in the FPGA fabric. For example, we assume the incoming task size is 30. The available blocks sizes are 5,10,15 and 20. In the figure, A and G blocks are occupied by tasks. The other blocks are ready to be used by incoming tasks. If the incoming task size is larger than block size, adjacent blocks must be merged to assign space for the incoming task. There are two possible ways to do this; one is to merge blocks B and C and the other is to merge D, E, and F blocks. Merging blocks B and C is the efficient solution for the above case.

6 Experimental Results

6.1 Simulation-Environment

We have simulated the proposed algorithm to experimentally investigate the effectiveness of the partitioning algorithm and placement techniques. The parameters of the simulator include the width of the reconfigurable device, the configuration time, minimum and maximum size of the tasks, number of tasks, and configuration and readback times for one column. We report on a number of selected experiments conducted with this simulation environment. This section illustrates an experiment to examine the influence of the partitioning algorithm and placement technology on the performance of the nonpreemptive First Come First Serve (FCFS) scheduler. We assume a sequence of task with no inter-task communication. Task sets with different number of tasks have been considered. The simulation environment is similar to Xilinx Virtex XC2V 6000 (6 million gates) where the configuration and readback time of the one column takes 193 μs.The dynamic power dissipation of a Virtex-II CLB is 5.9 W per MHz for typical designs [13]. There is no benchmark currently available to characterize the workload of the reconfigurable systems. In addition, there are no statistical data available from the real-world applications to model this. Therefore, we have to resort to randomly generated tasks. In hardware task management, a hardware design unit is called a task; it is executable on the partitioned FPGA. In order to execute a given hardware task some time is needed to configure the FPGA. In all our experiments, we place sets of 25,50,100,250,500,1000 tasks on a 96 X 88 FPGA. The arrival time, execution time and width of tasks are uniformly distributed as shown in Table 1. We compare our placement technique in two different ways: free mode (FM) and control mode (CM). In free mode, a task is allocated to any free block available. In control mode, a task

is placed to the block whose width is closest to and greater than the task size. Figure 5 shows the results for both placement techniques with average response time for FCFS. The free mode achieves better average response time than control mode. But there is no better area utilization. In Figure 6, we present the average configuration time for the task sets; the control mode saves more configuration time than free mode. According to the Figure 7 and Figure 8, we can say that control mode in more energy efficient.

	Minimum	Maximum
Task Width (CLB)	5	20
Arrival Time (ms)	0.5	50
Execution Time (ms)	2	20

TABLE 1: *Task Information*

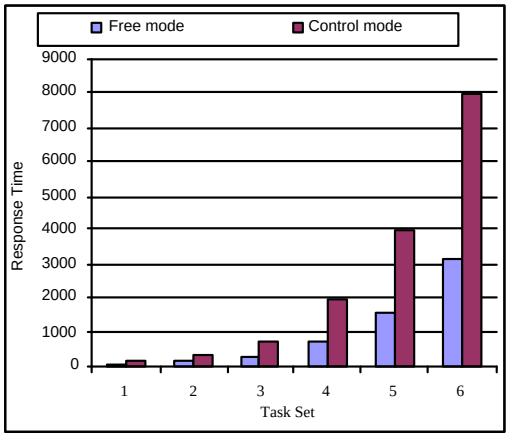

FIGURE 5: *Response Time for task execution*

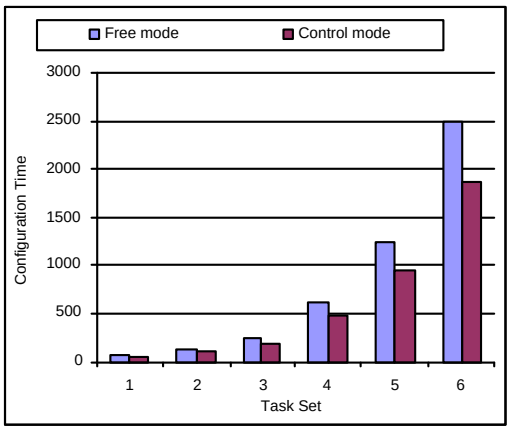

FIGURE 6: *Configuration time*

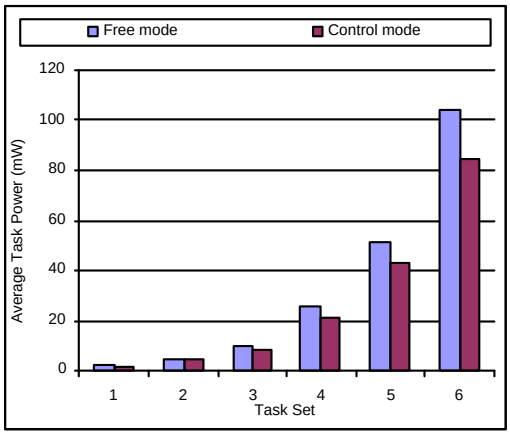

FIGURE 7: *Average Task Power*

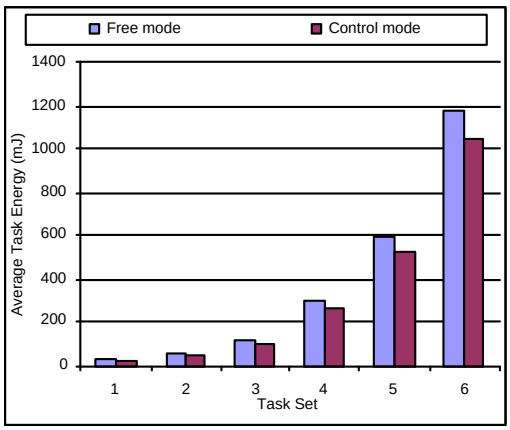

FIGURE 8: *Average Task Energy*

The Formal Description Of The Initial Partitioning Algorithm

Input:
Integer $\bar{w}$ – the size of the reconfigurable device;
Integer w_{min} – the lower bound of the sizes of the tasks;
Integer w_{max} – the upper bound of the sizes of the tasks;
Output:
Integers k, w_i and m_i such that $\bar{w} = \sum_{i=1}^{k} m_i \cdot w_i$ and $\max_{1 \le i,j \le k} \{|m_i - m_j|\}$ is minimized.
Algorithm Initial_Partitioning$(\bar{w}, w_{min}, w_{max})$
/* Find a good partitioning for the initial stage. */
Begin
/* fix the size of k */
1. Let $k = \lfloor w_{max}/w_{min} \rfloor$;
/* fix each w_i */
2. **for** $i = 1$ **to** $k - 2$ **do** $w_i = (i + 1) \cdot w_{min}$;

$w_k = w_{max}$;
3. $s = \sum_{i=1}^{k} w_i$;
4. $t = \lfloor \bar{w}/s \rfloor$;
5. **for** $i = 1$ **to** k **do** $m_i := t$;
/* partition the remainder interval*/
6. $r := \bar{w} - t \cdot s$;
7. **for** $i = k$ **downto** 1 **do**
/* get a block of size w_n from the remainder interval */
if $r \geq w_n$ **then**
begin $m_n := m_n + 1$;
$r := r - w_n$; n:=k,k-1,.....1;
end;
8. **if** $r > 0$ **then** $m_1 := m_1 + 1$;
End.

The Formal Description Of The Run-Time Partitioning Algorithm
Input:
The initial arrangement of the logic blocks: $b_1, b_2, \cdots, b_n$. Let $B = \{b_1, b_2, \cdots, b_n\}$. Assume a_i is the length of the side of the block i. The coming block is b with the size of a.
The objective function $F(B, b_i, b_j, b)$: the cost by replacing b_i to b_j by b, $i \leq j$.
Output:
The optimal arrangement of the current blocks.

Boolean Function Decision_of_Feasible_Solution $(B, current\_sum, left\_boundary)$;
/*find a feasible solution bounded by $left\_boundary$ and $right\_boundary$ */
begin
$part\_sum\_of\_blocks := current\_sum$;
$i := left\_boundary$;
while $(part\_sum\_of\_blocks < a)$ and $(i < n)$ **do**
$part\_sum\_of\_blocks := part\_sum\_of\_blocks + a_i$;
$i := i + 1$;
end;
if $i < n$ **then**
$Right\_boundary := i$;
$current\_sum := part\_sum\_of\_blocks$;
$Feasible\_solution :=$
$\{b_{lef\_boundary}, b_{left\_boundary+1}, \cdots, b_{right\_boundary}\}$;
return ('yes');
else return ('no');
end

Algorithm Schedule(B, b);
Begin
$Current\_sum := a_1$;
$Left\_boundary := 1$;
while $left\_boundary < n$ **do**
begin
if $Decision\_of\_Feasible\_Solution$
$(B, current\_sum, left\_boundary)$=='yes'
/* there exits feasible solution */
then
Calculate the current objective function $F(B, Feasible\_solution, b)$;
Update the current optimal solution if necessary;
$Left\_boundary := left\_boundary + 1$;
else
/* stop the while-loop*/
$left\_boundary := n + 1$;
end;
End

7 Conclusion

In this paper, we introduced an efficient partitioning algorithm for runtime reconfigurable platform. We presented a simulation framework that can be used to estimate average response time, configuration time, task power and task energy at different blocks on the FPGA. We use this model to increase area efficiency of the placement methodology. Our results show that partitioning based placement techniques provide better reconfigurable FPGA area utilization. The investigation of different types of scheduling algorithms, study on how to partition application-specific devices for reconfigurable computing can be taken up as future work.

References

[1] C. Plessl et al. 'Reconfigurable Hardware in Wearable Computing Nodes', In 'Proceedings of the 6th International Symposium on Wearable' Computers (ISWC), pages 215-222. IEEE Computer Society, October 2002.

[2] IMEC Interuniversity Micro Electronic Center, T-ReCS Gecko, http://www.imec.be.

[3] J.Villasenor, C.Jones, and B.Schoner, 'Video communication using rapidly reconfigurable hardware', IEEE Trans. Circuits Syst. Video Technol., 1995,5, (6), pp.565-567

[4] Xilinx Inc., Virtex Family Series http://www.xilinx.com.

[5] H. Simmler, L. Levinson, and R. Manner, 'Multitasking on FPGA Coprocessors'. In proceedings of the 10th International Workshop on Field Programmable Gate Arrays (FPL), pp 121-30. Springer, 2000.

[6] J.-Y. Mignolet et al, 'Infrastructure for Design and Management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-Chip, 'In proceedings of Design, Automation and Test in Europe (DATE), pp 986-991. IEEE Computer Society, March 2003.

[7] G. Wigley and D. Kearney. 'Research Issues in Operating Systems for Reconfigurable Computing', In proceedings of the International Conference on Engineering of Reconfigurable System and Algorithms (ERSA), pp 10-16. CSREA Press, Junie2002.

[8] Herbert Walder and Marco Platzner 'Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing' 17th International Parallel & Distributed Processing Symposium (IPDPS); Reconfigurable Architectures Workshop (RAW) April 2003

[9] Herbert Walder and Marco Platzner 'Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations' Conference on Engineering of Reconfigurable Systems and Architectures (ERSA), pages 284-287, June 2003

[10] Kiarash Bazargan, Ryan Kastner, and Majid Sarrafzadeh, 'Fast Template Placement for Reconfigurable Computing Systems', In IEEE Design and Test of Computers, volume 17, pp 68-83, 2000.

[11] M. Gericota, G. Alves, M. Silva, and J. Ferreira. 'Run-Time Management of Logic Resources on Reconfigurable Systems', In Proc. of Design, Automation and Test in Europe, Mar. 2003.

[12] M. Handa and R. Vemuri. 'Area Fragmentation in Re-configurable Operating Systems', In Proc. of the International Conference on Engineering of Reconfigurable Systems and Algorithms. CSREA Press, Jun. 2004.

[13] L. Shang, A. S. Kaviani, and K. Bathala"Dynamic power consumption in VirtexTM-II FPGA," 10th ACM International Symposium on Field-Programmable Gate Arrays(FPGA), Feb. 2002, pp.157-164.