Efficient On-Chip Task Scheduler and Allocator for Reconfigurable Operating Systems

Chuan Hong, Khaled Benkrid, Xabier Iturbe, Ali Ebrahim, and Tughrul Arslan

***Abstract*—This letter presents efficient and modular task scheduler and allocator support for dynamically and partially reconfigurable electronic systems. This enables hardware tasks to be preempted and arbitrarily placed at an optimal position on the chip on-the-fly. In particular, we present a novel fault-tolerant allocating algorithm called "best-fit empty area compact (BF-EAC)," and its on-chip implementation on a Xilinx Virtex-4 field-programmable gate array (FPGA), which circumvents emerging faults while maintaining more compact empty areas for emerging tasks. We also present an implementation of the early deadline first (EDF) scheduling heuristic used to optimize the chronological order of execution of hardware tasks to meet real time constraints. Put together, the placement and scheduling architecture efficiently exploits chip resources with a μs-grade computing speed and a lightweight footprint (less than 500 slices).**

***Index Terms*—Allocator, dynamic reconfiguration, field-programmable gate array (FPGA) partial reconfiguration, placement algorithm, scheduler.**

I. Introduction and Related Work

THE reprogrammability of field programmable gate arrays (FPGAs) has given rise to the possibility of hardware tasks behaving just like software tasks as these can be preempted, and mapped to different computing resources on the fly. However, this idea is still in its infancy in practice, mainly because of the lack in system and technology support. To date, the majority of approaches to dynamic hardware multitasking use a large static on-chip region [1]–[3], with predefined and often limited regions for partial reconfigurable modules (PRMs). However, the architecture of Xilinx Virtex FPGAs allows PRMs to be placed arbitrarily on chip. Therefore, by logically mapping the PRMs onto the chip, not only a considerable number of CLBs could be more efficiently used, but also some degree of fault tolerance could be achieved as emerging damaged areas could be avoided on-the-fly. Towards these goals, we proposed a reliable, reconfigurable real-time operating system (R3TOS) in [4] as a systematic solution to deal with dynamic hardware multitasking. In it, a task scheduler and allocator form the kernels for PRM management.

One of the barriers for easy module placement and replacement on-chip is intermodule communication routing, the details of which are proprietary and not open to the public. One way to circumvent this problem is to harness the internal configuration access port (ICAP) for intermodule communication. Since the throughput of ICAP in Virtex family has increased to 100 MHz with a 32 bits wordlength, the ICAP could now be considered a reasonable medium for cross-chip communication [5], [6]. In order to read the output of a PRM, the PRM only needs to buffer its output to an adjacent BRAM. The ICAP will then perform a "read back" to fetch the output data from that BRAM. Similarly, to perform a writing instruction, the ICAP writes reconfiguration frames to the PRMs' corresponding input-BRAM. This approach has been successfully used to eliminate intermodule communication routing and hence avoid the use of resource-hungry static BusMacros used in traditional partial reconfiguration flows. Position-independent PRMs or hardware tasks could thus be separately loaded anywhere in the reconfigurable region on-chip without much regard to communication routing.

With such approach, 2-D-packing algorithms could be applied to define PRM locations on the fly. Algorithms using the KAMER-based "maximum empty rectangular" [7] and the "vertex list set (VLS)" [8] dominate other state-of-Art 2-D-packing algorithms, in their lucid explanation and satisfactory simulation results. However, both of these lack the ability to cope with emerging faults. KAMER's "Partition Tree" structure collapses as soon as a fault is detected in "overlapping" partitions; while VLS only records the adjacency of task edges therefore cannot detect faults emerging between edges.

To circumvent these two obstacles, we present a novel placement algorithm [called best-fit empty area compact (BF-EAC)] and an implementation of it on Xilinx Embedded soft core PicoBlaze. Simulations show performance advantages in comparison with the other two algorithms. The resulting implementation confirms the feasibility of this heuristic in practice. A task scheduler implementation based on the early deadline first (EDF) heuristic is also presented in this letter. The scheduler passes appropriate PRMs to the allocator according to their deadlines, and update tasks' information in a pipelined fashion with other operating system (OS) processes.

The remainder of this letter is organized as follows. First, Section II will present the modeling context. Then, Section III presents the novel BF-EAC algorithm as well as the EDF-based scheduler and its interaction with the placement algorithm. After that, Section IV presents a Virtex-4 based implementation of an overall hardware architecture which includes the BF-EAC allocator, the EDF-based scheduler, a configuration controller, and efficient memory architecture for task configurations. Finally, conclusions are drawn.

II. Context Modeling

A hardware task or PRM can be characterized in both time and area domains as illustrated in Fig. 1.

Manuscript received June 30, 2011; accepted August 20, 2011. Date of publication September 12, 2011; date of current version October 28, 2011. This manuscript was recommended for publication by R. Kastner.

The authors are with the School of Engineering, System Level Integration Group, University of Edinburgh, EH9 3JL, U.K. (e-mail: C.Hong@ed.ac.uk; k.benkrid@ed.ac.uk; x.iturbe@ed.ac.uk; a.ebrahim@ed.ac.uk; t.arslan@ed.ac.uk).

Digital Object Identifier 10.1109/LES.2011.2167737

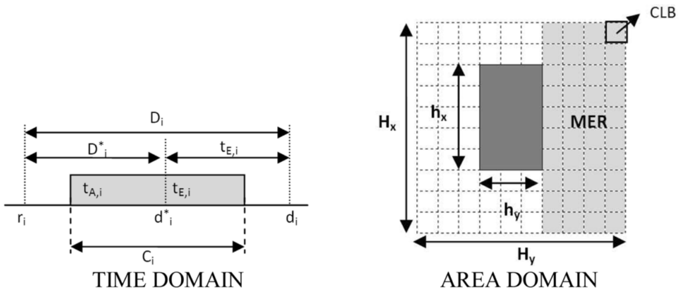


Fig. 1. Hardware task and FPGA modeling in time and area domain.

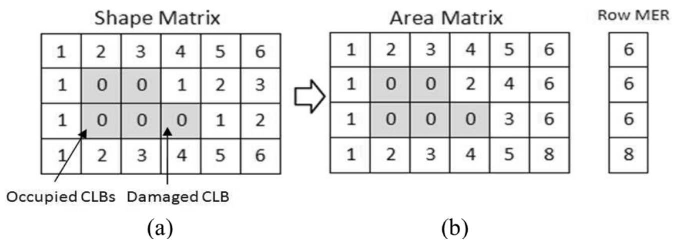


Fig. 2. EAD's two matrixes.

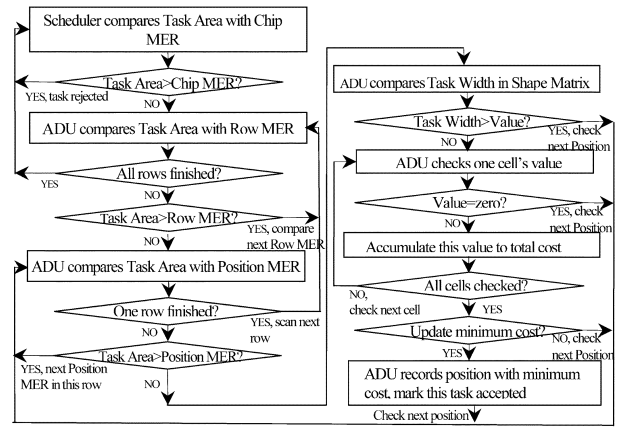
Dealing only with homogeneous resources, a chip or a reconfigurable area is modeled as a rectangle composed of $H_x \times H_y$ configurable logic blocks (CLBs). Similarly a task θ_i can be described as a $h_x \times h_y$ rectangle, where h is a multiple number of CLBs. The maximum empty area (MER) is defined as the maximum number of adjacent empty CLBs in a rectangular area. In the time domain, a hardware task θ_i has its computing time C_i, which is the sum of the allocation (or configuration) time $t_{A,i}$ which is proportional to the task area, and the Execution Time $t_{E,I}$ which is function dependant. The task's relative deadline D_i (see Fig. 1) implies the maximum relative affordable elapsing time before task completion, whereas D_i^* refers to the maximum relative affordable time before θ_i starts executing. Assuming a task arrival time r_i, the task's absolute deadlines d_i and d_i^* could be deduced.

In this context, the scheduler manages tasks in the time domain whereas the area domain is supervised by the allocator. The allocator can be conceptually divided into two components: 1) an empty area descriptor (EAD) which uses matrices to represent the FPGAs' CLBs array, it tracks the occupation/damaged state of FPGA CLBs by consecutively updating these matrices; and 2) the allocation decision unit (ADU) which consults the EAD to determine a position to load emerging hardware tasks or PRMs.

III. Software Approach: Allocator, Scheduler

A. Novel Best-Fit Empty Area Compact (BF-EAC) Algorithm

The EAD evaluates the chip area as a rectilinear grid formed by cell arrays. It uses two matrixes to represent cell values. The first matrix is named "shape matrix" (SM), in which the damaged or occupied CLBs are labeled with zeros. The other cells are labeled with sequential values starting from one and incrementing horizontally [see Fig. 2(a)]. From the SM, the "area matrix" (AM) is derived [see Fig. 2(b)]. Each value in AM's cell arrays is called "Position MER," which represents the MER value in its bottom-right corner, e.g., the dash encircled MER's area in Fig. 2(b) is six, accordingly the value of the cell (2,6) (MER's bottom-right vertex) is six. This indicates that this is an invalid position for tasks with area larger than six. Furthermore, the greatest Position MER in one row is selected as the Row MER and largest Row MER represents the Chip MER. To accelerate the computations needed to find valid positions, task parameters are examined through a series of thresholds as illustrated in Fig. 3.



Scheduler compares Task Area with Chip MER
Task Area>Chip MER?
YES, task rejected
NO
ADU compares Task Area with Row MER
All rows finished?
YES
NO
Task Area>Row MER?
YES, compare next Row MER
NO
ADU compares Task Area with Position MER
One row finished?
YES, scan next row
NO
Task Area>Position MER?
YES, next Position MER in this row
NO
ADU compares Task Width in Shape Matrix
Task Width>Value?
YES, check next Position
NO
ADU checks one cell's value
Value=zero?
YES, check next Position
NO
Accumulate this value to total cost
All cells checked?
NO, check next cell
YES
Update minimum cost?
NO, check next Position
YES
ADU records position with minimum cost, mark this task accepted
Check next position

Fig. 3. Steps to filter in a valid location.

TABLE I
Task Sets Parameter

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6
$h_{x,max}$	10	10	20	20	40	40
$h_{x,min}$	1	5	1	5	1	5
$h_{y,max}$	10	10	20	20	40	40
$h_{y,min}$	1	5	1	5	1	5
$H_x \times H_y$	100×100 CLBs					
n_{task}	500 tasks					

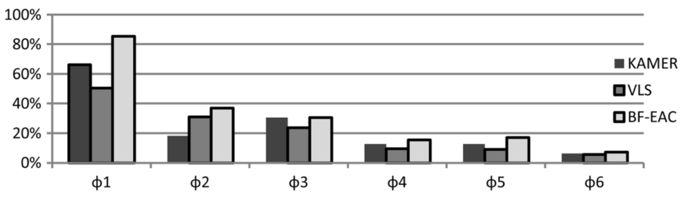


Fig. 4. Task Acceptance Rate.

If a position (x, y) is determined to be valid, i.e., the task's bottom-right vertex is mapped to (x, y), its cost will be the sum of all the position MERs covered by this task

$$\text{Cost}(x, y) = \sum_{i=x-h_x, j=y-h_y}^{i=x, j=y} (\text{Position MER}(i, j))$$

For example, for a task (2×3) placed at position (2,6) in Fig. 2(b), the cost will be 27. The least cost consumed, the more compact area would be retained. All valid positions' costs are calculated. Then the position with the minimum cost is certified as best-fit. Note that to accelerate the allocation time, we can choose the first valid position as the allocation position, the so-called first-fit (FF) approach. This bypasses the cost calculation for all valid positions.

A software simulation of the BF-EAC algorithm was conducted and compared with the KRAMER and VLS algorithms, under the situation where tasks have infinite execution times and deadlines. This shows the placement algorithm efficacy without time constraints.

Table I depicts six task-sets with various task sizes. Fig. 4 shows the corresponding acceptance results, indicating that the BF-EAC outperforms the other two algorithms by up to 25%.

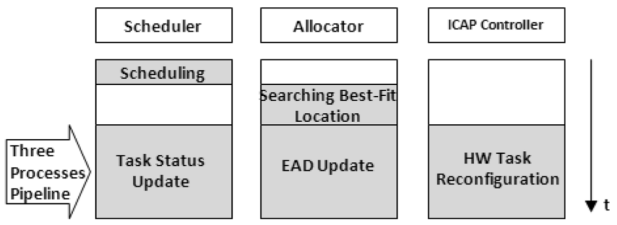


Fig. 5. OS pipeline.

The derivation of the area matrix (AM) is illustrated by the following pseudo-code:

```
1: PROCESS Updating Area Matrix
2: WHILE ⟨ not all cells updated ⟩
3:   $width\_factor = SM[i][j]$;
4:   FOR($i2 = i, i2 >= 0, i2 = i2 - 1$)
5:     $counter = 1$;
6:     IF($width\_factor > SM[i2][j]$)
7:       $width\_factor = SM[i2][j]$;
8:     ENDIF
9:     FOR($i3 = i2, i3 >= 0, i3 = i3 - 1$)
10:      IF($SM[i3][j] >= SM[i2][j]$)
11:        $counter++$;
12:      ELSE
13:        break;
14:      ENDIF
15:    ENDFOR
16:    FOR($i3 = i2, i3 <= i, i3 = i3 + 1$)
17:      IF($SM[i3][j] >= SM[i2][j]$)
18:        $counter++$;
19:      ELSE
20:        break;
21:      ENDIF
22:    ENDFOR
23:    $temp\_MER[i][j] = width\_factor \times counter$
24:    IF($temp\_MER[i][j] > MER[i][j]$)
25:      $MER[i][j] = temp\_MER[i][j]$;
26:    ENDIF
27:  ENDFOR
28:  $AM[i][j] = MER[i][j]$;
29: ENDWHILE
```

In order to update the *Position MER* $[i][j]$ (line 3–28), cells within the same column but above [i][j] are analyzed (line 4-27). Every cell being analyzed keeps a counter and a temp_MER value. The counter increases in two directions [upwards (line 9–15) and downwards (line 16–22)], when an equal or larger value is scanned. temp_MER is obtained by multiplying the cell's corresponding counter with its width_factor. The largest temp_MER is elected to be the Position MER[i][j].

The time complexity of updating AM is $O(n_{\text{cells}}^3)$. However, since the time complexity of searching the Best-Fit position is $O(n_{\text{cells}})$, the overall efficiency can be improved by pipelining as illustrated in Fig. 5.

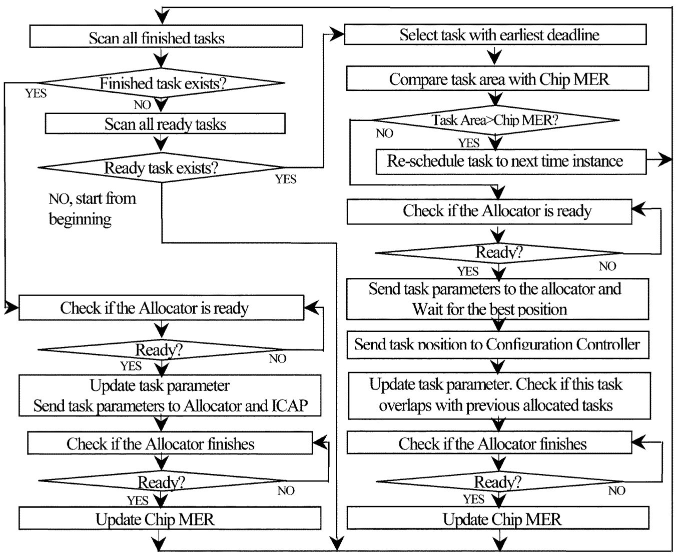

Scan all finished tasks
Finished task exists?
YES
NO
Scan all ready tasks
Ready task exists?
YES
NO, start from beginning
Check if the Allocator is ready
Ready?
YES
NO
Update task parameter
Send task parameters to Allocator and ICAP
Check if the Allocator finishes
Ready?
YES
NO
Update Chip MER
Select task with earliest deadline
Compare task area with Chip MER
Task Area>Chip MER?
NO
YES
Re-schedule task to next time instance
Check if the Allocator is ready
Ready?
YES
NO
Send task parameters to the allocator and Wait for the best position
Send task position to Configuration Controller
Update task parameter. Check if this task overlaps with previous allocated tasks
Check if the Allocator finishes
Ready?
YES
NO
Update Chip MER

Fig. 6. Scheduler processing.

TABLE II
TASK PARAMETERS MAPPING

Byte1:	Task ID	Byte6:	Absolute Deadline
Byte2:	Task Width	Byte7:	Task Status
Byte3:	Task Height	Byte8:	Task area
Byte4:	Task allocated position	Byte9:	Next Task ID
Byte5:	Execution/Finishing time	...	...

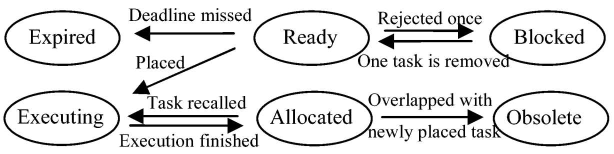


Fig. 7. Task status conversion.

B. Scheduler's Heuristic

The scheduler operates at the top level. The EDF algorithm (see Fig. 6) is used to schedule the task with the earliest deadline first. The scheduler manages task information by accessing a Task BRAM, where eight parameters for a single task are mapped in bytes (see Table II).

The conversion of the task status is a fundamental role of the scheduler (see Fig. 7). Once a task is rejected (space limited chip), it will be blocked and not attempted until a task is removed from the chip (i.e., more resources are available). Finished tasks remain allocated on-chip, ready to be reexecuted until a new task occupies its space.

IV. ON-CHIP IMPLEMENTATION

A. Architecture and Functioning

The overall architecture is illustrated in Fig. 8. Both the scheduler and the allocator are implemented on an 8-bit Xilinx PicoBlaze soft core. The scheduler masters a 2-bit Address Bus to switch communications between the configuration controller and the allocator. An 8-bit Control Bus is used for the synchronization of the processing status. Task parameters

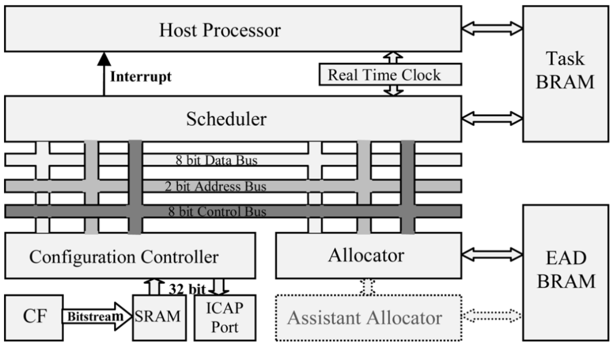


Fig. 8. Overall system architecture.

TABLE III
NEW ICAP TIMING MEASUREMENT

Process (System Clock 100MHz)	Time spent on one frame (41words)
CF card to SRAM (one time only)	1101 µs
SRAM to ICAP FIFO (burst read)	1.72 µs
ICAP FIFO to Configuration Memory (including initialization overhead)	1.4 µs

are transparent to both scheduler and host processor by sharing its two channels. Because of the dual-port nature of BRAMs, the EAD BRAM can be optionally accessed by two allocator instances simultaneously, hence accelerating the allocation time by up to 2×.

Instead of using the built-in HWICAP IP core provided by Xilinx, we developed a new ICAP controller with simplified bus interface and infrastructure, which extends the ICAP throughput beyond its maximum "official" speed of 100 MWord/s. Since the BRAM (16 kbit) is insufficient to store bitstreams of a generic size PRM (>50 kByte), an external memory SRAM is used to hold task configuration bitstreams at run time. In effect, when the board is booted up, the ICAP controller is initially configured as a DMA engine. The DMA transfers bitstream from an external nonvolatile memory (e.g., compact flash memory) to SRAM. Afterwards, the DMA's hardware mechanism is replaced by the SRAM interface. Therefore the painful delay in accessing the Flash memory is eliminated, and the configuration data could be rapidly fetched from SRAM. Table III depicts the data transfer throughput of the new ICAP Controller.

In order to configure tasks to a defined position, the new ICAP Controller is also responsible for changing the Frame Address Register (FAR), which is embedded in the bitstream to represent tasks' on-chip location.

B. Implementation Results

The above architecture was implemented on Xilinx ML403 board with a Virtex-4 XC4VFX12 FPGA chip, running at 100 MHz. Table IV gives out an analysis of the allocator algorithm execution time with various chip granularities. The algorithm execution time increases linearly with the granularity by a factor of three. The granularity could be parameterized to trade off the algorithm accuracy and execution time, depending on system constraints.

TABLE IV
ALLOCATOR EXECUTION TIME BREAK-DOWN

Chip granularity	8×8	10×10	16×12
Finding Best Location Average Time	41 µs	118 µs	465 µs
Insert Task Updating Average Time	232 µs	674 µs	2191 µs
Remove Task Updating Average Time	210 µs	630 µs	1866 µs
Finding First Location Average Time (FF)	3.5 µs	4.25 µs	9.8 µs

TABLE V
SYSTEM RESOURCE CONSUMPTION

Resource	Allocator	Scheduler	ICAP Controller	Total
Slices	96	138	93	430
RAMB16s	2	2	1	5

Benefiting from the PicoBlaze's light footprint, the whole architecture consumes only 430 slices (see Table V) which represents less than 8% of the total Virtex-4 XC4VFX12 FPGA chip slice count.

V. CONCLUSION

In this letter, we presented the design and implementation of an efficient hardware task allocator and scheduler, which form the kernel of a reconfigurable operating system, which promises to achieve hitherto unattainable levels of flexibility and fault-tolerance. A novel BF-EAC placement algorithm was presented and implemented efficiently on Xilinx Virtex-4 FPGAs. Together with an EDF-based scheduler, a configuration controller and optimized memory architecture for task configurations, the overall system produces satisfactory configuration speeds (in the μs range) and a lightweight footprint (less than 500 slices) paving the way for future high performance, autonomous and fault-tolerant systems.

REFERENCES

[1] E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in *Proc. 39th Annu. Design Autom. Conf.*, New Orleans, LA, 2002, pp. 343–348.
[2] B. Blodget, S. McMillan, and P. Lysaght, "A lightweight approach for embedded reconfiguration of FPGAs," in *Proc. Design, Autom. Test Eur. Conf. Exhibition*, Munich, Germany, 2003, pp. 399–400.
[3] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, and T. Becker, "Modular partial reconfiguration in Virtex FPGAs," in *Proc. Int. Conf. Field-Program. Logic and Appl.*, Tampere, Finland, 2005, pp. 211–216.
[4] X. Iturbe, K. Benkrid, A. T. Erdogan, T. Arslan, M. Azkarate, I. Martinez, and A. Perez, "R3TOS: A reliable reconfigurable real-time operating system," in *Proc. NASA/ESA Conf. Adapt. Hardware Syst.*, Anaheim, CA, 2010, pp. 99–104.
[5] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, "Metawire: Using FPGA configuration circuitry to emulate a network-on-chip," *IET*, vol. 4, pp. 159–169, 2010.
[6] X. Iturbe, K. Benkrid, T. Arslan, R. Torrego, and I. Martinez, "Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs," in *Proc. Int. Conf. Field-Program. Logic Appl.*, Crete, Greece, 2011.
[7] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *Design & Test Comput.*, vol. 17, pp. 68–83, 2000.
[8] J. Tabero, J. Septi'en, H. Mecha, and D. Mozos, "A low fragmentation heuristic for task placement in 2D RTR HW management," in *Proc. Int. Conf. Field-Program. Logic Appl.*, Leuven, Belgium, 2004, pp. 241–250.