An Effective Design System for Dynamically Reconfigurable Architectures *

Sriram Govindarajan, Iyad Ouaiss, Meenakshi Kaul, Vinoo Srinivasan, and Ranga Vemuri
University of Cincinnati, Cincinnati, OH 45221-0030. Email: `ranga.vemuri@uc.edu`

Abstract: *The* SPARCS *system is an integrated partitioning and synthesis environment for reconfigurable architectures. In this paper, we use the Joint Photographic Experts Group (*JPEG*) image compression algorithm as a design example to demonstrate the effectiveness of dynamic reconfiguration achieved using* SPARCS*. We present a typical design process using the* SPARCS *system consisting of temporal partitioning, spatial partitioning, and design synthesis. The results, obtained on a commercial* RC *architecture, show that the multiply-reconfigured version of the* JPEG *compression algorithm achieves reasonable improvement in execution times compared to the one-time configured version.*

1 Introduction

With the advancement of the field-programmable device technology, the Reconfigurable Computer (RC) that consists of a multi-FPGA board with memory banks and interconnection fabric, is being widely used for realizing fast implementation of wide classes of algorithms. Over the last couple of years, there have been research efforts [1, 2] toward design automation techniques for *dynamic reconfiguration* of the RC, leading to better performance and cost advantages.

The SPARCS (Synthesis and Partitioning for Adaptive Reconfigurable Computing Systems) system [2] is a prototype design environment that has an integrated collection of algorithms (Figure 1) for dynamic reconfiguration of a wide class of RC architectures. The SPARCS system consists of a *Temporal Partitioner* based on a non-linear programming model, a Genetic Algorithm based *Spatial Partitioner*, and a *High-Level Synthesis* tool integrated with a Floorplanner. The reader may refer [2] for a detailed study of SPARCS.

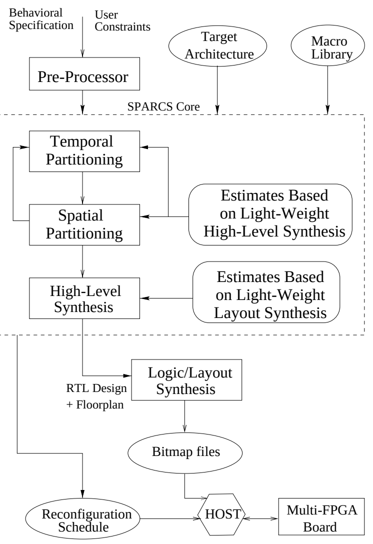


Figure 1: SPARCS System

*This work is supported in part by the US Air Force, Wright Laboratory, WPAFB, under contract number F33615-97-C-1043.

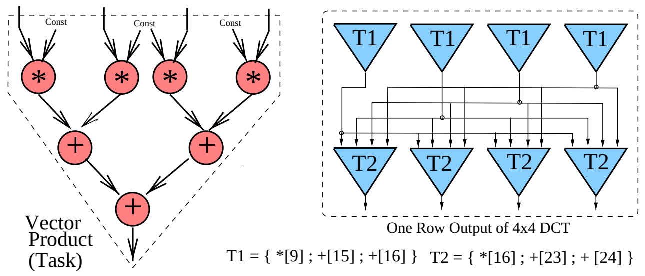

Const
Const
Const
Vector Product (Task)
T1
T2
One Row Output of 4x4 DCT
T1 = { *[9] ; +[15] ; +[16] } T2 = { *[16] ; +[23] ; + [24] }

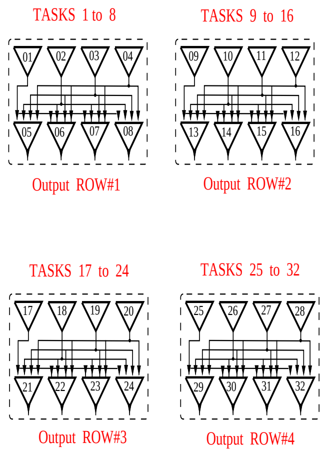
Figure 2: Task Partitions for 4x4 DCT

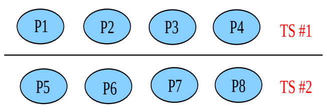
2 The Design Process

We modeled the JPEG image compression algorithm [3] as a *Hardware-Software Codesign*, where the Discrete Cosine Transform (DCT) being the most computationally intense subtask, was implemented in hardware, and the rest of the JPEG subtasks (Quantization, Zig-Zag, and Huffman Encoding) were in software. The RC considered, the *Wildforce board from Annapolis Micro Systems Inc.*, consists of four Xilinx XC4013 FPGAs having 576 CLBs each and four 32K memory banks with a 32-bit word. The host computer was a Pentium PC with a 200MHz processor. The host can communicate to the RC by reading/writing data on the board memory, using a simple handshaking protocol through the PCI bus.

As the first step, the behavioral specification of the 4x4 DCT was modeled in the form of 32 vector products (16 for each matrix multiplication) and simulated in VHDL. The entire DCT dataflow graph was then partitioned into a collection of 32 tasks, where each task is a vector product, represented by a cone in Figure 2. A collection of 8 tasks, classified into two types T1 and T2, forming a row of the 4x4 output matrix is also shown in Figure 2. The entire DCT task graph is shown in Figure 3. While deciding on task boundaries, we ensured that each task did not exceed an area of 576 CLBs for the XC4013. This is because the SPARCS system requires that each task should individually fit on the FPGA. Another goal while deciding the task boundaries was to minimize the number of memory read/write operations for the task I/O.

Given the DCT task graph and the RC architecture constraints, the SPARCS temporal partitioner produced two temporal configurations (temporal segments TS#1 and TS#2) with 16 tasks in each, as shown in Figure 3. Consider the temporal partitioning of the 8 tasks shown in Figure 2. Under the given area constraints, only 4 tasks can fit in any FPGA. Taking a closer look at the intercon-





Partition	Set of Tasks	CLB's	Nsecs
P1	{ 01, 09, 17, 25 }	267	52.5
P2	{ 02, 10, 18, 26 }	233	53.2
P3	{ 03, 11, 19, 27 }	240	53.4
P4	{ 04, 12, 20, 28 }	233	57.3
P5	{ 05, 06, 07, 08 }	408	72.3
P6	{ 13, 14, 15, 16 }	349	69.1
P7	{ 21, 22, 23, 24 }	364	70.9
P8	{ 29, 30, 31, 32 }	349	60

Figure 3: Temporal and Spatial Partitions

nection between these tasks, partitioning the top four and the bottom four into two groups leads to minimal communication between these groups. Partitioning in this fashion involves storing only four intermediate results and no other data transfers.

The SPARCS spatial partitioner produced four spatial partitions in each temporal segment, as shown in Figure 3. The table shown in Figure 3 lists the collection of tasks in each of these eight spatial partitions. The first four spatial partitions, P1 - P4, consist of the top sixteen DCT tasks and the next four spatial partitions, P5 - P8, consist of the bottom sixteen DCT tasks. The spatial partitions P1 - P4 were intelligently created so as to minimize the amount of hardware required to implement all tasks in a partition. Spatial partitions P5 - P8 were also efficiently created so that each spatial partition in itself requires only four common inputs and produces an entire row of the output matrix.

Thus the spatial and temporal partitioning were done so as to minimize the amount of data transfer through the memory and the interconnect. Following spatial partitioning, RTL designs for the partition segments were synthesized using the SPARCS' HLS tool. The RTL designs were then taken through logic synthesis (Synplify tool from Synplicity Inc.) and layout synthesis (Xilinx M1 tools) to produce the FPGA configuration files. For the eight spatial partitions, P1 - P8, the table in Figure 3 shows the area estimates provided by the Xilinx PAR (Partitioning And Routing) tool, timing estimates provided by the Xilinx TRACE (Timing Analyzer) tool.

3 Experimental Results

Two codesign versions of the JPEG compression algorithm were developed: *static* and *dynamic*. In both cases, DCT was implemented on the RC hardware and the rest of the subtasks in software. For the static-JPEG version, the board was configured only once. In the dynamic-JPEG version, multiple temporal configurations for DCT were generated using SPARCS. The host PC automatically downloads the configurations one at a time. Each configuration performs a partial computation on the entire image and communicates intermediate results through the on-board memory.

Image	# of 4x4 blocks	DCT Exec. time per 4x4 block in μsec		Improv. %
		Static	Dynamic	
Parrots	245,760	8.61	5.16	40.07
Rabbit	194,400	9.13	6.19	32.2
Grapes	172,800	9.43	6.81	27.8
Scenery	148,200	9.89	7.73	21.84
Group	120,000	10.66	9.25	13.20
XV	90,300	11.97	11.88	0.752

Table 1: **Average Execution Times**

Both JPEG versions were tested on the six image files of varying sizes as shown in Table 1. The images are listed in the decreasing order of their sizes. The table shows the average time spent by both DCT versions on each 4x4 block of an image. We measured the execution times by inserting probes in the software code at points where the RC board was invoked to execute the DCT subtask. The dynamically reconfigured DCT version shows up to 40% improvement when compared to the static version, for the larger images. The improvement factor will be much higher (orders of magnitude) if 8x8 DCT was used, since the amount of computation performed on hardware is much more than the available area. In Table 1, as the image size decreases, the average execution time for the dynamically reconfigured DCT increases and consequently the improvement factor decreases. This is because *for smaller images, the reconfiguration overhead defeats the gain obtained due to dynamic reconfiguration.*

4 Conclusion

In this paper we have presented results that demonstrate: (1) A well-defined design flow, such as the one in the SPARCS system, can provide a realistic design environment for dynamically reconfigurable architectures. (2) Dynamic reconfiguration provides performance/cost advantages over static configuration for computationally intense applications that demand high-performance low cost implementations.

References

[1] M. Vasilko and D. Ait-Boudaoud, "Architectural Synthesis Techniques for Dynamically Reconfigurable Logic", *FPL'96.*

[2] I. Ouaiss, S. Govindarajan, V. Srinivasan, M. Kaul, and R. Vemuri, "An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures", *Fifth Reconfigurable Architectures Workshop*, 1998.

[3] G. K. Wallace, "The JPEG Still Picture Compression Standard", *ACM Communications*, pg. 30-44, 1991.