

Register File Architecture Optimization in a Coarse-Grained Reconfigurable Architecture

Zion Kwok, Steven J. E. Wilton

Dept. of Electrical and Computer Engineering
University of British Columbia
Vancouver, B.C., Canada

Abstract. This paper investigates the impact of the local and global register file architecture on a reconfigurable system based on the ADRES architecture [3]. The register files consume a significant amount of area on the reconfigurable device, and their architecture has a strong impact on the performance. We found that the global registers should be tightly connected to as many functional units as possible, while the connection of the local register files to their neighbours is less critical. We found that the global register file should contain between 12 and 16 registers, while each local register file should only contain one or two registers. We used these results to propose a new architecture that has between 60% and 95% higher performance per unit area compared to the original architecture over the set of benchmarks.

1 Introduction

Coarse-grained reconfigurable architectures promise high computing parallelism and low power consumption. Published architectures such as MorphoSys [1], PipeRench [2], and ADRES [3] demonstrate higher computing performance than general-purpose processors and VLIW processors, especially in loop-intensive DSP functions. Coarse-grained architectures are more suitable for applications requiring datapaths than FPGAs, which are more effective for random logic [2]. Compared to fine-grained architectures and FPGAs in particular, coarse-grained architectures have larger reconfigurable functional units which require fewer programming bits and routing resources within each block, leading to power, area and delay reductions inside each functional unit. These power savings are important, especially in mobile systems.

As in any compute engine, the memory architecture has a significant effect on the performance of coarse-grained reconfigurable architectures [4]. MorphoSys [1] uses a frame buffer to stream image data. In REMARC [5], the MIPS processor loads the memory into the data register for the co-processor to access. The instruction memory is also significant; instructions can be stored in instruction registers, in context registers, or in RAM [5-9]. Furthermore, many coarse-grained architectures contain additional register files within the reconfigurable fabric for use as scratch-pad memory [1,5-14].

The area devoted to memory in a reconfigurable system is significant. Estimates in this paper show that approximately 62% of the area of a representative reconfigurable processor

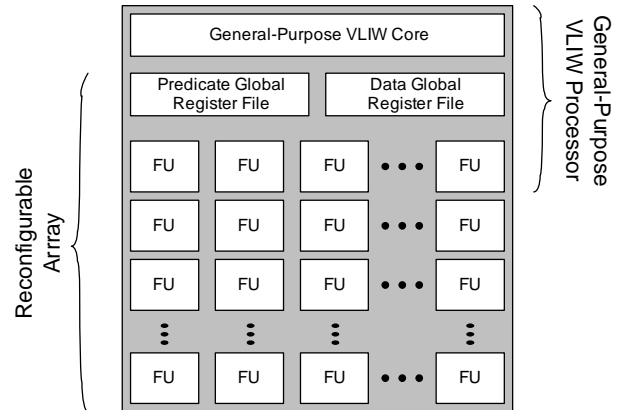


Figure 1: Baseline Architecture Based on ADRES [3]

(the ADRES architecture [3]) is memory. Of this, just under half is devoted to register files used as scratch-pad memory and used to communicate between the host processor and the reconfigurable fabric. Measurements in this paper will show that this register file architecture has a significant impact on the performance of the system. Clearly, a well designed reconfigurable processor needs a well-optimized register file architecture.

This paper presents an experimental analysis of the register file architecture in one representative reconfigurable processor, the ADRES architecture from IMEC [3]. This architecture contains two types of register files: global register files that can be accessed by some (or all) of the reconfigurable functional units on the fabric, and local register files, each of which is associated with a specific reconfigurable functional unit. In particular, we ask three questions for each type of register file:

- 1) How big should each register file be?
- 2) How should the register file be connected to the rest of the fabric?
- 3) How many read and write ports should each register file have?

We will then use these answers to construct a modified version of the ADRES architecture which has 17% to 27% smaller area and runs code between 32% and 42% faster than the original architecture.

Although the numerical results are specific to this processor, the trends that we observe may apply to many other reconfigurable systems.

The paper is organized as follows. In Section 2, we will describe our baseline architecture, which is very similar to the ADRES architecture [3]. Section 3 will describe our experimental methodology and Section 4 will present experimental results illustrating the impact of various architectural parameters on both the area and performance of the device. Finally, Section 5 will use the results from Section 4 to propose two new architectures that are significantly smaller and faster than the baseline architecture.

2 Architectural Framework

In this section, we first describe the baseline architecture. We then indicate which architecture parameters we investigate in this paper.

a) Overall Architecture

The baseline architecture is based on the ADRES architecture described in [3,15]. As shown in Figure 1, this architecture consists of a reconfigurable array coupled with a general-purpose VLIW processor. Highly pipeline-able loops are identified by a compiler [16,17] and executed on the reconfigurable array, and sequential code is executed by the processor. The processor and the reconfigurable unit communicate via two global register files.

The reconfigurable fabric consists of a heterogeneous array of configurable functional units (FU's). In this paper, we will consider two array sizes: 4x4 and 8x8. Each FU can perform a subset of forty-five 32-bit operations in each cycle. All FU's can perform a variety of arithmetic (signed and unsigned), logic, shifting, and comparison instructions. One out of every four columns has FU's that can also perform multiplication.

The FU's in the top row are different than the others. These FU's perform two tasks: they make up the functional units of the general-purpose VLIW processor when this processor is executing code, and act as additional configurable FU's when the device is executing parallel code on the reconfigurable array. These FU's have direct connections to the two global register files through dedicated read and write ports.

Figure 2 shows a single configurable functional unit (FU). Each FU contains both a configurable ALU (the function of which can be specified using context memories, analogous to the configuration memories in an FPGA or the instruction memory in a processor). Each FU also contains two local register files; these register files will be described in more detail below.

The FU's are interconnected using the "closest" architecture as described in [15]. In this architecture, each of the two data ALU inputs is driven by registered versions of four neighbouring data FU outputs including the data output from the local FU. The single predicate ALU input is driven by registered versions of seven neighbouring predicate FU outputs including the predicate output from the local FU.

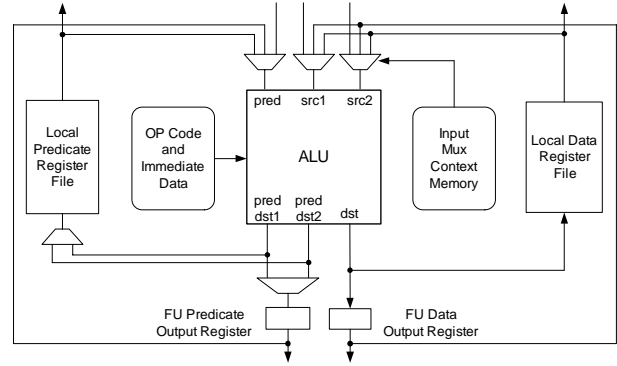


Figure 2: Configurable Functional Unit in the ADRES Architecture

b) Register Files

As described above, this architecture has two types of register files: two local register files within each FU, and two global register files.

First consider the two global register files. These register files are used for two purposes: (1) to transfer data between the general-purpose VLIW processor and the reconfigurable array, and (2) as scratch-pad memory for the functional units in the top row of the reconfigurable array. There is one global register file (32 bits wide) to store data values, and one global register file (1 bit wide) to store predicate values.

As described above, in the baseline architecture, each data global register file is connected to each FU in the top row through two dedicated read ports, one for each data ALU input, and one dedicated write port. Thus, in a 4x4 array, each data global register file has eight read ports and four write ports; in an 8x8 array, each data global register file has sixteen read ports and eight write ports.

Each predicate global register file is connected to the FU's in the top row through one dedicated read port and one dedicated write port. Thus, in a 4x4 array, each predicate global register file has four read ports and four write ports; in an 8x8 array, each predicate global register file has eight read ports and eight write ports.

There are also two local register files within each functional unit. These register files can be used as scratch-pad memory by the functional units in the reconfigurable array, and can be used to transmit data between functional units. Each functional unit contains one 32-bit wide register file to store data values, and one 1-bit wide register file to store predicate values.

In the baseline architecture, each local register file has one read port and one write port. As shown in Figure 2, the write port of each local register file is driven by the unregistered output of the ALU within the local FU. The output of each local register file, however, is connected to six sinks: the two inputs of the ALU in the local FU, and one input in each of four neighbouring FU's (the four FU's diagonally adjacent to the local FU).

Table 1: Typical Contributions of Various Memory Resources to Chip Area

Type of Memory		Area
Register Files	Global VLIW Data RF's	10%
	Global VLIW Predicate RF's	2%
	Local Data RF's	13%
	Local Predicate RF's	3%
FU Output Registers	FU Data Output Registers	3%
	FU Predicate Output Registers	1%
Context Memory	Input Mux Context Memory	4%
	Op Code Memory	3%
	Immediate Data Memory	23%
Total		62%

Table 2: Architectural Parameters Used in This Paper

Parameter		Value in Baseline Architecture	Range Considered in Section 4
Global Register Files	Number of FU's connected to each register file	4 (4x4 array)	4-16 (4x4 array)
		8 (8x8 array)	8-64 (8x8 array)
	Number of read and write ports per register file	4 (4x4 array)	4-16 (4x4 array)
		8 (8x8 array)	8-64 (8x8 array)
	Number of registers per register file	16 (4x4 array)	4-48 (4x4 array)
		64 (8x8 array)	8-48 (8x8 array)
Local Register Files	Number of FU's connected to each register file	5	1 to 5
	Number of input and output ports per register file	1	1 to 6
	Number of registers per register file	4	0 to 16

Table 1 shows a breakdown of the memory resources in the 4x4 baseline architecture. Both types of register files consume a significant amount of area.

Clearly, there is a tradeoff between the area devoted to global and local register files. The global register files are more widely connected to functional units (in the top row); however, the local register files provide additional storage throughout the reconfigurable array, increasing the potential usefulness of this array. In addition, there are tradeoffs between the connections between these register files and the compute elements. For example, the global register file can be connected to more than just the top row of FU's, possibly reducing the need for local register files. By answering the three questions set out in the introduction, this paper investigates these tradeoffs, and proposes two modified architectures that are smaller and faster than the original architectures based on ADRES.

Table 3: Description of Benchmark Kernels

Benchmark	Application	Loop Description
1	LDL	ldl
2	LDL	fnorm
3	FFT	radix4
4	IDCT 8x8	vertical
5	IDCT 8x8	horizontal
6	MPEG-2 Decoder	dequantize intra
7	MPEG-2 Decoder	dequantize non-intra
8	MPEG-2 Decoder	saturate
9	MPEG-2 Decoder	add block non-intra
10	MPEG-2 Decoder	add block intra
11	MPEG-2 Decoder	clear block
12	MPEG-2 Decoder	fast idct
13	MPEG-2 Decoder	form component prediction
14	MPEG-2 Decoder	form component prediction
15	MPEG-2 Decoder	form component prediction
16	MPEG-2 Decoder	form component prediction
17	MPEG-2 Decoder	form component prediction
18	MPEG-2 Decoder	form component prediction
19	MPEG-2 Decoder	form component prediction
20	MPEG-2 Decoder	form component prediction

c) Architectural Parameters

Table 2 shows a list of the architectural parameters that describe the register files in the baseline architecture for both a 4x4 array and an 8x8 array. In Section 4, one parameter is varied at a time, and the impact on area and performance is measured. The range investigated for each parameter is also shown in Table 2.

3 Experimental Methodology

In the next section, we will describe experiments aimed at determining the optimum values of various architectural parameters. This section describes the flow we used to obtain the results.

Figure 3 shows our experimental flow. Common DSP functions, including an FFT, an IDCT solver, and an MPEG-2 Decoder were used as benchmark programs. These programs were compiled using IMPACT [16] and loops with high amounts of loop-level parallelism were extracted. Twenty such loops were obtained; these loops were used as our benchmark workloads. These loops are listed in Table 3.

Each loop was mapped to each potential reconfigurable architecture using DRES [17]. The DRES compiler uses a modulo-scheduling algorithm to schedule each loop on the target architecture, assigning each operation in the loop to a specific FU and a specific time slice, and assigning each variable to one of the register files. The DRES tool takes into account the characteristics of the target architecture, including the capabilities of each FU, the sizes of each register file, and the interconnect between the FU's and the register files. This ensures that the resulting schedule can be implemented on the reconfigurable device. Using techniques from simulated annealing [18] and Pathfinder [19], the DRES tool performs the tasks of scheduling, placement, and routing simultaneously.

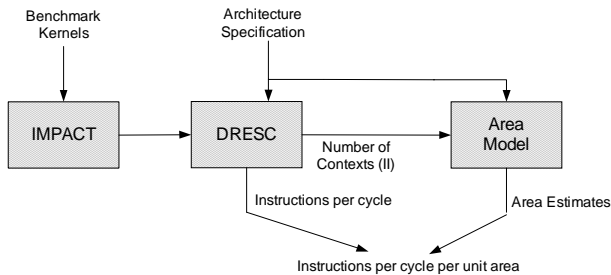


Figure 3: Experimental Flow

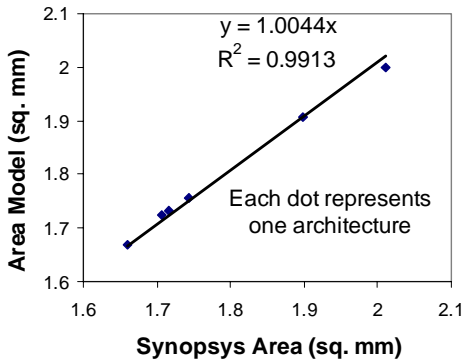


Figure 4: Validation of Area Model

During mapping, DRESC attempts to minimize the Iteration Interval (II) of the scheduled loop. The Iteration Interval is the number of clock cycles between the initiation of successive iterations of the loop. As explained in [17], the II is equal to the number of contexts required in the configuration RAM of the reconfigurable fabric. Therefore, the achievable II directly impacts the area of the architecture. Intuitively, if an architecture is more flexible (either because there are more registers or they are better located or connected to the computation units), the DRESC compiler will be able to find a schedule with a smaller value of II. On average, this means that if the fabric was constructed, a smaller number of contexts would be used, meaning a smaller area is required. As explained in [17], the value of II also has a strong influence on the achievable Instructions per Cycle (IPC) which dictates the overall performance of the device.

To estimate the area, we used a detailed area model. In [15], potential architectures were synthesized using Synopsys, and the area metric was obtained from this synthesized fabric. In this paper, we consider larger architectures, and thus the flat synthesis of these architectures is too time-consuming. Instead, we use the following approximation. For each component in the reconfigurable fabric (multiplexer, configuration memory, register, etc.) we created a parameterized model based on measurements obtained from Synopsys for a 0.18 μ m TSMC process. By adding the area contributions of each architectural component, we estimated the overall area of each potential architecture. To verify the accuracy of this approximation, we constructed and synthesized six full architectures as in [15]. In selecting these six architectures, we attempted to "span" the design space as much as possible. We then compared the areas obtained to the area estimates from our approximate scheme.

Table 4: Impact of Changing the Number of FU's that can Connect to the Global Register Files (4x4 Array)

Number of FU's	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
4 FU's (Baseline)	7.23	2.10	3.45
8 FU's	9.24	1.91	4.85
12 FU's	9.58	1.90	5.05
16 FU's	9.74	1.90	5.14

Table 5: Impact of Changing the Number of FU's that can Connect to the Global Register Files (8x8 Array)

Number of FU's	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
8 FU's (Baseline)	13.59	6.48	2.10
16 FU's	19.68	5.81	3.39
24 FU's	20.40	5.73	3.56
32 FU's	20.38	5.76	3.54
40 FU's	20.40	5.79	3.53
48 FU's	20.88	5.78	3.61
56 FU's	21.90	5.71	3.84
64 FU's	20.85	5.83	3.58

As shown in Figure 4, the area estimates correlate well with the synthesis results.

To quantify the performance of the device, we measured the achievable Instructions per Cycle (IPC) that could be achieved for each loop on each architecture. This does not take into account differences in the cycle time of each architecture (different architectural options might have slightly different cycle times). However, we believe that the delay of the multiplier in the ALU dominates the critical path delay by using an estimated 7.4 ns of 10.4 ns. Since we don't modify the ALU in this paper, we expect that all architectures investigated here have similar critical path delays.

Finally, we divide the IPC by the area estimate to compute the *number of instructions per cycle per unit area*. This is the metric we use to rank our architectures.

4 Results

In this section, we experimentally investigate the effects of changing the number of registers in each register file, the number of ports on each register file, and the manner in which the register files are connected to the logic, for both the global and local register files.

a) Global Register Files

This section presents results for the global register files.

Degree of Connectivity

We first consider the manner in which these register files are connected to the rest of the device. In the baseline architecture, the global register files are connected to the top row of FU's only. Each FU in this top row has dedicated read and write ports to each of the two global register files. Table 4 shows the performance and area measurements if we connect the register

files to more than just the top row (for a 4x4 array). In each case, the number of read and write ports on the global register file is held constant at four respectively (except there are twice as many read ports on the data global register file). Thus, as more FU's are connected to the register file, these FU's need to be multiplexed onto the same four read and write ports. In Table 4, if x FU's are connected to the register file, there are $x/4$ FU's multiplexed onto each port. As the table shows, increasing the reach of the global register file decreases the number of instructions per cycle required to complete the loop (averaged over all benchmark loops). This makes sense; the more paths there are between the global register files and the computing elements, the easier it is for DRESC to find an efficient schedule. The area required to implement the architecture goes down as the number of connections increases. As more connections are available, DRESC is able to find schedules which have a lower iteration interval (II), meaning less context memory is required (on average). This decrease in area is more significant than the small increase in size in the input multiplexers of the FU's. Combining the area and IPC results, it is clear that connecting the global register files to as many FU's as possible results in the best IPC per unit area (this ratio is recorded in the last column of Table 4).

We repeated the experiment for an 8x8 array and the results are in Table 5. In this case, the number of read and write ports in the each register file was held constant at eight (except for the data global register which has 16 read ports). This table shows the same trends, with the exception of the last entry, which corresponds to an architecture in which the global register files are connected to all 64 FU's. In this case, DRESC found schedules that are actually worse than the 56 FU case. We have observed that often the optimization algorithm within DRESC has problems dealing with architectures with too much flexibility; a better understanding of this is on-going work. However, with the exception of this point, the conclusion is the same: the more connected the global register file, the better.

Number of Ports

In the previous results, the number of read and write ports was held constant at four or eight. In this subsection, we allow the number of ports to float as we increase the connectivity of the global register files.

Table 6 shows the results for a 4x4 array. In these results, all FU's that are connected to the global register file have a set of

Table 6: Impact of Changing the Number of Ports on the Global Register Files (4x4 Array)

Number of Ports	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
4 (Baseline)	7.23	2.10	3.45
8	9.38	2.06	4.56
12	9.57	2.21	4.32
16	9.57	2.40	3.99

Table 7: Impact of Changing the Number of Ports on the Global Register Files (8x8 Array)

Number of Ports	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
8 (Baseline)	13.59	6.48	2.10
16	21.40	6.69	3.20
24	22.29	7.65	2.91
32	22.56	8.72	2.59
40	23.35	9.71	2.41
48	23.35	10.77	2.17
56	23.35	11.83	1.97
64	23.35	12.89	1.81

dedicated read and write ports. Compared to Table 4, we can see that there is very little increase in the achievable IPC (in fact, it is smaller in some cases, likely because of DRESC's difficulty dealing with very flexible architectures as described above). There is a significant area penalty, however, in increasing the number of ports. Therefore, we conclude that increasing the number of read/write ports to match the number of FU's connected to the global register file is not a good idea. Instead, sharing ports between multiple FU's results in a better performance per unit area. Table 7 shows the results for an 8x8 array; the same conclusions hold.

Register File Size

Figure 5 shows the impact of the number of registers in each global register file on the performance and area of a 4x4 array. For a very small number of registers, DRESC is unable to find efficient loop implementations, leading to low IPC's and many contexts (and hence large area). As the number of registers increases, the IPC increases; however, beyond 16, the increase is outweighed by the extra area required for the registers.

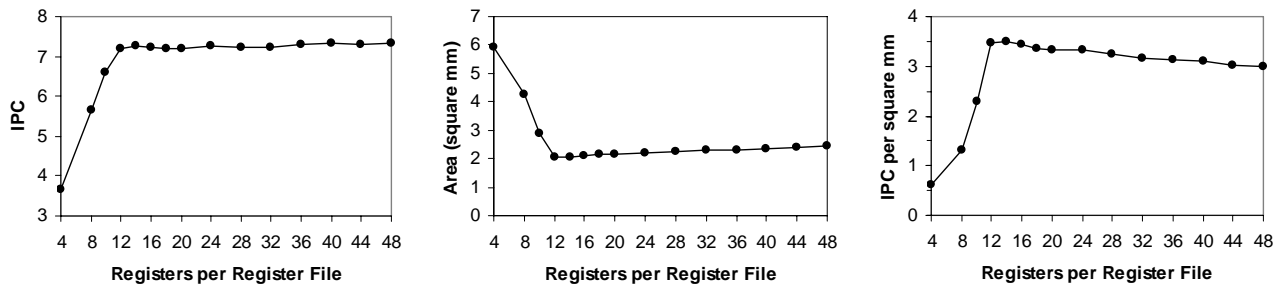


Figure 5: Impact of Changing the Number of Registers in Each Global Register File in the 4x4 Array

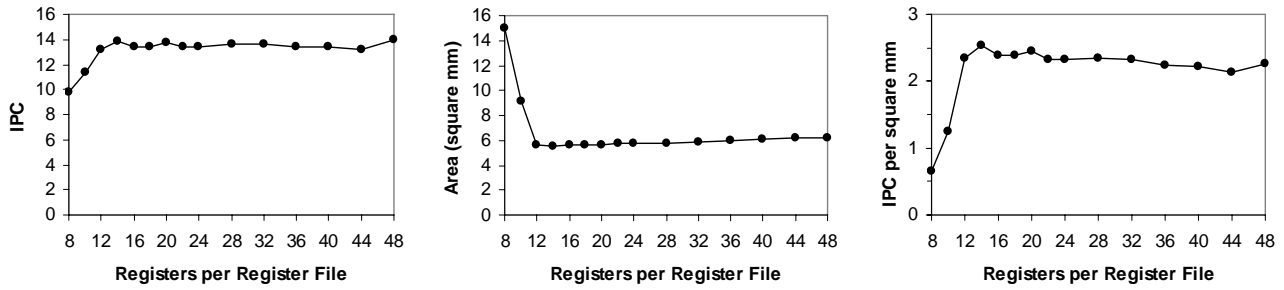


Figure 6: Impact of Changing the Number of Registers in Each Global Register File in the 8x8 Array

Figure 6 shows the results for an 8x8 array. In this case we see that 14 registers in each global register file are sufficient to achieve peak performance. Intuitively, we might have thought that a larger array would prefer a larger number of global registers than a smaller array, yet the best choice of register file size is the same in both cases. This may be because in the larger array, there are more FUs, and hence more local register files. The larger number of local register files may mean that smaller global register files are suitable.

b) Local Register Files

This section presents results for the local register files. As shown in Figure 2, there are two local register files per FU: one 32-bit wide register file to store data values; and one 1-bit wide register file to store predicate values.

Degree of Connectivity

As described in Section 2, in the baseline architecture, the local register files are connected to the computing resources as follows. The input of each local register file (within a FU) is driven by the unregistered output of the ALU within the local FU. The output of each local register file, however, is connected to six sinks: the two inputs of the ALU in the local FU, and one input in each of four neighbouring FU's (the four FU's diagonally adjacent to the local FU).

Table 8 shows the performance and area results for this baseline architecture and three other architectures. Architecture 1 is less connected than the baseline; in this architecture, each local register file drives *only* the two inputs to the local ALU; it does not drive ALU's in neighbouring FU's. In Architecture 2, the inputs of each local register file are connected to the output of the local ALU as well as the outputs of four neighbouring ALU's, while the output of the register file is connected to only the local FU. Architecture 3 is the most flexible; it contains the flexible register input connection pattern of Architecture 2 and the flexible register output connection pattern of Architecture 1.

From Table 8, with the exception of Architecture 1, we can see that all architectures provide roughly the same IPC, and there is only a small difference in the area. Results for an 8x8 array are shown in Table 9. In this case, there was a larger improvement in performance when both inputs and outputs were connected to multiple FU's.

Comparing the global register file results in Tables 4 and 5 to the local register file results in Tables 8 and 9, we see that that

Table 8: Impact of Changing the Local Register File Connection Pattern (4x4 Array)

Local Register File Connection Pattern	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
Architecture 1	6.01	3.03	1.98
Baseline	7.23	2.10	3.45
Architecture 2	7.25	2.15	3.37
Architecture 3	7.21	2.21	3.26

Table 9: Impact of Changing the Local Register File Connection Pattern (8x8 Array)

Local Register File Connection Pattern	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
Architecture 1	11.95	6.67	1.79
Baseline	13.59	6.48	2.10
Architecture 2	13.16	6.68	1.97
Architecture 3	14.29	6.67	2.14

Table 10: Impact of Changing the Number of Output Ports on the Local Register Files (4x4 Array)

Number of Output Ports	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
1 (Baseline)	7.23	2.10	3.45
2	7.20	2.18	3.31
3	7.16	2.26	3.18
4	7.28	2.30	3.17
5	7.36	2.35	3.13
6	7.22	2.43	2.97

Table 11: Impact of Changing the Number of Output Ports on the Local Register Files (8x8 Array)

Number of Output Ports	Performance (IPC)	Area (sq. mm)	IPC per sq. mm
1 (Baseline)	13.59	6.48	2.10
2	13.90	6.61	2.10
3	13.76	6.84	2.01
4	13.59	7.07	1.92
5	13.73	7.23	1.90
6	13.55	7.42	1.83

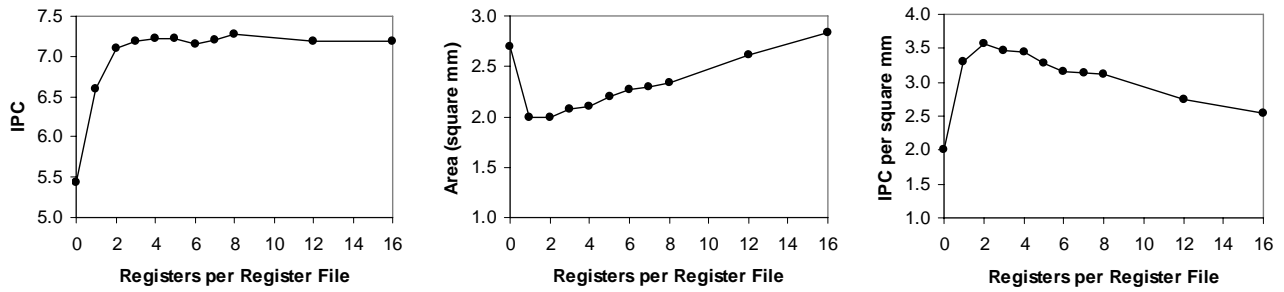


Figure 7: Impact of Changing the Number of Registers in Each Local Register File in the 4x4 Array

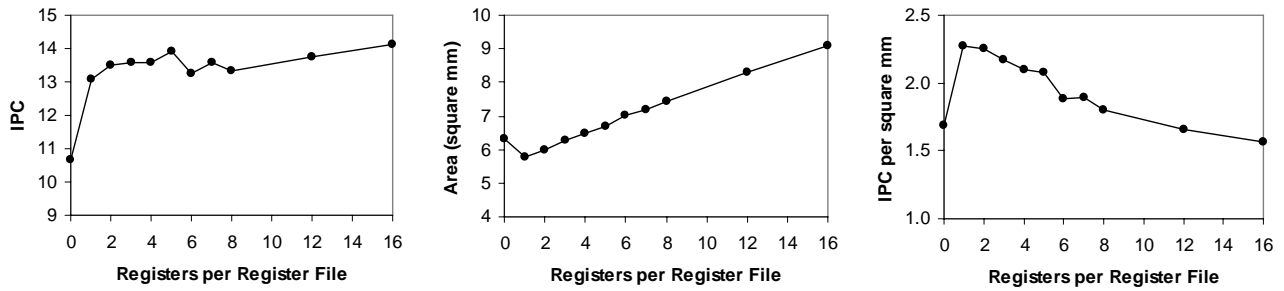


Figure 8: Impact of Changing the Number of Registers in Each Local Register File in the 8x8 Array

Table 12: Number of Local Registers in Each Configurable Functional Unit in Various Published Architectures

Published Architectures	Local Registers Per FU
MorphoSys [1]	4
REMARc [5]	8
PADDI [6]	12
DReAM [14]	16
Montium [13]	16
MaRS [9]	16
CRISP [12]	32
MATRIX [10]	128

the impact of increasing the connectivity of the global register file has a much larger impact on the performance of the array than does increasing the connectivity of the local register files. It is clear that the global register files, which serve as the storage locations for input and output variables for each loop, as well as a central repository, play a critical role in the storage and sharing of data. This is less true of the local register files.

Number of Ports

In the previous results, although the register file inputs and outputs may have multiple sources and sinks, there is only one input port and one output port on each register file. Table 10 shows the impact of changing the number of output ports in the baseline 4x4 array architecture. Since, in the baseline architecture, the output drives up to six ALU inputs, the register file could potentially make use of up to six output

ports. As Table 10 shows, however, one port provides sufficient performance. Table 11 shows that this also holds for an 8x8 array, but two output ports is also a good choice.

Number of Registers

Figures 7 and 8 show the impact of the number of registers in the local register files on IPC, area, and IPC per square mm for the 4x4 and 8x8 arrays respectively. As the figures show, only a small number of registers are required in each local register file. In the 4x4 array, two registers are sufficient, and in the 8x8 array, one register per local register file is the best choice.

It is surprising that such a small number of local registers should be included within each register file, especially considering the larger number of local registers (4 to 128) reported for other architectures (see Table 12). Note that some of these architectures do not have global register files. To gather insight into this, the register utilization of the local register files was extracted from the loop scheduling and mapping information, and recorded in Tables 13 and 14. These tables show the total number of variables stored in all of the local register files, the average number of variables stored in each local register file, and the peak number of variables stored in any local register file in the 4x4 and 8x8 arrays. These results were gathered for an architecture with 128 registers per local register file, so as to maximize the local register file utilization.

Table 13 shows that the peak number of local data registers used from a single register file (the most highly utilized register file in the 4x4 array) is eight registers. It also shows that the

Table 13: Per Benchmark Register Utilization (4x4 Array)

Bench- mark	Total Number of Variables in All Local RF's		Average Number of Variables in Each Local RF		Peak Number of Variables in a Local RF	
	data	pred	data	pred	data	pred
	1	n/a	n/a	n/a	n/a	n/a
2	2	0	0.13	0.00	1	0
3	n/a	n/a	n/a	n/a	n/a	n/a
4	53	0	3.31	0.00	8	0
5	n/a	n/a	n/a	n/a	n/a	n/a
6	2	1	0.13	0.06	1	1
7	3	1	0.19	0.06	1	1
8	24	12	1.50	0.75	6	4
9	17	4	1.06	0.25	4	1
10	6	4	0.38	0.25	2	1
11	0	0	0.00	0.00	0	0
12	27	0	1.69	0.00	4	0
13	17	0	1.06	0.00	6	0
14	4	1	0.25	0.06	2	1
15	15	0	0.94	0.00	2	0
16	8	0	0.50	0.00	3	0
17	13	0	0.81	0.00	3	0
18	6	0	0.38	0.00	2	0
19	34	0	2.13	0.00	6	0
20	15	0	0.94	0.00	3	0
Average	14.5	1.4	0.90	0.08	3.2	0.5

Table 14: Per Benchmark Register Utilization (8x8 Array)

Bench- mark	Total Number of Variables in All Local RF's		Average Number of Variables in Each Local RF		Peak Number of Variables in a Local RF	
	data	pred	data	pred	data	pred
	1	105	0	1.64	0.00	6
2	2	0	0.03	0.00	1	0
3	26	0	0.41	0.00	4	0
4	80	0	1.25	0.00	6	0
5	58	10	0.91	0.16	4	2
6	3	1	0.05	0.02	1	1
7	3	0	0.05	0.00	1	0
8	22	12	0.34	0.19	2	3
9	12	4	0.19	0.06	2	2
10	10	3	0.16	0.05	2	2
11	0	0	0.00	0.00	0	0
12	28	0	0.44	0.00	2	0
13	13	0	0.20	0.00	2	0
14	0	0	0.00	0.00	0	0
15	10	0	0.16	0.00	2	0
16	10	0	0.16	0.00	2	0
17	8	0	0.13	0.00	2	0
18	4	0	0.06	0.00	1	0
19	24	0	0.38	0.00	3	0
20	6	0	0.09	0.00	2	0
Average	21.2	1.5	0.33	0.02	2.3	0.5

Table 15: Per Benchmark Comparison Between Results for Enhanced and Baseline Architectures with a 4x4 Array

Bench- mark	Performance (IPC)		Area (sq. mm)		IPC per sq. mm	
	new	old	new	old	new	old
1	8.94	9.47	3.69	3.69	2.42	2.57
2	8.75	4.38	1.45	2.23	6.02	1.97
3	9.88	7.18	2.09	2.71	4.72	2.65
4	10.33	9.30	2.25	2.55	4.58	3.65
5	11.08	11.08	2.89	3.04	3.83	3.65
6	5.67	5.67	1.29	1.41	4.39	4.01
7	9.50	6.33	1.13	1.41	8.39	4.47
8	11.14	8.67	1.93	2.39	5.76	3.63
9	11.00	7.33	1.45	1.90	7.57	3.85
10	10.00	8.00	1.45	1.74	6.88	4.60
11	8.00	4.00	0.97	1.25	8.23	3.19
12	9.88	9.88	2.09	2.23	4.72	4.44
13	10.25	6.83	1.45	1.90	7.06	3.59
14	6.50	4.33	1.13	1.41	5.74	3.06
15	9.50	8.14	1.77	2.06	5.36	3.94
16	8.25	5.50	1.45	1.90	5.68	2.89
17	10.80	9.00	1.61	1.90	6.70	4.73
18	10.00	6.00	1.29	1.74	7.74	3.45
19	11.17	7.44	1.77	2.39	6.30	3.11
20	10.75	6.14	1.45	2.06	7.40	2.97
Average	9.57	7.23	1.73	2.10	5.52	3.45
% Diff.	+32%		-17%		+60%	

local predicate register file usage is much lower, with at most four registers being used from a single register file. On average, 14.5 data variables and 1.4 predicate variables are stored in local register files. The average number of variables stored in *each* register file is less than one. The average peak usage of each register file is 3.2 for the data registers and 0.5 for predicate registers. These low utilizations explain the low register requirement illustrated in Figures 7 and 8. Table 14 shows similar results for the 8x8 array.

5 Enhanced Architectures

In this section, we use the results from Section 4 to propose two new architectures, one containing a 4x4 array and one containing an 8x8 array. Based on the results from Section 4, our new 4x4 array architecture has the following parameters:

1. All sixteen functional units are connected to the global register files using shared read and write ports.
2. The predicate global register file has four read ports and four write ports; the data global register file has eight read ports and four write ports.
3. The global register files each have 12 registers.
4. The local register files each have one input port and one output port.
5. The local register file outputs are connected to four neighbouring functional units.
6. The local register files each have two registers.

Table 15 shows a comparison between the baseline and enhanced architectures broken down by benchmark loop. As

Table 16: Per Benchmark Comparison Between Results for Enhanced and Baseline Architectures with an 8x8 Array

Bench- mark	Performance (IPC)		Area (sq. mm)		IPC per sq. mm	
	new	old	new	old	new	old
1	20.12	20.12	7.89	9.14	2.55	2.20
2	17.50	8.75	4.11	6.60	4.25	1.33
3	19.75	13.17	5.37	7.87	3.68	1.67
4	13.29	18.60	7.26	7.24	1.83	2.57
5	24.00	28.80	6.63	7.24	3.62	3.98
6	8.50	8.50	4.11	5.33	2.07	1.59
7	19.00	9.50	3.48	5.33	5.45	1.78
8	19.50	15.60	5.37	7.24	3.63	2.16
9	22.00	14.67	4.11	5.97	5.35	2.46
10	40.00	13.33	3.48	5.97	11.48	2.23
11	8.00	8.00	3.48	4.70	2.30	1.70
12	19.75	19.75	5.37	6.60	3.68	2.99
13	20.50	10.25	4.11	6.60	4.98	1.55
14	13.00	6.50	3.48	5.33	3.73	1.22
15	19.00	14.25	4.74	6.60	4.01	2.16
16	16.50	11.00	4.11	5.97	4.01	1.84
17	27.00	13.50	4.11	6.60	6.56	2.04
18	15.00	10.00	4.11	5.97	3.65	1.68
19	22.33	16.75	4.74	6.60	4.71	2.54
20	21.50	10.75	4.11	6.60	5.23	1.63
Average	19.31	13.59	4.71	6.48	4.10	2.10
% Diff.	+42%		-27%		+95%	

the table shows, the enhanced architecture has 17% less area and a 32% higher IPC than the baseline architecture. The IPC per square mm is 60% higher in the new architecture than in the baseline over the set of benchmarks.

We also propose a new architecture based on an 8x8 array. This architecture has the following parameters:

1. Fifty-six of the 64 functional units are connected to the global register files using shared read and write ports.
2. The predicate global register file has eight read ports and eight write ports; the data global register file has sixteen read ports and eight write ports.
3. The global register files each have 16 registers.
4. The local register files each have one input port and two output ports.
5. The local register file inputs and outputs are connected to four neighbouring functional units.
6. The local register files each have one register.

A comparison between this architecture and the baseline architecture is shown in Table 16. In this case, the enhanced architecture has 27% less area and a 42% higher IPC than the baseline architecture. The IPC per square mm is 95% higher in the new architecture than in the baseline over the set of benchmarks.

6 Conclusions

In this paper, we have investigated the impact of the local and global register file architecture on a reconfigurable system

based on the ADRES architecture from IMEC. The register files consume a significant amount of area on the reconfigurable device, and their architecture has a strong impact on the performance. Starting with a baseline architecture, we considered the global and local register files separately, and found that:

- The global register files should be tightly connected to as many functional units as possible. Not all functional units need dedicated read and write ports, however.
- A global register file depth of between 12 and 16 provides the best performance per unit area.
- The local register files should be connected to several neighbours, although the importance of this is small compared to the importance of providing wide access to the global register files.
- One input and one output port is sufficient for each local register file, but it is also good to have two output ports.
- The local register files should be small; one or two registers per register file is sufficient.

Using these results, we developed two new register file architecture that have a 60% higher performance per unit area for a 4x4 array and 95% higher performance per unit area for an 8x8 array over the set of benchmarks. Although the numerical results are specific to this processor, the trends that we observe might apply to many other reconfigurable systems.

With the clear difference in usage between data and predicate register files, further area savings could possibly be obtained by trimming the predicate register files. On the other hand, since their area cost is relatively small, having larger predicate register files may be an inexpensive way to improve performance. Initial results have shown that varying the size of the predicate register files has a small impact on both area and performance, so they are not considered separately in this paper.

References

- [1] H. Singh, M-H Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, E. Chaves, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Compute Intensive Applications", IEEE Transactions on Computers, vol. 49, no. 5, May 2000, pp. 465-481.
- [2] S. C. Goldstein, H. Schmit, M. Budiuh, S. Cadambi, M. Moe, R. Taylor, "PipeRench: A Reconfigurable Architecture and Compiler", IEEE Computer, Vol. 33, No. 4, 2000, pp. 70-77.
- [3] B. Mei, S. Vernalde, D. Verkest, H. De Man, R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix", Proc. of International Conference on Field-Programmable Logic and Applications, Sept. 2003, pp. 61-70.
- [4] J. Becker, M. Glesner, A. Alsolaim, J. Starzyk, "Fast Communication Mechanisms in Coarse-grained Dynamically Reconfigurable Array Architectures", Proc. of Second International Workshop on Engineering of Reconfigurable Hardware/Software Objects (ENREGLE'00, in conjunction with PDPTA 2000), Las Vegas, USA, June 2000.

- [5] T. Miyamori and K. Olukotun, "A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications", Proc. of IEEE Symposium on FPGAs for Custom Computing Machines, April 1998.
- [6] D. Chen and J. Rabaey, "Reconfigurable Multi-Processor IC for Rapid Prototyping of Algorithmic-Specific High-Speed Datapaths", IEEE Journal of Solid-State Circuits, vol. 27, no. 12, Dec. 1992.
- [7] A. Marshall, T. Stansfield, I Kostarnov, J. Vuillemin, B. Hutchings, "A Reconfigurable Arithmetic Array for Multimedia Applications", ACM/SIGDA International Symposium on FPGAs, Feb. 1999, pp. 135-143.
- [8] M. Frank, S. Amarasinghe, A. Agarwal, "The RAW Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", IEEE Micro, vol. 22, no. 2, March/April 2002, pp. 25-35.
- [9] N. Tabrizi, N. Bagherzadeh, A. H. Kamalizad, H. Du, "MaRS: A Macro-pipelined Reconfigurable System", Proc. of the First Conference on Computing Frontiers, Ischia, Italy, April 2004, pp. 343-349.
- [10] E. Mirsky and A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources", Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines, April 1996.
- [11] R. W. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger, "On Reconfigurable Co-Processing Units", Proc. of Reconfigurable Architectures Workshop (RAW98), Orlando, Florida, March 1998.
- [12] F. Barat, M. Jayapala, T. Vander Aa, R. Lauwereins, G. Deconinck, H. Corporaal, "Low Power Coarse-Grained Reconfigurable Instruction Set Processor", FPL 2003, pp. 230-239, Sept. 2003.
- [13] P.M. Heysters, G.J.M. Smit, and E. Molenkamp, "A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems", The Journal of Supercomputing, vol. 26, no. 3, Kluwer Academic Publishers, Boston, USA, Nov. 2003, ISSN 0920-8542.
- [14] J. Becker, M. Glesner, A. Alsolaim, J. Starzyk, "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems", Proc. FCCM'00, Napa, CA, USA, April 2000, pp. 205-215.
- [15] S.J.E. Wilton, N. Kafafi, B. Mei, S. Vernalde, "Interconnect Architectures for Modulo-Scheduled Coarse-Grained Reconfigurable Arrays", Proc. of the International Conference on Field-Programmable Technology, Brisbane, Australia, Dec. 2004, pp. 33-40.
- [16] P. Chang, S. Mahike, W. Chen, N. Warter, W. Hwu, "IMPACT: An Architectural Framework for Multiple-instruction-issue Processors", International Symposium on Computer Architecture, May 1991, pp. 266-275.
- [17] B. Mei, S. Vernalde, D. Verkest, H. De Man, R. Lauwereins, "Exploiting Loop-level Parallelism on Coarse-grained Reconfigurable Architectures Using Modulo Scheduling", IEE Proc. of Computers and Digital Techniques, vol. 150, no. 5, Sept. 2003, pp. 255-261.
- [18] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," Science, May 1983, pp. 671-680.
- [19] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and Routing Tools for the Triptych FPGA," IEEE Transactions on VLSI, vol. 3, Dec. 1995, pp. 473-482.