A Reconfigurable Unit for a Clustered Programmable-Reconfigurable Processor

Richard B. Kujoth, Chi-Wei Wang, Derek B. Gottlieb, Jeffrey J. Cook, Nicholas P. Carter
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{kujoth, cwang12, dgottlie, jjcook, npcarter}@crhc.uiuc.edu

ABSTRACT

In a clustered programmable-reconfigurable processor, multiple programmable processors and blocks of reconfigurable logic communicate through a register-based communication mechanism, which reduces the impact of wire delay on clock cycle time. In this paper, we present a circuit-level design for the reconfigurable clusters used on the Amalgam programmable-reconfigurable processor. We outline our interleaved reconfigurable array design, which provides high bandwidth to and from the register file without requiring large amounts of register control logic. We characterize the latency of operations in our array, and present results that show the impact that this latency has on overall system performance in a range of fabrication processes. Finally, we present a pipelining scheme that enables the array to operate at clock rates closer to those of programmable processors and allows for better scaling in future technologies.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*; C.1 [**Processor Architectures**]: Miscellaneous

General Terms

Design, Performance

Keywords

FPGA, Reconfigurable processor, Technology scaling

1. INTRODUCTION

Clustered programmable-reconfigurable processors [6] are a new class of processor architectures that integrate multiple programmable processors and blocks of reconfigurable logic onto a single chip, using a distributed, or *clustered* organization to reduce wire lengths. As shown in Figure 1, the independent processing resources, or clusters, communicate with each other and with the shared memory system through an on-chip network. This organization allows the clock rate of the processor to be determined by the longest path through a cluster, instead of the wire delay to communicate with centralized issue logic and register files. When clusters must communicate over the on-chip network, its wire delay manifests as additional cycles of latency, which are visible to the compiler, allowing it to schedule operations in an order that tolerates the network latency.

Clustered processors are similar in many ways to chip multiprocessors (CMPs). However, the key difference between these two types of architecture is that, in CMPs, processing resources may only communicate with each other through the shared memory system, while clustered processors provide a more direct means of inter-cluster communication. In the case of Amalgam, this mechanism takes the form of a distributed register file. Each cluster, programmable or reconfigurable, contains a 32-entry register file. Operations executing on a cluster may only read from that cluster's register file, but may write their results into any cluster's register file.

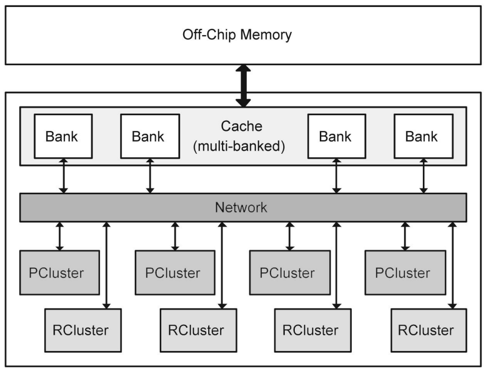

Figure 1: Block diagram of an 8-cluster Amalgam processor

Previous clustered processor architectures have integrated multiple programmable processors onto an integrated circuit. Amalgam differs from these systems in that half of its clusters contain reconfigurable logic, while the remainder contain conventional dual-issue in-order microprocessors. Our previous studies [6] have shown that this architecture is extremely effective, achieving an average speedup of almost 14x over a system containing just one programmable cluster, and almost 3x over an architecture with an equal number of clusters, all of which contain programmable processors. The register-based inter-cluster communication mechanism has also been shown to significantly improve performance. In our experiments, programs that took advantage of the inter-cluster communication mechanism achieved up to 69% better performance than versions of the same program that communicated solely through memory.

In this paper, we present the logical and circuit-level design of the reconfigurable clusters used in Amalgam. While Amalgam's clustered architecture delivers high performance, it places of a number of constraints on the design of the reconfigurable cluster. Chief among these is the need to integrate a register file with the reconfigurable logic in each cluster, followed by the need to communicate with the on-chip network. These constraints have led us to develop a reconfigurable cluster that interleaves portions of the register file with rows of the reconfigurable array, providing the array with high-bandwidth access to the register file without the need to implement register file control logic in the reconfigurable array. An array control unit (ACU) controls the overall flow of computation through the array and the transfer of data to and from the network. This increases the efficiency of the reconfigurable cluster by freeing up resources in the reconfigurable array to implement computational datapaths.

The body of this paper begins with a description of our reconfigurable cluster architecture. This is followed by a discussion of the circuit-level design and layout of the cluster and its performance. We then present an analysis of the impact that improvements in fabrication technology will have on the performance of the reconfigurable cluster relative to programmable processors, and a pipelining scheme designed to improve performance in advanced fabrication technologies. Finally, we discuss related and future work, and then conclude.

2. OVERVIEW OF THE AMALGAM RECONFIGURABLE CLUSTER

As shown in Figure 2, each of Amalgam's reconfigurable clusters (RCs) [16] contains an array of reconfigurable logic, a 32-entry register file, and an array control unit. The reconfigurable array is organized as 32 rows of 32 logic blocks. To better support multi-bit computations, each row of logic blocks is divided into two 16-block sections, each containing a carry-select carry chain to reduce the latency of arithmetic operations.

One of the early challenges in the design of the RC was determining how the reconfigurable array would communicate with the register file. Our main goals were to provide extremely high bandwidth between the array and the register file to support parallel computation, and to limit the amount of logic required to control the register file in order to leave more of the reconfigurable array available for computation. To achieve these goals, we divided the register file into four banks of eight registers each and the reconfigurable array into four segments of eight rows. Register file banks and array segments are interleaved in a ring topology, as illustrated in Figure 2.

Rather than requiring the reconfigurable array to generate the indices of the registers it needs to access on each

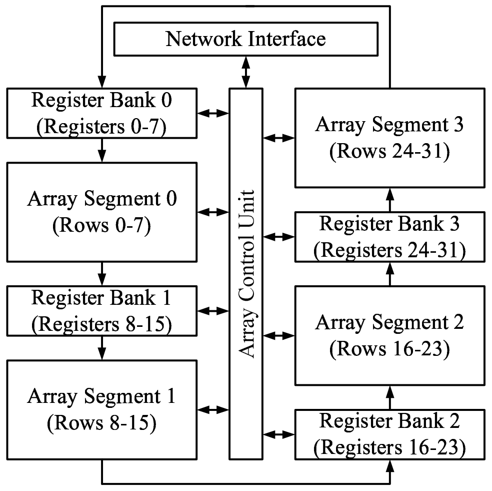
Network Interface
Register Bank 0 (Registers 0-7)
Array Segment 0 (Rows 0-7)
Register Bank 1 (Registers 8-15)
Array Segment 1 (Rows 8-15)
Array Control Unit
Array Segment 3 (Rows 24-31)
Register Bank 3 (Registers 24-31)
Array Segment 2 (Rows 16-23)
Register Bank 2 (Registers 16-23)

Figure 2: Block diagram of an Amalgam reconfigurable cluster

cycle, each bit in the register file drives its output on a read channel (RCH) that is visible to all the logic blocks in the corresponding column of the array segment "below" the bank. Similarly, the input to each register file bit is a write channel (WCH) that can be driven by any of the logic blocks in the segment "above" the bank. Logic blocks can be configured to read any of the registers in the bank "above" them by simply configuring their input multiplexors to route the appropriate read channel to one of the inputs of the logic block's LUT. Similarly, logic blocks can be configured to write any of the registers in the bank "below" them by driving their outputs onto the appropriate write channel.

This organization provides very high register bandwidth with low overhead, potentially allowing the entire contents of the register file to be read and written in each cycle. It also leads to a counter-clockwise flow of data through the reconfigurable cluster, similar to the structure found in PipeRench [5]. Each register has a write enable input driven by the array control unit (ACU).

The segmented and circular nature of the reconfigurable cluster significantly reduces the length of the longest wire that a signal must traverse in a single clock cycle, which helps reduce the effects of wire delay on clock cycle time. Also, to help reduce the effects of wire delay for the ACU, which handles all control and configuration for the entire RC, we have situated the ACU in the center of the four segments.

Communication between logic blocks occurs over a set of horizontal and vertical wires (HWIREs and VWIREs). Each logic block has a dedicated VWIRE that it may drive, which is visible to the 8 rows of logic blocks directly below it, ensuring that each row of logic blocks can communicate directly with at least one row in the next segment. As shown in Figure 3, HWIREs run in channels between rows of a segment and span the entire width of the segment. The pattern of HWIREs between two rows can be configured to allow any logic block to read the output of any of the logic blocks in the row above it. In addition, each VWIRE can be configured to drive one of the HWIREs in each row that the VWIRE crosses to broadcast its value across the row, and

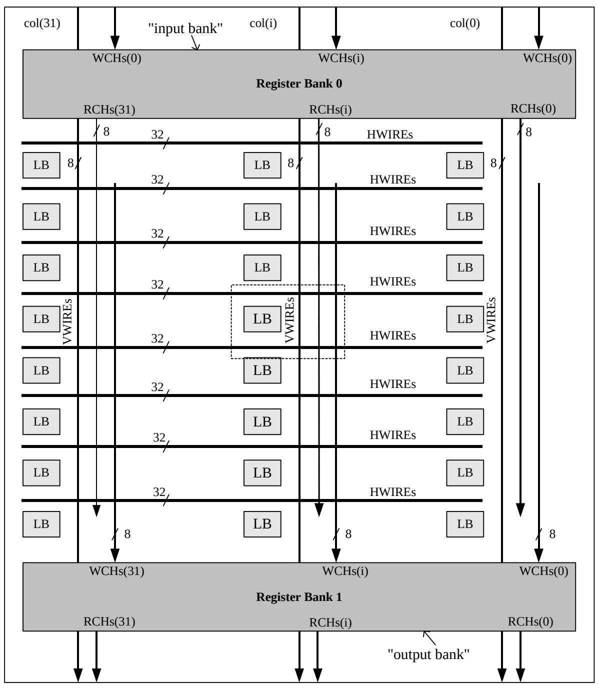

Figure 3: Detailed view of an RC segment, illustrating the routing resources

the HWIRE's in the first row of a segment can be set to drive the read channels in the segment, creating a broadcast mode that makes the value on an HWIRE visible to every logic block in the segment. Finally, a local feedback network connects each logic block's output to its two nearest neighbors on either side, allowing propagation of data across a row. This set of wiring resources provides a very flexible set of communication paths within the array, allowing a large set of applications to be efficiently mapped onto the array.

3. DESIGN OF THE RECONFIGURABLE CLUSTER

In the last section, we described the architecture of Amalgam's reconfigurable clusters at a high level. We now proceed to cover the design of selected elements of the reconfigurable cluster in more detail, including the translation of the architecture into circuit-level designs and layout.

3.1 Logic Block

As shown in Figure 4, Amalgam's logic blocks (LBs) consist of a four-input lookup table (LUT), a data register (DREG) that latches the output of the LUT, input and output multiplexors, and optimized carry chain logic. The four-input LUT was selected because 4-LUTs have been shown to allow area-efficient mappings of a wide range of circuits [14]. However, our carry chain design required that the logic block be able to generate multiple outputs. To accommodate this, the 4-LUT is implemented as a pair of 2-LUTs and a 3-LUT, which can either be configured as a single 4-LUT or as separate LUTs to generate the values required by the carry chain.

The initial design of the logic block called for eight modes

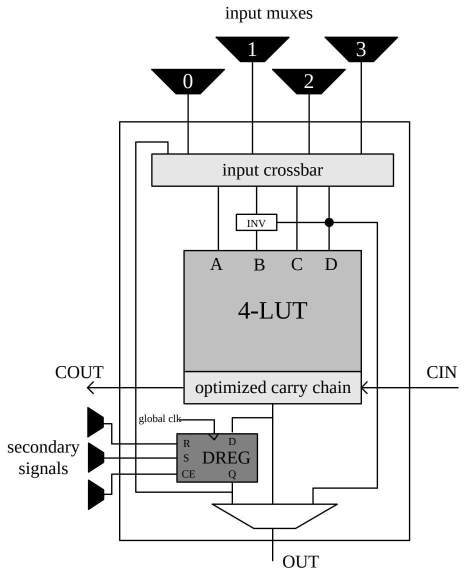

Figure 4: High-level view of the LB

of operation: registered and non-registered versions of combinational, arithmetic, ALU, and shortcut (pass-through) operation. After implementing the fast carry chain, it became apparent that two types of arithmetic and ALU modes were required: ones that utilize the carry chain to perform wide computations and ones that use a ripple carry for computations on small bit widths. This brings the total number of modes of operation of the LB to 12, registered and non-registered versions of each of the following:

- LBM_ARITH: Arithmetic mode. The 4-LUT is split into two 3-LUTs to compute the sum and carry in parallel. The LUTs are indexed using the a, b, and c inputs. Carry-ins are provided via the carry-select carry chain or can be an arbitrary value, depending on the programming.
- LBM_RARITH: Arithmetic mode w/ripple chain. The same as LBM_ARITH mode, but using the ripple carry chain instead of the fast carry chain.
- LBM_ALU: ALU mode. The same as LBM_ARITH mode, but the d input controls the inversion of the b input to implement both addition and subtraction with the d input being the operation select.
- LBM_RALU: ALU mode w/ripple chain. The same as LBM_ALU, but using the ripple carry chain.
- LBM_COMB: Combinational mode. The a, b, c, and d inputs are used as indices into the 4-LUT.
- LBM_SHORTCUT: Shortcut mode. Drives the d input onto the output of the LB. This mode is used for HWIRE to VWIRE routing.

3.2 Carry-select Carry Chain

The reconfigurable cluster is intended to support computations, and is therefore optimized for multi-bit operations. A key part of this optimization was the decision to include

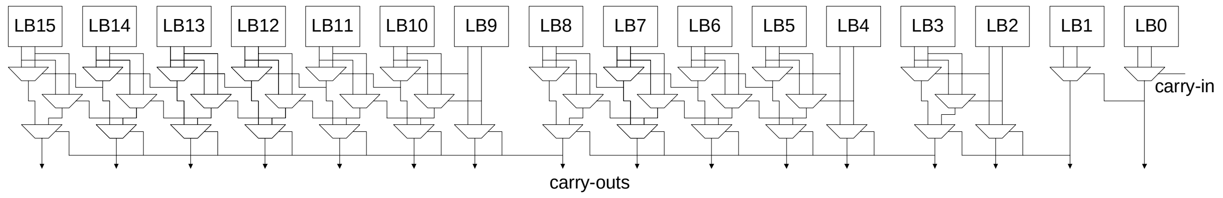

Figure 5: Block diagram of the carry-select carry chain

two fast 16-bit carry chains in each row of the array, which can be used to either implement two parallel 16-bit operations in one cycle or a 32-bit operation with a latency of two cycles. Two 16-bit carry chains were used instead of a single 32-bit chain because we believed that it was unreasonable to expect to perform a 32-bit operation in one clock cycle, an assumption that was supported by our circuit studies.

Implementation of the carry chain required that the 4-LUT be split into two 2-LUTs and one 3-LUT, as mentioned in the previous section. One 2-LUT, which we will call the C1 2-LUT, calculates the carry-out assuming the carry-in will be a logic 1. The C0 2-LUT, in turn, calculates the carry-out assuming the carry-in will be logic 0. These results are propagated to the next LB's carry chain hardware, which calculates its carry-outs based on these two predicted carry-ins using a 2-to-1 multiplexor. These chains create several short adders (of increasing length) from the 16-bit half-row of LBs. These short adders execute in parallel, with the output being selected by the previous short adder's carry-out. For our circuit, short adders of length 1, 3, 5, and 7 were optimal for our multiplexor delays.

The 16-bit carry chain can be split into smaller bit lengths via programming bits. Carry chain splits can be made at the short adder boundaries and each short adder can be extended by one bit by including the most significant bit of the previous short adder in its chain. This allows for several lengths of adders, most notably two 8-bit computations. Additional computation lengths beyond the splits between the short adders can be also implemented by using a combination of the ripple carry chain and the carry-select carry chain, but are limited by the latency of the ripple carry chain. Our current timing estimates show that a 3-bit ripple carry chain has the same latency as a 16-bit carry-select carry chain.

3.3 Input Multiplexor and Crossbar

Four input multiplexors (muxes) and a crossbar allow the logic block to select its inputs from among the routing resources available to it. The crossbar is necessary because the 4-LUT in each logic block is implemented as two 2-LUTs and a 3-LUT. In modes where these smaller LUTs operate independently, the crossbar allows the inputs to the logic block to be routed to the LUT that needs them. Initially, the design of the RC allowed each input mux to select from any of the 32 HWIREs in the row above it as well as the read channels and VWIREs in its column. This led to a total of 46 inputs at each mux, making them unacceptably large. To reduce multiplexor area, the number of HWIRE inputs to each mux was reduced from 32 to 16, lowering the total number of inputs to 28 (16 HWIREs, 2 VWIREs, 8 read channels, and 2 local feedback connections).

To determine the subset of 16 HWIREs available to each input mux, we tested our cycle-accurate Amalgam simulator, *amalsim*, with several different possible input mux configurations and chose the best from these. Each configuration required that two of the four input muxes are able to select each HWIRE, so that each HWIRE could be connected to an alternate input mux if any routing conflicts occurred in one input mux.

The four outputs from the input muxes are passed into the crossbar, which selects the final inputs into the LB. In addition to selecting from the input mux outputs, the crossbar can also select from a logic 0 or 1, the ripple-carry in, and a feedback loop from the LB's DREG to be driven on any of the LUT's a, b, c, or d inputs. The crossbar also includes logic to invert the b input into the LUTs in ALU mode and a bypass mux for the carry-select carry-in.

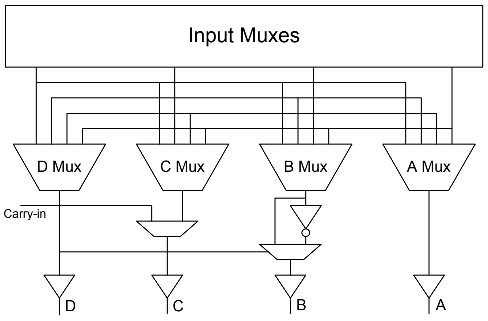

Figure 6: Crossbar design, with the selective inverting of the b input and selection of carry-in

Our circuits use a transmission gate architecture because we felt that the stronger signal propagation provided by transmission gates was more critical than the area savings that would have been achieved through the use of nmos-only pass gates. In spite of this, buffers are still required at several points in the logic block (generally every three or four transmission gates) to provide sufficient drive strength. An example of this can be seen in Figure 6.

3.4 Register File

As described earlier, the register file in each RC is divided into four banks of eight registers, which are interleaved with the segments of the reconfigurable array. Each column in the array has access to the values in the corresponding bits of each register in the bank above it and can write the corresponding bits of registers in the bank below it. To support this architecture, the register file is divided into columns, and each column is pitch-matched to a column of the array. The structure of each column of a register file bank is shown in Figure 7. The eight register bits in the column are organized into two rows of four bit cells to match the width of the logic block. Write channels enter the column from above, and the outputs of the column are continuously driven onto the read channels. Control bits from the ACU allow it to determine when each register is written, and a data channel allows the ACU to read and write one register in each bank per cycle.

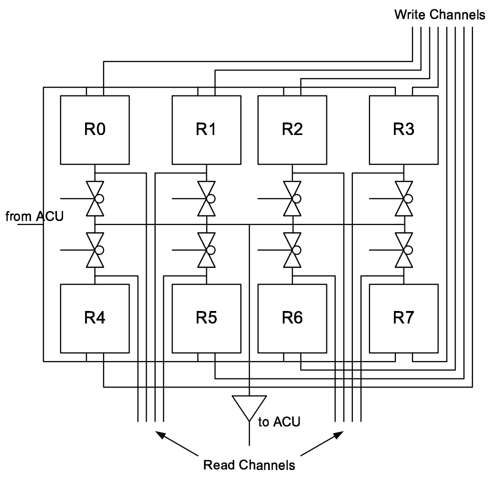

Figure 7: One column of a register file bank

Each of the 32 registers in a cluster is implemented as 32 flip-flops with set and reset capabilities. This allows initialization of the register file and control from the ACU. The ACU can also initialize the register file with arbitrary values by initiating register transfers to each of the registers one at a time.

3.5 Array Control Unit

The array control unit (ACU) controls all computation and configuration in the RC. It is composed of four main parts: the fixed control unit (FCU), the programmable control unit (PCU), the network interface controller (NIC), and the configuration cache (CC). The FCU controls reconfiguration of the array and all register-based communication. The PCU drives several control signals into the array: the read/write wires to the register banks, the set/reset wires to the DREGs, and a subset of the HWIREs. The NIC is the interface for the RC to the intercluster network and the CC stores configurations to support run-time reconfiguration of the array.

Due to its irregular structure, the ACU is the one component of the RC that we have not performed a circuit-level design of by hand. Instead, the ACU is currently being implemented in Verilog for synthesis down to a gate-level design. We expect that the ACU will be the component of the RC most likely to change as we continue our studies of Amalgam; implementing it in Verilog will allow us to implement these changes much more quickly than would be possible in a full circuit-level design.

3.6 Programming the RC

One of the major design issues in reconfigurable systems is the method by which configurations are loaded into the reconfigurable fabric. Programming bits are required for configuring routing between VWIREs and HWIREs, reading data off of the RCHs, and configuring the LB's inner components, the LUTs and crossbar.

There are several methodologies for programming described in [3]. Each methodology revolves around one of two central ideas: modeling the programming bit registers as a long shift register and modeling them as a parallel loading register. Most architectures find a compromise between the two, balancing the lengthy programming times of the shift approach and the area requirements for the parallel approach. Our programming bit registers for each LB are simple D flip-flops connected in a chain, similar to a shift register or a scan chain. These flip-flops are located throughout the LB. Each LB has 32 pbits for its configuration and 20 pbits for routing the LB result for a total of 52 pbits per LB.

As illustrated in Figure 8, we have intelligently positioned the programming bits within the shift registers to allow one portion of the configuration to be changed without changing the entire RC configuration, to reduce configuration times if only the contents of the LUTs need to change. The first programming bits in the shift register are the LUT memory bits, which can be changed by only activating the clock to their shift registers and feeding them with data from the ACU. Our study of run-time reconfiguration in this architecture is just beginning, but we expect that this feature will help reduce the overhead of reconfiguring the array at run-time.

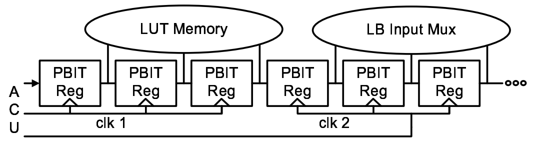

Figure 8: Example of a programming bit register chain such that a subset of the registers can be reconfigured without changing the entire set

Programming bits are propagated to the programming bit registers in the LBs via wires from the ACU. These wires are driven by shift registers in the ACU whose contents are loaded out of the configuration cache. These ACU shift registers are loaded with programming bits from the configuration cache (CC) prior to programming the RC. This configuration architecture allows the array to be configured very quickly as long as the desired configuration is in the configuration cache, with reconfiguration time being limited only by the number of programming bits in each logic block.

3.7 Layout

Because Amalgam is a high-performance architecture, we chose to layout the reconfigurable cluster by hand using Cadence Virtuoso. We did not feel that we could achieve either the performance or the area required by the RC with a synthesized design. As mentioned above, the ACU is the one exception to this because of its irregular structure and the expectation that its design will change over time. To perform our layout, we used the NCSU Cadence Design Kit [1] for their design rules for the TSMC 180nm technology.

The regular structure of the RC helped laying it out immensely. Since the RC is primarily an array of LBs, most of the time in the layout phase was devoted to design of the LB. A complete layout of the LB annotated with the floorplan can be seen in Figure 9. It is worthwhile to note that despite reducing the number of inputs into the input muxes, the four of them still take up nearly half of the area of the

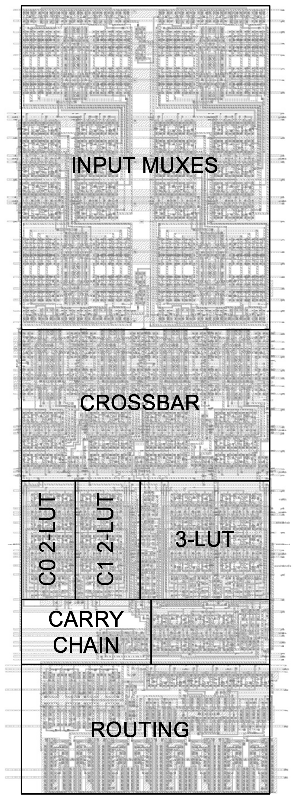

Figure 9: Floor-plan of the LB overlayed on top of its layout

total LB. We have created a full layout of the RC without the ACU in a 180nm process, the results of which will be discussed in the next section.

3.8 Results

We have created a full circuit schematic and layout of the entire RC except for the ACU. SPICE simulation of this schematic has provided us with accurate estimates of the latencies and power requirements of each component in the RC, while the hand layout has given us a good idea of the area required. An annotated layout of one segment of the RC can be seen in Figure 10. Table 1 gives a breakdown of the area required for each component of the segment. The worst case power consumption for a single row is 27.79mW, in the case that all logic blocks are switching simultaneously every clock cycle, which leads to a worst case power consumption for the entire array of 0.889W. We are planning to run simulations to determine an average case power consumption for the array.

It is useful to compare the chip area required for the programmable and reconfigurable clusters to determine whether the reconfigurable clusters provide enough performance benefit to justify their area cost. Using the methods described in [7], we estimate the area of a programmable cluster to be 6.26 mm^2, approximately half of the 11.34 mm^2 required for a reconfigurable cluster.

Through simulation in HSPICE, we have obtained estimates of latencies in the RC, a selection of which are shown in Table 2. A 16-bit addition, the main component of the RC's clock period, has been found to have a latency of

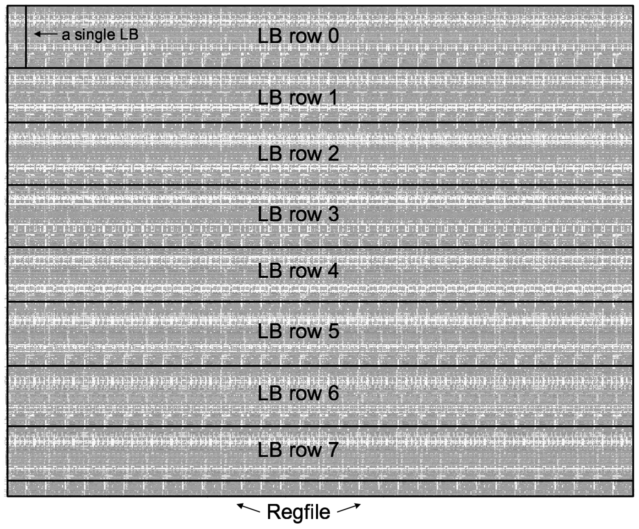

Figure 10: Layout of one segment from the reconfigurable cluster in 180nm

Component Name	Area for Each (mm^2)	% of Tot. RC Area
RC	11.35	100
RC Segment	2.27	80
LB	8.58e-3	77.44
Bit Reg.	5.35e-5	25.11
ACU (est.)	2.27	20
Reg. Bank	5.95e-2	8.39
Carry Chain	1.26e-3	0.71

Table 1: Layout-extracted component area values and their percentage of the total area of the RC in 180nm

18.29 FO4 delays. This is in line with the cycle times of recent Intel processors, which have ranged from 12-20 FO4 delays [11]. However, future programmable processors are expected to include much less logic than current designs in each pipeline stage, leading to greater increases in clock rates than would be caused by technology scaling alone. The next section will describe the impact that this scaling has on the performance of our architecture.

We chose to add the carry-select chain because we believed that it would increase performance greatly at a negligible area cost. Our simulations in HSPICE showed that a 16-bit add required 18.29 FO4 delays with the carry-select chain and 77.83 FO4 delays with the ripple chain, a 76.5% improvement in performance. The carry-select logic only comprised of 0.71% of the total RC area, which we considered an acceptable usage of area, considering the performance benefits.

4. IMPACT OF TECHNOLOGY SCALING ON AMALGAM'S RC

We have now presented a design for the Amalgam reconfigurable cluster and described its performance, as designed in 180nm technology. A large body of research has been done to determine the effects that technology scaling has

Component	Latency (FO4)
Carry-select 16-bit ADD	18.29
Ripple-carry 16-bit ADD	77.84
LB (COMB Mode)	9.55
LB (COMB_SEQ Mode)	13.70
LB (SHORTCUT Mode)	3.04
VWIRE	0.37
HWIRE	0.63
RCH	0.37

Table 2: HSPICE simulated values for FO4 delays of several components of the RC in 180nm

on circuit design; some of these challenges are discussed in [2] and [10]. One of the main difficulties of technology scaling is the discrepancy of latency scaling between wires and transistors. Wire delay is becoming the bottleneck in high performance circuit designs. We will now study the effect that technology scaling has on the performance of the RC, and its effect on the performance of Amalgam.

To begin technology scaling of the RC, we assume that the latency of the RC's clock cycle will continue to be determined by the critical path through the traversal of an HWIRE, VWIRE, RCH, and a 16-bit add. This path has been determined to have a latency of 19.96 FO4 delays in 180nm technology through simulations in HSPICE. The 16-bit add's latency is the majority of the total path's latency, accounting for 18.29 FO4 delays. For our technology scaling studies that follow, we assume that the 16-bit add's latency only increases slightly in future technologies, due to the traversal of the carry chain. This is a good assumption, as FO4 delays are technology independent. High-level interconnects, including the intercluster communication network, VWIREs, HWIREs, etc., have been scaled using the models in [17] with the future technology parameters found in [13]. The interconnect latencies increase the critical path of the RC from 19.96 FO4 delays in 180nm to 27.57 FO4 delays in 22nm.

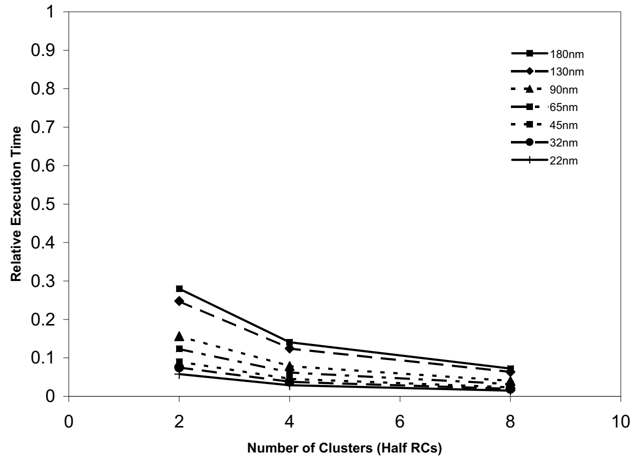

Figure 11: Average multicluster speedup over future technologies

To estimate the clock rate of the programmable clusters in each technology, we used the values presented in the 2001 International Technology Roadmap for Semiconductors (ITRS) [12]. Relative cycle times of the programmable and reconfigurable clusters were estimated by converting the cycle times given in the ITRS to FO4 delays, as shown in Table 3. Our FO4 delays for future technologies were estimated as 500 picoseconds times the transistor's drawn gate length, which is a worst case estimate. This model of FO4 delays appears to be accurate model of worst case FO4 delay through devices with drawn dimensions of 18nm [10].

We used our delay findings to model an Amalgam's performance in future technologies in our Amalgam simulator, *amalsim*. To model the increased latency of the RC with respect to the PC, we scaled the clock frequency of the RC using the RC/PC ratios found in Table 3. We accounted for the network latency by introducing multicycle transfers across the network, the number of cycles being determined by the technology.

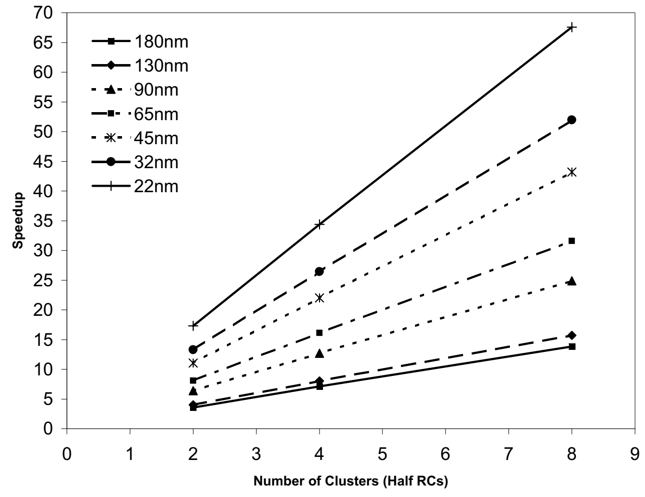

Figure 12: Average speedup of Amalgams with RCs over 1 PC in future technologies

To compare performance across all of the future technologies, we used a standard set of benchmarks which we have developed for the Amalgam architecture. The benchmarks we include are Dither, Rijndael, nQueens, and DNA. Dither uses Floyd-Steinberg error diffusion to convert a 512x512 pixel image from 24-bit RGB format into an 8-bit/pixel one. Rijndael is an implementation of the AES block encryption algorithm using a block size of 128 bits and 10 iterations over a sequence of 512 blocks. nQueens determines the number of arrangements of N queens on an NxN chessboard such that none of the queens can capture any of the others in a single move. DNA uses a dynamic programming algorithm to compute the edit distance between two sequences of genetic information. All of these benchmarks were hand-coded in assembly language.

Figure 11 shows how the execution time of our benchmarks scales with the number of clusters used in each of the fabrication technologies. This graph shows that the overall performance of clustered programmable-reconfigurable processors will improve significantly in future fabrication processes, although an increasing fraction of that performance

Gate Length (nm)	PC Clock Freq. (GHz)	Delay of FO4 (ps)	PC Cycle Delay (FO4)	RC Cycle Delay (FO4)	RC/PC Delay Ratio	RC Eff. Clock Freq (GHz)
180	0.75	90	14.81	19.96	1.35	0.56
130	1.68	65	9.14	19.53	2.14	0.79
90	3.99	45	5.57	20.60	3.70	1.08
65	6.74	32.5	4.57	21.36	4.68	1.44
45	11.51	22.5	3.86	22.74	5.89	1.95
32	19.35	16	3.23	24.39	7.55	2.56
22	28.75	11	3.16	27.57	8.72	3.30

Table 3: Projected cycle delays and frequencies for the PC and RC

will come from cycle time improvements instead of parallelism. While this result was not unexpected given the growing importance of wire delays, it is a strong motivation for research into techniques to minimize the amount of communication required during program execution.

Figure 12 shows how the speedup of a full Amalgam processor over a processor with a single programmable cluster will change as fabrication technologies improve. As expected, total speedups decline as technology advances, due to the impact of wire delays and the fact that merely implementing our reconfigurable cluster in future technologies will not allow its clock rate to scale to match expectations for future programmable processors.

In past publications, we have shown how an Amalgam processor composed of RCs and PCs performs much better than an Amalgam composed of solely PCs. In Figure 13, we compare the performance of an 8-cluster PC Amalgam with an 8-cluster PC/RC Amalgam. In current technologies, our previous findings hold true, but in future technologies, we find that the PC Amalgam performs better with the crossover point being right at the 65nm technology.

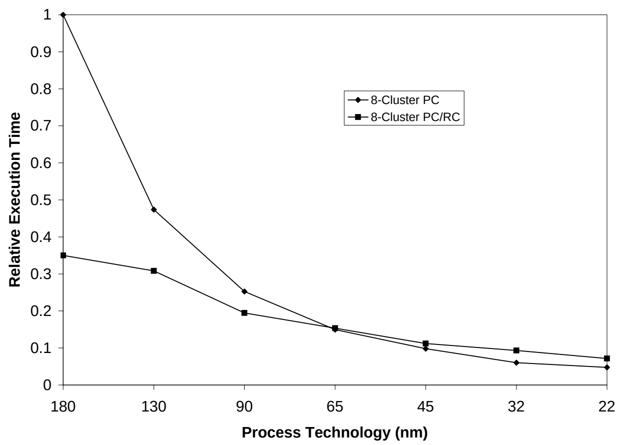

Figure 13: Comparison of average execution times of an 8-clustered Amalgam composed of PCs and an 8-clustered Amalgam of PCs and RCs over future technologies

We attribute this result to the fact that our RC scaling does not take into account any pipelining support and that the estimates in the ITRS for future clock rates are extremely aggressive. Matching the predictions in the ITRS requires that the clock period of a programmable cluster decrease to 3.16 FO4 delays in 22nm processes. Results from [11] indicate that reducing clock periods to less than 6 to 8 FO4 delays per cycle hurts overall performance, leading us to believe that programmable processor clock rates will not scale at the rates predicted by the ITRS.

5. PIPELINING

Achieving predicted clock rates for programmable processors in advanced fabrication processes will require increasing pipeline depths to reduce the amount of work performed in each stage. As we have seen in the previous sections, this will lead to an increasing clock-rate gap between the programmable and reconfigurable clusters if the current reconfigurable cluster design is merely re-implemented in each fabrication process. In this section, we present a pipelining scheme for the reconfigurable cluster that will reduce this gap in clock rates.

5.1 Clock Rate Limitations in the Original RC

In the baseline reconfigurable cluster design, the output D-flip-flop (DFF) in each logic block provides a natural mechanism for pipelining computations. Using the DFFs in each row of logic blocks as pipeline latches, we can derive maximum clock-rates for our design in two ideal cases: the case where the computation in each stage is performed within a single logic block, and the case where each pipeline stage makes use of the carry chain to perform a 16-bit computation. In both cases, the maximum clock rate of the RC is equal to the time to perform the computation in a pipeline stage plus the longest possible wire delay to the input of the next stage.

The longest inter-stage communication path in our design goes from the output of a logic block, through a VWIRE, a HWIRE, and a read channel (RCH). HSPICE simulations with logical effort techniques [15] have determined that the delay along this path is 1.7ns for our design when implemented in a 180nm process[1].

In the case where each logic stage's computation is performed within a single logic block, the minimum clock cycle time is 1.7ns (interconnect) + 0.5ns (input mux) + 1.108ns (crossbar/LUT) = 3.308ns. If we assume that each pipeline stage contains a single 16-bit computation using the embedded carry chains, the cycle time becomes 1.7ns (interconnect) + 2.121ns (16-bit add) = 3.821ns (263MHz).

[1]This value includes the setup and hold times for the logic block's output DFF, as well as the delays of the buffers required in the signal path. The numbers given previously for HWIRE, VWIRE, and RCH delays considered only the actual propagation delays along these wires, for technology scaling purposes.

5.2 Pipelined RC Architecture

We pipeline the reconfigurable cluster by adding two sets of flip-flops to the design: one set that breaks up all delays longer than the delay through a logic block into multiple cycles, and another set that provides retiming at logic block inputs to handle cases where the inputs to a logic block take different numbers of cycles to arrive.

5.2.1 Pipelining Long Delays

As described above, our worst-case interconnect delay is fairly close to the time required to perform a 16-bit addition in the RC. Therefore, our first attempt at pipelining the RC will be to add pipelining latches at the output of each logic block's input multiplexers, breaking the longest path into two cycles: one for interconnect delay and input multiplexing, and one for computation. Placing the pipeline flip-flops on the outputs of the input multiplexers rather than the inputs significantly reduces the number of latches required to implement this pipelining. With this change, the minimum cycle time of the RC becomes 2.8ns when performing a 16-bit addition in each stage.

To further reduce the minimum cycle time, we need to pipeline both the interconnect and 16-bit computations that use the carry chain. Adding flip-flops to the C0 and C1 outputs as well as the carry out of each logic block allows us to partition a 16-bit add into two operations with latencies of 1.504 and 1.617ns (including the flip-flop delay). Similarly, we can pipeline our interconnect delays by adding flip-flops at the junctions between VWIREs, HWIRES, and RCHs. These flip-flops serve two purposes: they allow wire delay to be pipelined across multiple cycles, and they insulate paths from the resistance and capacitance of segments that they do not use. With this set of pipeline latches, the interconnect becomes capable of supporting clock periods down to 1.7ns in our 180nm process.

5.2.2 Retiming

With all large delays pipelined, the target clock period becomes 1.75ns, the delay from the input flip-flop of a logic block to its output register, giving a clock rate of 570 MHz. However, adding these pipeline latches to our design creates problems for the designer, because different inputs to a logic block may now take different numbers of cycles to reach the logic block. To address this, we add a set of retiming latches to each input of the logic block, which are configured as a variable-length FIFO queue.

Because of the reconfigurable cluster's segmented structure, rows of logic blocks near the top of each segment tend to need less retiming than rows near the bottom, as they are more likely to read their inputs from the register file. Therefore, we use two different configurations of retiming registers. The top four rows in each segment have five retiming flip-flops on each input, while the bottom four rows have seven retiming flip-flops on each input. Since logic blocks in the top rows will occasionally require large numbers of retiming registers, our architecture allows each input an LB in one of these rows to "borrow" up to two flip-flops from the adjacent input's retiming chain if they are not needed by that input.

The final logic block architecture is shown in Figure 14.

By adding pipelining to the reconfigurable array, we are able to significantly reduce the number of FO4 delays in each clock cycle at the cost of some increase in the number of cycles required to perform a computation. Each path through the array can be pipelined independently, and our pipelining scheme isolates pipelining overheads so that paths that do not require pipelining do not see the delay of the retiming flip-flops. The area cost of pipelining was also a major factor in our design. Adding these flip-flops to the reconfigurable cluster will increase the area of each segment of the array by 25% and the total area by 20%. We believe that the clock rate increases made possible by these pipelining techniques will allow the reconfigurable cluster to achieve significantly better performance in future fabrication processes than the base design. We are currently porting our benchmarks to take advantage of pipelining, and expect to have application results in the near future.

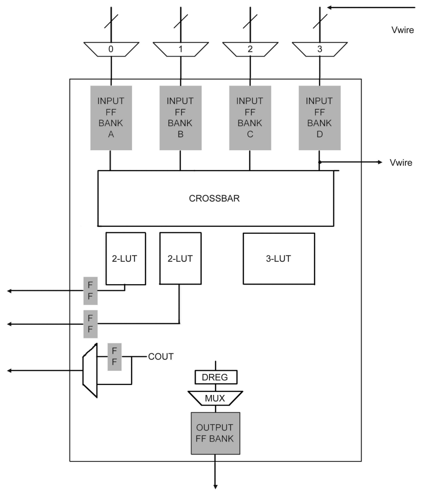

Figure 14: New LB Architecture

6. RELATED WORK

Clustered multiprocessors have been shown to be a successful alternative to traditional programmable processor designs. Amalgam builds on this success with the addition of reconfigurable logic in the form of reconfigurable clusters. The integration of programmable and reconfigurable logic has been attempted in Garp [9] and Chimaera [18]. The row-based design of the reconfigurable cluster is based heavily off of the architecture of Garp. Inclusion of fast carry chains in reconfigurable logic to increase performance of arithmetic options has also been investigated by the Chimaera project [8].

Pipelined reconfigurable computing systems have varied from coarse-grain to more flexible fine grain architectures. RaPiD [4] is a 1-dimensional linear pipeline. The datapath is formed by a row of coarse-grain word-based functional units connected to a programmable interconnect with a variable configurable length. PipeRench [5] is another coarse-grain architecture that organizes its functional units into multiple rows, or stripes, consisting of 8 bit ALUs, registers and a local communication interface. Similar to Amalgam's RC, global communication is performed by a vertical global bus which gives the architecture a direction of computational flow.

7. FUTURE DIRECTIONS

Despite having a nearly complete RC design, we are still actively researching improvements on Amalgam's RC design. Runtime reconfiguration offers a large advantage over traditional systems, but is limited by long reconfiguration latencies. In order to support quick reconfigurations, we are currently investigating techniques for configuration caching. After the design for the RC is finalized, the ultimate goal is to have Amalgam and its RCs fabricated onto a chip. This drove the selection of tools for our implementation as described in this paper. We are also developing a suite of tools to automatically compile standard C-code to the reconfigurable clusters.

8. CONCLUSIONS

In this paper, we have presented a design for Amalgam's reconfigurable cluster. We have described several key components of the reconfigurable cluster in detail, namely the logic block, the fast carry chain, the register file, and a method of programming the reconfigurable cluster. We have also shown that Amalgam reconfigurable cluster's performance will change in future fabrication technologies.

Our results from our technology scaling study show that reconfigurable cluster performance will remain superior to programmable cluster performance on near future technologies, but will lose its performance margin in the distant future. A redesign of the reconfigurable cluster will be needed for it to maintain its substantial edge over the programmable cluster. Adding pipelining support for the reconfigurable cluster will improve its performance and allow for better scaling in future technologies, which we have also detailed in this paper.

9. ACKNOWLEDGMENTS

This work was funded by the Office of Naval Research under award number N00014-01-1-0824. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

10. REFERENCES

[1] NCSU Cadence Design Kit. http://www.cadence.ncsu.edu/.

[2] S. Borkar. Design Challenges of Technology Scaling. *Proceedings of the International Symposium on Microarchitecture*, 19(4):23–29, July 1999.

[3] P. Chow, S. O. Seo, J. Rose, K. Chung, G. Paez-Monzon, and I. Rahardja. The Design of a SRAM-Based Field-Programmable Gate Array–Part II: Circuit Design and Layout. *IEEE Transactions on VLSI Systems*, 7(3):321–330, Sept. 1999.

[4] D. Cronquist, P. Franklin, C. Fisher, M. Figueroa, and C. Ebeling. Architecture Design of Reconfigurable Pipelined Datapaths. In *Proceedings of the Twentieth Anniversary Conference on Advanced Research in VLSI*, pages 23–40, 1999.

[5] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer. PipeRench: A Coprocessor for Streaming Multimedia Acceleration. In *Proceedings of the International Symposium on Computer Architecture*, pages 28–38, 1999.

[6] D. B. Gottlieb, J. J. Cook, J. D. Walstrom, S. Ferrera, C.-W. Wang, and N. P. Carter. Clustered Programmable-Reconfigurable Processors. In *Proceedings of the IEEE International Conference on Field-Programmable Technology*, pages 134–141. IEEE, Dec. 2002.

[7] S. Gupta, S. W. Keckler, and D. Burger. Technology Independent Area and Delay Estimates for Microprocessor Building Blocks. Technical Report 2000-05, Department of Computer Sciences, The University of Texas at Austin, 2000.

[8] S. Hauck, M. M. Hosler, and T. W. Fry. High-Performance Carry Chains for FPGA's. *IEEE Transactions on VLSI Systems*, 8(2):138–147, Apr. 2000.

[9] J. R. Hauser and J. Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In K. L. Pocek and J. Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 12–21, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[10] R. Ho, K. W. Mai, and M. A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, Apr. 2001.

[11] M. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar. The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 14–24. IEEE, May 2002.

[12] *The International Technology Roadmap for Semiconductors 2001*. Semiconductor Industry Association, 2001.

[13] *The International Technology Roadmap for Semiconductors 2002 Update*. Semiconductor Industry Association, 2002.

[14] J. Rose, R. J. Francis, D. Lewis, and P. Chow. Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency. *IEEE Journal of Solid-State Circuits*, 25(5):1217–1225, Oct. 1990.

[15] I. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast Cmos Circuits*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[16] J. D. Walstrom. The Design of the Amalgam Reconfigurable Cluster. Master's thesis, University of Illinois at Urbana-Champaign, 2002.

[17] S.-C. Wong, G.-Y. Lee, and D.-J. Ma. Modeling of Interconnect Capacitance, Delay, and Crosstalk in VLSI. *IEEE Transactions on Semiconductor Manufacturing*, 13(1):108–111, February 2000.

[18] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee. CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit. In *Proceedings of the International Symposium on Computer Architecture*, pages 225–235, 2000.