Appeared in Third Canadian Workshop of Field-Programmable Devices (FPD'95)

A First Generation DPGA Implementation

Edward Tau Derrick Chen Ian Eslick Jeremy Brown
{edtau,kuang,beethovn,jhbrown}@ai.mit.edu
André DeHon
andre@ai.mit.edu
MIT Artificial Intelligence Laboratory
NE43-791, 545 Technology Sq., Cambridge, MA 02139
Phone: (617) 253-5868 FAX: (617) 253-5060

Abstract

Dynamically Programmable Gate Arrays (DPGAs) represent a hybrid architecture lying between traditional FPGAs and SIMD arrays. Notably, these arrays can efficiently support computations where the function of the array elements needs to vary both among array elements during any single cycle and within any single array element over time. We describe our minimal, first generation DPGA. This DPGA uses traditional 4-LUTs for the basic array element, but backs LUT and interconnect programming cells with a 4-context memory implemented using dynamic RAM. Additionally, this DPGA supports non-intrusive background loads of non-active contexts and automatic refresh for the dynamic memory cells. We draw several lessons from this design experience which may be relevant to future DPGA and FPGA designs.

1 Introduction

Traditional Field-Programmable Gate Arrays (FPGAs) have a set of programmable elements which can be configured to personalize the FPGA to implement a user-defined function. Reprogrammable FPGAs implement these programmable elements as memory cells, commonly static RAMs. These cells are configured during a slow programming phase which typically takes 10's of milliseconds due to limited bandwidth to off-chip memories [Xil93] [Atm94]. Once configured, the array personality is static until reloaded. Each array element, consequently, performs the same function throughout the epoch of operation for the device. Devices such as the AT6000 from Atmel can support partial reloads during operation, but the reload still takes a millisecond or more due to bandwidth limitations, and cells cannot be used during the reload operation.

Dynamically Programmable Gate Arrays (DPGAs) [BDK93] [DeH94] differ from traditional FPGAs by providing on-chip memory for multiple array personalities. The configuration memory resources are replicated to contain several configurations for the fixed computing and interconnect resources. In effect, the DPGA contains an on-chip cache of array configurations and exploits high, local on-chip bandwidth to allow reconfiguration to occur rapidly, on the order of nanoseconds instead of milliseconds. Loading a new configuration from off-chip is still limited by low off-chip bandwidth. However, the multiple contexts on the DPGA allow the array to operate on one context while other contexts are being reloaded.

In this paper, we describe our minimal DPGA

[1]**Acknowledgements:** This research was supported by the Advanced Research Projects Agency of the Department of Defense under Rome Labs contract number F30602-94-C-0252.

implementation. This design represents our first generation effort and contains considerable room for optimization. Nonetheless, the design demonstrates the viability of DPGAs, underscores the costs and benefits of DPGAs as compared to traditional FPGAs, and highlights many of the important issues in the design of programmable arrays.

Our DPGA Features:

- 4 on-chip configuration contexts
- DRAM configuration cells
- non-intrusive background loading
- automatic refresh of dynamic memory elements
- wide bus architecture for high-speed context loading
- two-level routing architecture

We begin by detailing our basic DPGA architecture in Section 2. Section 3 provides highlights from our implementation including key details on our prototype DPGA IC. Section 4 summarizes the major lessons from this effort.

2 Architecture

Figure 1 depicts the basic architecture for this DPGA. Each array element is a conventional 4-input lookup table (4-LUT). Small collections of array elements, in this case 4×4 arrays, are grouped together into subarrays. These subarrays are then tiled to compose the entire array. Crossbars between subarrays serve to route inter-subarray connections. A single, 2-bit, global context identifier is distributed throughout the array to select the configuration for use. Additionally, programming lines are distributed to read and write configuration memories.

DRAM Memory The basic memory primitive is a 4×32 bit DRAM array which provides four context configurations for both the LUT and interconnection network (See Figure 2). The memory cell is a standard three transistor DRAM cell. Notably, the context memory cells are built entirely out of N-well devices, allowing the memory array to be packed densely, avoiding the large cost for N-well to P-well separation. The active context data is read onto a row of standard, complementary CMOS inverters which drive LUT programming and selection logic.

Array Element The array element is a 4-LUT which includes an optional flip-flop on its output. Each array element contains a context memory array. For our prototype, this is the 4×32 bit memory described above. 16 bits provide the LUT programming, 12 configure the four 8-input multiplexors which select each input to the 4-LUT, and one selects the optional flip-flop. The remaining three memory bits are presently unused.

Subarrays The subarray organizes the lowest level of the interconnect hierarchy. Each array element output is run vertically and horizontally across the entire span of the subarray. Each array element can, in turn, select as an input the output of any array element in its subarray which shares the same row or column. This topology allows a reasonably high degree of local connectivity.

This leaf topology is limited to moderately small subarrays since it ultimately does not scale. The row and column widths remains fixed regardless of array size so the horizontal and vertical interconnect would eventually saturate the row and column channel capacity if the topology were scaled up. Additionally, the the delay on the local interconnect increases with each additional element in a row or column. For small subarrays, there is adequate channel capacity to route all outputs across a row and column without increasing array element size, so the topology is feasible and desirable. Further, the additional delay for the few elements in the row or column of a small subarray is moderately small compared to the fixed delays in the array element and routing network. In general, the subarray size should be carefully chosen with these properties in mind.

Local Interconnect In addition to the local outputs which run across each row and column, a number of non-local lines are also allocated to each row and column. The non-local lines are driven by the

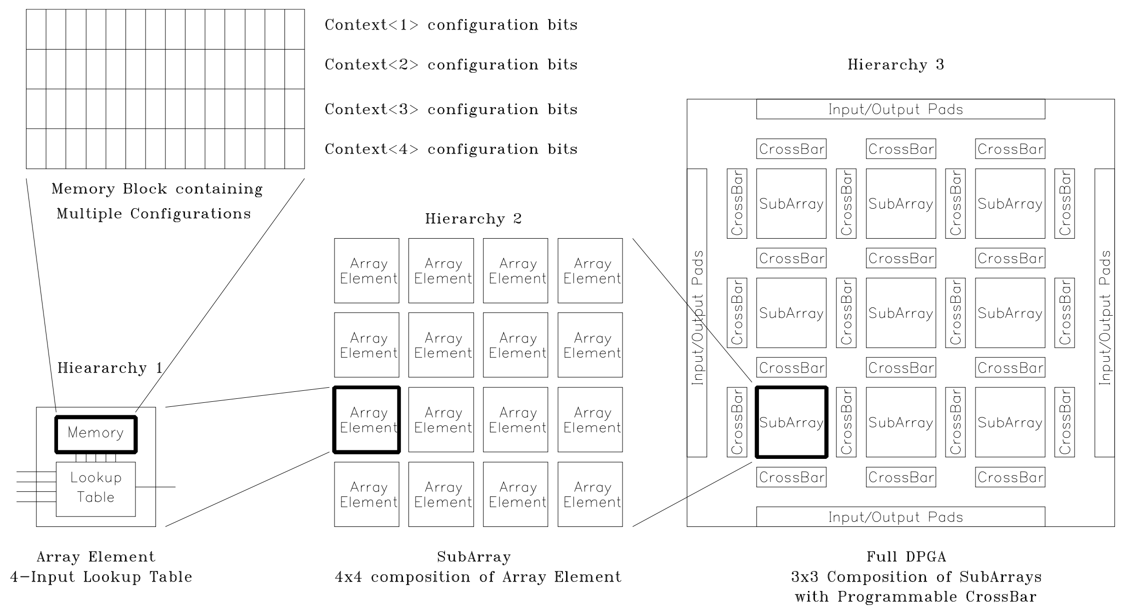
Context<1> configuration bits
Context<2> configuration bits
Context<3> configuration bits
Context<4> configuration bits
Memory Block containing
Multiple Configurations
HiearArchy 1
Memory
Lookup Table
Array Element
4-Input Lookup Table
Hierarchy 2
Array Element
SubArray
4x4 composition of Array Element
Hierarchy 3
Input/Output Pads
CrossBar
SubArray
Full DPGA
3x3 Composition of SubArrays
with Programmable CrossBar

Figure 1: Architecture and Composition of DPGA

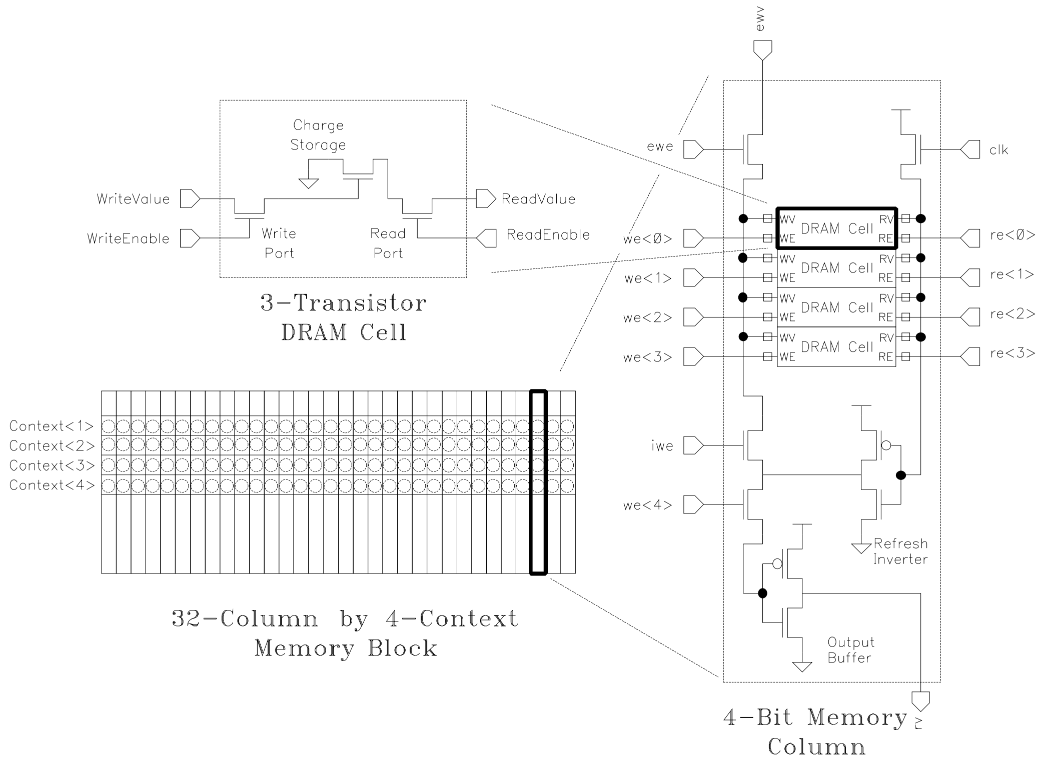

Figure 2: DRAM Memory Primitive

global interconnect. Each LUT can then pick inputs from among the lines which cross its array element. In the prototype, each row and column supports four non-local lines. Each array element could thus pick its inputs from eight global lines, six row and column neighbor outputs, and its own output. Each input is configured with an 8:1 selector as noted above.

Local Decode Row select lines for the context memories are decoded and buffered locally from the 2-bit context identifier. A single decoder services each row of array elements in a subarray. One decoder also services the crossbar memories for four of the adjacent crossbars. In our prototype, this placed five decoders in each subarray, each servicing four array element or crossbar memory blocks for a total of 128 memory columns. Each local decoder also contains circuitry to refresh the DRAM memory on contexts which are not being actively read or written.

Global Interconnect Between each subarray a pair of crossbars route the subarray outputs from one subarray into the non-local inputs of the adjacent subarray. Note that all array element outputs are available on all four sides of the subarray. In our prototype, this means that each crossbar is a 16×8 crossbar which routes 8 of the 16 outputs to the neighboring subarray's 8 inputs on that side. Each 16×8 crossbar is backed by a 4×32 DRAM array to provide the 4 context configurations. Each crossbar output is configured by decoding 4 configuration bits to select among the 16 crossbar input signals.

While the nearest neighbor interconnect is sufficient for the 3×3 array in the prototype, a larger array should include a richer interconnection scheme among subarrays. At present, we anticipate that a mesh with bypass structure with hierarchically distributed interconnect lines will be appropriate for larger arrays.

Programming The programming port makes the entire array look like one large, 32-bit wide, synchronous memory. The programming interface was designed to support high-bandwidth data transfer from an attached processor and is suitable for applications where the array is integrated on the processor die. Any non-active context may be written during operation. Read back is provided in the prototype primarily for verification.

Function	**Elements**	**Percent**	
Logic	Total		16
	Memory array	10	
	Memory decode	3	
	Fixed Logic	3	
Network	Total		66
	Memory array	15	
	Memory decode	5	
	Switching	19	
	Wiring	27	
Blank			18
Total			100

Table 2: Array Core Area Breakdown by Programmable Function

3 Implementation

The DPGA prototype is targeted for a 1μ drawn 0.85μ effective CMOS process with 3 metal layers and silicided polysilicon and diffusion. The basic composition and area for the prototype is summarized in Table 1. From Table 2, we see that 40% of the area used on the chip goes into supporting the programmable configuration memories. Table 2 also shows that 80% of the area used supports the configurable network.

Layout Inefficiencies The prototype could be packed more tightly since it has large blank areas and large areas dedicated to wire routing. A more careful co-design of the interconnect and subarray resources would eliminate much or all of the unused space between functional elements. Most of the dedicated wiring channels are associated with the local interconnect within a subarray. With careful planning, it should be possible to route all of

Unit	Size	Composition
Die	6.8mm$\times$6.8mm	Core with pads
Core	5.6mm$\times$4.7mm	All internal logic except pads
Array Core	5.25mm$\times$4.4mm	3$\times$3 subarrays including crossbars (no pads)
Subarray+crossbar tile	1460$\mu\times$1750μ	Subarray + 4 adjacent 16$\times$8 crossbars and memory
Crossbar	495$\mu\times$270μ	Crossbar including memory
Subarray	1150$\mu\times$1400μ	4 $\times$ 4 Array Elements + 4 Local Decodes
Local Decode	253$\mu\times$167μ	
Array Element	275$\mu\times$240μ	Includes local routing channels

Table 1: Basic Component Sizes for Prototype

Element	#	Size
DRAM Cell	4	7.6$\mu\times$19.2μ
Output Buffer	1	7.6$\mu\times$28.0μ
Pass Gates	1	7.6$\mu\times$26.4μ
Column		7.6$\mu\times$131.2μ

Table 3: DRAM Column Breakdown

these wires over the subarray cells in metal 2 and 3. As a result, a careful design might be 40-50% smaller than our first generation prototype.

Memory Area From the start, we suspected that memory density would be a large determinant of array size. Table 2 demonstrates this to be true. In order to reduce the size of the memory, we employed a 3 transistor DRAM cell design. To keep the aspect ratio on the 4$\times$32 memory small, we targeted a very narrow DRAM column. Unfortunately, this emphasis on aspect ratio did not allow us to realize the most area efficient DRAM implementation (See Table 3).

One key reason for targeting a low aspect ratio was to balance the number of interconnect channels available in each array element row and column. However, with 8 interconnect signals currently crossing each side of the array element, we are far from being limited by saturated interconnect area. Instead, array element cell size is largely limited by memory area. Further, we route programming lines vertically into each array element memory. This creates an asymmetric need for interconnect channel capacity since the vertical dimension needs to support 32 signals while the horizontal dimension need only support a dozen memory select and control lines.

For future array elements we should optimize memory cell area with less concern about aspect ratio. In fact, the array element can easily be split in half with 16 bits above the fixed logic in the array element and 16 below. This rearrangement will also allow us to distribute only 16 programming lines to each array element if we load the top and bottom 16 bits separately. This revision does not sacrifice total programming bandwidth if we load the top or bottom half of a pair of adjacent array elements simultaneously.

Timing Table 4 summarizes the key timing estimates for the DPGA prototype at the slow-speed and nominal process points. As shown, context switches can occur on a cycle by cycle basis and contribute only a few nanoseconds to the operational cycle time. Equation 1 relates minimum achievable cycle time to the number of LUT delays, n_l, and crossbar crossings, n_x in the critical path of a design.

$$t_{cycle} = t_{mem} + n_l \cdot t_{lut} + n_x \cdot t_{xbar} \qquad (1)$$

These estimates suggest a heavily pipelined design which placed only one level of lookup table logic ($n_l = 1$) and one crossbar traversal ($n_x = 1$) in each pipeline stage could achieve 60-100MHz operation allowing for a context switch on every cycle.

Path	Delay		
	symbol	slow-speed	nominal
CLK→configuration memory stable	t_{mem}	4 ns	2.5 ns
CLK→XBAR out	t_{xbar1}	8.5 ns	5 ns
XBAR in→XBAR out	t_{xbar}	4.5 ns	2.5 ns
LUT in→LUT output (1 level)	t_{lut}	9 ns	3.5 ns
CLK→CLK (maximum, DRAM leakage)	$t_{clk_{max}}$		200 ns

Table 4: Estimated Timings

Our prototype, however, does not have a suitably aggressive clocking, packaging, or i/o design to actually sustain such a high clock rate. DRAM refresh requirements force a minimum operating frequency of 5MHz.

4 Conclusions

We have presented the full design for a first-generation DPGA prototype. The prototype demonstrates that efficient, dynamically programmable gate arrays can be implemented which support a single cycle, array-wide context switch.

Multiple context programmable gate arrays make the most sense when moderately small amounts of additional memory area can increase the amount of logic available for use. While the configurable memory is clearly a major element of programmable array size in our 4-context prototype, the memory component does not dominate fixed logic area. With the current memory cell design, the tradeoff is worth considering, but not necessarily compelling. If the basic memory cell were much larger relative to the fixed logic, the memory area would dominate and the multiple context support would make little sense. Consequently, SRAM-based DPGAs may not be desirable in the architecture described here.

If the basic memory cell were much smaller relative to the fixed logic, the logic area would dominate making the cost of multiple contexts marginal and making it sensible to include even more on-chip contexts. DPGAs with a more aggressive dynamic memory cell look appealing. Further, DPGAs based on native DRAM cells or flash memories look very attractive in this architecture.

References

[Atm94] Atmel Corporation, 2125 O'Nel Drive, San Jose, CA 95131. *Configurable Logic Design and Application Book*, 1994.

[BDK93] Michael Bolotski, André DeHon, and Thomas F. Knight Jr. Unifying fpgas and simd arrays. Transit Note 95, MIT Artificial Intelligence Laboratory, September 1993.

[DeH94] André DeHon. DPGA-Coupled Microprocessors: Commodity ICs for the Early 21st Century. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, April 1994.

[Xil93] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *The Programmable Logic Data Book*, 1993.