Embedded DRAM for a Reconfigurable Array

Stylianos Perissakis, Yangsung Joo[1], Jinhong Ahn[1], André DeHon, John Wawrzynek

Computer Science Division
University of California, Berkeley
{sper,johnw,amd}@cs.berkeley.edu

[1]DRAM Design 5 Team
LG Semicon, Cheongju, Korea
{jooys,jhahn}@lgsemicon.co.kr

Abstract

A field-programmable gate array, coupled with an on-chip 2 Mb DRAM bank has been designed, to aid in the study of the tradeoffs involved in the design of embedded DRAM for FPGAs. The memory can be used both as configuration storage, enabling reconfiguration in under 5 μs, and application data memory, providing application logic executing on the array with up to 2 GB/sec data bandwidth. The variable latency of the DRAM is hidden from the logic by a stall mechanism and an SRAM-like interface.

Introduction

With the increasing number of gates on a single chip, it is necessary to integrate a large amount of memory on the die as well, as is evident from the large caches on most microprocessors [1]. FPGAs in particular lag other architectures in the amount of on-chip memory, mainly because of their initial application as "glue logic", where little or no memory was needed. Devices available in the market today provide limited amounts of fine-grain SRAM, currently up to 96 blocks of 4 Kbits[1]. This is insufficient for the data sets of many applications, which makes it necessary to manage the internal memory as a cache and store the full data set in external memory, accessible through the low bandwidth external interface.

In addition, when an FPGA needs to be reconfigured "on the fly", the reconfiguration time can be a performance limitation. For example, in common speech recognition systems, a signal processing front end, a multilayer perceptron and a Viterbi decoder must alternate at frame rate, typically 800 Hz. If the configuration bitstreams reside in external memory, the low bandwidth of the external interface makes this alternation impractical. If, on the other hand, bitstreams are preloaded in an internal memory, improvement by an order of magnitude is possible, making this model of computation possible.

DRAM, with an order of magnitude higher density than SRAM, provides the opportunity to integrate large memory blocks on-chip. This however comes at the expense of increased latency and somewhat degraded logic performance. *Trumpet* is a test chip that we designed in order to study the tradeoffs involved in the design of reconfigurable arrays coupled with coarse-grain DRAM banks.

[1]Xilinx XCV1000

The design

A. Trumpet

Our long-term architecture goal consists of a network of compute pages (configurable subarrays) and memory pages (Configurable Memory Blocks, or CMBs). Compute pages are based on a 5-input lookup table (5-LUT) logic block. Compute and memory pages are interconnected by a fat pyramid network [2] – a fat tree with additional nearest neighbor connections, or "shortcuts". This network extends into each subarray, reaching individual logic blocks at the leaves, while the root serves as a connection to an on-chip microprocessor. The design relies heavily on pipelining to maintain a high clock rate. Many of the switches in the network are registered, so it takes multiple cycles for a signal to travel across the chip. Retiming registers within each logic block, as well as at the CMB inputs, are used to balance the delays between signals that travel different distances [3].

Each subarray, CMB and switchbox requires one or more application-dependent configuration bitstreams. Such bitstreams can be preloaded in the DRAM within the CMBs and loaded on demand into their respective destinations at the full DRAM bandwidth. Subarrays and CMBs are arranged in pairs for configuration purposes, while configuration of each switchbox can be assigned to the nearest CMB. In this way it is possible to achieve partial or full reconfiguration in the time it takes to configure one subarray, one CMB and one or two switchboxes. Of course, it is also possible to configure any of the above from the external interface, at a slower rate. In addition, state bitstreams for both CMB and subarray can be loaded or dumped in a similar fashion for the purposes of initialization, diagnostics and context switching.

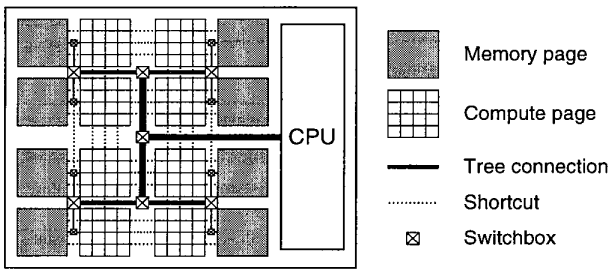

Figure 1: Architecture overview

Table 1: TEST CHIP CHARACTERISTICS

Chip size	10 x 5 mm^2
Feature size (DRAM/logic)	0.25/0.35 μm
Supply voltage (DRAM/logic)	2.1/2.5 V
Target cycle time (DRAM/logic)	8/4 ns
DRAM capacity	2 Mbits
Logic capacity	64 5-LUTs
Reconfiguration time (CMB/subarray)	0.5/4.5 μs

For data access, each subarray can reach the nearest CMB via the shortcuts or tree connections, or any remote CMB via the tree. Routes are determined by the switchbox configurations and are static for the lifetime of a configuration.

Trumpet consists of a single 64 logic block subarray and a single 2 Mbit DRAM bank. While it does not expose the interactions of multiple compute and memory pages, it provides a means to evaluate some of the related architecture and circuit design issues. Trumpet is currently in fabrication and is expected back in March of 1999.

B. DRAM macro

The CMB design is based on a 2 Mbit DRAM macro, designed on a triple well, 0.25 μm, 64 Mbit DRAM process. Measuring 3.7 by 2.4 mm^2, it is organized as 2 subbanks, each 1K rows by 1K bits. It has a 128-bit wide SDRAM interface, running at 125 MHz. A row buffer in each subbank can hold a row of data while the subbank is being precharged. Row and column latencies are 5 and 2 cycles respectively. For the logic design, the design rule is relaxed to 0.4 μm (0.35 μm gate length), while 4 layers of metal are available.

C. CMB logic

The CMB consists of the DRAM macro and an additional layer of logic, that implements the functionality needed by the subarray. The CMB can operate in two basic modes. After reset and initialization, it will accept commands from the external interface and execute block transfer operations for the purpose of configuration load/dump, state load/dump or I/O. The **RUN** command will set the CMB into the application execution mode, in which it can receive and process read/write requests from the logic running in the subarray.

An internal address register, the *Local Address* is used as the running address for the block transfers. A command from the external interface can be used to load the starting address. The Local Address is autoincremented after each word transferred. A 64-bit data bus, the "global data" bus, shared by the CMB, the subarray and the external interface, is used for the data transfer.

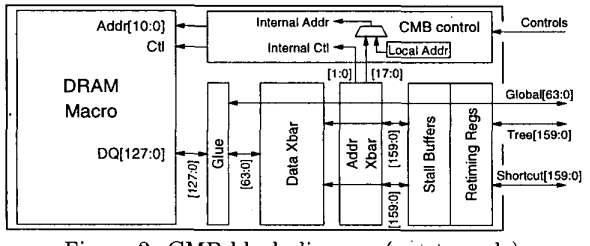

Figure 2: CMB block diagram (not to scale)

After a configuration and initial state have been loaded into both the subarray and the CMB, the application can start executing. At that time, the local address and the global data bus are disconnected and the address/control source and data source/sink are mapped onto the tree or shortcut wires, directly accessible from the subarray. Separate read and write data buses are provided. The width of each data bus is configurable, from 0 to 8 bytes wide, with a maximum combined width of 80 bits. The address width can range from 4 to 18 bits, depending on the total address space in use by the application and the required addressing granularity.

Out of each addressed word, each bit must be routed independently to any logic block in the compute page, depending on the layout of the logic. The tree network is designed so that any connection can be made by traversing the tree up towards the root, then going down towards the destination logic block. This scheme works well for connections within the compute page, but for connections between the memory and the subarray, one part of the route has to be completed in the CMB. Initial studies indicated that the area required to replicate the tree in the CMB would be quite large. Therefore an alternative structure was pursued. A routing network consisting of three stages of crossbar switches, combined with 2:1 and 4:1 (de)multiplexers was chosen. This structure is not entirely non-blocking, but it is possible to choose compatible routes to the logic blocks in a simple procedural way. The latency through this network does not exceed a single cycle.

A similar but smaller and somewhat simpler structure is used to route address and control bits from the subarray to the CMB logic. The internal address is always 18 bits wide, addressing the entire memory space to the byte. When the application provides fewer address bits, the uppermost bits are filled in from the Local Address register. This provides a basic means of relocation: a base address

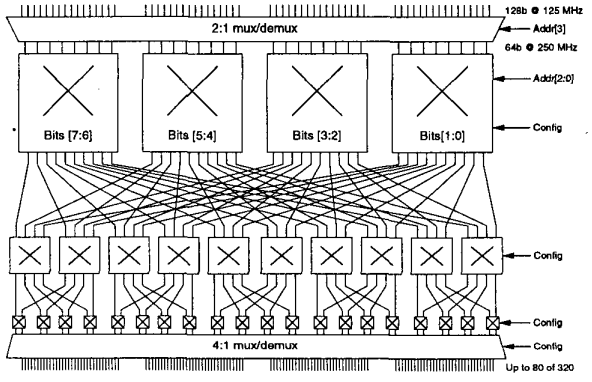

Figure 3: Data crossbar block diagram

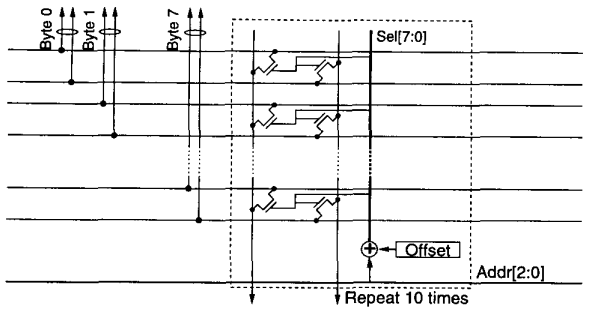

Figure 4: Data crossbar detail

is loaded into the Local Address register, while the offset bits are provided by the application.

The target clock frequency for the logic is twice that of the DRAM macro, or 250 MHz. Sequential accesses can be performed internally at the full width of 128 bits at the DRAM clock rate and "folded" to 64 bits at the logic clock rate. The 2:1 multiplexer shown in figure 3, controlled by one address bit, performs this folding. Back-to-back random accesses must be aligned to the rising edge of the DRAM clock, reducing the sustained data rate by a factor of 2.

When the addressing granularity is less than 64 bits, the appropriate number of least-significant address bits, `Address[2:0]`, are used to index a sub-word of the appropriate width. This is implemented by the uppermost crossbars in figure 3. Each of these 4 crossbars routes 2 bits out of every byte accessed. To reduce the number of configuration bits and control wiring required, each bit pair is routed as a unit. Pairs can be split in the final crossbar stage. As shown in figure 4, each vertical slice of the upper crossbars is controlled by a configuration word, the *byte offset*, and the appropriate number of address bits. The byte offset indicates which byte of the accessed word, if any, will be routed over this slice. When a nonzero address is applied, it is added to the byte offset, selecting the appropriate byte.

To ease application development, the interface provided to the logic is similar to a synchronous pipelined SRAM. A request and read/write signal are mapped onto a pair of wires at the interface. The row buffers of the DRAM macro are used as a simple row cache. When the requested word resides in one of the row buffers, it is returned within the minimum latency of six logic cycles. This consists of the two DRAM cycle latency of the macro, plus two logic cycles because the inputs and outputs of the CMB are registered[2]. If the requested word is not in the row buffer, the appropriate subbank is precharged, the new row is accessed and the requested word is returned, after a total latency of 16 cycles. This variable latency is hidden from the application logic with a stall mechanism. Whenever a row miss occurs, the subarray clock is masked for 10 cycles, so the logic will still experience a 6-cycle latency

[2]Simulations have shown that the CMB controller will not meet the 4ns cycle target in this test chip. An additional stage in the pipeline is required to meet the target, with a latency of 7 cycles at 250 MHz.

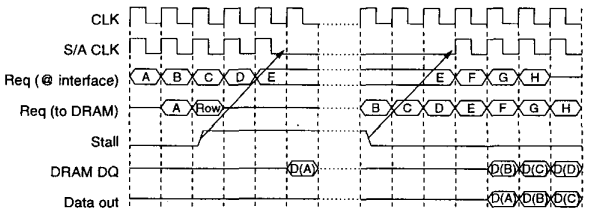

Figure 5: Stall timing (assuming sequential accesses)

in terms of its gated clock.

Given the high operating frequency, the stall signal will need more than a clock cycle to reach the subarray and take effect. This delay is 2 cycles in Trumpet and would be longer in larger chips. In the meantime, the clock in the subarray is running and more requests are generated. These are buffered in a set of *input stall buffers*, to be processed after the current row miss has been completed. Data returned from the memory while the subarray is stalled is buffered in a set of *output stall buffers* until the subarray resumes. The 6-cycle read latency sets the required depth of the stall buffers to 6 slots. As shown in figure 5, request B results in a row miss. Before the subarray clock is stalled, requests C, D and E have been generated. These are buffered for processing after B. In the meantime, the data from request A is returned by the macro. This is also buffered, to be returned after the proper latency of 6 subarray clock cycles.

Configuration bits are implemented as 5T SRAM cells, linked into a 162-bit wide, 20-deep shift register. During a configuration load from DRAM, 3 successive 64-bit words are demultiplexed to a 192-bit configuration word (including 30 don't care bits). Every 3 cycles a new word is pushed into the shift register. Configuration of the CMB can thus be achieved in under 0.5 μsec (with some overhead cycles), when the bitstream is preloaded in the DRAM. Configuration from the external interface is limited by the external bandwidth, 125 MB/sec, resulting in 4 μsec configuration time.

The DRAM macro clock is derived from the logic clock by simple division by 2. This results in some skew, with the DRAM clock lagging, so hold time is a concern. The DRAM clock is buffered internally in the macro, resulting in a hold time requirement of 1 ns for all inputs. The clock division adds about 400 ps to that. To guarantee the hold requirement, all DRAM macro inputs are latched with level- sensitive latches, controlled by the logic clock. This guarantees a 2ns hold time, at the expense of about 350 ps of latch delay.

Evaluation

The motivation behind using DRAM was the high density it offers. However, the average density of the CMB is far from typical DRAM densities. Its overall dimensions are 5.05 by 2.6 mm^2, yielding an average density of 156 Kbits/mm^2. Compared to typical DRAM densities, in the 700 Kbit/mm^2 to 1 Mbit/mm^2 range for similar technologies, we see a difference by a factor of about 5 to 6. This difference is mainly because of i) the area required to im-

plement the CMB functionality, and ii) the small capacity of the macro, which magnifies the overhead caused by the macro controller and CMB logic (about 40% and 30% of the total area respectively).

There is definitely room for improvement in these numbers. There are significant inefficiencies arising from the fact that the DRAM macro and the CMB have been designed independently, with the macro being a "general purpose" component. In fact, much of the macro functionality is not used in the CMB. In an alternative implementation, the macro and CMB controllers could be combined and unused functionality removed, resulting in a much smaller control overhead and increased density.

In a possible SRAM implementation, assuming a typical 48 Kbits/mm^2 for the SRAM core [4] [5], it would be possible to fit 192 Kbits of SRAM in the same area as the DRAM core. The macro and CMB controllers would be much simpler, while the CMB datapath would also be somewhat simplified, mainly because of the absence of stall buffers. For a rough estimate of the area of the SRAM implementation, assume that the controller takes one third the area of the combined DRAM and CMB controllers, and the 0.65 mm^2 taken by the stall buffers (1.85 by 0.35) are saved. This results in an overall density of 25 Kbits/mm^2. Although this is an approximate number, it shows that for the given functionality there is still about an order of magnitude difference between SRAM and DRAM implementations.

The price to pay for this density advantage is the increased latency, which frequently results in reduced bandwidth available to the application logic. The effect on bandwidth is largely dependent on the access patterns. For example, during a configuration load, the accesses are sequential read-only. The state machine in the CMB controller prefetches rows, so that no stalls occur and the full bandwidth is sustained. If the access pattern is completely random, so that almost every access requires a row fetch, the bandwidth will be limited by the row cycle, 7 DRAM clock cycles.

In many cases the access patterns are predictable, frequently consisting of sequential segments. This gives the programmer the opportunity to arrange the data layout so that few stalls occur. In many image processing applications for example, pixels are accessed sequentially in row- or column-major order. Such access patterns will result in stalls only at the DRAM page boundary.

Another situation where the read latency is exposed is a sequence of alternating reads and writes. Although separate read and write buses are used in the subarray and the CMB interface, a single bidirectional datapath is used internally in the macro. So, a write has to wait for the previous read to complete, to avoid contention. This situation is handled by stalling, resulting to execution of a read and a write every 8 cycles (equal to the read latency plus 2). It is possible of course to improve on this by rearranging the access pattern, but this requires larger programming effort and possibly more logic or memory resources. Potential hardware support could consist of reduced read latency and/or separate read and write datapaths in the DRAM macro – possibly at the expense of having only half the width available to each direction. To overcome this width limitation, a single configuration bit could be used to select between a wide bidirectional bus and two narrower unidirectional buses, depending on the intended access pattern.

An additional source of bandwidth loss is refresh overhead. In Trumpet refresh is handled by the external control. At the appropriate interval, operation is halted and one or more rows of the DRAM are refreshed. Based on earlier characterization of this process and with expected high junction temperature caused by the logic operation, a retention time of 8 to 16 μsec is expected. With 2K refresh cycles and some overhead for halting and restarting the application, this results in about 2.5% to 5.0% reduction in the available bandwidth. More accurate data will be available once the testchip is characterized. However, in many cases, like FIFOs and temporary buffers, refresh is not required if the lifetime of the data is shorter than the retention time. In such cases refresh can be omitted.

Conclusion

Trumpet is a first approach to a memory-rich reconfigurable architecture. Large memory banks make it possible to store configurations on-chip, enabling rapid run-time reconfiguration. In addition, application kernels can benefit from high bandwidth access to data.

It is important for application development that the complexity of the DRAM is hidden from the programmer. We have demonstrated a simple SRAM-like interface that provides a familiar abstraction to the application developer.

Although no claim is made that Trumpet represents exactly the right balance between processing and memory resources, we believe that it is a step to the right direction. Experience gained by using it in real problems will be invaluable for a better understanding of the tradeoff between processing and memory.

References

[1] C.E.Kozyrakis, et al. "Scalable processors in the billion-transistor era: IRAM." *IEEE Computer*, pages 75–78, September 1997.

[2] R.I.Greenberg. "The fat-pyramid: A robust network for parallel computation." In *Advanced Research in VLSI*, pages 195–213, 1990.

[3] W.Tsu, et al. "HSRA: High-speed, hierarchical synchronous reconfigurable array." In *FPGA*, 1999.

[4] K.Nakamura, et al. "A 500MHz 4Mb CMOS pipeline-burst cache SRAM with point-to-point noise reduction coding I/O." In *ISSCC*, page 406, 1997.

[5] K.Ishibashi, et al. "A 300MHz 4-Mb wave-pipelined CMOS SRAM using a multi-phase PLL." In *ISSCC*, page 308, 1995.