Power-aware RAM Mapping for FPGA Embedded Memory Blocks

Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA, USA
tessier@ecs.umass.edu

Vaughn Betz, David Neto
Altera Toronto Technology Centre
151 Bloor St, Suite 200
Toronto, ON, CANADA

Thiagaraja Gopalsamy
Altera Corporation
101 Innovation Drive
San Jose, CA, USA

ABSTRACT
Embedded memory blocks are important resources in contemporary FPGA devices. When targeting FPGAs, application designers often specify high-level memory functions which exhibit a range of sizes and control structures. These logical memories must be mapped to FPGA embedded memory resources such that physical design objectives are met. In this work a set of power-aware logical-to-physical RAM mapping algorithms are described which convert user-defined memory specifications to on-chip FPGA memory block resources. These algorithms minimize RAM dynamic power by evaluating a range of possible embedded memory block mappings and selecting the most power-efficient choice. Our automated approach has been integrated into a commercial FPGA compiler and tested with 40 large FPGA benchmarks. Through experimentation, we show that, on average, embedded memory dynamic power can be reduced by 21% and overall core dynamic power can be reduced by 7% with a minimal loss (1%) in design performance.

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Design Aids

General Terms
Algorithms

Keywords
FPGA, Embedded memory block, Dynamic power

1. INTRODUCTION
On-chip memory is an essential component of programmable logic devices. Most on-chip data storage is implemented in large RAM blocks integrated into the FPGA architecture. These storage blocks allow for the implementation of a variety of memory structures, including FIFOs, scratch pad memories, and shift registers, within close physical proximity of logic resources. Due to their extensive use, embedded memory blocks have been found to consume between 10-20% of core dynamic power in typical FPGA designs [1]. As the amount of FPGA logic and on-chip memory grows rapidly over the next few years, the power-efficient use of memory blocks will become increasingly important.

FPGA'06, February 22-24, 2006, Monterey, California, USA.

Embedded memory blocks in contemporary FPGAs are typically implemented with synchronous SRAM [2][17] to improve design performance. Like other synchronous SRAM architectures, FPGA embedded memory accesses are performed in concert with a design clock and a series of interface signals including read/write (R/W) enables, clock enables, address, and data signals. During application development, designers may directly specify the source of control signals that are used to manipulate design RAMs. More typically, a higher-level RAM representation is specified and automatically converted to physical RAMs and associated control circuitry. Control signals, such as R/W enable and clock enables are generated by this control circuitry.

Synchronous FPGA embedded memories primarily consume dynamic power as a result of internal RAM clocking. To save power, RAM control signals can be configured to suppress internal clocking when RAM access is unnecessary on a specific clock cycle. Although user-defined or generated control signals provide for valid functional embedded memory behaviour, their configuration may not efficiently suppress unnecessary clocked memory accesses, leading to wasted RAM dynamic power. These limitations motivate a need for RAM mapping algorithms that take power objectives into account while maintaining valid functional behavior.

In this paper we describe a series of algorithms to automatically map user-specified *logical* memories to available *physical* embedded memory block resources with the goal of reducing overall FPGA dynamic power consumption. In considering feasible RAM mappings, our approach estimates the relative dynamic power consumption of each potential implementation and selects the most power-efficient implementation subject to on-chip RAM availability constraints. When necessary, user-specified RAM control signals (R/W enable, clock enables) are remapped to achieve a logically-equivalent RAM implementation with reduced dynamic power consumption. If an FPGA contains embedded memory blocks of different sizes, a mapping using each block type is considered.

Our mapping techniques have been integrated into the Altera Quartus II synthesis system [1] and targeted to a variety of Altera FPGA families which contain embedded memories. Through experimentation with 40 RAM-based Stratix II designs, we show

an average embedded memory dynamic power reduction of 21% and overall core dynamic power reduction of 7%.

In the next section we discuss related power-aware memory mapping techniques. In Section 3 the basic operation of FPGA embedded memories are described along with details of the basic mapping flow used to translate user-specified logical memory to physical embedded memory blocks. Section 4 provides the details of our power-aware RAM mapping techniques and supporting algorithms. Experimental results are presented in Section 5. Section 6 concludes the paper and offers directions for future work.

2. Related Work

We are unaware of any prior CAD tools that produce a power-efficient mapping of design RAM to FPGA or structured ASIC embedded memory. Previous research efforts that map design logic to embedded memory blocks in ASICs [4][14] and FPGAs [9] do not consider power optimization as a mapping goal. Although FPGA logic and routing dynamic power reduction has been studied [10], these techniques were not applied to embedded memory blocks.

RAM dynamic power-reduction techniques for ASICs and microprocessor systems have been considered at the application-mapping, compiler, and circuit levels. Although these approaches provide insight into reducing FPGA embedded memory power, none are directly applicable. Several synthesis techniques for application-specific embedded systems create power-optimized memory structures based on application address traces. In Benini et al. [5], the memory trace of an embedded application is analyzed by an algorithm to determine the portion of program and data memory that is most frequently accessed. These addresses are then grouped into memory banks which are implemented with scratch pad memories. Infrequently accessed addresses are grouped into larger physical memory blocks. Later work by Cao et al. [6] extends this optimization to consider data width scaling. Wuytack et al. [16] have developed techniques to optimize the entire memory hierarchy of an application for power consumption based on application information. These previous approaches rely on application trace information to perform memory partitioning.

A number of compiler techniques have been developed for processor-based systems which optimize power while mapping data to fixed system memory resources. For example, in Unsal et al. [15], a series of memory locations for multimedia applications are remapped to a small, local scratch pad memory to save dynamic power. In Petrov and Orailoglu [13], the organization and power consumption of a translation look-aside buffer are adjusted on a per-application basis. In Gebotys [8], memory energy is managed through memory and register allocation using a network flow algorithm. In Ferrahi et al. [7], a compiler technique to optimize sleep mode operation for memories is described. Memory reactivations are minimized via scheduling to save dynamic power.

Numerous circuit-level techniques for power reduction have been explored [11] including reduced swing pre-decode lines, multi-stage address decoding, and divided word and bit lines, among others. These techniques may be used in the future by FPGA designers to reduce FPGA embedded memory block power and are additive to the approaches described in this paper.

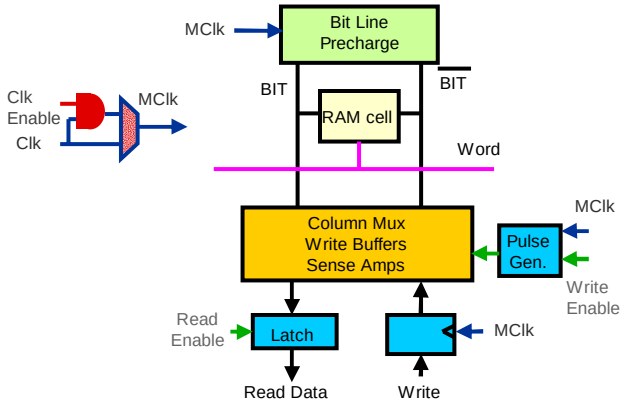

Figure 1: Internal view of embedded memory read/write port

3. Background

The development of a power-efficient embedded RAM mapping strategy requires insight into the internal behaviour of synchronous SRAM. Typically, each port of an embedded memory block is controlled by one or more read/write (R/W) enable signals, clock (Clk) enable signals, and clock signals. As shown in Figure 1, these signals directly or indirectly control data movement in different parts of the embedded memory port.

During a typical memory **read** operation the following events occur in sequence, in response to a rising clock edge:

- The memory port clock (MClk) is strobed causing the BIT lines to be precharged to Vcc.
- The read address is decoded and one word line is activated.
- The BIT line difference is identified by sense amps causing the read data to be strobed into a column multiplexer.
- Read data passes through the column multiplexer and a latch conditioned by Read Enable to the RAM external Read Data lines.

Memory **write** operations require a similar sequence of operations which occur in the following order:

- The memory port clock (MClk) is strobed causing the BIT lines to be precharged to Vcc.
- The Write Enable signal, conditioned by MClk, creates a write pulse which transfers write data to the write buffers and a word line is activated following write address decode.
- Write buffer data is stored in the RAM cells

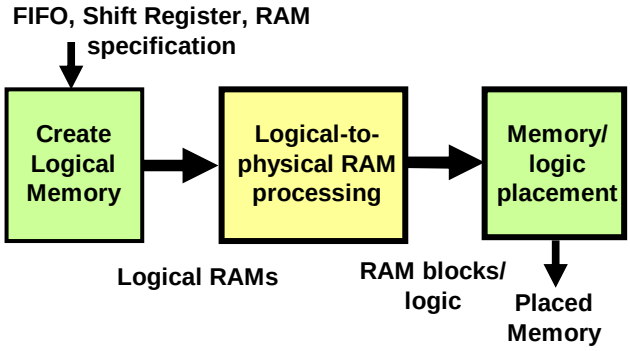

Figure 2: Typical Logical RAM to Embedded Memory Block Mapping Flow

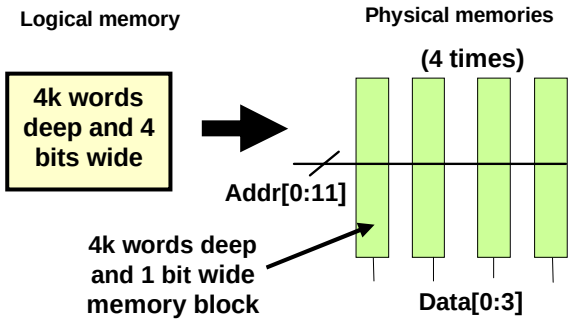

Figure 3: Area-efficient mapping of a 4Kx4 logical RAM to 4 Kbit memory blocks

For both synchronous read and write RAM operations, most dynamic power is consumed via BIT line precharging [12]. To control clocking, embedded memory ports often have a clock enable signal which can eliminate internal precharging, word-line decoding, and RAM cell access. The disabling of the clock enable signal when memory port access is not required provides the best technique to eliminate embedded memory dynamic power consumption for a memory port. If a RAM port is inactive on a given clock cycle and its clock can be suppressed via an inactive clock enable, the RAM port will not consume significant dynamic power.

A number of contemporary FPGAs support embedded memory blocks with R/W enable and clock enable signals. Altera Stratix [3] and Stratix II [2] devices support both R/W enables and clock enables on each port of TriMatrix embedded memory block dual-port memory. Each Xilinx Virtex-II [18] and Virtex-4 [17] embedded SelectRAM block contains write enable and clock enable control signals on each port, but no separate read enable.

The goal of power-aware RAM mapping is to implement the functionality of a user-defined RAM module (logical memory) in one or more FPGA embedded memory blocks so that memory precharges are limited. This optimization goal attempts to minimize RAM dynamic activity through the use of RAM port clock enables whenever possible. The effective use of clock enable signals ensures that the bulk of embedded memory block dynamic power is consumed when a *required* access to data within a RAM is performed. In some cases this goal may require the synthesis of one of more clock enable signals during the mapping process. This mapping must achieve the same functional behaviour for the RAM as specified by the designer while allowing for possible tradeoffs regarding design power consumption and design area and performance.

3.1 Typical FPGA RAM Mapping Flow

FPGA embedded memory blocks are used to implement a variety of RAM components including FIFOs, shift registers, and single and dual-port memories. Logical RAMs are specified by the designer in RTL or schematic form, created by the FPGA compiler and mapped [1], as shown in Figure 2:

1. **Logical memory creation** – User-defined RAM descriptions are processed by the FPGA compilation software to create logical memories with desired characteristics.
2. **Logical-to-physical RAM processing** - Logical RAMs are converted into one or more RAM blocks which match the external interface and size constraints of available embedded memory blocks.
3. **Embedded memory block placement** – RAM blocks and associated control logic are assigned to available on-chip embedded memory block and logic resources.

The power-aware algorithms developed in this work are applied in the logical-to-physical RAM processing step. Traditionally, RAM mapping has targeted logical RAM performance and FPGA area minimization [9] rather than power consumption. To conserve dynamic power it is desirable to map memory functions specified by designers to available physical memories so that power consumption is optimized within area and delay constraints. As shown in Section 4.2, an area-optimal embedded memory implementation does not always consume the least amount of dynamic power.

The size of both logical and physical (embedded) memory blocks can be defined in terms of the number of addressable locations (*depth*) and output bits per memory (*width*). The number of address bits required for both logical and physical memories is directly related to memory block depth. The number of data in and data out bits is related to memory block width. To promote flexibility, an FPGA embedded memory block may typically be programmed to support a range of depth versus width configurations [2][17].

Until the relatively recent adoption of synchronous SRAMs, most user-defined RAM designs targeted asynchronous memories (both external and internal to FPGAs) which use read and write enable for data access control. Although embedded memory blocks now allow for the use of either operation-specific enable or clock enable signals to provide access control, many designers continue to use the operation-specific enable approach, ignoring the clock enable. Contemporary RAM mapping flows (e.g. Figure 2) automatically map these user-defined enable signals to the R/W enable signals located on the embedded memory block ports. Unspecified clock enables are set to be continuously active. The use of read and write enable signals for data access control instead of associated clock enable signals leads to sub-optimal power consumption in many cases.

A second impediment to reduced RAM power dissipation is related to logical RAM size. In most cases the size of a user-

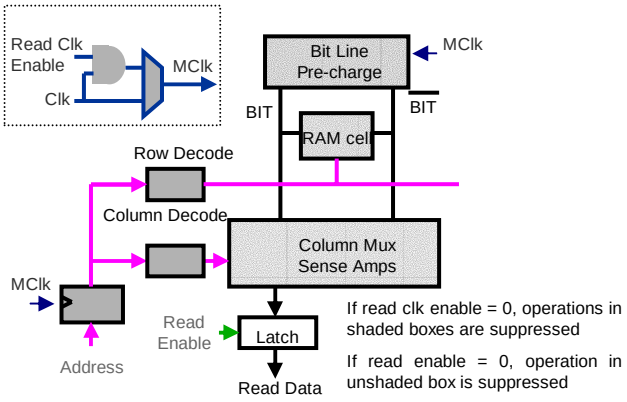

Figure 4: Functional equivalence of embedded memory read enable and read clock enable

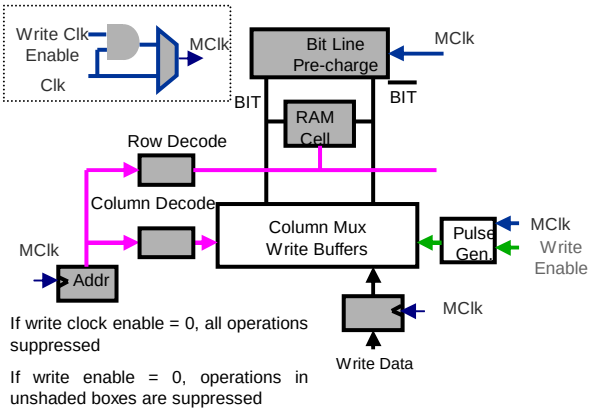

Figure 5: Functional equivalence of embedded memory write enable and write clock enable

specified logical memory will not exactly match the width and depth dimensions of an embedded memory block. Since existing RAM mapping flows focus on optimizing delay and resource usage, rather than power, logical memories are typically mapped using a minimum of external logic. As an example, Figure 3 illustrates the mapping of a 4Kx4 logical memory to four 4Kx1 embedded memory blocks. In this case, each memory block is configured as 4Kx1 so that a single bit of each addressable location is located in each block. This configuration requires no external logic. However, all four memory blocks must be active during each logical memory access, so this is a high-power implementation.

4. Power-Aware RAM Mapping

Our RAM mapping approach consists of two algorithms that obtain a power-efficient mapping of logical memories to FPGA embedded memory blocks. Two specific cases are targeted:

1. Since most embedded memory block dynamic power is a result of clock-induced precharging, we identify cases where user-specified logical RAM read and write enable signals can be automatically converted to or combined with corresponding read and write clock enable signals while maintaining correct functional behaviour.

2. For cases where more than one embedded memory block is required to implement a logical RAM, we implement a multi-banked RAM mapping. As a result of this banked mapping, only one embedded memory block is clocked per access. In some cases the banked structure may require the inclusion of supporting logic.

4.1 Conversion of read and write enable to read and write clock enable

In general, synchronous embedded memory blocks exhibit the same functional RAM behaviour if either an enable or a clock enable is used to control a read (or write) access and the alternate signal is set to an active state. To illustrate this observation, all four configurations of active-high enable and clock enable signals are considered for read and write accesses. For a successful read (or write) access both enable and clock enable signals must be set to active-high.

The functional equivalence of embedded memory read enable and read clock enable can be observed in Figure 4 based on the discussion of RAM read steps in Section 3. Using the figure, the behaviour of the following four read cases can be considered:

1. Read Clk Enable = 0, Read Enable = 0 – New data will not be transferred to the column multiplexer since the BIT lines are not precharged.
2. Read Clk Enable = 1, Read Enable = 0 – New data is obtained from the RAM cell following BIT line precharge but will not be transferred to the Read Data lines since the latch conditioned by Read Enable is closed.
3. Read Clk Enable = 0, Read Enable = 1 – New data will not be transferred to the column multiplexer since the BIT lines are not precharged. Indeterminate data passes through the latch and is driven onto the Read Data lines.
4. Read Clk Enable = 1, Read Enable = 1 – New data is obtained from the RAM cell following bit line precharge and passed through the latch to the Read Data lines.

Consider a scenario where Read Enable is attached to a control signal and Read Clk Enable is always tied to active logic 1. From the enumeration it can be seen that since the AND of Read Enable and Read Clk Enable is needed for a successful read, the signals are functionally equivalent for reads, the Read Clk Enable signal can be driven by the signal previously tied to Read Enable, and Read Enable can be tied to logic 1.

Similarly, the functional equivalence of embedded memory write enable and write clock enable can be observed in Figure 5 based on the discussion of RAM write steps in Section 3.

Using the figure, the behaviour of the following four write cases can be considered:

1. Write Clk Enable = 0, Write Enable = 0 – A write enable pulse will not be generated by the pulse generator preventing write data from being loaded onto the BIT lines. The BIT lines are not precharged.
2. Write Clk Enable = 1, Write Enable = 0 – A write enable pulse will not be generated by the pulse generator preventing write data from being loaded onto the BIT lines. The BIT lines are precharged.

3. Write Clk Enable = 0, Write Enable = 1 – A write enable pulse will not be generated by the pulse generator preventing write data from being loaded onto the BIT lines. The BIT lines are not precharged.
4. Write Clk Enable = 1, Write Enable = 1 – A write enable pulse is generated, the BIT lines are precharged, write data is loaded onto the BIT lines and into RAM cells

Consider a scenario where Write Enable is attached to a control signal and Write Clk Enable is always tied to active logic 1. From the enumeration it can be seen that since the AND of Write Enable and Write Clk Enable is needed for a successful write operation, the signals are functionally equivalent for writes, the Write Clk Enable signal can be driven by the signal previously labelled Write Enable, and Write Enable can be tied to logic 1. The **conversion** of user-defined read and write enable signals to respective clock enables primarily reduces power by eliminating BIT line precharging when embedded memory block data access is not required. The same functional RAM behaviour is maintained.

For some logical memories, a designer may specify both an enable and a clock enable signal for an embedded memory port. In these cases, additional logic (an AND gate) must be added to the user design to allow the user-defined enable signal to condition the associated memory port clock. The **combining** of the enable and clock enable signal forms a new combined clock enable signal which can be attached to the memory port clock enable input. Depending on designer timing constraints, the addition of logic delay to the clock enable path may negatively impact mapped design performance. As a result, this approach may only be appropriate if design power reduction is considered more important than design performance or preliminary timing information is available to determine if performance is not likely to be affected.

The mapping steps in Figure 6 are performed on each logical RAM. These steps perform enable-to-clock enable conversion and combining for embedded memory block inputs Clken and Enable and designer signals User Clken and User Enable.

If Clken = 1 and Enable = User Enable

Set Clken = User Enable and Enable = 1

If Clken = User Clken and Enable = User Enable

Set Clken = User Enable & User Clken and Enable = 1

If Clken = User Clken and Enable = 1

Perform no change

Figure 6: Steps required for enable signal conversion and combining. This analysis is performed on each design logical RAM

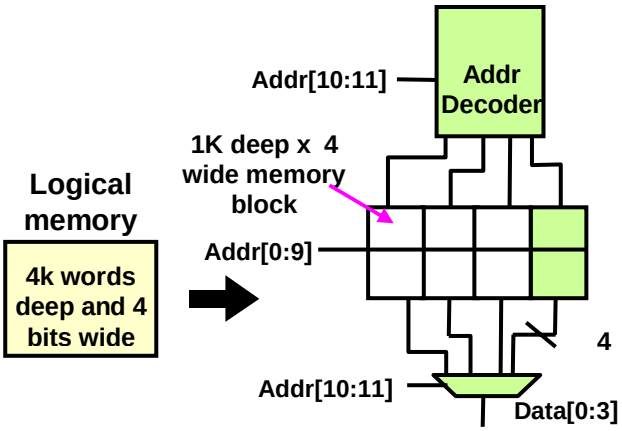

Figure 7: Alternate mapping of a 4Kx4 logical RAM to 4 Kbit memory blocks

4.2 Power-aware RAM Partitioning

As shown in Figure 3, a logical memory which exceeds the size of an embedded memory block must be mapped to multiple blocks. Although the mapping shown in Figure 3 does not require any supporting logic, each memory block is active during each memory access, requiring substantial power consumption. In this case, the *depth* of each physical memory block matches the depth of the logical memory and the width of each physical memory block is smaller than its logical memory counterpart. This mapping is an example of *vertical* memory slicing.

In general, an FPGA embedded memory block can be structured to have a variety of depth and width configurations, each with the same bit storage capacity. For example, an Altera M4K 4608 bit embedded memory block can be organized into configurations ranging from 4096x1 to 256x18 [2]. This allows a range of choices in mapping a logical RAM to physical memory blocks. For example, Figures 3 and 7 provide two example mapping alternatives. In the mapping in Figure 7, the *width* of each physical memory block matches the width of the logical memory while the depth of each physical memory block is reduced compared to its logical memory counterpart. This mapping can be considered an example of *horizontal* memory slicing. This second mapping requires the inclusion of address decoding circuitry to determine which memory block contains the requested data. Additionally, a multiplexer is required on the read port to select the requested word during read requests. Although dynamic power is consumed by the added address decoder and multiplexer, all but one of the embedded memory blocks is disabled during RAM accesses, saving considerable dynamic power. Unused memory blocks are disabled by connecting the outputs of the address decoder to memory block clock enable signals.

The vertical and horizontal RAM slicing implementations shown in Figures 3 and 7 represent the end points of a spectrum of feasible logical-to-physical RAM mappings (e.g. 2Kx2 RAM block configurations are also possible). If, as a result of a mapping change, an embedded memory block is converted from a given depth to one that is half as deep, the following additional mapping changes are required:

1. Each write port data line must be tied to twice the number of source/destination embedded memory blocks.
2. The size of the address decoder increases by a factor of 2.
3. The bit input size of each multiplexer on the embedded memory read port increases by a factor of 2.
4. One address line is removed from each of the embedded memory blocks.

The *relative* power consumed by each logical-to-physical mapping can be evaluated by assessing the power consumed by the memory blocks during a data access, the address decoder, the output multiplexer, and associated routing. As mappings approach the vertical slicing implementation (maximum physical block depth), memory block power is increased and multiplexer and address decoder power is decreased. As mappings approach the horizontal slicing implementation, multiplexer and address decoder power is increased and memory block power is decreased. Figure 8 shows the dynamic power consumed by various mappings of a 4Kx32 logical RAM in a Stratix II device for a selection of embedded memory block depths as reported by the Quartus II PowerPlay power analyzer. The plot shows that the power optimal mapping for this logical RAM falls between the horizontal slicing on the left and vertical slicing on the right. All mappings achieve the same functional behaviour.

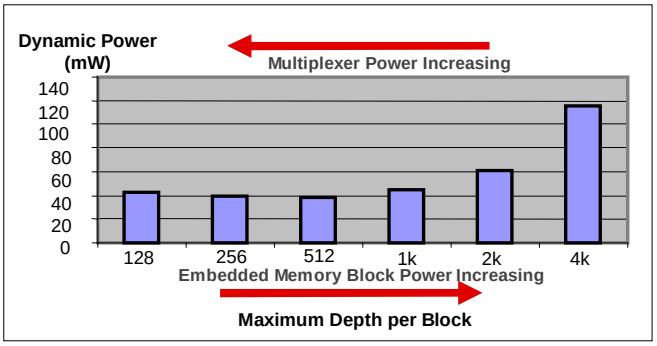

Figure 8: Dynamic power consumption of a 4Kx32 logical RAM at 100 MHz in different slicing configurations

4.3 Logical RAM Partitioning Algorithm

A power-aware RAM partitioning algorithm has been developed to evaluate the relative power consumption of a series of logical-to-physical RAM mappings. Each mapping is evaluated based on the number of active embedded memory blocks per port, the amount of associated address decoder and multiplexer circuitry required, and associated routing. Since contemporary FPGAs contain a set of different embedded memory block sizes, mapping evaluation is performed for each block type to determine the most power-efficient choice. The relative cost for each mapping is determined based on the estimated dynamic power consumption of the mapping. This cost can be expressed for each port of each logical RAM as:

$$\mathbf{Cost} = \mathbf{W} * \mathbf{P}_{\mathbf{mux}} + \mathbf{N} * \mathbf{P}_{\mathbf{ram}} + \mathbf{P}_{\mathbf{addr\_decode}} \quad (1)$$

1. For each embedded memory block type
   a. For each possible embedded memory block depth and width configuration
      i. Determine **N** and the size of the address decoder.
      ii. For each logical memory port
         1. Look-up memory block power, $\mathbf{P}_{\mathbf{ram}}$
         2. Scale $\mathbf{P}_{\mathbf{ram}}$ by number of memories, **N**
         3. Look up per-bit dynamic power of bit of output read port multiplexer, $\mathbf{P}_{\mathbf{mux}}$
         4. Scale $\mathbf{P}_{\mathbf{mux}}$ by read port width, **W**
         5. Look up dynamic power of address decoder, $\mathbf{P}_{\mathbf{addr\_decode}}$
         6. Sum power components to determine **Cost** via Eq. (1)
      iii. Sum **Cost** values across logical memory ports
   b. Save lowest power configuration
2. Rank possible implementations by power consumption
3. Select lowest-power feasible implementation
   a. Check if memory block usage overflowed by selection
   b. If yes, select next best feasible implementation

Figure 9: Power-aware memory partitioning algorithm applied to each logical RAM

Where **Cost** is the relative power cost for the mapping, **W** is the width of the logical RAM, $\mathbf{P}_{\mathbf{mux}}$ is the per-bit dynamic power of a read port multiplexer, **N** is the number of required embedded memory blocks, $\mathbf{P}_{\mathbf{ram}}$ is the per-block dynamic power, and $\mathbf{P}_{\mathbf{addr\_decode}}$ is the dynamic power consumption of the address decoder. Specific algorithm steps for a logical memory are shown in Figure 9.

Our approach is effective for both single- and dual-port logical RAMs. The key power savings aspect of the approach is the connection of address decoder outputs to embedded memory block clock enables. Only the *addressed* memory block is precharged on a given clock cycle saving considerable RAM dynamic power.

The inclusion of a memory block read port multiplexer can negatively impact design performance for designs which include the RAM block output on the design critical path. Design performance is not explicitly considered by the partitioning algorithm. However, to minimize performance impact, only configurations which require a 4-to-1 or smaller multiplexer on each read port output bit are considered.

In addition to possibly affecting performance, the inclusion of multiplexers consumes device logic. This added logic may result in an overflow of required design logic elements for a target device.

4.4 Parameter Evaluation

The algorithms described in Sections 4.1 and 4.3 have been integrated into Quartus II version 5.1 and applied to Stratix II FPGAs [2]. Experimental results were determined in two phases. First, the technology parameters noted in Equation (1) were determined via parameter evaluation experiments with a representative set of logical RAMs. After parameter evaluation, the algorithms were tested with 40 commercial benchmark designs containing logical RAMs.

The logical RAMs used for parameter evaluation included ROMs and single and dual port RAMs of sizes ranging from 512x2 to 8Kx132. Parameter evaluation was performed for the Altera Stratix II architecture, which contains three types of embedded memory blocks, each of a different size: 576-bit (M512), 4,608 bit (M4K), 589,824 bit (M-RAM) [2]. Each memory block allows for implementation of both single and dual-port synchronous RAMs.

Each logical RAM used for parameter evaluation was mapped to each of the three Stratix II memory block types using multi-block partitioning ranging from horizontal slicing to vertical slicing. Following synthesis with Quartus II, the memory designs were placed and routed using Quartus II. All synthesis, place, and route steps used an unattainable 1 GHz timing constraint to ensure maximum fitting effort by the CAD software. Designs were simulated at 100 MHz with random input vectors and dynamic power analysis was performed using the Quartus II PowerPlay power analyzer. All compiled designs were able to satisfy a minimum clock frequency of 100 MHz.

Statistical averaging was then used to determine the following values based on measured values for all RAM implementations:

- Power consumed by single bit of an **n**-to-1 multiplexer, $\mathbf{P_{mux}}$, Values for only 2-to-1 and 4-to-1 multiplexers were determined since shallower embedded memory blocks depth slicings are not performed by our system due to performance concerns.
- Per-port design power consumed by an active physical memory block, $\mathbf{P_{ram}}$, for an M512, M4K, and M-RAM embedded memory block.
- Power consumed by a **k**-to-**n** address decoder, $\mathbf{P_{addr\_decode}}$, for a 2-to-4 and 1-to-2 decoder.

Because the power analyzer takes detailed placement and routing into account when producing a power estimate, the averaged values for $\mathbf{P_{mux}}$ and $\mathbf{P_{addr\_decode}}$ take the effects of control signal, address, and data fanout into account.

Although the calculated parameters measure dynamic power values averaged across the RAM parameter evaluation design set, the access patterns of user logical RAMs may differ. Since our algorithm considers relative rather than absolute dynamic power values in making tradeoffs, we consider the subsequent use of these parameters across a range of user benchmarks to be acceptable and representative of most RAM access patterns.

Table 1: Benchmark Design Statistics

Design	LUTs	Memory bits	Flip flops	Target Device
1	8005	254680	6247	EP2S15F672
2	9106	47264	6971	EP2S15F672
3	15988	548	12948	EP2S60F1020
4	9802	292608	5363	EP2S15F672
5	8853	63744	7349	EP2S60F1020
6	5751	168	1750	EP2S60F1020
7	5743	168	1030	EP2S60F1020
8	13121	426512	10394	EP2S60F1020
9	23464	327680	3215	EP2S60F1020
10	215	331776	27	EP2S15F484
11	243	331776	47	EP2S15F484
12	26154	327680	3215	EP2S90F1508
13	5295	1134	3587	EP2S15F484
14	5488	512	4915	EP2S60F1020
15	7409	6432	5944	EP2S60F1020
16	23550	128452	22063	EP2S60F1020
17	8071	43008	3863	EP2S30F672
18	17857	66336	12199	EP2S90F1508
19	17857	66336	12199	EP2S90F1508
20	35849	89600	19745	EP2S90F1508
21	12039	1206785	8542	EP2S60F1020
22	11785	65536	8131	EP2S30F672
23	11149	36096	5297	EP2S30F672
24	13714	51456	6415	EP2S60F1020
25	5881	111872	4673	EP2S15F672
26	4816	98684	3875	EP2S15F672
27	10066	227010	7384	EP2S15F672
28	18987	184320	14940	EP2S60F1020
29	3082	124290	2702	EP2S15F672
30	30352	88048	25489	EP2S60F1020
31	8458	168416	6966	EP2S15F672
32	23283	337501	18868	EP2S30F672
33	13112	293856	9149	EP2S30F672
34	36741	1402661	16492	EP2S90F1508
35	16731	524288	15547	EP2S30F484
36	12560	1057428	9181	EP2S60F672
37	2136	171098	1618	EP2S15F484
38	4183	286956	3913	EP2S15F484
39	5384	153864	2809	EP2S60F1020
40	28199	1009920	12148	EP2S60F1020

5. Results

Following the determination of the tuning parameters and the integration of our algorithms with Quartus II, experimentation was performed on 40 commercial designs provided by Altera. This benchmark set includes designs which contain RAM from encryption, signal processing, and communications processing domains. LUT, memory bit and flip flop counts for each design are shown in Table 1. As seen in Figure 2, optimization occurs after complex memory functions (e.g. FIFOs, shift registers) are converted to logical RAMs, but before structures are assigned to specific embedded memories. The 40 designs were targeted to the smallest Stratix II device which would hold them. The specific device used for each design is listed in Table 1.

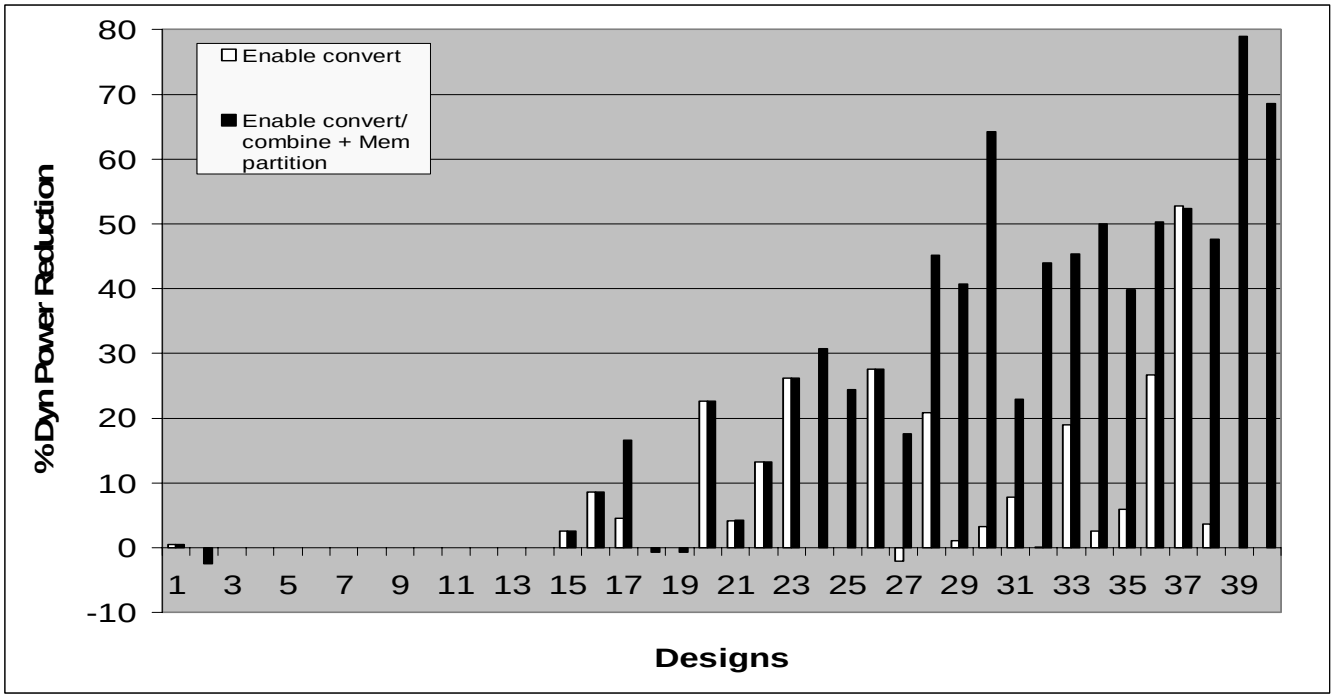

Figure 10: Data RAM power savings for benchmark designs due to RAM power optimizations

Dynamic power consumption for all designs was evaluated using the same power analysis flow noted in Section 4.4 for parameter evaluation except for the inclusion of our new RAM mapping algorithms. A series of test vectors were used to simulate each design at 100 MHz following compilation with an unattainable 1 GHz clock constraint. Dynamic power analysis was performed with the Quartus II PowerPlay power analyzer using switching activity values determined via simulation. All Quartus II power optimizations, except for our new algorithms, were shut off during experimentation.

To validate our approach, a series of experiments were performed using combinations of the algorithms with the 40 benchmark circuits. Dynamic power statistics related to the benchmarks appear in Table 2 for initial compilation with default parameters and no RAM power optimizations. Dynamic power percentages were determined versus overall design core dynamic power. In addition to compilation without RAM power optimizations, each design was compiled using the following automatic RAM power optimization cases described in Section 4.

1. Read/write enable conversion to read/write clock enable.
2. Read/write enable combining with an existing clock enable in addition to read/write enable conversion.
3. Memory partitioning in addition to read/write enable conversion and combining.

Table 2: Benchmark power statistics

Average % dynamic power – embedded block memory	25.3%
Average % dynamic power – combinational logic	22.7%
Average % dynamic power - registers	33.5%

As shown in Table 2, RAM dynamic power forms a significant part of average design core dynamic power. A bar graph illustrating the per-design percent reduction in memory dynamic power due to these optimizations for Cases 1 (enable conversion) and 3 versus base case compilation with no RAM power optimization appears in Figure 10. A bar graph illustrating the reduction in overall core dynamic power appears in Figure 11. The designs appear in the same order numerically in each plot and in Table 1. Case 3 data for each graph includes any increase in combinational logic and register dynamic power due to logic added for multiplexing, address decoding, and clock enable combining. These plots show that although some designs achieve no benefit from the new approaches, others benefit significantly (up to 78% of RAM power and 34% of overall core dynamic power). Table 3 shows the average percentage improvement for core and RAM dynamic power for all three cases. The use of memory partitioning more than doubles the average core dynamic power savings (6.8% vs. 2.6%) and RAM dynamic power savings (21.0% vs. 9.7%).

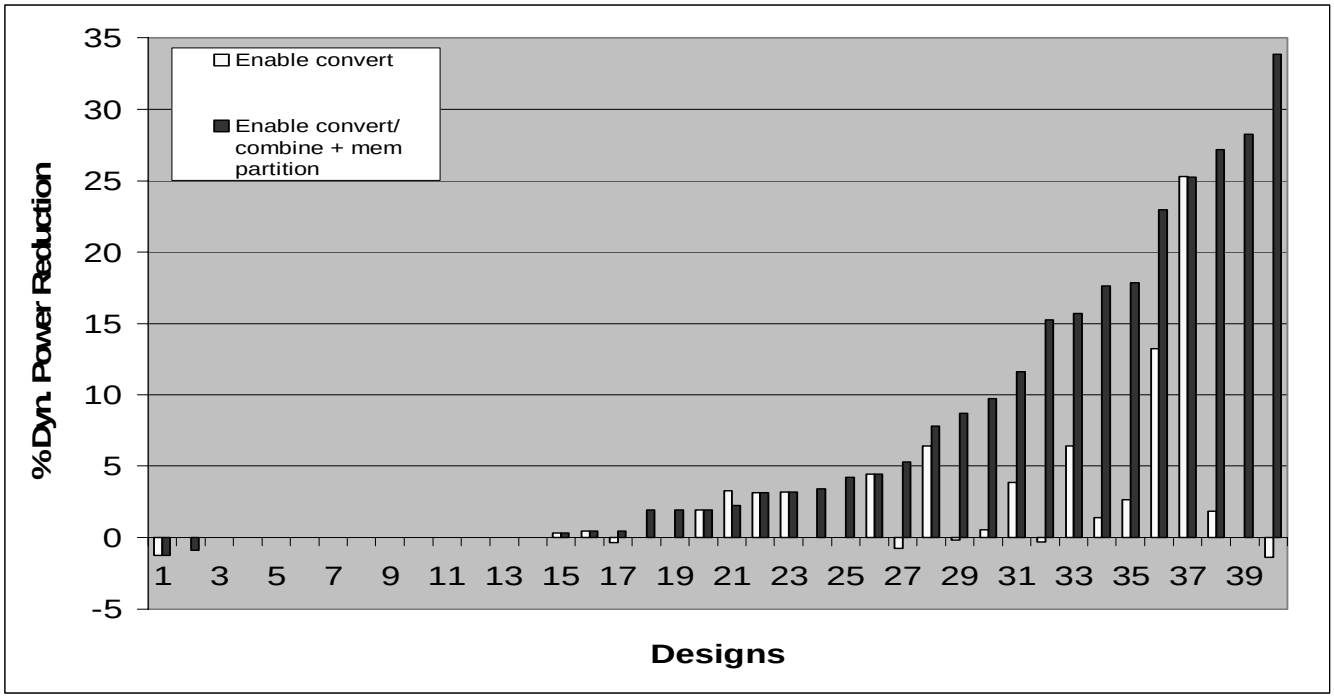

Figure 11: Overall core dynamic power savings for benchmark designs due to RAM power savings

Table 3: Summary of RAM optimization results for 40 benchmark designs (all averages geometric)

	Enable convert	Enable convert/ combine	Enable convert/ combine + Mem partition
Core dynamic power	-1.8%	-2.6%	-6.8%
Memory dynamic power	-6.3%	-9.7%	-21.0%
Max clk freq	-0.1%	-0.2%	-1.0%
LUT count	0.0%	0.1%	0.7%

Table 3 also shows that the RAM dynamic power optimizations have little effect on area or performance. The percentage reduction in the achievable average design clock is shown in the table for all three cases. As expected, Case 3, which includes memory partitioning, exhibits the largest performance loss due to the inclusion of multiplexers at the logical RAM output (1.0%). As discussed in Section 4.3, this performance loss was mitigated by our restriction of a maximum 4-to-1 read port output bit multiplexer size. Case 3 also shows the largest increase in required LUTs (0.7%), primarily used to implement multiplexing logic. Case 1 (enable conversion) requires no additional logic and shows minimal performance decrease.

As stated in Section 4.3, the memory partitioning algorithm considers mapping each logical memory to each type of embedded memory block on a target device and selects the most power-efficient implementation relative to available resources. To illustrate the dynamic power benefits of the availability of multiple embedded memory block sizes on a target FPGA we re-mapped each of the 40 benchmark designs to a Stratix II EP2S180 using the constraints described in Section 4.4, except for the smallest device constraint. Four separate compiles were performed for each design, each using one of the following constraints:

a. Memory partitioner selects the target physical embedded memory for each logical memory
b. All logical memories mapped to M512s
c. All logical memories mapped to M4Ks
d. All logical memories mapped to M-RAMs

For each compile, all RAM power optimizations were used, including memory partitioning.

Due to RAM resource limitations it was not possible to successfully map all designs for Cases b, c, and d. Table 4 shows the number of designs that were successfully mapped for each case and the percentage increases for Cases b, c, and d mapping versus Case a for several parameters. Although it was possible to map all but 2 designs using solely M4Ks for embedded memory, a 6.6% core dynamic power and 33.3% RAM power penalty was observed. More drastic results versus the base case were observed by restricting memory mapping to solely M512s and M-RAMs.

Table 4: Summary of RAM optimization results for logical RAMs targeted to specific embedded memory blocks versus unconstrained RAM placement using 40 benchmark designs

	M512	M4K	M-RAM
Designs completed	23	38	4
Core dynamic power	40.4%	6.6%	47.3%
Memory power	279.5%	33.3%	754.0%
Max clk freq.	-2.2%	0.6%	-1.0%
LUT count	0.4%	-0.5%	0.0%

6. Conclusion and Future Work

In this paper we have presented a set of RAM mapping algorithms that are targeted to FPGA embedded memory blocks. These techniques take advantage of the internal structure of FPGA embedded memory to reduce memory dynamic power dissipation. When possible, embedded memory block clock enables are used to deactivate RAM block precharging. Our mapping algorithms maintain the functional behaviour of each designer-specified RAM. These techniques achieve a 21% RAM dynamic power reduction and a 7% core dynamic power reduction for 40 large benchmark designs with a performance and logic cost of about 1%. Possible extensions to this work include packing multiple logical memories into a single physical memory to save power and using application signal activity profiling to guide power-aware RAM mapping decisions.

7. Acknowledgments

The authors wish to thank Elden Chau, Aaron Egier, Marcel LeBlanc, David Lewis, and David Lin for their insights regarding this work.

8. REFERENCES

[1] Altera Corp. *Quartus II Handbook, Chapter 7*, vol. 1, July 2005.

[2] Altera Corp. *Stratix II Device Handbook*, vol. 2, July 2005.

[3] Altera Corp. *Stratix Device Handbook*, vol. 1, July 2005.

[4] S. Bakshi and D. Gajski. A memory selection algorithm for high-performance pipelines, In *Proceedings of the European Design Automation Conference*, Brighton, England, Sept. 1995, pp. 124-129.

[5] L. Benini, A. Macii, and M. Poncino. A recursive algorithm for low-power memory partitioning, In *Proceedings of the International Symposium on Low Power Electronics and Design*, Rapallo, Italy, July, 2000, pp. 78-83.

[6] Y. Cao, H. Tomiyama, T. Okuma and H. Yasuura. Data memory design considering effective bitwidth for low-energy embedded systems, In *Proceedings of the IEEE International Symposium of System Synthesis*, Kyoto, Japan, Oct. 2002, pp. 201-206.

[7] A. Ferrahi, G. Tellez, and M. Sarrafzadeh. Memory segmentation to exploit sleep mode operation, In *Proceedings of the ACM/IEEE Design Automation Conference*, San Francisco CA, Jun. 1995, pp. 36-41.

[8] C. Gebotys. Low energy memory and register allocation using network flow, In *Proceedings of the ACM/IEEE Design Automation Conference*, Anaheim, CA, Jun. 1997, pp. 435-440.

[9] W. Ho and S. Wilton. Logical-to-physical memory mapping for FPGAs with dual-port embedded memories, In *Proceedings of the International Workshop of Field Programmable Logic and Applications*, Glasgow, UK, Aug. 1999, pp. 111-123.

[10] J. Lamoureux and S. Wilton. On the interaction between FPGA CAD algorithms, In *Proceedings of the IEEE International Conference on Computer-Aided Design*, San Jose, CA, Nov. 2003, pp. 701-708.

[11] M. Margala. Low-power SRAM circuit design, In *Proceedings of the IEEE International Workshop on Memory Technology, Design, and Testing*, San Jose, CA, Aug. 1999, pp. 115-122.

[12] M. Mamidipaka and N. Dutt. *An Enhanced Power Estimation Model for On-Chip Caches.* CECS Technical Report #04-28, University of California, Irvine, 2004.

[13] P. Petrov and A. Orailoglu. Virtual page tag reduction for low-power TLBs, In *Proceedings of the IEEE International Conference on Computer Design*, San Jose, CA, Oct. 2003, pp. 371-374.

[14] H. Schmit and D. Thomas. Address generation for memories containing multiple arrays, *IEEE Transactions on VLSI Systems*, vol. 17, pp. 377-385, May 1998.

[15] O. Unsal, R. Ashok, I. Koren, C. Krishna, and C. Moritz. Cool-cache for hot multimedia, In *Proceedings of the ACM/IEEE International Symposium on Microarchitecture*, Austin, TX, Dec. 2001, pp. 274-283.

[16] S. Wuytack, F. Catthoor, L. Nachtergaele and H. De Man. Power exploration for data dominated video applications, In *Proceedings of the IEEE International Symposium on Low Power Design*, Monterey, CA, Aug. 1996, pp. 359-364.

[17] Xilinx Corp. *Virtex-4 User's Guide*, July 2005.

[18] Xilinx Corp. *Virtex II Platform FPGAs: Complete Data Sheet*, March 2005.