Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 751863, 13 pages
doi:10.1155/2008/751863

Research Article

On the Power Dissipation of Embedded Memory Blocks Used to Implement Logic in Field-Programmable Gate Arrays

Scott Y. L. Chin, Clarence S. P. Lee, and Steven J. E. Wilton

Department of Electrical and Computer Engineering, Faculty of Applied Sciences, University of British Columbia, Vancouver, B.C. V6T 1Z4, Canada

Correspondence should be addressed to Scott Y. L. Chin, scottc@ece.ubc.ca

Received 20 September 2007; Revised 23 November 2007; Accepted 29 January 2008

Recommended by Gustavo Sutter

We investigate the power and energy implications of using embedded FPGA memory blocks to implement logic. Previous studies have shown that this technique provides extremely dense implementations of some types of logic circuits, however, these previous studies did not evaluate the impact on power. In this paper, we measure the effects on power and energy as a function of three architectural parameters: the number of available memory blocks, the size of the memory blocks, and the flexibility of the memory blocks. We show that although embedded memories provide area efficient implementations of many circuits, this technique results in additional power consumption. We also show that blocks containing smaller-memory arrays are more power efficient than those containing large arrays, but for most array sizes, the memory blocks should be as flexible as possible. Finally, we show that by combining physical arrays into larger logical memories, and mapping logic in such a way that some physical arrays can be disabled on each access, can reduce the power consumption penalty. The results were obtained from place and routed circuits using standard experimental physical design tools and a detailed power model. Several results were also verified through current measurements on a 0.13 μm CMOS FPGA.

1. INTRODUCTION

On-chip user memory has become an essential and common component of field programmable gate arrays (FPGAs). FPGAs are now used to implement entire systems on a chip (SoC's) rather than the small logic subcircuits that have traditionally been targeted to FPGAs. One of the key differences between these large systems and smaller logic subcircuits is that the large systems often require storage. Although this storage could be implemented off-chip, implementing these systems on a single chip has a number of advantages. Besides the obvious advantages of integration, on-chip storage will often lead to higher-clock frequencies since input/output (I/O) pins need not be driven with each memory access. In addition, on-chip storage will relax I/O pin requirements, since pins need not be devoted to external memory connections.

FPGA vendors have responded by incorporating two types of embedded memory resources into their architectures: fine-grained and coarse-grained. The Xilinx distributed SelectRAM, in which the lookup-table circuitry can be configured as small RAMs, is an example of a fine-grained memory architecture [1]. The Altera TriMatrix memory and Xilinx block SelectRAM, which are dedicated memory arrays embedded into the FPGA fabric, are examples of coarse-grained memory architectures [1, 2].

The coarse-grained embedded memory arrays lead to much denser memory implementations [3] and are therefore very efficient when implementing large SoC's that require storage. However, for logic intensive circuits that do not require storage, the chip area devoted to the embedded FPGA memory is wasted. This need not be the case if the FPGA memories are configured as ROMs to implement logic. Previous work has presented heterogeneous technology mapping algorithms that automatically map logic circuits to FPGAs with both large ROMs and small lookup tables [4, 5]. Given a logic circuit, these algorithms attempt to pack as much of the logic into the available ROMs as possible, and implement the rest of the logic using lookup-tables. These studies have shown that significant density improvements can

be obtained by implementing logic in these unused memory arrays, compared to implementing all the logic in lookup-tables. This technique can be especially beneficial for designs that are up against the logic capacity of the target FPGA. As far as these studies have shown, there is no reason not to use the memory arrays to implement logic.

These previous studies, however, did not consider the impact on power. Power has become a first-class concern among designers and is often the limiting factor in handheld battery-powered applications. FPGAs are power-hungry for several reasons. First, the logic is typically implemented in small lookup-tables, which have not been optimized for a specific application. Second, the prevalence of programmable switches in the interconnect leads to high-interconnect capacitances, and hence, high-switching energy.

Intuitively, implementing logic in memory arrays will impact the overall power dissipated by the system in two ways. If large amounts of logic can be implemented in a memory array, not only are fewer lookup-tables required (which would save a small amount of power), but also the interconnect between these lookup-tables is not required (which would save a significant amount of power). On the other hand, memory arrays contain word-lines, bit-lines, amplifiers, and decoders, all of which consume power [6].

In this paper, we make two contributions. First, we investigate whether implementing logic in FPGA memory arrays leads to a net reduction or increase in power for a range of memory architectures. In particular, we answer the following questions.

(1) How does the number of available FPGA memories used for logic affect power?
(2) How does the size of the available FPGA memories used for logic affect power?
(3) How does the flexibility of the available FPGA memories used for logic affect power?

The answers to these questions are important. As FPGA capacity grows, more embedded memory arrays will be included to facilitate the efficient implementation of larger systems that require more storage. Therefore, understanding the power implications of using memories to implement logic will be important for logic intensive circuits. In addition, the understanding of the impact on memory block architecture on power will lead to more power-efficient FPGAs.

The second contribution of this paper is a power-aware algorithm for packing logic into memory arrays. Heterogeneous technology mapping algorithms, such as those in [4, 5], optimize for area, possibly under delay constraints. In this paper, we show how the algorithm in [5] can be modified to optimize for power. The key idea is to combine physical memory arrays into larger logical memories, such that some physical memories can be disabled with each access. We will show that this technique leads to a significant reduction in the power of the overall system.

This paper is organized as follows. Section 2 describes the necessary background to this work. Section 3 investigates the power implications of implementing logic in FPGA memory blocks and presents experimental results. Section 4 then presents our power-aware heterogeneous technology mapping algorithm. Section 5 discusses the limitations of our study, and Section 6 concludes the paper. An early version of the results in Section 3 was presented in [7].

2. BACKGROUND

This section describes the architecture of the embedded memories found in commercial FPGAs, as well as existing heterogeneous technology mapping algorithms, and some background information on FPGA power estimation models.

2.1. Embedded memories in commercial FPGA

Embedded FPGA memory array architectures can be described by three parameters: N, B, and w_{eff}. N denotes the number of available arrays, and B denotes the number of bits available in each array. Typically, each memory array can be configured to implement one of several aspect ratios; for example, the M4k block in the Altera Stratix II architecture can be configured as one of 4096×1, 2048×2, 1024×4, 512×8, 256×16, 128×32 [2]. We will refer to the set of allowable widths of each memory block as w_{eff}; in the above example, $w_{\text{eff}} = \{1, 2, 4, 8, 16, 32\}$.

It is important to note that the shape of the physical array does not change as the chip is configured (it is fixed when the FPGA is manufactured). The appearance of a configurable data width is obtained by using multiplexers and column enable signals to read and update only certain columns in each word [3]. Note that the width of the physical array (the number of columns) must be at least as large as the largest value in w_{eff} so that w_{eff} bits can be read from the array in one access.

2.2. Heterogeneous technology mapping

Two algorithms that map logic to memory arrays have been described: EMB_Pack [4] and SMAP [5]. Both algorithms operate similarly and produce similar results; in this paper, we use SMAP. In this subsection, we briefly review the SMAP algorithm.

In the following, we will use the following terminology (primarily from [8]). The combinational part of a circuit is represented by a directed acyclic graph G(V,E) where the vertices V represent combinational nodes, and the edges E represent dependencies between the nodes. V also contains nodes representing each primary input and output of the circuit. Flip-flop inputs and outputs are treated as primary outputs and inputs. A network is k-feasible if the number of inputs to each node is no more than k. Given a node v, a cone rooted at v is a subnetwork containing v and some of its predecessors. As in [5], we define a cone rooted at a set of nodes W to be a subnetwork containing each node in W along with nodes that are predecessors of at least one node in W. A fanout-free cone is a cone in which no node in the cone (except the root) drives a node not in the cone. The maximum-fanout free cone (MFFC) for a node v (or set of nodes W) is the fanout-free cone rooted at v (or W) containing the largest number of nodes. Given a cone C rooted at v, a cut (X, X') is a partitioning of nodes such that $X' = C$. A cut-set of a cut is

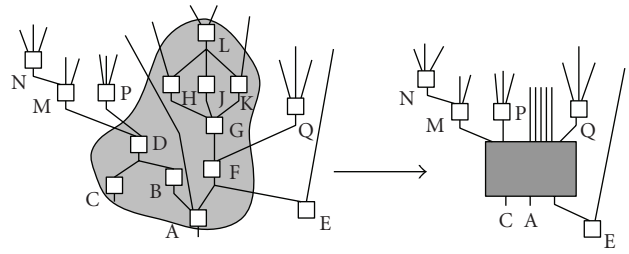

Figure 1: Example of mapping logic to a memory array.

the set of all nodes v such that $v \in X$ and v drives a node in X'. If the size of the cut set is no more than d, the cut is said to be d-feasible. Given a cone rooted at v, the maximum-volume d-feasible cut is the d-feasible cut (X, X') with the largest number of nodes in X'.

Given a logic circuit and a set of memory arrays, SMAP tries to pack as much logic into each memory array as possible and implements the rest of the logic in lookup-tables. It does this one memory array at a time. For each memory array, the algorithm chooses a seed node (as described below). Given a seed node, it determines the maximum-volume k-feasible cut of the node's fan-in network. The k cut nodes become the inputs to the memory array. Given the seed node and the cut nodes, SMAP then determines which nodes become the memory array outputs. Any node that can be expressed as a function of the cut nodes can be a memory output. SMAP chooses the output nodes with the largest maximum fan-out-free cone (MFFC). The nodes in the chosen output node's MFFC are packed into the memory array. Figure 1 shows an example of an eight input cut, where nodes C, A, and F are chosen as memory outputs, and the resulting circuit implementation.

To find the best solution using this method, the algorithm is repeated for all possible seed nodes and the solution that results in the highest number of packed nodes is selected. For arrays with a configurable aspect ratio, SMAP tries all possible configurations with the selected seed node and chooses the configuration that results in the highest number of packed nodes.

When there is more than one memory array, the algorithm is repeated iteratively for each array. In this way, the algorithm is greedy; the decisions made when packing nodes into the first array do not take into account future arrays. However, experiments have shown that this works well. For a more complete description of SMAP, see [5].

2.3. Power estimation for FPGAs

Both commercial and academic power estimation tools exist for FPGAs [9–14]. Vendors typically provide power estimation spreadsheets for power estimation in the early design stage [12, 13]. However, power estimates at early design stages are not very accurate. For power estimates in later design phases, vendors provide tools in their CAD suites such as Altera's PowerPlay power analyzer [11] and Xilinx's Xpower [10].

The poon power model (PPM) [14] is a probabilistic academic FPGA power model. This model is built on top of the popular versatile place and route (VPR) [15] CAD suite and is used for the experiments conducted in this paper. The power estimates are calculated after placement and routing and are broken down into three components: dynamic, static, and short-circuit power.

Dynamic power dissipation arises from the charging and discharging of the node capacitances in the FPGA. PPM uses (1) to model the dynamic power dissipation of a net. Where α is the switching activity of the net, C is the effective capacitance of the net, V_{supply} is the supply voltage, V_{swing} is the voltage swing when switching, and f_{clock} is the clock frequency. Dynamic power is very significant in FPGAs due to the numerous programmable connections. These connections add substantial parasitic capacitances to the capacitance of the prefabricated wire segments. To model the dynamic power dissipated within the logic blocks and clock network, PPM employs a number of transistor-level models:

$$P_{\text{Dynamic}} = \frac{1}{2} \cdot \alpha \cdot C \cdot V_{\text{Supply}} \cdot V_{\text{Swing}} \cdot f_{\text{Clock}}. \quad (1)$$

Static power is caused by leakage current when a transistor is turned off. There are many sources of leakage currents but the most significant source, and only source modeled by PPM, is the subthreshold current. To model this, PPM uses a first-order approximation to estimate the subthreshold leakage current of each transistor and multiplies this value with the supply voltage. The PPM model for static power has been shown to be within 13.4% of SPICE simulations.

The last source of power dissipation is short-circuit power. When a CMOS gate switches, there is a brief moment when the pull-up and pull-down networks are both partially on. This allows current to flow directly between the supply and ground rails. The amount of current that actually flows is dependent on the rise and fall times. Based on commercial device data sheets, PPM estimates the short-circuit power as 10% of the dynamic power. For more details on PPM, please refer to [14].

3. POWER IMPLICATIONS OF IMPLEMENTING LOGIC IN MEMORY BLOCKS

In this section, we quantify the power implications of implementing logic in memory blocks using the SMAP algorithm. We also investigate the impact of the memory block architecture on the power efficiency of the resulting implementation.

3.1. Experimental methodology

To investigate the power implications when using memories to implement logic, we employ an experimental methodology. We gather results in two ways: we use an enhanced version of VPR [15] and PPM [14] that supports embedded memories, as well as performing current measurements on a modern 0.13 μm CMOS FPGA (Altera Stratix EP1S40). Although the second technique provides the most accurate results, it is not possible to use it to investigate alternative

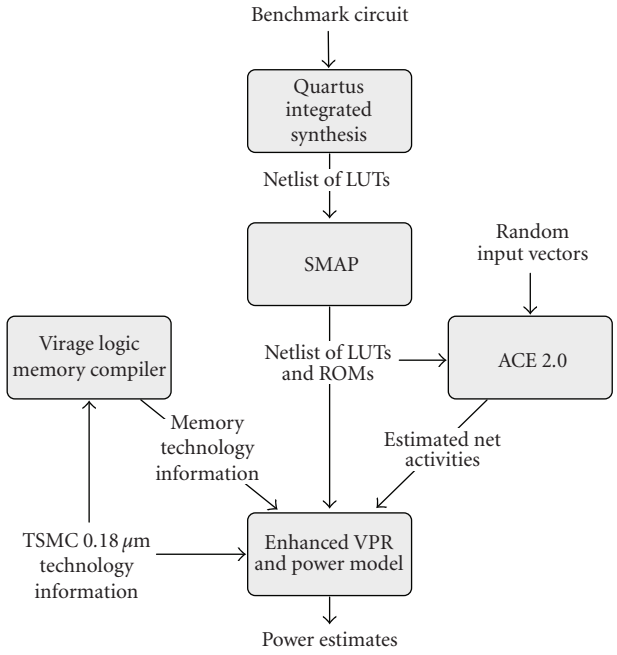

Figure 2: Flow for VPR-based experiments.

Table 1: Benchmark circuits.

Circuit name	Inputs	Outputs	4-LUTs	Flip flops
alu4	14	8	989	0
apex2	38	3	1023	0
apex4	9	19	844	0
bigkey	229	198	1032	224
clma	62	83	4682	33
des	256	245	1242	0
diffeq	64	40	924	314
dsip	229	198	924	224
elliptic	131	115	2021	886
ex5p	8	63	210	0
ex1010	10	10	876	0
frisc	20	117	2167	821
misex3	14	14	939	0
pdc	16	40	2218	0
s298	4	7	738	8
s38417	29	107	3998	1391
s38584	38	305	4501	1232
seq	41	35	1118	0
spla	16	46	1916	0
tseng	52	123	626	225

memory architectures; for those experiments, we need to use our VPR flow. In this section, we describe both methodologies.

3.1.1. Enhanced VPR and power model flow

Figure 2 shows the flow for our VPR-based experiments. We first technology map twenty large MCNC circuits (combinational and sequential) to 4-LUTs using Quartus integrated synthesis (QIS), and export the netlist to BLIF [16] format using the Quartus University Interface Program (QUIP) [17]. Although SIS [18] and Flowmap [19] could also be used to perform technology mapping, we chose QIS since we need to eventually feed the same circuit through the Quartus flow to perform measurements on an actual device, and we want to ensure that the measured circuit is identical to the one used in the VPR flow. Statistics for our benchmark circuits are shown in Table 1.

To map logic to memories, we use a modified version of SMAP. The embedded memories found in commercial FPGAs are typically synchronous whereas the memories generated by SMAP are asynchronous. To make the SMAP memories synchronous, we add a new constraint that only allows SMAP to make input cuts at register inputs. If the fan-outs of these registers become completely packed into the memory, then we can also remove these registers from the circuit. For circuits that are purely combinational, we add registers to all primary inputs. Although this new constraint may restrict the solutions produced by SMAP, we found that it actually had no impact on SMAP's packing efficiency. The reason is that registers are intrinsic boundaries for the SMAP algorithm.

To perform power estimation using VPR and PPM, switching activities for the circuits are required. We use a modified version of ACE 2.0 to generate the switching activity information [20]. The original version of ACE 2.0 supports three elements: primary I/Os, combinational nodes, and registers. Our extension allows ACE to estimate activities for ROM memories as well. Each column of a ROM can be represented as a multi-input single-output combinational node; we construct these nodes using the results from SMAP and insert them into the circuit before performing activity estimation.

To place and route the circuits we use a modified version of VPR that supports memories [21]. An architecture in which memories are arranged in columns was assumed; the ratio of memory columns to logic columns was fixed at 1 : 6. It was further assumed that each logic cluster contains ten four-input lookup tables, as in the Altera Stratix device.

A power estimate of the resulting implementation was obtained using an enhanced version of PPM [14]. The original power model estimates the power dissipated in logic blocks and the configurable routing; we enhanced the model to estimate the power dissipated in the embedded memory blocks as well. The power dissipated in each memory block was modeled as the sum of three components: the pin power, the cycle power, and the leakage power. The pin power is calculated from the activity and pin capacitance of each pin. The cycle power is the power dissipated within the memory array during each read access. Estimating the cycle power and the leakage power requires technology information; we obtained this information by creating memories of different shapes and sizes using the Virage Logic memory compiler [22]. This memory compiler also provided timing information which

was used by the timing-driven place and route algorithms within VPR. A 0.18 μm TSMC CMOS process was assumed throughout.

3.1.2. Current measurements on a commercial FPGA

In order to validate the trends found through our VPR-based flow, we implemented some of these circuits on an Altera Nios Development Kit (Stratix professional edition) which contains a 0.13 μm Stratix EP1S40F780C5 device. For each implementation, we measured the current entering the board, subtracted the quiescent current when the chip is idle, and multiplied the result by the voltage to get an estimate of the power dissipation.

For these experiments, we created a test harness for the benchmark circuit similar to the one in [23], as shown in Figure 3. Driving the external input and output pins of the FPGA can consume significant power, and we did not want to include this power in our comparisons. This harness consists of a linear feedback shift register (LFSR) connected to the primary inputs of the benchmark circuit; this allows the circuit to be stimulated by vectors that are generated on-chip. The harness also contains a wide exclusive-or gate connected to all of the primary outputs. The harnessed circuit only has a single input, the clock, and a single output, the exclusive-or gate output. The harnessed circuit was then replicated several times to fill the FPGA.

3.2. Experimental results

In this section, we investigate the impact of three architecture parameters on power dissipation. Results are presented in terms of energy per cycle to make the results independent of operating frequency. For combinational circuits, it is assumed that the cycle time is the maximum combinational delay through the circuit.

3.2.1. Number of arrays used for logic

We first determine the impact of the number of memory arrays used to implement logic on the energy consumption of a circuit. Intuitively, as the number of memory arrays increases, more logic can be packed into the memories. Whether this provides an overall reduction in energy depends on how much logic can be packed into each array. As described earlier, SMAP is a greedy algorithm, meaning we would expect to pack more logic into the first few arrays than in later arrays. This suggests that the energy reduction (increase) will be smaller (larger) as more memory arrays are available.

Figure 4 shows the overall energy (dynamic and leakage) results obtained using the VPR flow averaged across the 20 MCNC benchmark circuits. The number of memory arrays is varied from 0 (all logic implemented in lookup-tables) to 8, and the array size was fixed at 512 bits. The vertical axis is normalized to the total energy of the all-logic implementation. Since the agreement with our technology provider prohibits us from publishing the absolute power characteristics of the Virage memory compiler output (the memory power), we have normalized the vertical axis in several graphs in this paper.

The bottom line corresponds to the energy dissipated in the logic blocks (the logic energy). As expected, the logic energy goes down as the number of arrays used by SMAP increases. This is because as the number of used memory arrays increases, more logic can be packed into the memory, meaning there are fewer LUTs in the final circuit. The second line indicates the sum of the logic energy and the energy consumed by the routing and the clock (so the area between the lower two lines represents the routing and clock energy). Again, as expected, more memory arrays means there are fewer LUTs, leading to fewer connections, and hence, lower routing energy. Finally, the top line is the overall energy; the difference between the top line and the middle line represents the energy consumed by the memories. As the graph shows, overall, mapping logic to memory arrays does not reduce energy consumption. In fact, the energy increases significantly as more memory arrays are used. This suggests that the extra power dissipated during a read access of the memory is larger than the power dissipated if the corresponding circuitry is implemented using lookup-tables. We repeated this experiment for other memory sizes; the results are shown in Figure 5.

To verify this trend, we implemented a number of the circuits on an Altera NIOS development kit containing a 0.13 μm Stratix EP1S40F780C5 device. The Stratix device contains two types of memory arrays: 512-bit blocks and 4 Kbit blocks. Figure 6(a) shows the measured results for a single representative circuit when only 512-bit blocks are used. The bottom line represents the power dissipated in the memories and clock network (this was obtained by keeping the inputs constant, but toggling the clock, forcing each memory to perform a read access each cycle). The upper line presents the total power dissipated in the FPGA. In both cases, the static power (both the static power of the FPGA and the power dissipated by the board) was subtracted. As discussed in Section 3.1.2, the static current is subtracted when the quiescent current entering the board is subtracted. As shown in the graph, the measured results show the same trends as the VPR results presented in Figure 5. We repeated the experiment using only 4 Kbit blocks; the results are shown in Figure 6(b).

3.2.2. Array size

Although the results in the previous subsection show that the energy, and hence power, is increased when implementing logic in memories, there are still situations in which we may wish to do this. In particular, significant density improvements are reported in [5] and when the design is up against the logic capacity of the target FPGA. Therefore, it is important that we measure the impact of the other architectural parameters, in an effort to reduce the power penalty imposed by this technique.

In this subsection, we investigate the impact of the array size on the energy dissipated by the circuits. Intuitively, a larger array means more logic can be packed into each array,

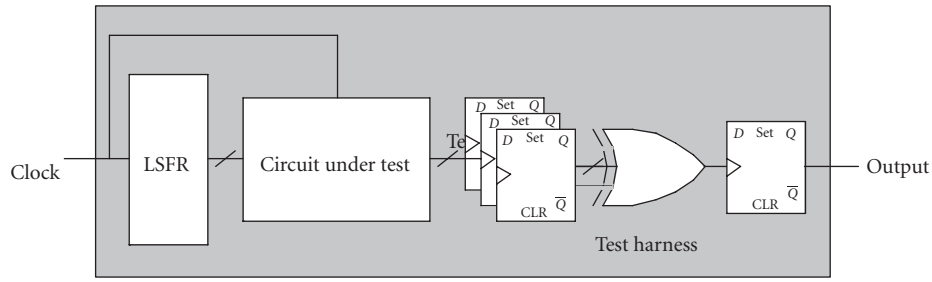

Figure 3: Test harness.

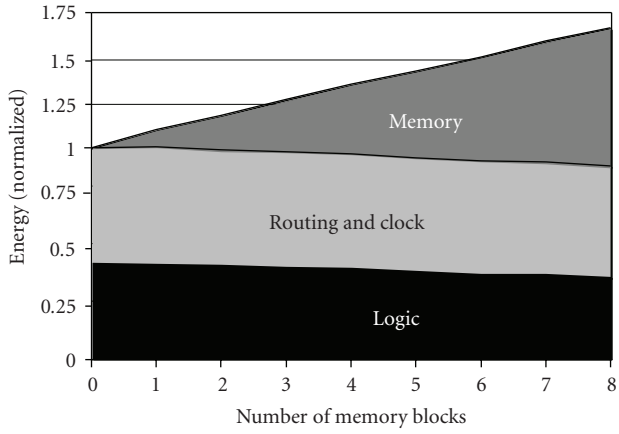

Figure 4: Impact on energy when increasing the number of memory arrays (512-bit arrays in VPR flow).

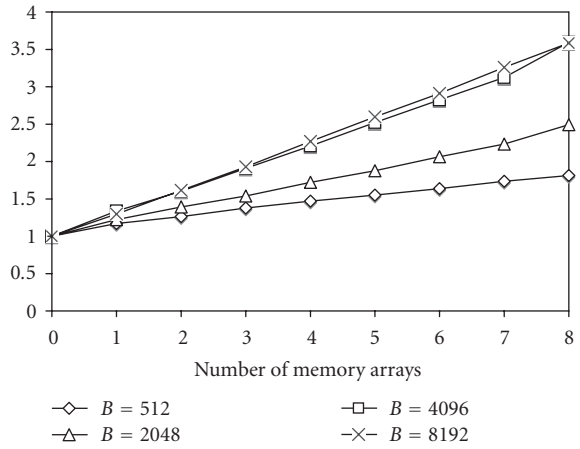

Figure 5: Impact on energy when increasing the number of memory arrays (VPR Flow).

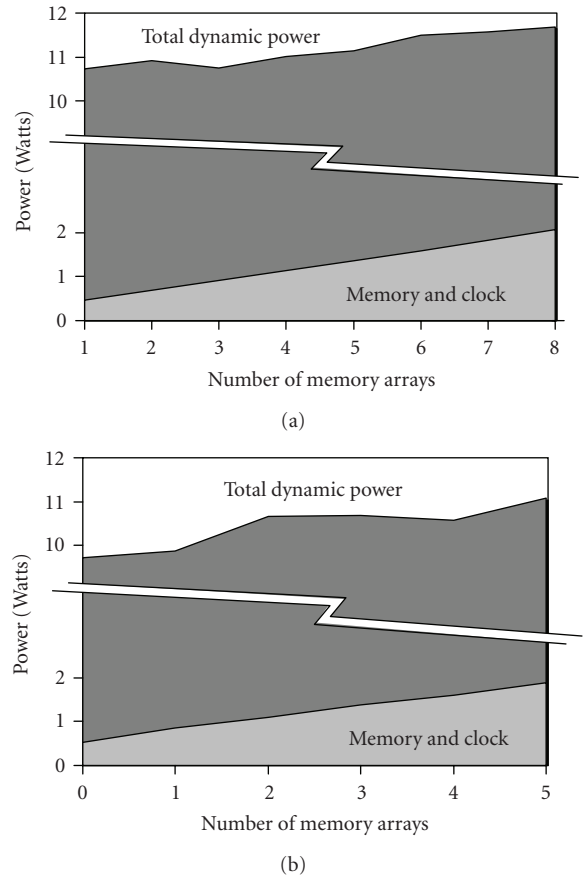

Figure 6: Impact on power when increasing the number of memory arrays (measured flow): (a) 512-bit arrays, (b) 4096-bit arrays.

however, the word-lines and bit-lines are longer, meaning more energy is dissipated with each memory access.

To investigate this, we fix $N = 1$ and vary B from 256 to 8192. N is fixed at one because this is the optimistic case as we saw in the previous subsection. We repeat the experiment for various values of w_{eff}. Clearly, as B increases, the power dissipated in each memory during each access increases. This is shown graphically in Figure 7. In all graphs in this section, we normalize the energy values to the overall energy consumed when mapping to a single 256×1 memory. For each memory, we assume that the number of columns in the array is the same as the number of rows in the array. For values of B which do not have an integral square root, it is assumed that the number of rows is twice the number of columns. Since the power dissipated in our memory depends more on the number of columns than the number of rows, the power dissipation of blocks in which B does not have an integral square root is not significantly larger than the next smaller value of

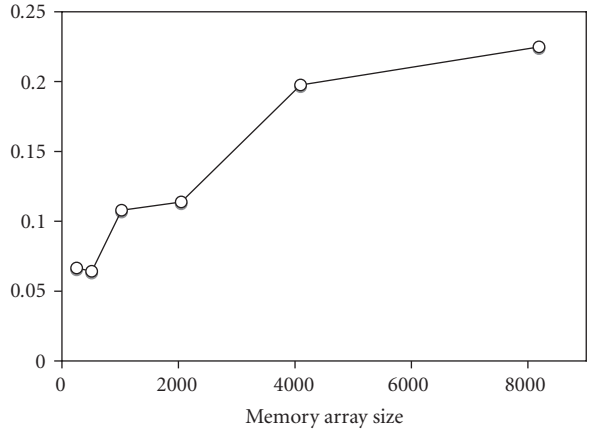

Figure 7: Impact on memory energy when increasing memory array size.

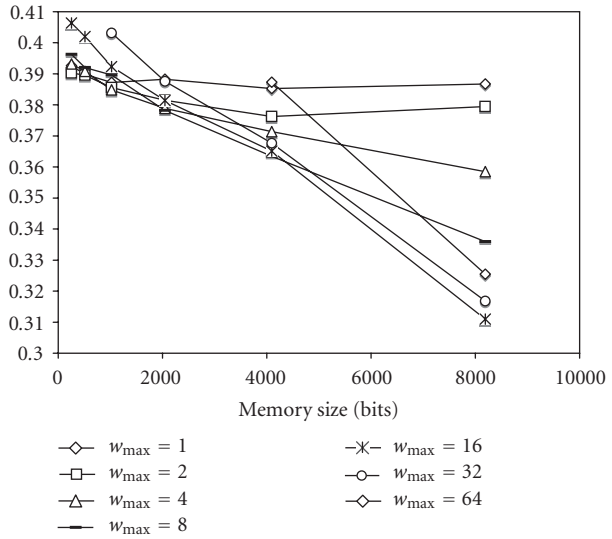

Figure 8: Impact on logic energy when increasing memory size.

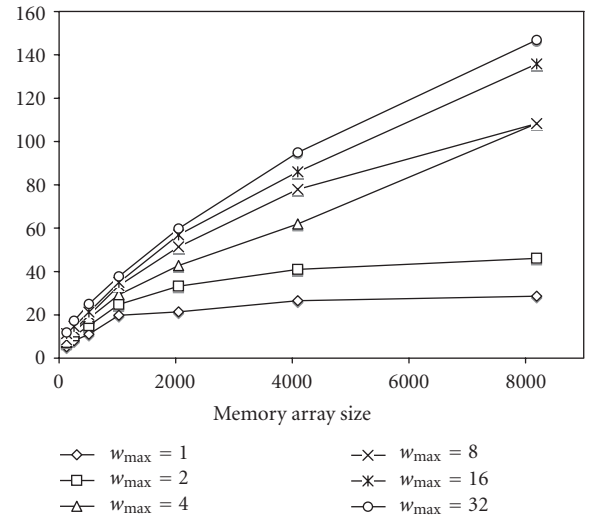

Figure 9: Impact on amount of packable LUTs when increasing memory size.

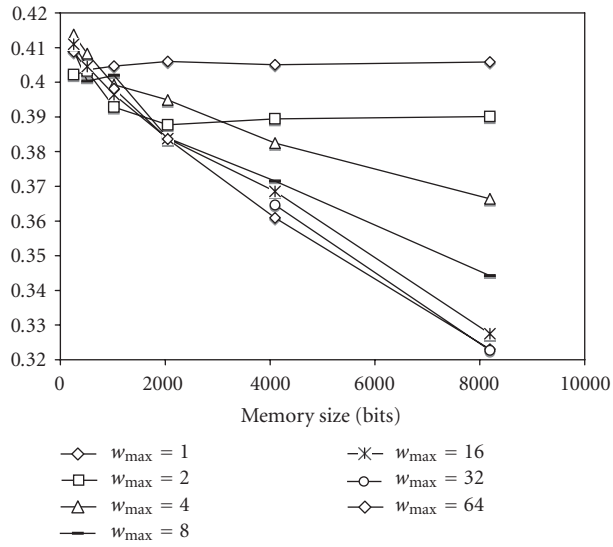

Figure 10: Impact on routing energy when increasing memory size.

B (which would have an integral square root). This explains the "bumpy" appearance of the lines in Figure 7.

Next consider the impact on the logic energy. The results are shown in Figure 8 (note that the y-axis does not cross at the origin). As expected, a larger array means that more logic can be packed into the array, which reduces the logic energy. However, for $w_{eff} = \{1\}$ and $w_{eff} = \{1, 2\}$, the trend is relatively flat. This is because, as shown in Figure 9, when the flexibility is low, increasing the array size does not lead to any significant increase in the amount of packed logic.

Figure 10 shows the impact on routing energy when increasing the memory array size (again note that the y-axis does not cross at the origin). The significant reduction in the number of lookup tables directly translates to a reduction in the number of nets, and hence a reduction in the routing energy.

Figure 11 shows the results for the overall energy (including dynamic and leakage). Despite the reduction in logic and routing energy, the memory power still dominates. Hence, the overall energy increases as the memory size increases.

Since we are not able to vary B on our Altera part, we are not able to perform this experiment on the actual commercial device. However, by comparing the results summarized in Figures 6(a) and 6(b), we found that using a 4 kbit block consumes approximately 8% more power on average than the 512-bit block when $N = 1$; this matches the conclusion drawn from the VPR experiments.

3.2.3. *Memory block flexibility*

In this section, we investigate the power implications of changing the flexibility of each memory block. As described earlier, FPGA memory blocks typically have a configurable output width; the set of output widths in which an array can

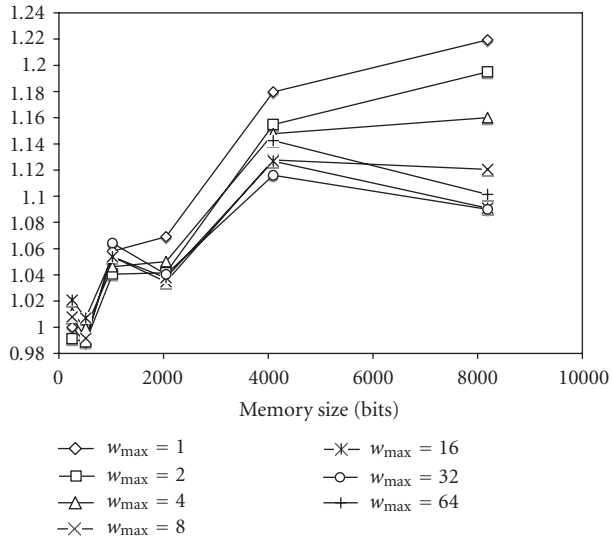

Figure 11: Impact on overall energy when increasing memory size.

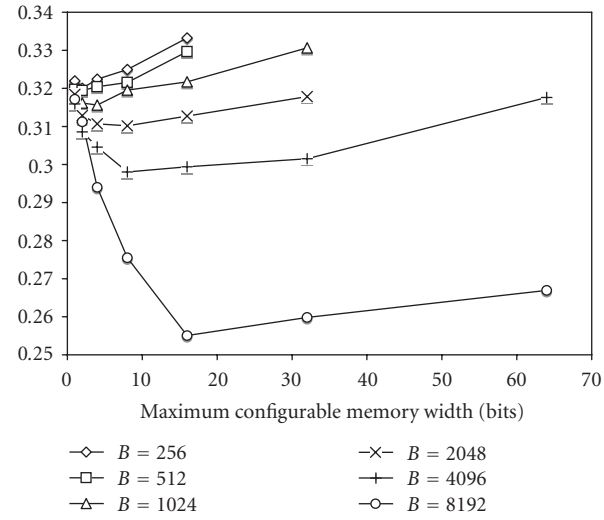

Figure 13: Impact on routing energy when increasing memory flexibility.

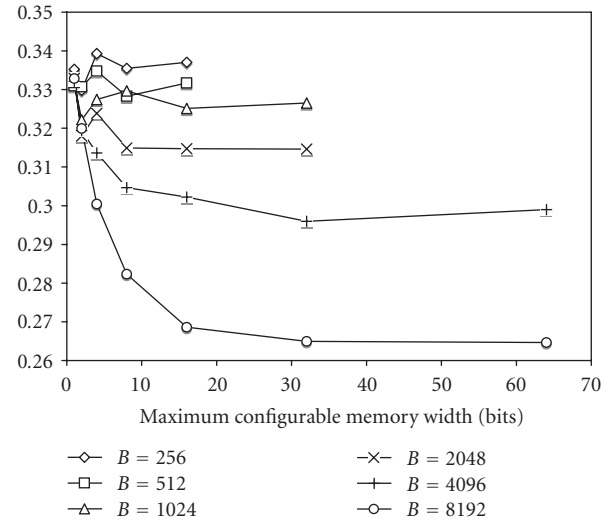

Figure 12: Impact on logic energy when increasing memory flexibility.

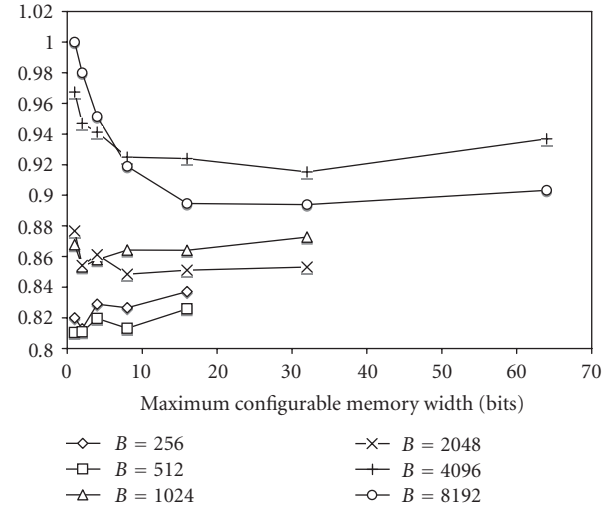

Figure 14: Impact on overall energy when increasing memory flexibility.

be configured is denoted w_{eff}. In this section, we vary the maximum value in w_{eff}, which we will denote w_{max} and measure the impact on energy dissipation.

Intuitively, changing w_{eff} will have little affect on the power dissipated in the memory array itself. Changing w_{eff} does not affect the shape of the physical array; it only affects the multiplexers that are used to select values from the array. Since these multiplexers are small, they would affect only a very small increase in array power as we increase the value of w_{max}. In our experiments reported in this section, we ignore this increase.

However, as the flexibility increases, the amount of logic that can be packed into each array might increase. In that case, we would expect the overall power dissipation to drop as the flexibility is increased.

To investigate this, we fix the number of memory arrays at one and vary the maximum output width, w_{max}. SMAP is then free to choose a configuration for each memory ranging from $B \times 1$ to $(B/w_{max}) \times w_{max}$, where the widths are in powers of two.

First consider the energy dissipated in the logic blocks. Figure 12 shows the impact of increasing w_{max}, on logic energy. As w_{max} increases, the logic energy decreases. This is because the amount of logic implemented using lookup-tables is reduced. For large values of w_{max} , this impact is much less pronounced than for smaller values; this is because SMAP rarely produces solutions which use a very large number of outputs. In Figure 12, there are several cases where increasing flexibility increases logic power. This is counterintuitive. Since we are using a simulated-annealing-based placement

algorithm, small differences in the netlist may result in different placements; these differences may overshadow any small differences caused by SMAP.

Figure 13 shows the routing energy, when w_{max} is increased. In general, as w_{max} increases, the routing energy decreases. However, for large values of w_{max}, the power dissipation increases; this is because the signals driven by the memory outputs also need to be routed, and a larger value of w_{max} leads to solutions which use more memory outputs.

Figure 14 shows the impact on the total energy per cycle. Since logic and routing energy decrease and memory energy is constant, the overall impact is a decrease in energy.

Again, since we are not able to vary w_{eff} on our Altera part, we are not able to perform this experiment on the commercial device.

4. POWER-AWARE HETEROGENEOUS TECHNOLOGY MAPPING

The previous section showed that implementing logic in memory using the SMAP algorithm results in a severe power penalty. Yet, the significant density improvements reported in [4, 5] may motivate designers (or CAD tools) to implement logic in memory blocks anyway, especially if the design is up against the capacity limits of target FPGA. In this case, it is important to map logic to memories in as power-efficient a manner as possible. In this section, we describe and evaluate an enhancement to SMAP to make it power-aware.

4.1. Algorithm enhancement

Although it would be possible to perform the cut selection, output selection, or seed selection in a power-aware manner similar to [24], our experiments (not reported here) indicate that this does not work well. These methods helped to reduce routing energy slightly but did not address the large amounts of energy consumed by the memory blocks. Instead, we change the way multiple memory arrays are used to implement logic.

The key idea in this technique is to combine two or more *physical* memory arrays into larger *logical* memories and use SMAP to map logic to these larger logical memories. The idea of combining physical memories into larger logical memories was first presented in [5] as an attempt to reduce the run-time of SMAP. The original SMAP algorithm maps to the memory arrays sequentially, which can lead to long run-times if there are a large number of memory arrays. By combining physical memories to create fewer larger arrays, fewer iterations of SMAP are required, leading to significantly improved run-time. An example of this is shown in Figure 15(a). In this example, two physical arrays with $B = 512$ and $w_{eff} = \{1, 2, 4, 8, 16\}$ are combined to implement a single *logical* memory with $B = 1024$ and $w_{eff} = \{2, 4, 8, 16, 32\}$. In this example, each physical array supplies half of the bits in each word. This larger logical array is then treated as a single entity in SMAP, meaning only one iteration of the SMAP algorithm is required.

Figure 15(b) shows another way in which the two memory arrays can be combined to create a single larger logical memory. In this case, the resulting logical memory has $B = 1024$ and $w_{eff} = \{1, 2, 4, 8, 16\}$. In this organization, all bits for each word are stored in the same physical array. Nine of the address lines are provided to each physical array, and the tenth address line is used to select the output bits from one of the two arrays. This latter organization has the potential for lower power, since the array that is not currently being accessed can be turned off (using the memory enable signal). This is the key to the enhancement described in this section; we combine memory arrays into larger logical arrays such that all but one of the arrays can be turned off with each access. Note that this is similar to the technique described in [25], however, they did not evaluate this idea in the context of heterogeneous technology mapping.

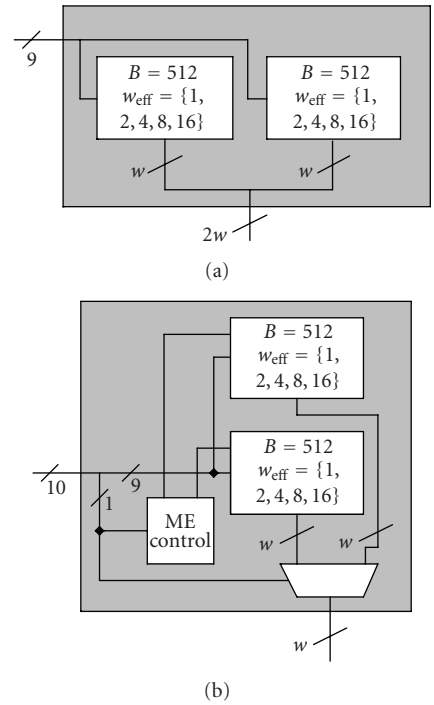

FIGURE 15: Forming logical memories: (a) area efficient and (b) power efficient.

In general, more than two arrays can be combined into a larger logical memory. In [5], the number of physical memories used to form a logical memory is termed the *blocking factor, (BF)*. In the examples in Figure 15, BF = 2. Although this technique will reduce the memory power, it has two potential drawbacks.

(1) Extra LUTs are needed to implement the ME control and output multiplexers. These extra logic elements will consume power, and will also reduce the overall packing efficiency of the technique.

(2) As shown in [5], increasing BF tends to reduce the amount of logic that can be packed into a set of physical memory arrays. Again, this will tend to increase the power dissipation and reduce the packing efficiency of our technique.

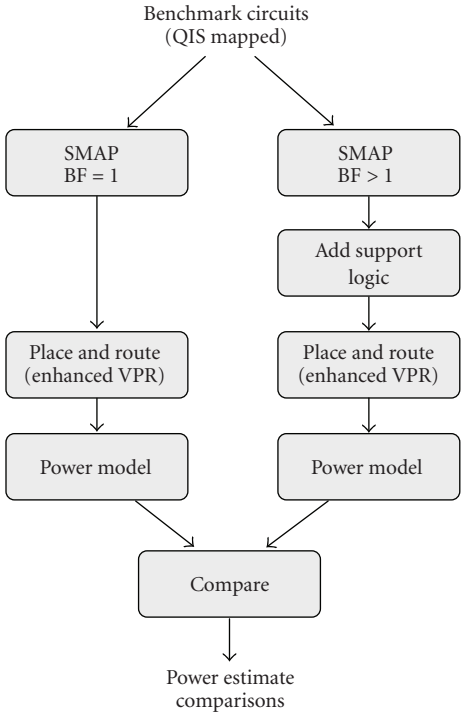

Figure 16: Methodology for evaluating enhanced algorithm.

4.2. Experimental methodology

In the remainder of this section, we determine whether the proposed technique reduces the energy consumption of implementations generated using SMAP, or whether the energy increase due to the extra logic and reduced packing efficiency results in an overall energy increase.

Figure 16 shows our experimental methodology. The original SMAP flow from Section 3 is our baseline; in this flow, the memory arrays are filled sequentially, and memories are not combined in any way. In the enhanced flow, we first map each circuit using SMAP with BF > 1. For each value of BF, we chose the "deepest" organization possible, that is, we choose an organization in which all bits that make up each word are stored in a single memory array, since this allows us to turn off all but one memory array on each access. This version of SMAP is also aware of the LUTs that need to be introduced for output multiplexing. Due to this awareness, SMAP will only choose wider configurations when the number of packed LUTs can overcome the overhead required for the output multiplexers. We then add the support logic needed (multiplexers and a memory enable controller); this extra logic is implemented in LUTs. Finally, both the baseline and enhanced flow mappings are placed and routed using our enhanced version of VPR, and the enhanced power model is used to estimate the power and energy of the resulting implementation.

Table 2 summarizes the values of N and BF that we explored. The left half of the table shows the experiments when using 512-bit physical memories, and the right half of the table shows the experiments when using 4096-bit physical memories.

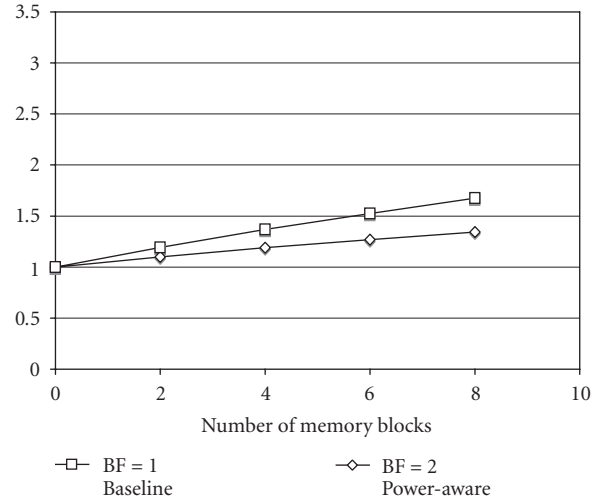

Figure 17: Impact on energy when increasing the number of 512-bit memories.

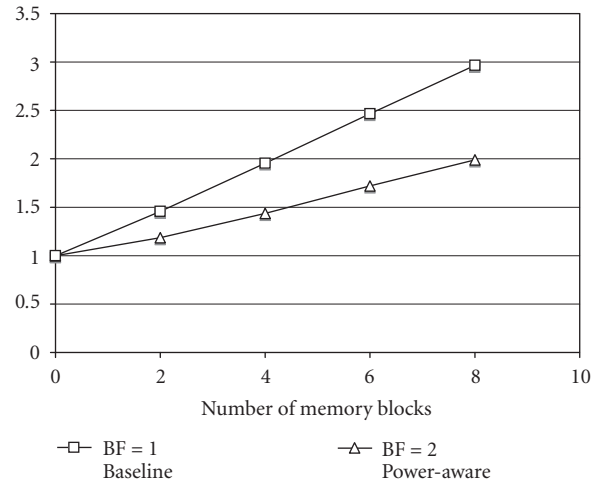

Figure 18: Impact on energy when increasing the number of 4096-bit memories.

4.3. Experimental results

We first consider the packing efficiency of our new mapping technique. As previously explained, we would expect a decrease in the amount of logic that can be mapped to each memory array. The number of LUTs that can be packed into the memory arrays for each benchmark circuit is shown in Table 3 (for $B = 512$) and Table 4 (for $B = 4096$). The columns labeled BF = 1 correspond to the original SMAP algorithm. The columns labeled BF = 2, BF = 4, and BF = 8 correspond to the power-aware technique described in this section. For BF > 1, the number of LUTs required to implement the memory enable control and output multiplexers has been subtracted from the number of LUTs packed into memory; if the result is negative, a "—" is shown in the table (this means that our version of SMAP actually increased the number of LUTs in the design).

Table 2: Summary of experiments for evaluating enhanced algorithm.

(a) $B = 512$

Experiment	Logical size	N			
Baseline	512	2	4	6	8
BF = 2	1024	1	2	3	4
BF = 4	2048	—	1	—	2
BF = 8	4096	—	—	—	1

(b) $B = 4096$

Experiment	Logical size	N			
Baseline	4096	2	4	6	8
BF = 2	8192	1	2	3	4
BF = 4	16384	—	1	—	2
BF = 8	32768	—	—	—	1

Table 3: Number of LUTs packed into memory blocks ($B = 512$).

Circuit	$N = 2$		$N = 4$			$N = 6$		$N = 8$			
	Base BF = 1	BF = 2	Base BF= 1	BF = 2	BF = 4	Base BF = 1	BF = 2	Base BF = 1	BF = 2	BF = 4	BF = 8
alu4	34	42	68	50	39	100	56	132	62	57	96
apex2	32	2	63	3	—	85	4	103	5	—	—
apex4	106	103	212	206	198	318	309	421	404	386	354
bigkey	15	3	21	5	—	26	7	28	9	—	—
clma	34	6	68	13	4	101	18	133	23	15	9
des	18	3	34	5	—	50	7	66	9	—	—
diffeq	15	3	23	6	—	31	9	39	12	—	—
dsip	19	8	23	9	—	26	10	28	12	—	—
elliptic	13	1	23	2	—	32	3	40	4	—	—
ex5p	46	37	79	61	62	104	80	125	94	94	51
ex1010	95	93	187	182	159	277	271	365	358	312	304
frisc	15	2	27	3	—	37	4	45	6	—	—
misex3	34	9	68	17	10	101	24	133	30	13	8
pdc	63	54	108	90	61	147	118	181	144	89	16
s298	106	104	212	207	181	316	310	358	331	294	234
s38417	34	5	68	10	8	89	14	107	18	15	—
s38584	39	12	77	24	9	114	36	150	46	21	—
seq	33	13	65	17	11	97	21	129	24	10	—
spla	60	51	100	82	55	135	107	168	127	67	7
tseng	16	2	27	3	—	37	3	47	3	3	—

As the tables show, the number of LUTs packed into the memory arrays decreases as BF is increased. For BF = 4 or BF = 8, there are many circuits in which our new version of SMAP could not find a mapping solution that could overcome the overhead of the memory support logic. Thus, in the remainder of this section, we do not consider BF > 2.

Figures 17 and 18 show the impact on energy averaged across all twenty benchmarks for $B = 512$ and $B = 4096$, respectively. The horizontal axis is the number of physical memory arrays, and the vertical axis is the overall energy (including dynamic and leakage), normalized to the case when no memories are used. The upper line in each graph corresponds to the original SMAP, while the lower line corresponds to the power-aware version described in this section, with BF = 2. As the graphs show, the enhancements described in this section reduce the energy required to implement the benchmark circuits by an average of 19.8% and 32.9% for eight 512-bit and 4096-bit memories, respectively, when compared to the original SMAP algorithm.

5. LIMITATIONS OF THIS STUDY

There are several limitations to this study related to the tools used in the experimental investigation, the FPGA architecture, and the circuit-level design of the FPGA.

The conclusions rely heavily on the tool used to perform logic synthesis, technology mapping, and heterogeneous technology mapping. For the logic synthesis and technology

Table 4: Number of LUTs packed into memory blocks ($B = 4096$).

Circuit	$N = 2$		$N = 4$			$N = 6$		$N = 8$			
	Base BF = 1	BF = 2	Base BF = 1	BF = 2	BF = 4	Base BF = 1	BF = 2	Base BF = 1	BF = 2	BF = 4	BF = 8
alu4	163	152	239	209	263	311	261	379	288	552	545
apex2	64	5	110	10	—	143	13	165	16	—	—
apex4	780	763	843	820	786	845	824	847	828	—	692
bigkey	25	7	31	9	—	33	11	35	13	—	—
clma	72	18	140	35	22	207	50	272	64	33	36
des	36	22	69	56	44	101	85	133	107	90	62
diffeq	19	3	35	6	—	47	9	56	12	—	—
dsip	26	8	30	10	—	32	12	34	14	—	—
elliptic	15	1	29	2	—	41	3	53	4	—	—
ex5p	159	122	175	134	82	177	136	179	138	—	—
ex1010	714	705	877	864	826	879	869	880	871	—	878
frisc	22	3	40	5	—	53	7	62	8	—	—
misex3	76	29	143	54	187	209	80	273	96	242	234
pdc	297	141	504	231	82	652	308	764	373	120	84
s298	477	437	697	657	640	735	677	737	686	651	610
s38417	75	15	104	29	12	132	38	156	47	18	—
s38584	52	20	102	38	17	152	53	198	68	25	21
seq	80	21	145	29	15	193	34	229	38	31	—
spla	243	124	369	177	62	442	211	508	239	116	78
tseng	23	2	37	4	—	48	5	58	6	—	—

mapping, we used the commercial Altera Quartus II tool. Alternatives such as SIS/Flowmap were also attempted [8, 18]. In those experiments, the number of lookup-tables required to implement each benchmark circuit was slightly higher. As a result, on average, SMAP was able to pack more LUTs into each memory array, leading to more optimistic results. Even though the results were different, the conclusions remained the same. Technology mapping and logic synthesis for FPGAs are well-studied, so we do not feel that future algorithms will provide significantly different LUT mappings, and hence result in significantly different conclusions. Power-aware mappers also exist [24, 26, 27], however, again we would not expect these to lead to significantly different conclusions.

The heterogeneous technology mapping algorithm, SMAP, has a more significant impact on the results. Other mapping approaches, in which logic subcircuits are mapped directly to memories, may lead to different conclusions. We suggest that designers of future heterogeneous technology mapping algorithms should perform an evaluation of the power characteristics of their results, similar to the studies presented in this paper.

Throughout this study, we assumed a relatively realistic FPGA architecture. Yet, it does deviate from commercial architectures in some ways; in particular, commercial FPGAs contain much more support logic within each logic block, such as carry chains. Including these features in our model would affect our results in two ways. First, the power penalty might be increased, since LUTs can more efficiently implement logic circuits. On the other hand, these extra features all consume power and would not be required in a memory array. A different routing architecture might also slightly affect the results; low-power routing architectures might make the memory implementation of logic less appealing. However, we expect that this impact would be small.

Finally, our study made several assumptions regarding the circuit-level design of the FPGA, in particular, the embedded memory arrays. To make our results as realistic as possible, we used the memory implementation obtained from a commercial memory generator. However, if low-power memory techniques were employed, it may be possible to improve the power-efficiency of these blocks, perhaps even to the point where implementing logic in memories is more power-efficient than implementing logic in lookup-tables. The use of low-power memories in FPGAs is an area of future work.

6. CONCLUSIONS

In this paper, we have shown that implementing logic in FPGA embedded memory arrays leads to an increase in power dissipation of the resulting circuit. This is an important result. Previous papers have reported significant density increases when embedded memory is used in this way, and suggested that there is no reason not to do this. As a result, more and more embedded arrays are being included in FPGAs, with the understanding that if they cannot be used for storage, they can be used for logic. The results of this paper show that if power is a concern, this may be a bad idea. If designers (or CAD tools) wish to implement logic in memory arrays, it is important to carefully tradeoff the

power penalties with the potential increase in density. Even if a memory array is not required for storage, these results suggest that, from a power perspective, it is better to leave the array unused, rather than use it to implement logic.

That being said, there are times when the density improvement may motivate the mapping of logic to embedded memory arrays. In that case, optimizing the size and flexibility of the memory blocks to reduce this power penalty is important. We have shown that smaller-memory arrays are more power efficient than large arrays, but that for most array sizes, the arrays should be as flexible as possible.

Finally, we showed that we could achieve power savings by combining multiple physical memories into larger logical memories and mapping logic to the logical memories. To form the logical memories from the physical memories, we used a power-efficient arrangement that allows one or more of the physical memories to be disabled in each cycle. However, using this technique requires additional support logic implemented in LUTs. Overall, when using eight memories and BF = 2, we found an average reduction in overall energy of 19.79% and 32.93% for 512-bit and 4096-bit memories, respectively.

ACKNOWLEDGMENT

This research was supported by Altera and the National Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Xilinx Corporation, "Virtex-4 User Guide," September 2005.

[2] Altera Corporation, "Stratix II Device Handbook 2," December 2005.

[3] T. Ngai, J. Rose, and S. J. E. Wilton, "An SRAM-Programmable Field Configurable Memory," in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 499–502, Santa Clara, Calif, USA, May 1995.

[4] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proceedings of the 6th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 179–188, Monterey, Calif, USA, February 1998.

[5] S. J. E. Wilton, "Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 1, pp. 56–68, 2000.

[6] D. Hodges, H. Jackson, and R. Saleh, "Analysis and Design of Digital Integrated Circuits," in *Deep Submicron Technology*, McGraw-Hill, New York, NY, USA, 3rd edition, 2004.

[7] S. Y. L. Chin, C. S. P. Lee, and S. J. E. Wilton, "Power implications of implementing logic using FPGA embedded memory arrays," in *Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL '06)*, pp. 1–8, Madrid, Spain, August 2006.

[8] J. Gong and Y. Ding, "Combinational logic synthesis for LUT based field programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 1, no. 2, pp. 145–204, 1996.

[9] Actel Corporation, "SmartPower User's Guide," http://www.actel.com/documents/smartpower_ug.pdf, 2006.

[10] Xilinx Corporation, "XPower analyzer. In Xilinx ISE 8.2i Software Manual," 2006.

[11] Altera Corporation, "PowerPlay Power Analysis. In Quartus II 6.0 Handbook," volume 3, 2006.

[12] Xilinx Corporation, "XPower Estimator (Spreadsheet)," http://www.xilinx.com/products/design_resources/power_central/index.htm, 2006.

[13] Altera Corporation, "PowerPlay Early Power User Guide," 2006.

[14] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 279–302, 2005.

[15] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD For Deep-Submicron FPGAs*, Kluwer Academic Publisher, New York, NY, USA, 1999.

[16] Berkeley Logic Synthesis Verification Group, "Berkeley Logic Interchange Format (BLIF)," http://vlsi.colorado.edu/ vis/blif.ps.

[17] Altera Corporation, "Quartus II University Interface Program," 2007.

[18] E. M. Sentovich, "SIS: a system for sequential circuit analysis," UCB/ERL M92/41, Electronics Research Laboratory, University of California, Berkeley, Calif, USA, 1992.

[19] J. Cong and Y. Ding, "Flowmap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[20] J. Lamoureux and S. J. E. Wilton, "Activity estimation for field-programmable gate arrays," in *Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL '06)*, pp. 87–94, Madrid, Spain, August 2006.

[21] S. Y. L. Chin, "Power Implications of Implementing Logic using FPGA Embedded Memory Blocks," M.S. thesis, M.S. thesis, University of British Columbia, Vancouver, Canada, 2006.

[22] "Virage Logic Memory Compiler," http://www.viragelogic.com.

[23] S. J. E. Wilton, S.-S. Ang, and W. Luk, "The impact of pipelining on energy per operation in field-programmable gate arrays," in *Proceedings of the 14th International Conference on Field Programmable Logic and Application (FPL '04)*, vol. 3203 of *Lecture Notes in Computer Science*, pp. 719–728, Leuven, Belgium, August 2004.

[24] J. Lamoureux and S. J. E. Wilton, "On the interaction between power-aware computer-aided design algorithms for field-programmable gate array," *Journal of Low Power Electronics*, vol. 1, no. 2, pp. 119–132, 2005.

[25] R. Tessier, V. Betz, D. Neto, and T. Gopalsamy, "Power-aware RAM mapping for FPGA embedded memory blocks," in *Proceedings of the 14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '06)*, pp. 189–198, Monterey, Calif, USA, February 2006.

[26] D. Chen, J. Cong, F. Li, and L. He, "Low-power technology mapping for FPGA architectures with dual supply voltages," in *Proceedings of the 12th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '04)*, vol. 12, pp. 109–117, Monterey, Calif, USA, February 2004.

[27] J. H. Anderson, F. N. Najm, and T. Tuan, "Active leakage power optimization for FPGAs," in *Proceedings of the 12th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '04)*, vol. 12, pp. 33–41, Monterey, Calif, USA, February 2004.