POWER IMPLICATIONS OF IMPLEMENTING LOGIC USING FIELD-PROGRAMMABLE GATE ARRAY EMBEDDED MEMORY BLOCKS

by

SCOTT YIN LUNN CHIN

B.Eng., University of Victoria, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

August 2006

© Scott Yin Lunn Chin, 2006
ABSTRACT

POWER IMPLICATIONS OF IMPLEMENTING LOGIC USING FIELD-PROGRAMMABLE GATE ARRAY EMBEDDED MEMORY BLOCKS

Modern field-programmable gate arrays (FPGAs) are used to implement entire systems, and these systems often require storage. FPGA vendors have responded by incorporating two types of embedded memory resources into their architectures: dedicated and non-dedicated. The dedicated embedded memory blocks lead to much denser memory implementations and are therefore very efficient for implementing large systems 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 algorithms that automatically map logic circuits to FPGAs with both large ROMs and small lookup tables. These previous studies, however, did not consider the impact on power. Power has become a first-class concern among FPGA vendors.

In this thesis, we develop a power model for FPGAs that contain embedded memories, and apply it to investigate the impact of various embedded memory architectural parameters on power dissipation when using memories to implement logic. From this study, we find that mapping logic to memories incurs a significant power penalty due to the power consumed in the embedded memories. We then investigate two possible ways to reduce this power penalty at the CAD level, one of which we found to be effective.
TABLE OF CONTENTS

ABSTRACT .................................................................................................................. II
TABLE OF CONTENTS................................................................................................. III
LIST OF TABLES........................................................................................................ V
LIST OF FIGURES....................................................................................................... VI
ACKNOWLEDGEMENTS ............................................................................................... VIII

1 INTRODUCTION ...................................................................................................... 1
  1.1 MOTIVATION ................................................................................................... 1
  1.2 RESEARCH GOALS AND CONTRIBUTIONS .................................................. 2
  1.3 THESIS ORGANIZATION ............................................................................... 3

2 BACKGROUND AND PREVIOUS WORK .................................................................. 4
  2.1 FPGA ARCHITECTURE ................................................................................... 4
    2.1.1 Logic Element Architecture .................................................................... 5
    2.1.2 Clusters .................................................................................................... 7
    2.1.3 Embedded Block Memory Architecture ................................................ 8
    2.1.4 Routing Architecture .............................................................................. 10
    2.1.4.1 Wire Segments .................................................................................... 11
    2.1.4.2 Connection Blocks and Switch Blocks ............................................... 12
    2.1.4.3 Programmable Connections .............................................................. 12
  2.2 FPGA CAD ....................................................................................................... 13
    2.2.1 Technology Mapping .............................................................................. 14
    2.2.2 Clustering ................................................................................................ 15
    2.2.3 Placement ................................................................................................ 15
    2.2.4 Routing ................................................................................................... 16
  2.3 HETEROGENEOUS TECHNOLOGY MAPPING .................................................. 17
    2.3.1 Terminology ............................................................................................ 17
    2.3.2 Existing Algorithms ............................................................................... 18
    2.3.3 SMAP ....................................................................................................... 19
  2.4 POWER ESTIMATION ...................................................................................... 21
    2.4.1 Switching Activity ................................................................................... 21
    2.4.1.1 Transition Density Model ................................................................... 22
    2.4.1.2 Lag One Model ................................................................................... 23
    2.4.1.3 ACE2.0 .............................................................................................. 24
    2.4.2 Power Estimation for FPGAs ................................................................. 25
    2.4.3 Power Estimation for Memories and Caches ......................................... 26
  2.5 FOCUS AND CONTRIBUTION OF THESIS .................................................... 29

3 POWER MODEL FOR FPGAS CONTAINING EMBEDDED MEMORIES .................. 31
  3.1 ACTIVITY ESTIMATION ............................................................................... 31
    3.1.1 Read Only Memory .................................................................................. 32
    3.1.2 Random Access Memory ....................................................................... 33
    3.1.3 Framework and Integration into ACE2.0 ............................................... 34
  3.2 POWER ESTIMATION ...................................................................................... 37
    3.2.1 Fixed-Size Memory ................................................................................ 38
    3.2.2 Programmable Column Decoder ............................................................ 38
    3.2.3 Framework and Implementation of the Power Estimator ....................... 40
  3.3 SUMMARY ....................................................................................................... 43

4 POWER IMPLICATIONS OF MAPPING LOGIC TO MEMORIES ............................. 44
LIST OF TABLES

Table 2-1. Aspect Ratios of Memories in Commercial FPGAs [2, 3, 19, 20] .................. 8
Table 3-1. VPR Memory Power Parameters ............................................................................. 41
Table 4-1. Benchmark Characteristics ..................................................................................... 46
Table 4-2. Parameters Under Investigation ............................................................................. 50
Table 5-1. Percentage Change in Routing Energy When Using the Activity Aware Cost Function and Memories with B = 512 bits ................................................................. 75
Table 5-2. Percentage Change in Logic Energy When Using the Activity Aware Cost Function and Memories with B= 512 ........................................................................ 76
Table 5-3. Percentage Change in Overall Energy When Using the Activity Aware Cost Function and Memories with B= 512 ........................................................................ 76
Table 5-4. Number of LUTs Needed For Power Efficient Logical Memories ......................... 80
Table 5-5. Summary of Experiments (left: B=512 bits, right: B=4096 bits) .......................... 81
Table 5-6. LUTs Removed After Mapping (B=512) ................................................................. 82
Table 5-7. LUTs Removed After Mapping (B=4096) ............................................................... 83
Table 5-8. Average Percent Change in Energy When Using BF=2 ....................................... 87
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Conceptual FPGAs. Left: Traditional. Right: Heterogeneous</td>
<td>5</td>
</tr>
<tr>
<td>2-2</td>
<td>2-LUT Configured as an AND Gate</td>
<td>6</td>
</tr>
<tr>
<td>2-3</td>
<td>LUT Paired with Flip-Flop</td>
<td>6</td>
</tr>
<tr>
<td>2-4</td>
<td>Cluster Architecture</td>
<td>7</td>
</tr>
<tr>
<td>2-5</td>
<td>High Level Embedded Memory Block Architecture</td>
<td>9</td>
</tr>
<tr>
<td>2-6</td>
<td>Programmable Column Decoder Architecture</td>
<td>10</td>
</tr>
<tr>
<td>2-7</td>
<td>Island-Style FPGA Routing Architecture</td>
<td>11</td>
</tr>
<tr>
<td>2-8</td>
<td>Connection Types a) unbuffered b) buffered uni-directional c) buffered bi-directional</td>
<td>13</td>
</tr>
<tr>
<td>2-9</td>
<td>CAD Flow</td>
<td>13</td>
</tr>
<tr>
<td>2-10</td>
<td>Technology Mapping Example</td>
<td>14</td>
</tr>
<tr>
<td>2-11</td>
<td>Example of Mapping Logic to a Memory Array</td>
<td>20</td>
</tr>
<tr>
<td>2-12</td>
<td>Glitch Filtration</td>
<td>24</td>
</tr>
<tr>
<td>2-13</td>
<td>Typical SRAM Memory Architecture</td>
<td>27</td>
</tr>
<tr>
<td>3-1</td>
<td>Replacing a ROM with Equivalent Nodes and Registers</td>
<td>32</td>
</tr>
<tr>
<td>3-2</td>
<td>Integration of Memory Activity Estimation into ACE2.0</td>
<td>35</td>
</tr>
<tr>
<td>3-3</td>
<td>Pseudo-Code for RAM Simulator</td>
<td>36</td>
</tr>
<tr>
<td>3-4</td>
<td>Transistor Level Modelling of the Programmable Column Decoder</td>
<td>39</td>
</tr>
<tr>
<td>3-5</td>
<td>Modelling of of LUTs in the Poon Power Model (from [57])</td>
<td>40</td>
</tr>
<tr>
<td>4-1</td>
<td>Flow for VPR-Based Experiments</td>
<td>46</td>
</tr>
<tr>
<td>4-2</td>
<td>Test Harness for Board Measurements</td>
<td>49</td>
</tr>
<tr>
<td>4-3</td>
<td>Impact on Energy When Increasing the Number of 512bit Memory Arrays</td>
<td>51</td>
</tr>
<tr>
<td>4-4</td>
<td>Impact on Energy When Increasing the Number of 4kBit Memory Arrays</td>
<td>53</td>
</tr>
<tr>
<td>4-5</td>
<td>Number of Packed 4LUTs When Increasing the Number of Memories</td>
<td>53</td>
</tr>
<tr>
<td>4-6</td>
<td>Impact on Energy When Increasing the Number of 512bit Memory Arrays</td>
<td>55</td>
</tr>
<tr>
<td>4-7</td>
<td>Impact on Energy When Increasing the Number of 4kBit Memory Arrays</td>
<td>55</td>
</tr>
<tr>
<td>4-8</td>
<td>Impact on Memory Energy When Increasing Memory Array Size</td>
<td>57</td>
</tr>
<tr>
<td>4-9</td>
<td>Impact on Logic Energy When Increasing Memory Size</td>
<td>58</td>
</tr>
<tr>
<td>4-10</td>
<td>Impact on Amount of Packable LUTs When Increasing Memory Size</td>
<td>58</td>
</tr>
<tr>
<td>4-11</td>
<td>Impact on Routing Energy When Increasing Memory Size</td>
<td>59</td>
</tr>
<tr>
<td>4-12</td>
<td>Impact on Overall Energy When Increasing Memory Size</td>
<td>60</td>
</tr>
<tr>
<td>4-13</td>
<td>Impact on Logic Energy When Increasing Memory Flexibility</td>
<td>62</td>
</tr>
<tr>
<td>4-14</td>
<td>Impact on Routing Energy When Increasing Memory Flexibility</td>
<td>64</td>
</tr>
<tr>
<td>4-15</td>
<td>Impact on Overall Energy When Increasing Memory Flexibility</td>
<td>64</td>
</tr>
<tr>
<td>5-1</td>
<td>Node Replication in SMAP</td>
<td>70</td>
</tr>
<tr>
<td>5-2</td>
<td>Reducing Cut-Set Fanout</td>
<td>71</td>
</tr>
<tr>
<td>5-3</td>
<td>Number of Packed LUTs Using the Activity Aware Cost Function</td>
<td>74</td>
</tr>
<tr>
<td>5-4</td>
<td>Forming Logical Memories a)Area Efficient b)Power Efficient</td>
<td>78</td>
</tr>
</tbody>
</table>
Figure 5-5. Methodology For Power-Efficient Super-Arrays ........................................... 81
Figure 5-6. Distribution of How the Number of LUTs That Can be Removed Are
Affected for 512Bit Memories When BF=2 ................................................................. 84
Figure 5-7. Distribution of How the Number of LUTs That Can be Removed Are
Affected for 4096Bit Memories When BF=2 ............................................................... 84
Figure 5-8. Impact on Energy When Increasing the Number of 512bit Memories .......... 85
Figure 5-9. Impact on Energy When Increasing the Number of 4096bit Memories ...... 86
ACKNOWLEDGEMENTS

The first person that I'd like to thank is my supervisor Dr. Steve Wilton. Although there were many candidates more qualified than me, Dr. Wilton took a chance by giving me the opportunity to be a part of his research group. Through his dedication to his students, I gained more than just a technical education in my masters program; Dr. Wilton has exposed me to every aspect of a career in research. Without his guidance, encouragement, and humor, this thesis would not have been possible.

To all the members of the System on Chip research group, I would like to thank you for all the insightful conversations and above all else, the friendship and company. Special mention to the FPGA research group – Brad, Cary, David Grant, David Yeager, Eddy, Eric, Jason, Julien, Mark, Martin, Marvin, Nathalie, and Usman; the boys from MCLD315 – Amit, Karim, Reza, and XiongFei; and other SoC students who talked to me – Derek, David Chiu, Dipanjan, Melody, Neda, Rod, and Shirley. I especially thank all the professors who have taught me at UBC, and my committee members for taking the time to read my thesis and giving me thoughtful feedback.

I greatly appreciate the financial support provided by the Altera Corporation and the Natural Sciences and Engineering Research Council of Canada. Without their support, this work would not be possible.

For their unending love and support, I dedicate this thesis to my wonderful parents Yvonne and Philip Chin. Finally, I want to thank Marie O’Connor for always being there to love, support, encourage, and inspire.
1 INTRODUCTION

1.1 Motivation

On-chip user memory has become an essential and common component of modern field-programmable gate arrays (FPGAs). Modern FPGAs are used to implement entire systems, and these systems often require storage. FPGA vendors have responded to this by incorporating two types of memory resources into their architectures: non-dedicated memories and dedicated memories. The Xilinx Distributed SelectRAM [1], in which lookup-tables can be configured as small RAMs, is an example of a non-dedicated memory architecture. The Altera TriMatrix Memory [2] and Xilinx Block SelectRAM [3], which are dedicated memory arrays embedded into the FPGA fabric, are examples of dedicated memory architectures.

The dedicated embedded memory arrays lead to much denser memory implementations and are therefore very efficient for implementing large systems that require storage [4]. However, for logic intensive circuits that do not require storage, the chip area devoted to the dedicated 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 algorithms that automatically map logic circuits to heterogeneous FPGAs with both large ROMs and small lookup tables [5-7]. 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 of the logic in lookup-tables.

These previous studies, however, did not consider the impact on power. Power has become a first-class concern among FPGA vendors, and is often the limiting factor in handheld battery-powered applications. FPGAs are power-hungry for two 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.

1.2 Research Goals and Contributions

There are three objectives to this research:

1. To design a flexible power model for FPGA embedded memory blocks that can be integrated into the Poon Power model [8] and the commonly used academic Versatile Place and Route (VPR) CAD Suite [9]. This model must be flexible enough to target different memory architectures.

2. To use the power model to investigate how the FPGA embedded memory architecture impacts the overall power consumption when the memories are used to implement logic.

3. To apply power-aware techniques to existing algorithms that map logic to memories and investigate their impact on power using the power model.
1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 provides the background material for this research: an overview of FPGA architecture and CAD, algorithms for mapping logic to memories (heterogeneous technology mapping), and power estimation techniques. Chapter 3 proposes the new flexible power model for FPGAs that contain embedded memory blocks. Chapter 4 presents a study on how power is affected by the architecture of the embedded memory blocks when used to implement logic. Chapter 5 presents two possible power-aware modifications to the existing heterogeneous technology mapping algorithm SMAP, and investigates how these algorithms perform in terms of density and power. Finally, the thesis concludes with a brief summary and possible future research directions. Parts of this thesis have been published in [10].
2 BACKGROUND AND PREVIOUS WORK

This chapter begins with an overview of field-programmable gate array (FPGA) architecture, and the computer aided design (CAD) algorithms used to map circuits to an FPGA. This chapter also discusses power estimation techniques for digital circuits and memories in particular. After presenting the background material and previous work, the contributions and focus of this thesis will be stated.

2.1 FPGA Architecture

FPGAs are prefabricated integrated circuits (ICs) that can be programmed after fabrication to implement any digital circuit or system. This post-fabrication programmability is provided by three fundamental components: configurable logic resources, I/O resources, and a configurable interconnect. The functionality of the logic resources and the interconnect connections are programmed via a number of configuration memory bits. Although the most popular technology for the configuration memory is SRAM, FPGAs that use other technologies such as antifuse and flash are also commercially available [11]. This thesis focuses on SRAM-based FPGAs.

The most basic logic element in an FPGA, traditionally referred to as a logic element (LE), typically consists of a lookup-table and a flip-flop [14]. As the density of FPGAs
increased, and their uses expanded to larger digital systems, vendors introduced additional resources such as embedded memory blocks, embedded arithmetic logic units, and embedded processors to efficiently implement commonly used functions [11-13]. FPGAs that include these more dedicated resources are sometimes called *Platform FPGAs* or *Heterogeneous FPGAs*. Figure 2-1 illustrates the difference between traditional island-style FPGAs, and modern heterogeneous FPGAs. The following sections will review the architecture of the LE, the embedded memory blocks, and the programmable interconnect.

![Conceptual FPGAs](image)

*Figure 2-1. Conceptual FPGAs. Left: Traditional. Right: Heterogeneous*

### 2.1.1 Logic Element Architecture

Most FPGAs use LUTs in their basic logic element. A *K*-input LUT (K-LUT) works like a memory with $2^K$ configuration bits, $K$ address lines, and a single output line. Each $K$-LUT can be configured to implement any function of $K$ inputs by storing the truth table of the desired function in the $2^K$ configuration bits [14]. Figure 2-2 shows an example of a 2-input LUT configured as a 2-input AND gate.
The value, $K$, plays an important role on the efficiency of the FPGA architecture. Large values of $K$ reduce the number of LUTs that are required to implement a user circuit, and subsequently the demand on the programmable interconnect [15]. However, the number of configuration bits increases exponentially and hence the area overhead increases. Studies have shown that a value of $K=4$ is good for area [16] but a value of $K=7$ is good for speed [17]. Many commercial FPGAs use 4-input LUTs. However, the latest 90nm and 65nm FPGAs use 6-input LUTs [12, 13].

To implement sequential circuits, LUTs are typically paired with flip-flops as shown in Figure 2-3. In this structure, a configuration bit is used to control the state of the output multiplexer. Depending on the value of this configuration bit, the output signal of the LUT can either be sequential or combinational.
2.1.2 Clusters

To increase the speed of the FPGA, LEs are typically grouped together into Clusters. The use of clusters has a similar benefit as using larger LUTs in the LEs, but it incurs a smaller area penalty [18]. An example of a cluster architecture is shown in Figure 2-4. The interconnect within a cluster (called intra-cluster routing) is faster than the general purpose routing (called the inter-cluster routing). This is because the intra-cluster wires have a smaller parasitic capacitance due to shorter lengths. The fast intra-cluster interconnect is used to connect cluster inputs to LEs and to implement fast connections between LEs. The cluster inputs and connections between LEs are steered to the appropriate LE inputs through a multiplexer-based crossbar. Clusters in high-density, high-performance FPGAs often contain specialized connections such as carry and arithmetic chains. Studies have found that cluster sizes of 3 to 10 provide the best speed and area [17].

![Figure 2-4. Cluster Architecture](image-url)
2.1.3 Embedded Block Memory Architecture

Today, FPGAs are often used to implement entire systems. These systems often require storage, and vendors have responded by including memory resources on the FPGA. There are two ways that FPGA vendors provide these memory resources: embedded block memory and distributed memory. Embedded memory solutions offer a number of relatively large dedicated memory blocks on the FPGA. Distributed memory on the other hand uses small memories spread across the entire FPGA chip implemented by allowing users to access the configuration bits of the LUTs. In this thesis we focus only on the dedicated embedded memory blocks.

Embedded FPGA memory block architectures can be described by three parameters: \( N \), \( B \), and \( w_{\text{eff}} \). The number of available arrays is denoted by \( N \), and the number of bits available in each array is denoted by \( B \). Typically, each memory array can be configured to implement one of several aspect ratios; Table 2-1 shows the various aspect ratios available on a number of embedded memory blocks found in commercial FPGAs. We will refer to the set of allowable widths of each memory block as \( w_{\text{eff}} \) and the maximum allowable width as \( w_{\text{max}} \). \( w_{\text{eff}} \) contains all powers of two up to and including \( w_{\text{max}} \).

<table>
<thead>
<tr>
<th>Stratix/StratixII</th>
<th>Virtex4</th>
<th>Virtex5</th>
</tr>
</thead>
<tbody>
<tr>
<td>576Bits</td>
<td>4608Bits</td>
<td>576kBits</td>
</tr>
<tr>
<td>512 x 1</td>
<td>4k x 1</td>
<td>64k x 8</td>
</tr>
<tr>
<td>256 x 2</td>
<td>2k x 2</td>
<td>64k x 9</td>
</tr>
<tr>
<td>128 x 4</td>
<td>1k x 4</td>
<td>32k x 16</td>
</tr>
<tr>
<td>64 x 8</td>
<td>512 x 8</td>
<td>16k x 32</td>
</tr>
<tr>
<td>64 x 9</td>
<td>512 x 9</td>
<td>16k x 36</td>
</tr>
<tr>
<td>32 x 16</td>
<td>256 x 16</td>
<td>8k x 64</td>
</tr>
<tr>
<td>32 x 18</td>
<td>256 x 18</td>
<td>8k x 72</td>
</tr>
<tr>
<td>128 x 32</td>
<td>4k x 128</td>
<td>512 x 32</td>
</tr>
<tr>
<td>128 x 36</td>
<td>4k x 144</td>
<td>512 x 36</td>
</tr>
</tbody>
</table>

Table 2-1. Aspect Ratios of Memories in Commercial FPGAs [2, 3, 19, 20]
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 aspect ratio is obtained by using a programmable column decoder to read and update only certain columns in each word [4]. Figure 2-5 shows how an embedded memory block is made up of two components: a fixed size memory component with $B$ bits and a minimum width of $w_{\text{max}}$, and a programmable column decoder network connected to the data inputs and the data outputs.

![Figure 2-5. High Level Embedded Memory Block Architecture](image)

Figure 2-6 shows the architecture of the programmable column decoder connected to the data outputs of the memory. An equivalent programmable column decoder is connected
to the data inputs of the memory. By appropriately setting the configuration bits of the programmable column decoder, outputs of the memory can be multiplexed together to form deeper columns. Once multiplexed together, the select signals of the multiplexers are controlled by the address lines. These deeper columns give the appearance of a deeper memory and a narrower word width.

![Programmable Column Decoder Architecture](image)

**Figure 2-6. Programmable Column Decoder Architecture**

Embedded memory blocks on many SRAM-based FPGAs can be configured to act as a single or dual port RAM or a single port ROM.

### 2.1.4 Routing Architecture

The configurable routing fabric connects the programmable logic resources and I/O resources via pre-fabricated metal wire segments and programmable switches. Several components make up the configurable routing fabric: wire segments, connection and
switch blocks, and the programmable switches. In modern FPGAs, the interconnect accounts for a majority of the area, delay, and power consumption [15, 21].

2.1.4.1 Wire Segments

Wire segments run along *tracks* and tracks are grouped together into *channels*. In an Island-Style FPGA, these channels run vertically and horizontally in a grid-like fashion as shown in Figure 2-7 [21].

![Figure 2-7. Island-Style FPGA Routing Architecture](Image)

Wire segments may span one logic block, or multiple logic blocks [22]. Short wire segments provide routing flexibility. However, if a signal must traverse multiple wire segments, it must also traverse multiple programmable switches. This leads to larger
signal propagation delays due to parasitic capacitances on the wire segments and switches. Therefore most vendors include wire segments of different lengths to provide a balance between routing flexibility and speed.

2.1.4.2 Connection Blocks and Switch Blocks

Switch Blocks are used to connect wire segments to other wire segments, and Connection Blocks are used to connect wire segments to logic resources. Switch Blocks lie at the intersection of channels and can connect incoming wire segments to a certain number of outgoing wire segments. Although maximum flexibility would be achieved by having a fully connected Switch Block (any incident wire segment may connect to any other incident wire segment), the area overhead makes this impractical. Therefore, Switch Blocks are typically sparsely populated (each incident wire segment may only connect to a subset of all other incident wire segments). A number of sparsely populated switch block topologies have been proposed [23-25]. Similar to the Switch Block, Connection Blocks are also typically sparsely populated [21].

2.1.4.3 Programmable Connections

The programmable connections in the Switch Blocks and Connection Blocks are implemented using pass transistors that are controlled by configuration memory bits. These connections may be buffered or un-buffered as shown in Figure 2-8. The un-buffered connections are more area and power efficient but the buffered connections speed up propagation of signals that need to traverse multiple wire segments. Therefore, FPGA vendors employ a combination of buffered and unbuffered connections. The buffered connections can be uni-directional or bi-directional. Uni-directional connections
are more area efficient and are shown to improve performance when compared to bi-directional connections [26]. The connections in commercial FPGAs are typically unidirectional.

![Connection Types](image)

**Figure 2-8. Connection Types a) unbuffered b) buffered uni-directional c) buffered bi-directional**

### 2.2 FPGA CAD

The configuration bits on an FPGA need to be programmed in a specific way for the FPGA to function as the desired user circuit. Computer Aided Design (CAD) tools are used to determine the states of these configuration bits. There are several steps in the CAD flow as shown in Figure 2-9.

![FPGA CAD Flow](image)

**Figure 2-9. CAD Flow**
The user typically describes the circuit that they want to implement on the FPGA using a schematic or a hardware description language. The first step of the CAD flow, called High Level Synthesis, transforms this input to a netlist of basic logic gates and flip flops. This netlist is then given to the FPGA CAD flow to transform into a bitstream that is used to program the FPGA configuration memory bits. The FPGA CAD flow consists of four steps: Technology Mapping, Clustering, Placement, and Routing.

### 2.2.1 Technology Mapping

The goal of technology mapping is to transform the netlist of basic gates into a netlist of components that are available on the FPGA such as $K$-input LUTs and flip flops. Several optimizations can be performed at this stage. Area optimization can be performed by minimizing the number of LUTs in the resulting netlist [27]. Speed can be optimized by minimizing circuit depth, or in other words the number of nodes that are traversed by the longest path between any primary input and any primary output [28]. Power can be minimized by encapsulating high activity nets of the original netlist inside an element within the solution netlist [29, 30]. Figure 2-10 shows an example of how a part of the original netlist can be mapped to a LUT.

![Figure 2-10. Technology Mapping Example](image-url)
2.2.2 Clustering

Clustering (also known as packing) produces a netlist of clusters and determines which LUTs and flip flops to group together within each LE [15]. One of the main goals in this step is to group together closely connected LUTs and flip flops to take advantage of the fast intra-cluster interconnect. Since the parasitic capacitance in the intra-cluster interconnect is smaller compared to the inter-cluster interconnect, using the intra-cluster interconnect for closely connected elements reduces both delay and power consumption of the circuit.

2.2.3 Placement

Placement takes the netlist of clusters produced by the clustering algorithm and assigns each cluster a physical location on the FPGA. The placement algorithm tries to simultaneously minimize the predicted routing demand and critical path delay of the user circuit. Routing demand is not actually known until the routing step but can typically be reduced by placing closely connected clusters near each other. High routing demand in a localized region is called congestion and increases the difficulty of the routing problem. Similarly, placing clusters on the critical path close together can reduce the critical path delay. FPGA CAD tools use analytical methods [31] or Simulated Annealing [15] to solve the placement problem.

Analytical placement algorithms specify the location of each cluster as a variable in a system of equations. These equations express the tradeoffs between various optimization objectives such as delay, congestion, and power as a function of the relative location of each cluster. After solving this system of equations, the placement solution goes through
a final legalizing step. The purpose of the legalizing step is to resolve contentions since solving the system of equations may produce a solution where some clusters occupy the same location in the FPGA.

The VPR tool used in this project uses a Simulated Annealing placement algorithm. The algorithm starts off by placing the clusters randomly onto the FPGA. Two clusters are randomly chosen and their locations are swapped. The swap is evaluated based on an optimization function. Good swaps are kept and a percentage of bad swaps are also kept to prevent the placement solution from getting trapped in a local minima. The quality of the placement solution is gradually improved with successive swaps until no more good swaps can be made.

2.2.4 Routing

The final step in the FPGA CAD flow is routing. This step determines which routing resources to use for each connection in the netlist. FPGA routing is different than ASIC routing. The difference arises from the fact that the FPGA interconnect is pre-fabricated with a fixed number of wires and connections, whereas the interconnect of an ASIC can contain as many wires and connections as necessary. In ASIC routing, a two-step global-detailed routing method is often employed [32, 33]. The first step, global routing, determines which channels to use for each net. The second step, detailed routing, determines where to add wire segments. Using this method for FPGAs, the detailed routing step becomes very difficult due to the limited routing resources. Therefore, single step routing algorithms are typically used for FPGAs [34, 35].
The VPR routing algorithm is based on the PathFinder routing algorithm [35]. PathFinder is an iterative negotiation-based algorithm. Each routing resource is assigned a cost for its usage. In the first iteration of the algorithm, each net is routed using the shortest path possible regardless of whether the routing resource has already been assigned to another net. At the end of each iteration, the cost for routing resources that have been assigned to multiple nets increases. All nets are then ripped up and re-routed. As the cost for routing resources with high demand increases, only the most critical nets will be assigned to those resources. This process is repeated until there is no more contention.

At the end of the FPGA CAD flow, the states of all the components in the FPGA are known and the states for all the configuration memory bits can be determined.

2.3 Heterogeneous Technology Mapping

Traditionally, the term *homogeneous technology mapping* has been reserved for FPGAs that contain only LUTs and Flip Flops. In this thesis, we will use the term *heterogeneous technology mapping* to refer to algorithms that map logic to FPGAs that contain LUTs, flip flops, and memories[5-7]. For this kind of heterogeneous technology mapping, we will also interchangeably use the term *logic* to refer to the LUTs produced by the traditional technology mapping step.

2.3.1 Terminology

We will review some terminology on *directed acyclic graphs* (DAGs) before discussing the heterogeneous technology mapping algorithms. In this thesis, we will use the
terminology defined primarily in [5, 36]. The combinational part of a logic circuit can be represented with a Boolean network which is a directed acyclic graph (DAG). For the DAG \( G(V, E) \), the set of vertices \( V \) represent combinational nodes, and the set of edges \( E \) represent directed connections between the nodes.

The set of nodes that drive a node \( x \) will be denoted as \( \text{fanins}(x) \) and the nodes that are driven by \( x \) will be referred to as \( \text{fanouts}(x) \). A node \( x \) is a predecessor of node \( z \) if there exists a directed path from \( x \) to \( z \). A cone rooted at \( z \) is a subgraph containing \( z \) and some of its predecessors. A fanout-free cone is a cone in which none of the nodes (except the root node) in the cone fans-out to a node that is not inside the cone. A maximum-fanout free cone rooted at \( z \), which we denote as \( \text{MFFC}(z) \), is the fanout-free cone rooted at \( z \) that contains the largest number of nodes. Given a cone \( H \) rooted at \( u \), a cut \( C_u = (X, X') \) is a partitioning of \( H \) such that \( u \in X' \) where \( X' \subset H \) and \( X \subset H \), and \( X \cup X' = H \). The cut-set of a cut \( C_u \) is the set of nodes in \( X \) that drive a node in \( X' \) and is denoted by \( \text{cutset}(C_u) \). \( C_u \) is said to be \( k \)-feasible if the number of nodes in \( \text{cutset}(C_u) \) is less than or equal to \( k \). A maximum-volume \( k \)-feasible cut is a \( k \)-feasible cut \( C_u = (X, X') \) with the largest number of nodes in \( X' \).

2.3.2 Existing Algorithms

Three heterogeneous technology mapping algorithms have been published: MemMap [7], EMB_Pack [6], and SMAP [5]. The SMAP and EMB_Pack algorithms operate similarly and are based on using FlowMap [28] and the Max-Flow Min-Cut theorem [37] to identify regions for mapping into memories. MemMap determines regions for mapping into memories by expanding around nodes with reconvergent fanout. All of these
algorithms produce similar results; in this thesis, we use SMAP. We will now briefly review the SMAP algorithm.

### 2.3.3 SMAP

The SMAP algorithm takes place between the Technology Mapping step and the Clustering step in the traditional FPGA CAD flow, but can also be considered a part of technology mapping. 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 cone rooted at the seed node. The set of $k$ cut nodes are the inputs to the memory array.

Given a seed node $s$ 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 is a potential memory output. The set of all potential nodes is denoted as $\text{potentials}(s)$. SMAP chooses the output nodes by ranking all potential outputs with the number of nodes in its MFFC. In other words, the cost function for choosing node $p$, where $p \in \text{potentials}(s)$, as a memory output is:

$$\text{Cost}_{\text{output}}(p) = |\text{MFFC}(p)|$$  \hspace{1cm} (2.1)

The $w$ highest scoring nodes are selected as the memory outputs where $w$ is the width of the memory. The nodes in the chosen output node's MFFC are packed into the memory array. For memories with a configurable width, SMAP iterates through all aspect ratios.
available for the memory, and the solution that results in the highest number of packed nodes is selected. Figure 2-11 shows an example of an six input cut where nodes I, J, and H are chosen as memory outputs and the resulting circuit implementation.

For seed selection, each node in the network is visited as a potential seed node, and the above algorithm for output selection is performed using the deepest memory aspect ratio (largest number of memory inputs). The node that leads to the largest number of nodes that can be packed is chosen as the seed node.

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].
2.4 Power Estimation

There are many ways to perform power estimation with various degrees of accuracy and computational complexity. The accuracy of power estimation depends on two things: the level of abstraction of the power model, and the stage of the design flow at which it is performed [38]. Power estimations with low levels of abstractions and performed later in the design flow when most of the physical implementation detail is known are the most accurate. The computational requirement and accuracy of a power estimation method both depend on the level of abstraction of the power model. For example, power estimation at the transistor level will be more accurate than at the architectural level but will likely require more computation.

In digital circuits, Equation 2.2 is commonly used to calculate dynamic power dissipation of a net. $\alpha$ is the switching activity of the net, $C$ is the effective capacitance of the net, $V_{\text{supply}}$ is the supply voltage, $V_{\text{swing}}$ is the voltage swing when switching, and $f_{\text{clock}}$ is the clock frequency. The parasitic capacitance on the transistors that make up the gates that drive the net are lumped with the wire capacitances and expressed by the effective capacitance $C$.

$$P_{\text{Dynamic}} = \frac{1}{2} \cdot \alpha \cdot C \cdot V_{\text{Supply}} \cdot V_{\text{Swing}} \cdot f_{\text{Clock}}$$ (2.2)

2.4.1 Switching Activity

The switching activity of a net describes the average number of times that the net toggles in each clock cycle. Switching activities of every net are required before the power dissipated by an entire circuit can be estimated. Activity estimation techniques can be
divided into simulation-based techniques and probabilistic techniques [39]. Simulation-based methods are more accurate but require input vectors to stimulate the circuit and are more computationally intensive. By simulating the circuit with input vectors, the switching of each net can be observed and tracked. At the end of the simulation, the average number of times that the net toggled can be calculated. Full simulations can make use of detailed delay models and account for dependencies among signals (such as spatial correlation and temporal correlation) to produce high quality estimates.

Probabilistic methods eliminate the need for input vectors and are more computationally efficient. Once the switching activities at the primary inputs are specified, the activities of successive nodes can be calculated using an activity propagation model. The most commonly used models are the Transition Density Model [40] and the Lag-One Model [41].

2.4.1.1 Transition Density Model

The Transition Density Model uses two numbers to represent the activities of each net: the static probability and the switching activity (or transition density). Static probability is defined as the average fraction of clock cycles in which the steady state value of the output of the node is logic high [40]. The static probability of an output of a node depends on the static probabilities of the node's inputs and the function of the node.

The switching activity at the output of a combinational node y can be calculated with the following equations [40].
\[ \text{Activity}(y) = \sum_{i=1}^{\text{all inputs}} P\left( \frac{\partial f(x)}{\partial x_i} \right) \cdot \text{Activity}(x_i) \quad (2.3) \]

\[ \frac{\partial f(x)}{\partial x_i} = f(x)\big|_{x_i=1} \oplus f(x)\big|_{x_i=0} \quad (2.4) \]

Where \( f(x) \) is the logic function of the node, \( x \) is the set of inputs, and \( P\left( \frac{\partial f(y)}{\partial x_i} \right) \) is the probability that a change in input \( i \) will cause a change at the output \( y \). One drawback of this model is that it does not account for simultaneous transitions of the inputs of the gate. This can lead to overestimation of the activity and hence overestimation of power.

### 2.4.1.2 Lag One Model

The Lag-One Model calculates static probability and switching probability. Switching probability is defined as the probability of a signal switching in each clock cycle and is a lower bound on switching activity. In other words, switching probability represents switching activity without glitching. The following equation is used to calculate the switching probability in the Lag One Model [41].

\[ P_{\text{switch}}(y) = \sum_{s_i \in S_i} \left[ P_i(s_i) \cdot \sum_{s_j \in S_0} P_{\text{switch}}(s_i, s_j) \right] + \sum_{s_i \in S_i} \left[ P_i(s_i) \cdot \sum_{s_j \in S_i} P_{\text{switch}}(s_i, s_j) \right] \quad (2.4) \]

The first term represents the probability of the output of a node \( y \) switching from a logic 1 to a logic 0. This term can be read as the probability of the inputs being in a state \( s_i \), where \( f(s_i)=1 \), multiplied by the probability of transitioning to all states \( s_j \) where \( f(s_j)=0 \). The summation is performed for all states where \( f(s_i) = 1 \). Similarly, the second term represents the probability of switching from a logic 0 to a logic 1. Since this model uses state transitions, and not input transitions, for its calculations simultaneous input transitions are accounted for.
2.4.1.3 ACE2.0

Our power model, which will be described later, is based on an existing academic activity estimation tool called ACE2.0 [42]. ACE2.0 is based on the Transition Density model and the Lag-One model and supports three elements: primary inputs and outputs, combinational nodes, and registers. It uses three steps to estimate switching activity. The first step uses simulation to estimate the static and switching probabilities for logic within sequential feedback loops. The second step uses the Lag One model to calculate static and switching probabilities for the remaining nodes. Nodes are visited from the primary inputs to the primary outputs. The third step calculates switching activity using a method that accounts for different input signal arrival times and the glitch filtering effect of logic gates. Glitches with very small pulse widths do not actually occur because the gate does not have enough time to fully charge the output as shown in Figure 2-12. This effect is not captured in the Transition Density model since it assumes single input transitions, and it is not captured in the Lag-One Model because it assumes equal arrival times. In ACE2.0, the arrival times are assumed to be normally distributed and the concept of a minimum pulse width is introduced through the variable τ. τ represents the minimum pulse width that can be propagated by a gate and is process dependant.

![Figure 2-12. Glitch Filtration](image)

24
The following equations show how $\tau$ is used to calculate switching activity from the switching probability.

$$\text{Activity}(y) = \frac{T}{\tau} \cdot P_{\text{switch}}(y)$$  \hspace{1cm} (2.6)

$$P_{0\rightarrow1} = \frac{P_{\text{switch}}(f)}{2 \cdot (1 - P_{1}(f))} \cdot \frac{\tau}{T} \quad \text{and} \quad P_{1\rightarrow0} = \frac{P_{\text{switch}}(f)}{2 \cdot P_{1}(f)} \cdot \frac{\tau}{T}$$  \hspace{1cm} (2.7)

where $T$ is the maximum delay from the primary inputs to the primary outputs.

### 2.4.2 Power Estimation for FPGAs

Both commercial and academic power estimation tools for FPGAs exist [8, 43-47]. Vendors typically provide power estimation spreadsheets for power estimation in the early design stages [46, 47]. However, as mentioned earlier, power estimates at early design stages is not very accurate. For power estimates in later design phases, vendors provide tools in their CAD suites. Altera’s PowerPlay Power Analyzer [45] and Xilinx’s Xpower [44] tool provide simulation-based activity estimation at the post technology mapping and post place and route stages. Altera’s PowerPlay Power Analyzer also performs vectorless (probabilistic) activity estimation for the StratixII devices.

The Poon Power Model (PPM) [8] is a flexible academic FPGA power model built on top of the popular Versatile Place and Route (VPR) [9] CAD suite. This model provides power estimates for homogeneous island-style FPGAs. Power estimates are broken down into three components: dynamic power, static power, and short circuit power. Dynamic power is calculated after placement and routing using Equation 2.2. Process technology information from the user, and the architectural description provided to VPR are used to calculate capacitances on nets. A transistor capacitance model is used along with the
transition density model to estimate the power dissipated within a cluster and clock network. For the clock network, an H-Tree clock distribution network is assumed.

2.4.3 Power Estimation for Memories and Caches

Techniques for estimating the power of fixed memories vary in complexity and accuracy. Understanding the operation of the memory is important to understanding how to estimate its power consumption. For the following discussion, we refer to Figure 2-13. An SRAM memory array has several components: the address decoder (which includes the row and column decoders), the array of SRAM cells (the core), sense amplifiers, write drivers, precharge circuitry, and control logic. The core consists of an array of SRAM cells, each of which can store one bit. Each SRAM cell is made up of six transistors, four of which are used for a cross-coupled inverter circuit that stores the data, and two transistors are used to access the cell. Word-lines run horizontally into the core and connect to the gates of the access transistors. Complementary bit-line pairs run vertically along each column and connect to the access transistors to transfer data into and out of the core. Since the word-lines and bit-lines are long metal tracks that are connected to many transistors, they accumulate a significant amount of parasitic capacitance and thus dominate the access time and power dissipation of the memory. The purpose of the other components will be come clear when we discuss the memory's operation.
At the beginning of each access cycle, the bit-lines are precharged to a high value regardless of what operation will be performed. For a read operation, the row decoder decodes the address and drives the word-line of the desired row. This connects the cells of that row to the bit-lines. Depending on whether a 0 or a 1 is stored in the cell, one of the two bit-lines in each column is pulled down. At this point, the column decoder selects the desired columns to pass to the sense amplifiers. Sense amplifiers are needed to detect the small drop in voltage across the discharging bit-line as they do not discharge completely. Depending on which bit-line in the column is being discharged, the sense amplifier will drive the dataout signals accordingly.
The operation of a write cycle is similar. After precharging the bit-lines, the write driver pulls down one of the bit-lines in each column depending on whether a 0 or a 1 is being written to the column. The address decoder then drives the appropriate word-line and connects the cells to the bit-lines for updating. The charge in the bit-lines is strong enough to force a change in the states of the cross-coupled inverters.

Dynamic and leakage power estimation for SRAM memories and caches is a well studied topic [48-56]. Methods are often analytical in nature and based on theoretical calculations of internally switched capacitances. Most techniques operate at the transistor level or the micro-architectural level. At the transistor level, power estimation can be performed using SPICE. Although very accurate, transistor level modeling of memories is very computationally intensive. The authors of [56] alleviate this problem somewhat by generating analytical models through characterization of memory implementations using transistor-level simulations.

Many tools exist at the micro-architectural level but often target specific implementations of the memory or cache. Models such as those in [50] model only the power dissipated in the core array. Power models for caches such as CACTI[53], PRACTICS[54], and WATCCH[49] employ detailed capacitance models and account for both the tag and data arrays and all components of the memory. Most of these tools have limited absolute accuracy and are useful only for relative comparisons. Recently, a more generic model called IDAP [51] has been shown to work for a variety of implementations for the components in the memory and was shown to provide estimates that are within 22.2% of SPICE simulations.
2.5 Focus and Contribution of Thesis

The goal for this research is two-fold: first we want to build a flexible power model for FPGAs that contain embedded memories to enable future FPGA architectural and CAD tool research, and secondly we want to use this tool to investigate the power implications of using memories to implement logic. The approach to implement the power model is to take an existing and validated FPGA power model for homogeneous FPGAs, called the Poon Power Model, and extend it to model power dissipation of embedded memory blocks. To enable architectural studies, our memory power model requires the following attributes. First of all, it needs to be simple enough such that estimates can be calculated quickly. Secondly, it must have high fidelity meaning that the absolute accuracy is not as important as accuracy of relative comparisons. Thirdly, the power model should not be implementation and layout specific.

In Chapter 3, we describe the details of our power model. In Chapter 4, we perform an architectural study by applying this power model to investigate the power implication of mapping logic to memories. In Chapter 5, we perform a CAD study with the tool to see whether these heterogeneous technology mapping algorithms can be made power-aware.

The contributions of this thesis are summarized as follows:

1. A novel power model for heterogeneous FPGAs that contain embedded memories
2. Experimental evaluation of the impact that mapping logic to memories has on power and energy.
3. Explore power-aware heterogeneous technology mapping algorithms by enhancing an existing algorithm and measuring their performance in terms of power and energy.
3 POWER MODEL FOR FPGAS CONTAINING EMBEDDED MEMORIES

This chapter describes the power model developed for estimating the power consumption of FPGAs that contain embedded memories. The power model is made up of two separate parts: an activity estimation tool, and a power estimation tool. We implement these tools by extending existing tools for homogeneous FPGAs to support embedded memory blocks. In the following sections, we will describe how the existing tools were enhanced.

3.1 Activity Estimation

To develop our activity estimation tool, we extended ACE2.0 (as described in Section 2.4.1.3) to support embedded memories. Although the activities from ACE2.0 are very accurate and fast, the model cannot be directly applied to memories because of the transistor level nature of the memory circuitry. Therefore, we require an alternative technique for memories. Activity estimation for the outputs of an embedded memory block can be carried out differently depending on whether the memory block is configured as a RAM or a ROM. The following sections will discuss methods for both.
3.1.1 Read Only Memory

The overall strategy for estimating the activities of ROMs is to decompose the ROM into registers and logic nodes that ACE2.0 can understand. Since the contents of a ROM are fixed, each column, or output, of a ROM can be represented by a multi-input single-output combinational node. The values stored in the column act as the truth table to describe the functional behavior of the combinational node. The inputs of the node are the address lines and the output of the node is the memory output. One node is required for each memory output to completely represent the ROM. For synchronous memories, the address lines are connected to registers before entering the node. If a memory enable signal is available, an additional AND gate is inserted to gate the memory output with the memory enable signal. Although gating the output in this manner is not exactly functionally correct, it is acceptable for the purpose of activity estimation. By replacing the ROM with registers and nodes in this manner, the existing probabilistic methods within ACE2.0 can be used. Figure 3-1 shows an example of replacing a 16x2 synchronous ROM memory with combinational nodes and registers.

![Figure 3-1. Replacing a ROM with Equivalent Nodes and Registers](image-url)
3.1.2 Random Access Memory

Activity estimation for an embedded memory block configured as a RAM needs to be performed differently because the contents of the memory are not static. Intuitively, the activities on the memory’s data out pins will depend on the activities of the memory inputs. These include the write enable signal, the address signals, and the datain signals. The activity estimation of each output pin can be simplified by observing that the activity of each dataout pin is independent of all but datain pins except one.

At least three different approaches to estimate the activities of the output pins of a RAM are possible. One approach is to determine a closed-form expression and use this expression to compute the activities. A second approach is to generate a profile of the output pin activities through simulation and store this profile in a table for fast lookup. This technique requires a characterization phase prior to usage. In the characterization phase, numerous simulations would be performed using vectors with different input switching activities. The simulated output activities would then be stored in a table and indexed with the input switching activities that were used for the simulation. This table would be generated only once for each memory architecture. At run-time, the output activities would be retrieved from the table using the input activities as the index. If the input activities do not fall exactly on an index position, linear interpolation can be used. A third option is to actually embed a simple memory simulator into ACE2.0. The simulator would use randomly generated vectors that match the calculated static and switching probabilities of the memory inputs to perform simulation-based activity estimation on the memory output nodes.
Each of the three methods described above has its advantages and drawbacks. The first method is the most elegant but finding a closed-form expression can be quite difficult. The second method is fast at run time but requires significant storage for the profiled values and significant time to characterize each memory architecture that may need to be explored in an architectural investigation. The third method would extend the execution time of the activity estimation tool, but it is accurate. A RAM simulator is required in all three methods because ACE 2.0 uses simulation to estimate the activities of nodes in feedback loops. For these reasons, we chose the third option of embedding a RAM simulator into ACE2.0.

### 3.1.3 Framework and Integration into ACE2.0

Figure 3-2 shows the pseudo-code of the new flow in ACE2.0. Lines that are in bold show the new components added for the activity estimation of the memories.

```plaintext
ACE2.0 (network, vectors, activities) {
    Replace_ROMS()

    /* Phase 1 */
    feedback_latch_and_memory = find_feedback_latches_and_memories(network)
    feedback_logic = find_feedback_logic(feedback_latch_and_memory)
    simulate_probabilities(feedback_logic)

    /* Phase 2 */
    foreach node n ∈ network {
        if(Status(n) != SIMULATED) {
            if(is_memory_output(n)) {
                Static_Prob(n) = simulate_memory_static_prob(n)
                Switch_Act(n) = simulate_memory_switch_act(n)
                Switch_Prob(n) = Switch_Act(n)
            } else {
                Static_Prob(n) = calc_static_prob(n)
                Switch_Prob(n) = calc_switch_prob(n)
            }
        }
    }
}
```
The flow first replaces all ROM memories with equivalent nodes as described in Section 3.1.1. As discussed in Section 2.4.1.3, ACE2.0 has three phases. The first phase identifies and simulates nodes, flip-flops and memories that are in sequential feedback loops. The second phase iterates through each remaining node and flip-flop and calculates their static probability and switching probability using the Lag One Model. If the node is a memory output, then the activities are simulated using the memory simulator. As described in the previous section, the simulator generates the switching activity and not switching probability which is required for the calculation of downstream nodes. To address this, we note that since our memory simulator assumes synchronous memories, the switching activity will be the same as the switching probability. In the final phase, the switching activities for each node are calculated from the static probabilities and switching probabilities. Since the switching activities for the memory output nodes have already been determined, they do not need to be visited in this phase.

Figure 3-3 shows the pseudo-code for the RAM simulator.
Inputs: static probabilities and switching activities for ME, WE, address and Datain signals  
Output: static probabilities and switching activities for Dataout signals  

Generate vectors with given average activities for ME, WE, Address, Datain signals  
for (number of simulation cycles)  
  
  /* get input states for current cycle */  
  ME_current = ME_vector[cycle]  
  WE_current = WE_vector[cycle]  
  Address_current = Address_vector[cycle]  
  Datain_current = Datain_vector[cycle]  
  
  /* idle cycle */  
  if(ME_current == 0)  
    {  
      /* do nothing */  
    }  
  /* write cycle */  
  else if(WE_current == 1)  
    {  
      S[Address_current] = Datain_current  /* write to memory */  
      Dataout = Datain  /* write through */  
    }  
  /* read cycle */  
  else  
    {  
      Dataout = S[Address_current]  /* read memory */  
    }  
  
  foreach bit  
  {  
    If (Dataout [bit] made a transition) toggled[bit]++ /* count if dataout toggled */  
    If (Dataout [bit] == 1) high[bit]++ /* count if dataout is logic 1 */  
  }  
  
  /* calculate static probabilities and switching activities */  
  foreach bit (i)  
  {  
    static_probability(Dout[i]) = high[i] / Num Cycles  
    activity(Dout[i]) = toggled[i] / Num Cycles  
  }  

Figure 3-3. Pseudo-Code for RAM Simulator  

When not simulating feedback loops, the inputs to the simulator are the static probabilities and switching activities of the memory input pins. First, the simulator generates vectors for each input signal that match the given statistics. The next step is the
actual simulation. In each iteration, the values of the inputs are retrieved for the given cycle. There are three possible events: an idle cycle, a write cycle, and a read cycle. If the state of the memory enable signal (ME) is a logic 0, then an idle cycle occurs and no changes to the memory contents or outputs occur. If this is not the case, then a write cycle or a read cycle is determined based on the state of the write enable signal (WE). In the pseudo-code, $S$ is a matrix used to store the contents of the memory. In a write access, the contents of $S$ are updated with the current datain values. This simulator assumes a write-through from the datain to the dataout pins when a write access is performed, as in some commercial devices, and therefore the simulator also updates the dataout values with the current datain values. In a read access, the dataout values are updated with the appropriate row in $S$ based on the current address values. At the end of each iteration, two counters for each dataout pin are updated. One counter counts the number of cycles that the dataout pin is in a logic 1 state. The other counter counts the number of times that the dataout pin toggles. At the end of the simulation, the static probability and switching activities of the dataout pins can be calculated by dividing the counters by the number of cycles that were used in the simulation. Using this method to determine the switching activities implicitly assumes that the memory is synchronous which is the case in commercial devices.

3.2 Power Estimation

As discussed in Section 2.1.3, the embedded memory blocks are made up of a fixed size memory array and a programmable column decoder network. The following sections describe how power estimation is performed on these two components and conclude with a discussion on the implementation of the tool. We also show that the power
consumption of the programmable column decoder is negligible compared to the fixed-size memory array and hence do not include it in our power model.

3.2.1 Fixed-Size Memory

In our power model, the power dissipated by the fixed-size memory component is modeled as the sum of two components: the dynamic cycle power, and the leakage power. We assume that the fixed-size memory component in the embedded memory block is a typical SRAM memory using 6-T SRAM cells with complementary bit-lines as shown in Figure 2-13. Due to the symmetry of this architecture, the same amount of dynamic power is dissipated within each write access. Although the power dissipated within the row decoder is dependant on the address line activities, it is very small compared to the other components. Because of this, a single power value can be used to estimate the power dissipated in a write access. Similarly, a single power value can be used to estimate the power dissipated in a read access. The dynamic cycle power of the memory can then defined as a weighted average of these two power values depending on how often the memory is performing read and write accesses. Since leakage power is independent of whether the memory is performing a read or a write, the leakage power dissipated by the memory can be represented by a single value.

3.2.2 Programmable Column Decoder

In this section, we model the programmable column decoder and show that it dissipates a negligible amount of power compared to the fixed-size memory thus allowing us to ignore this component in our power model.
We first assume that the programmable column decoder network consists of a two-input multiplexer tree, and a number of two-input multiplexers that are controlled by SRAM configuration bits. We further assume that these multiplexers can be modeled with NMOS pass transistor networks as shown in Figure 3-4. By modeling the programmable column decoder in this manner, we can draw on the methods used to model LUTs and multiplexers in PPM [8].

![Figure 3-4. Transistor Level Modelling of the Programmable Column Decoder](image)

In PPM, LUTs are also modeled as pass transistor networks as shown in Figure 3-5. The authors of PPM used transistor capacitance models from CACTI and the transition density model to predict the power dissipated in the multiplexer network. The power dissipated at each internal node of the network is estimated by summing the parasitic capacitances at each node, finding the activity of the node, and using Equation 2.2. The total power dissipated by the network is the sum of the power dissipated at each internal node. The authors of PPM showed that this method was accurate to within 14.5% of the maximum predicted value of HSPICE. A similar method was used to predict the power dissipated by SRAM controlled multiplexers. For these circuits they found that their model was within 5.3% of HSPICE. By using the same method, we found that the
programmable column decoder contributed to less than two percent of the power dissipated within an FPGA embedded memory block. By using the same method, we found that the power dissipated by the programmable column decoder was less than two percent of a 64x64 memory with 4096 bits. Therefore, we ignore the detailed modeling of this component in our power model.

![Figure 3-5. Modelling of LUTs in the Poon Power Model (from [57])]()  

**3.2.3 Framework and Implementation of the Power Estimator**

To implement our power estimation tool, PPM was modified to support embedded memories. As described earlier, we model the embedded memory block power as two components, dynamic cycle power and leakage power, and ignore the power dissipated in the programmable decoder.

In VPR (and PPM), technology parameters are specified in an architecture file. In the existing PPM, these technology parameters include the leakage power per SRAM cell as well as low-level capacitance and resistance information that allow PPM to calculate dynamic and short-circuit power of each element in the FPGA. We have extended this by adding the three new technology parameters indicated in Table 3-1.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{readcycle}}$</td>
<td>Equivalent Read Cycle Capacitance</td>
</tr>
<tr>
<td>$C_{\text{writecycle}}$</td>
<td>Equivalent Write Cycle Capacitance</td>
</tr>
<tr>
<td>$P_{\text{leak}}$</td>
<td>Memory Leakage Power</td>
</tr>
</tbody>
</table>

Table 3-1. VPR Memory Power Parameters

The parameter $P_{\text{leak}}$ is the leakage power of a memory block, and can be found either through careful SPICE simulations or read directly from memory generator datasheets (we use the latter approach in our experiments; our technology information is taken from Virage Memory Compiler output reports). The leakage power is constant for all memory blocks of the same size and organization; memory blocks with different sizes or organizations will have different values of $P_{\text{leak}}$.

The amount of dynamic energy dissipated during each read and write cycle can also be read directly from memory generator datasheets or found using SPICE. However, rather than creating architecture file parameters for these quantities, the read and write cycle power is specified indirectly, through the $C_{\text{writecycle}}$ and $C_{\text{readcycle}}$ parameters. These parameters are defined as the effective capacitance that is charged or discharged for each read and write cycle, respectively. Although $C_{\text{writecycle}}$ and $C_{\text{readcycle}}$ cannot be read directly from memory generator datasheets, these quantities can be calculated using the following equations:

\[
C_{\text{writecycle}} = \frac{2 \cdot Power_{\text{writecycle}}}{V_{\text{swing}} \cdot V_{\text{supply}} \cdot f_{\text{clock}}} \tag{3.1}
\]

\[
C_{\text{readcycle}} = \frac{2 \cdot Power_{\text{readcycle}}}{V_{\text{swing}} \cdot V_{\text{supply}} \cdot f_{\text{clock}}} \tag{3.2}
\]
where $V_{\text{Sw}}$, $V_{\text{Supply}}$, $f_{\text{clock}}$, $\text{Power}_{\text{write cycle}}$ and $\text{Power}_{\text{read cycle}}$ can be found from memory generator datasheets.

The advantage of using $C_{\text{write cycle}}$ and $C_{\text{read cycle}}$ as architecture file parameters instead of $\text{Power}_{\text{write cycle}}$ and $\text{Power}_{\text{read cycle}}$ is that they are independent of clock frequency. In addition, the use of capacitance numbers is more familiar to users of VPR and PPM, since that is how most other technology numbers are expressed.

Within our enhanced PPM, the dynamic cycle power is calculated by first calculating $\text{Power}_{\text{write}}$ assuming one write access is performed per cycle, using the following equation:

$$\text{Power}_{\text{write}} = \frac{1}{2} \cdot C_{\text{write cycle}} \cdot V_{\text{Swing}} \cdot V_{\text{Supply}} \cdot f_{\text{clock}}$$

(3.3)

In this expression, $f_{\text{clock}}$ is the actual post-place and route clock frequency found by VPR. Similarly, the quantity $\text{Power}_{\text{read}}$ is calculated assuming one read access is performed per cycle, using the following equation:

$$\text{Power}_{\text{read}} = \frac{1}{2} \cdot C_{\text{read cycle}} \cdot V_{\text{Swing}} \cdot V_{\text{Supply}} \cdot f_{\text{clock}}$$

(3.4)

In a real system, the memory is not accessed every cycle, and some accesses are reads and some are writes. Thus, the overall cycle power is:

$$\text{Power}_{\text{cycle}} = P_{ME} \cdot [P_{WE} \cdot \text{Power}_{\text{write}} + (1 - P_{WE}) \cdot \text{Power}_{\text{read}}]$$

(3.5)

where $P_{ME}$ is the static probability that the memory enable signal is high, and $P_{WE}$ is the static probability that the write enable signal is high (this assumes an active-high write
enable signal and an active-low memory enable signal). It is important to note that Equation 3.5 assumes that the write enable and memory enable signals are uncorrelated.

Leakage power is independent of the clock frequency and hence can be represented directly by a power value which we denote as $P_{\text{Leak}}$. We do not model short circuit power because there is very little short circuit power in the 6-T SRAM cells due to the fast transitions of the cross-coupled inverters. The contributions of short circuit power in the cell array is further reduced by the fact that no more than one row of cells is being written to at any time.

### 3.3 Summary

A power model for FPGAs that contain embedded memories was developed in two parts. The activity estimation part is implemented as an extension of ACE2.0. Activity estimation for ROMs is performed by replacing the memory with an equivalent network of combinational nodes and registers followed by using the standard ACE2.0 flow. Activity estimation for RAMs is performed by using a simple simulator to simulate the activities of the memory outputs. The power estimation step is implemented as an extension of the Poon Power Model and integrated into the VPR CAD suite. Since we found that the power dissipated by the programmable column decoder contributes to less than two percent of the embedded FPGA memory, we chose not to include this component in our power model. The power dissipated by the fixed-size memory is modeled with two components: dynamic cycle power and leakage power.
4  POWER IMPLICATIONS OF MAPPING LOGIC TO MEMORIES

In the previous chapter we described a method for activity and power estimation for the embedded memory blocks in FPGAs. In this chapter, we apply this model to investigate the power implications of configuring memories as large ROMs to implement user logic. Intuitively, implementing logic in memory arrays will impact the overall power dissipated by the circuit in two ways. If large amounts of logic can be implemented in a memory block, not only are fewer lookup-tables required (which would save a small amount of power), but also the interconnect between these lookup-tables are not required (which would save a significant amount of power). On the other hand, memory arrays contain long word-lines, bit-lines, sense amplifiers, and decoders, all of which consume power [58].

In this chapter, we investigate this intuition, and determine whether implementing logic in memory arrays leads to a net reduction or increase in power. In particular, we consider a range of memory architectures, and answer the following questions:

1. How does the number of FPGA memories used for logic affect power?
2. How does the size of the FPGA memories used for logic affect power?
3. How does the flexibility, or the maximum configurable width, of the FPGA memories used for logic affect power?
4.1 Experimental Methodology

To investigate the power and energy implications when using memories to implement logic, we employ an experimental methodology. The results are gathered in two ways: we use an enhanced version of VPR that supports the placement and routing of embedded memory blocks, and the power model described in Section 3.2.3, as well as performing current measurements on a 0.13μm CMOS FPGA (Altera Stratix EP1S40). Although the second technique provides the most accurate results, it is not possible to use it to investigate alternative memory architectures; for those experiments, we need to use the VPR flow. In this section, we describe both methodologies.

4.1.1 VPR Based Flow

Figure 4-1 shows the flow for the VPR-based experiments. First, the twenty largest MCNC circuits (ten are combinational and ten are sequential) are technology mapped to 4-LUTs using Altera’s Quartus Integrated Synthesis (QIS) [59], and the resulting netlist is exported using the Quartus University Interface Program (QUIP) [60]. Table 4-1 summarizes the characteristics of the QIS mapped MCNC benchmark circuits. Although SIS and Flowmap [28] could also be used to perform technology mapping, we chose QIS for two reasons. First, for all our circuits, QIS was able to find implementations requiring far fewer LUTs than SIS/Flowmap. This is partially because FlowMap achieves depth optimality at the expense of area by aggressively using node replication, whereas in QIS, a balanced area and timing driven algorithm was used. The second reason why we chose QIS is that we eventually feed the same circuit through the Quartus flow to perform measurements on an actual device (see Section 4.1.2), and we want to ensure that the measured circuit is identical to the one used in the VPR flow.
Figure 4-1. Flow for VPR-Based Experiments

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Primary Inputs</th>
<th>Primary Outputs</th>
<th>4-LUTs</th>
<th>Flip Flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>14</td>
<td>8</td>
<td>989</td>
<td>0</td>
</tr>
<tr>
<td>apex2</td>
<td>38</td>
<td>3</td>
<td>1023</td>
<td>0</td>
</tr>
<tr>
<td>apex4</td>
<td>9</td>
<td>19</td>
<td>844</td>
<td>0</td>
</tr>
<tr>
<td>bigkey</td>
<td>229</td>
<td>198</td>
<td>1032</td>
<td>224</td>
</tr>
<tr>
<td>clma</td>
<td>62</td>
<td>83</td>
<td>4682</td>
<td>33</td>
</tr>
<tr>
<td>des</td>
<td>256</td>
<td>245</td>
<td>1242</td>
<td>0</td>
</tr>
<tr>
<td>diffeq</td>
<td>64</td>
<td>40</td>
<td>924</td>
<td>314</td>
</tr>
<tr>
<td>dsip</td>
<td>229</td>
<td>198</td>
<td>924</td>
<td>224</td>
</tr>
<tr>
<td>elliptic</td>
<td>131</td>
<td>115</td>
<td>2021</td>
<td>886</td>
</tr>
<tr>
<td>ex5p</td>
<td>8</td>
<td>63</td>
<td>210</td>
<td>0</td>
</tr>
<tr>
<td>ex1010</td>
<td>10</td>
<td>10</td>
<td>876</td>
<td>0</td>
</tr>
<tr>
<td>frisc</td>
<td>20</td>
<td>117</td>
<td>2167</td>
<td>821</td>
</tr>
<tr>
<td>misex3</td>
<td>14</td>
<td>14</td>
<td>939</td>
<td>0</td>
</tr>
<tr>
<td>pdc</td>
<td>16</td>
<td>40</td>
<td>2216</td>
<td>0</td>
</tr>
<tr>
<td>s298</td>
<td>4</td>
<td>7</td>
<td>738</td>
<td>8</td>
</tr>
<tr>
<td>s38417</td>
<td>29</td>
<td>107</td>
<td>3998</td>
<td>1391</td>
</tr>
<tr>
<td>s38584</td>
<td>38</td>
<td>305</td>
<td>4501</td>
<td>1232</td>
</tr>
<tr>
<td>seq</td>
<td>41</td>
<td>35</td>
<td>1118</td>
<td>0</td>
</tr>
<tr>
<td>spla</td>
<td>16</td>
<td>46</td>
<td>1916</td>
<td>0</td>
</tr>
<tr>
<td>tseng</td>
<td>52</td>
<td>123</td>
<td>626</td>
<td>225</td>
</tr>
</tbody>
</table>

Table 4-1. Benchmark Characteristics
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. Therefore, we only allow SMAP to make memory input cuts at register inputs. If the fan-outs of these registers become completely packed into the memories, then these registers are removed from the circuit. For circuits that are purely combinational, we add registers to all primary inputs. We also modified SMAP to generate memory initialization files (MIF) that describe the contents of each ROM. The MIF files are used later for activity estimation. To verify that the circuit generated by SMAP is functionally equivalent to the original circuit, bounded sequential equivalence checking was performed using an academic tool from Berkeley called ABC [61].

Prior to performing activity estimation, the circuits generated by SMAP are placed in a test harness. This test harness consists of a linear feedback shift register and a wide XOR gate and is described in detail in the subsequent section. The harness is required in the Board Measurement flow and hence we include it in the VPR flow to make the circuits in both flows as similar as possible.

To estimate the activities on the nets, the activity estimator described in Section 3.1.3 was used. To place and route the circuits, a modified version of TVPACK and VPR that supports memories was used. The clustering, placement, and routing algorithms were timing-driven. We assume an architecture where memories are arranged in columns; the ratio of memory columns to logic columns was fixed at 1:6, which is similar to the Altera
Stratix device used in the Board Measurement Flow. 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 the power model described in Section 3.2.3. Estimating the memory cycle power and the leakage power with this power model requires technology information; we obtained this information by creating memories of different sizes using the Virage Logic Memory Compiler [62], and using the average performance read cycle, write cycle, and leakage power values. The memory compiler also provided timing information which was used by the timing-driven place and route algorithms within VPR. A 0.18μm TSMC CMOS process was assumed throughout.

Finally, in order to reduce the placement noise injected by the random nature of the simulated annealing placement algorithm, placement and routing was performed five times for each case using a different placement seed value. The power estimates reported in the following sections are arithmetically averaged over the five iterations.

### 4.1.2 Board Measurement Flow

In order to validate the trends of the VPR-based flow, we implement some of these circuits on an Altera Nios Development Kit (Stratix Professional Edition) which contains a 0.13μm 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 as shown in Figure 4-2. Driving the external input and output pins of the FPGA can consume significant power; we want to minimize this effect so that it will not obscure the trends that we want to measure. The strategy is to reduce the number of I/O connections to the FPGA and is achieved as follows. The 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 multi-input exclusive-or gate connected to all of the primary outputs. The harnessed circuit itself only has a single input, the clock, and a single output, the exclusive-or gate output which is also registered to prevent glitching of the output pin. The harnessed circuit was then replicated several times to fill the FPGA.

It is important to note that the VPR-based flow is based on a 0.18μm technology process whereas the Board Measurement Flow is based on a 0.13μm technology process. The goal of the Board Measurement flow is only to validate the trends found through the VPR-based flow. A direct comparison of the absolute power and energy values between the two flows is less meaningful.
4.2 Experimental Results

This section investigates the impact of the three architectural parameters shown in Table 4-2 on power and energy dissipation. Results are presented in terms of energy per cycle (for combinational circuits, it is assumed that the cycle time is the maximum combinational delay through the circuit).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Memories</td>
<td>$N$</td>
<td>The number of memory blocks used to implement logic</td>
</tr>
<tr>
<td>Memory Size</td>
<td>$B$</td>
<td>The total number of bits in each memory block</td>
</tr>
<tr>
<td>Flexibility</td>
<td>$w_{\text{max}}$</td>
<td>The maximum configurable width of the memory block</td>
</tr>
</tbody>
</table>

Table 4-2. Parameters Under Investigation

4.2.1 Energy vs. Number of Memories

We first determine the impact of the number of memory arrays used to implement logic on the power dissipation of a circuit. Intuitively, as the number of memory arrays goes up, more logic can be packed into the memories. Whether this provides an overall reduction in power 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 power reduction (increase) will be smaller (larger) as more memory arrays are used.
Figure 4-3 shows the results using the VPR flow averaged across all twenty circuits. The array size was fixed at B=512 bits, and the flexibility was fixed at $w_{max}=16$ as in the Altera Stratix device. The number of memory arrays is varied from 0 (all logic implemented in lookup-tables) to 8. Since the agreement with our technology provider prohibits us from publishing the absolute power characteristics of the Virage Memory Compiler output (the memory power), the vertical axis in the graph has been normalized to the case where no memories are used (the results are geometrically averaged before normalization). The bottom line corresponds to the energy dissipated in the logic blocks (the logic energy). The logic energy decreases by 14.5% when eight memory blocks are used. As expected, the logic energy goes down as the number of arrays increases. This is because as the number of 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 power and the power dissipated in the routing and the clock (so the area between the lower two lines represents the routing and clock power). The routing energy decreases by 3.6% when eight memory arrays are used. Again, more memory arrays means there are fewer LUTs, leading to fewer connections, and hence, lower routing energy. Finally, the top line is the overall power; the difference between the top line and the middle line represents the power dissipated in the memories. As the graph shows, overall, mapping logic to memory arrays does not reduce power. In fact, the power 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 and the programmable interconnect.

The experiment was repeated using a memory with $B=4096$ bits and $w_{\text{max}}=32$; the results are shown in Figure 4-4. This figure shows that although the reduction in logic and routing energy (50.0% and 33.8% respectively when using eight memories) is more significant than when using 512bit memories, the energy consumed per memory is also larger. As mentioned earlier, the rate of power increase (or the slope of the plot for overall power) is steeper when more memories are used. This is because, as shown in Figure 4-5, the number of LUTs that are packed decreases as more memories are used. This effect is more pronounced with the 4kBit memories. The prominence of this effect depends on the number of choices that SMAP has to make good cuts, which is dependent on the user circuit size and the memory block size.
Figure 4-4. Impact on Energy When Increasing the Number of 4kBit Memory Arrays (VPR Flow)

Figure 4-5. Number of Packed 4LUTs When Increasing the Number of Memories
To verify that energy consumption increases when the number of memories increases, we implemented a number of the circuits on an Altera NIOS development kit containing a 0.13µm Stratix EP1S40F780C5 device. The Stratix device contains two types of memory arrays: 512-bit blocks and 4Kbit blocks. Figure 4-6 shows the measured results when only 512-bit blocks are used for a representative circuit (misex3). The bottom line represents the power dissipated in the memories and clock network. This was obtained by disabling the LFSR and thus 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.

In this circuit, there is an 8.2% increase in overall power when seven memories are used; this matches the trend found in the VPR results. The experiment was repeated using only 4Kbit blocks; the results are presented in Figure 4-7 and show a 16.7% increase in overall power when seven memories are used.
Figure 4-6. Impact on Energy When Increasing the Number of 512bit Memory Arrays (Measured Flow)

Figure 4-7. Impact on Energy When Increasing the Number of 4kBit Memory Arrays (Measured Flow)
4.2.2 Energy vs. Memory Array Size

Although the results in the previous section show that the power increases 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]. 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 section, we investigate the impact of the array size on the power dissipated by the circuits. Intuitively, a larger array means more logic can be packed into each array, 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 in powers of two. We repeat the experiment for various values of $w_{max}$. Clearly, as $B$ increases, the power dissipated in each memory each access increases. This is shown graphically in Figure 4-8. Note that in our power model the memory energy is the same for all circuits when the memory size and number of memories used is fixed. Again, our agreement with our technology provider does not allow us to present absolute numbers, so we normalize all energy values in this section (Figure 4-8 - Figure 4-12) to the overall energy consumed when mapping to a single 256x1 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. The power dissipated by the memories produced by the Virage Logic Memory Compiler depends more on the number of columns than the number of

56
rows. This is because in the memory core only a single wordline toggles upon each access whereas one of the two bitlines from each column must toggle. Therefore, increasing the number of rows (and hence the number of wordlines) does not impact the power consumption of the memory as much as increasing the number of columns (and number of bitlines). Because of this, the power dissipation of memory blocks in which $B$ does not have an integral square root is not significantly larger than the next smaller value of $B$ (which would have an integral square root). This explains the “bumpy” appearance of the line in Figure 4-8.

![Figure 4-8. Impact on Memory Energy When Increasing Memory Array Size](image)

Now consider the impact on the logic energy. The results are shown in Figure 4-9. As expected, a larger array means that more logic can be packed into the array, which reduces the logic energy. However, for $w_{max} = \{1\}$ and $w_{max} = \{1, 2\}$, the trend is relatively flat. This is because, as shown in Figure 4-10, when the flexibility is low, increasing the array size does not lead to any significant increase in the amount of packed logic.
Figure 4-9. Impact on Logic Energy When Increasing Memory Size

Figure 4-10. Impact on Amount of Packable LUTs When Increasing Memory Size
In Figure 4-9, there are several cases where increasing the memory size increases logic power. This is counterintuitive. This is caused by noise in the placement which affects the critical path delay which in turn affects the logic leakage energy. We do not believe that this is a trend that is inherent in our results.

Figure 4-11 shows the impact on routing energy when increasing the memory array size. 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. For similar reasons discussed earlier, the trend for \( w_{max} = \{1\} \) and \( w_{max} = \{1, 2\} \) are also relatively flat.

![Figure 4-11. Impact on Routing Energy When Increasing Memory Size](image)

Figure 4-12 shows the results for the overall energy. Despite the significant reduction in logic and routing energy, the memory power still dominates. Hence, the overall energy increases as the memory size increases. However, an important observation is that non-square memories always perform better than the next smaller size (assuming a reasonable
This is due to the "bumpy" nature of the memory energy, as discussed earlier, and the ability to pack significantly more logic when using larger memory sizes as shown in Figure 4-10.

Figure 4-12. Impact on Overall Energy When Increasing Memory Size

Since we are not able to vary $B$ on the Altera device, we did not perform this experiment on the actual commercial device. However, by comparing the results summarized in Figure 4-6 and Figure 4-7, we found that using a 4kbit block consumes approximately 8% more power on average than the 512bit block when $N=1$; this matches the conclusion drawn from the VPR experiments.

4.2.3 Energy vs. Memory Flexibility

This section investigates the power implications of changing the flexibility of each memory array. As described earlier, FPGA memory arrays typically have a configurable output width; the set of output widths in which an array can be configured is denoted $w_{eff}$. 

60
In this section, we vary the maximum value in $w_{offs}$ which we denote $w_{max}$, and measure the impact on energy dissipation. All energy values in this section (Figure 4-13 - Figure 4-15) have been normalized to the overall energy value when mapping to a single 8192x1 memory.

Intuitively, changing $w_{max}$ will have little affect on the power dissipated in the configurable memory array itself. Changing $w_{max}$ does not affect the shape of the fixed-size array in the embedded memory block; it only affects the multiplexers in the programmable column decoder. Since these multiplexers are very small, they would cause a negligible increase in memory power as we increase $w_{max}$. Since the power dissipated in the programmable column decoder is ignored in our power model, it is also ignored in the experiments reported. Although changing $w_{max}$ would change the number of sense amplifiers needed in the fixed-size memory component, which would change the memory energy, this effect is ignored. 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, $w_{max}$, of the array. 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. It is important to remember that although a wider configuration may be allowed, SMAP may not choose to use it.

First consider the energy dissipated in the logic blocks. Figure 4-13 shows the impact of increasing $w_{max}$ on logic energy. For larger sized arrays, as $w_{max}$ increases, the logic
energy decreases. This is because the amount of logic implemented using lookup tables is reduced. The decrease in logic energy tapers off when $w_{\text{max}}$ becomes large because SMAP rarely produces solutions that use a configuration where the number of memory outputs is larger than the number of memory address inputs (which is a function of the configured depth). This is because the shape of circuits implemented with LUTs is typically triangular [63]. In other words, the number of LUTs at each combinational delay level decreases from the primary inputs. For small arrays, the number of memory outputs exceeds the number of memory inputs at very small widths. Thus there is little or no change in the logic energy for small arrays when the allowable width is increased.

![Figure 4-13. Impact on Logic Energy When Increasing Memory Flexibility](image)

In Figure 4-13, there are several cases where increasing flexibility increases logic power. Similar to the previous experiment in Section 4.2.2, this is caused by noise in the
placement which affects the critical path delay which in turn affects the logic leakage power.

Figure 4-14 shows the routing energy when \( w_{\text{max}} \) is increased. In general, the routing energy follows the same trend as the logic energy because the reduction in the number of LUTs directly translates into a reduction in the number of nets that need to be routed. However, there is a competing factor that may not be immediately obvious. Increasing the width of the memory increases the number of pins on the memory block that need to be accessible by the FPGA routing fabric. This means that additional programmable connections need to be added in the connection blocks that connect the memory to the routing fabric. This in turn adds parasitic capacitance to the routing tracks that surround the memory. Therefore, even if the additional width is not used by SMAP, the routing energy may increase if the routing tracks surrounding the memory are used. This explains the convex shape of the curves in Figure 4-14.

Figure 4-15 shows the impact on the total energy per cycle. Since memory energy is constant, the overall impact corresponds to the change in logic and routing energy. For large arrays, this results in a significant initial decrease in energy. For large values of \( w_{\text{max}} \), this decrease tapers off and begins increasing slightly again due to the routing energy. For smaller arrays, there is a slight increase in energy. This is caused by a negligible decrease in logic energy and an increase in routing energy. Again, since we are not able to vary \( w_{\text{max}} \) on our Altera part, we did not perform this experiment on the commercial device.
Figure 4-14. Impact on Routing Energy When Increasing Memory Flexibility

Figure 4-15. Impact on Overall Energy When Increasing Memory Flexibility
4.3 Sensitivity of Results

We have identified three aspects of this study to which our conclusions are sensitive. First, the results of the SMAP algorithm have been shown to be very sensitive to the technology mapped circuit that it is given [64]. For the most accurate results, the CAD tools should closely match the ones used in the final production software. In this case, we were able to use Altera’s commercial technology-mapper QIS.

The same study also showed that the use of different memory-to-logic mappers can significantly affect architectural conclusions. In our study we used SMAP which outperforms all other memory-to-logic mappers in terms of density and packable LUTs. Since total circuit energy is directly related to the amount of logic, we feel that using SMAP is the most appropriate.

Most importantly, the values used to describe the memory architecture power characteristics to VPR can drastically affect the conclusions. To address this, we used real values from memory architectures provided by TSMC, and used memory implementations provided by a commercial memory compiler from Virage.

4.4 Summary

This chapter has 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 studies have reported significant density increases when embedded memory is used in this way, and suggested that there is no reason not to do this. The results of this
study 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 trade-off 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 blocks. In that case, optimizing the size and flexibility of the memory blocks to reduce this power penalty is important. We have shown that for most array sizes, the arrays should be as flexible as possible but consideration should be given when many memory I/Os are involved. We have also shown that smaller memory arrays are more power efficient than large arrays. However, when larger arrays are desired to increase density, memories with non-integral square roots and a larger depth-to-width ratio should be used.
5 POWER AWARE METHODS FOR MAPPING LOGIC TO MEMORIES

The previous chapter investigated the power implications of mapping logic to memories using an area-driven algorithm called SMAP and showed that this technique results in a severe power penalty. This chapter investigates two possible modifications to SMAP to make it more power aware. The first modification changes the SMAP cost function from being area-aware to activity-aware. The second approach uses larger logical memory arrays and a power efficient logical-to-physical memory-mapping technique. The following sections will describe both techniques and compare them to the experimental results from Chapter 4.

5.1 Activity Aware Cost Function

This section begins by discussing activity-aware technology mapping techniques for homogeneous FPGAs. We apply these techniques along with some new ones to SMAP and present our new activity aware cost function. The performance of the activity-aware algorithm is compared against the results in the experiment performed in Section 4.2.1.
5.1.1 Power Aware Homogeneous Technology Mapping

Existing power aware technology mapping for LUTs seek to minimize power in two ways. The first method aims to minimize the routing power by choosing mappings that encapsulate as many high activity nets as possible [29, 30]. Since the tracks in an FPGA have large parasitic capacitance, reducing the number of nets that need to use the interconnect, especially the ones that switch frequently, can reduce the overall circuit power consumption. The second method aims to minimize node replication. Node replication occurs when a LUT is used to implement a cone of logic that contains a node that has a fanout outside of the cone. Node replication is required to produce depth-optimal solutions [28], but has been shown to be undesirable when power is a concern [29]. After replication, the signals that drive the replicated node now have to drive the replicated node as well as its replicant node. This means that more segments need to be routed through the FPGA interconnect.

5.1.2 Activity-Aware SMAP

The overall strategy for our power-aware algorithm is to change the way that SMAP ranks nodes when determining which nodes to select as a memory output. As described in Section 2.3.3, SMAP counts the number of nodes in the potential output's MFFC (the nodes that can be packed) and uses this to rank the desirability of selecting it as an output. In our power-aware version, we use the following cost function for ranking the desirability of selecting a potential node \( n \) as a memory output.

\[
\text{Cost}(n) = k_1 \cdot ECost(n) - k_2 \cdot RCost(n) + k_3 \cdot FCost(n) + k_4 \cdot GCost(n) \quad (5.1)
\]
The cost function has four components each of which focuses on a different way to reduce the number of high activity connections. The Edge Cost (ECost) is a modified version of the equation from [29], and is shown in Equation 5.2. The purpose of the Edge Cost is to favour mappings that encapsulate high activity edges inside the memory.

\[ ECost(n) = \sum_{v \in MFFC(n)} \text{activity}(v) \| \text{fanout}(v) \cap MFFC(n) \| \]  

(5.2)

This equation is a summation over all nodes in the MFFC of \( n \). Any fanout of a node in the MFFC that is also in the MFFC is encapsulated inside of the memory. We weight these edges with their switching activity and add it to the ECost value.

The second term, the Replication Cost (RCost), penalizes replication of high activity nets due to node replication. Since we do not know which nodes (if any) will be replicated until after all the outputs are actually selected, we use a heuristic which assumes that \( n \) is the only output of the memory and therefore the nodes in MFFC(\( n \)) are the only nodes that are packed. This implies that all nodes in a cone rooted at a potential node that drives a node in the MFFC of \( n \) will need to be replicated. This is illustrated in Figure 5-1.
Figure 5-1. Node Replication in SMAP

Given the cut that is shown in the left half of Figure 5-1, nodes \{a, b, c, n1, n2\} are all potential outputs. The right half of Figure 5-1 shows the two potential solutions. In the top solution, \(n2\) is chosen as the only output, while in the lower solution, \(n1\) is chosen as the only output. When \(n2\) is chosen as the only output, no replication occurs. When \(n1\) is chosen, the nodes in the cone rooted at \(n2\), which include \{a, b, n2\}, are replicated because \(n2\) drives \(n1\). Due to this replication, the fanouts of 13, 14, 15, and 16 each increase by one. \(RCost\) sums the switching activities of these new fanouts and is expressed in Equation 5.3 where \(RNodes\) is the set of replicated nodes and \(s\) is the seed node.

\[
RCost(n) = \sum_{v \in RNodes(n,s)} \left[ \sum_{u \in fanin(v)}\sum_{w \in cutter(s)}\sum_{w \in Fanin(MFFC(n))} activity(u) \right]
\]  

(5.3)

70
The third term in the cost function is the Fanin Cost, $F_{Cost}$. This term favours mappings that reduce the fanout of the cut-set and is expressed by Equation 5.4. This term only applies when mapping to memories with widths that are greater than one. The motivation behind this term is illustrated in Figure 5-2 which shows three possible mapping solutions for a 4-input 2-output memory.

$$W_{Cost}(n) = \begin{cases} \text{if } w > 1 & \sum_{v \in MPFC(n)} \sum_{u \in fanin(v)} \text{activity}(u) \\ \text{else} & 0 \end{cases} \quad (5.4)$$

The table in Figure 5-2 summarizes the number of fanouts for each node in the cut-set. In each mapping solution, the number of fanouts from the cut-set is reduced by different
amounts. The goal of the FCost term is to favour mappings that reduce the cut-set
fanouts while also taking into account their switching activities. Once again, we do not
know which nodes will be chosen as outputs when the cost function is evaluated, and
hence do not know which cut-set nodes will actually receive a reduction in their number
of fanouts. Therefore we again use a heuristic which assumes that each cut-set node that
fans-into the MFFC of the potential output node being evaluated will be shared by one of
the other memory output’s MFFC.

The final term is called the Glitch Cost (GCost) and is calculated using Equation 5.5.
The purpose of this term is to favour memory outputs that have high predicted glitching.
Since we are using synchronous memories, there is no glitching at the memory outputs.
Choosing a node that has glitching for a memory output not only reduces the switching
activity of the net driven by the node, but potentially also the switching activity of
downstream combinational nodes.

\[
G\text{Cost}(n) = \text{activity}(n) - \text{switch}\_\text{prob}(n)
\]  \hspace{1cm} (5.5)

The cost function in Equation 5.1 is a summation of four terms. In general, adding terms
within a cost function is not advisable since if the magnitudes of the terms are different,
or the units are different, changes in one term can easily overpower changes in another
term. Because of this, researchers often multiply (rather than add) individual terms, or
normalize each term so that the terms can be added [15]. In Equation 5.1, each term is a
measure of activity, and thus is of the same magnitude and has the same units. Thus, the
addition of terms is acceptable in this instance. Equation 5.1 contains four constants
which can be used to weight some terms more than others; however, in the results in this chapter, \( k_1 = k_2 = k_3 = k_4 = 1 \). We have not investigated other values of \( k_i \) to \( k_4 \).

### 5.1.3 Experimental Methodology

To evaluate the power dissipation of the circuits produced by our power-aware SMAP algorithm, we used the VPR-Flow described in Section 4.1.1. We repeated the experiment from Section 4.2.1 but replaced SMAP with our activity-aware version. For the discussion of the results, we will refer the results from Section 4.2.1 as the baseline.

### 5.1.4 Experimental Results

Intuitively, the new cost function will reduce the average activity of all nets in the circuit. However, since the mapping algorithm employing our new cost function does not explicitly optimize for the number of nodes packed into each memory, we would expect that the number of nodes packed may be slightly less than that in the original SMAP algorithm.

Figure 5-3 shows the impact that the activity-aware SMAP has on the number of packed LUTs when using memories with \( B=512 \) and memories with \( B=4096 \). In both cases, the number of packed LUTs is reduced on average by 12.58\% and 16.06\% when using eight 512 bit and 4096 bit memories respectively.
Table 5-1 summarizes the percentage change in routing energy when using our activity-aware version of SMAP with between one and eight 512 bit memories. As the table shows, the routing energy decreases in some cases, and increases in other cases. This is due to two factors. First, the actual changes that we expect as a result of our new cost function are small. Unlike techniques for power-aware homogeneous technology mapping [30], in which every node is impacted by the change in cost function, here, only a small fraction of all nodes in the circuit are actually mapped to memory, and hence, only a small amount of the routing power is impacted by the change in cost function. The second factor is that the VPR place and route solutions are sensitive to even slight perturbations in the netlist (this is far more significant than perturbations in the placement seed). Thus, as the circuit netlist is changed, the “random” noise from the VPR place and route results overwhelms any changes caused by the cost function. The net result is that
our activity-aware cost function is not effective in decreasing the overall routing energy of the circuits.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>N=1</th>
<th>N=2</th>
<th>N=3</th>
<th>N=4</th>
<th>N=5</th>
<th>N=6</th>
<th>N=7</th>
<th>N=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>-2.00</td>
<td>-3.32</td>
<td>-0.65</td>
<td>-0.26</td>
<td>-1.36</td>
<td>4.11</td>
<td>-0.89</td>
<td>-2.32</td>
</tr>
<tr>
<td>apex2</td>
<td>-4.27</td>
<td>2.16</td>
<td>3.58</td>
<td>-3.46</td>
<td>-1.24</td>
<td>-3.32</td>
<td>0.55</td>
<td>2.82</td>
</tr>
<tr>
<td>apex4</td>
<td>-1.82</td>
<td>-0.14</td>
<td>-1.64</td>
<td>-2.57</td>
<td>0.66</td>
<td>-3.19</td>
<td>-0.89</td>
<td>-2.45</td>
</tr>
<tr>
<td>bigkey</td>
<td>-6.86</td>
<td>-5.49</td>
<td>-1.50</td>
<td>-3.28</td>
<td>1.79</td>
<td>-3.74</td>
<td>8.02</td>
<td>9.55</td>
</tr>
<tr>
<td>clma</td>
<td>-2.97</td>
<td>4.63</td>
<td>3.24</td>
<td>3.91</td>
<td>2.11</td>
<td>-0.68</td>
<td>-0.62</td>
<td>-4.26</td>
</tr>
<tr>
<td>des</td>
<td>0.36</td>
<td>-0.97</td>
<td>0.25</td>
<td>1.20</td>
<td>7.23</td>
<td>5.47</td>
<td>4.41</td>
<td>3.88</td>
</tr>
<tr>
<td>diffeq</td>
<td>3.79</td>
<td>-1.11</td>
<td>-2.45</td>
<td>-1.15</td>
<td>-3.12</td>
<td>-7.57</td>
<td>-11.32</td>
<td>-0.89</td>
</tr>
<tr>
<td>dslp</td>
<td>-2.87</td>
<td>7.28</td>
<td>-0.10</td>
<td>-6.33</td>
<td>-0.67</td>
<td>8.01</td>
<td>2.40</td>
<td>3.28</td>
</tr>
<tr>
<td>elliptic</td>
<td>0.74</td>
<td>13.73</td>
<td>-1.19</td>
<td>-2.23</td>
<td>-2.83</td>
<td>-6.24</td>
<td>-2.28</td>
<td>-4.19</td>
</tr>
<tr>
<td>ex5p</td>
<td>-1.06</td>
<td>4.28</td>
<td>11.24</td>
<td>5.54</td>
<td>2.89</td>
<td>15.46</td>
<td>1.03</td>
<td>2.36</td>
</tr>
<tr>
<td>ex1010</td>
<td>-0.84</td>
<td>-2.18</td>
<td>0.06</td>
<td>-1.20</td>
<td>0.08</td>
<td>0.81</td>
<td>-0.52</td>
<td>-0.45</td>
</tr>
<tr>
<td>frisc</td>
<td>-1.05</td>
<td>-2.36</td>
<td>-1.93</td>
<td>-0.75</td>
<td>3.98</td>
<td>-0.27</td>
<td>-0.27</td>
<td>-0.28</td>
</tr>
<tr>
<td>mixe3</td>
<td>-3.68</td>
<td>-0.02</td>
<td>5.06</td>
<td>2.72</td>
<td>6.40</td>
<td>2.57</td>
<td>2.55</td>
<td>2.67</td>
</tr>
<tr>
<td>pdc</td>
<td>-1.00</td>
<td>3.06</td>
<td>1.56</td>
<td>8.09</td>
<td>5.47</td>
<td>3.07</td>
<td>3.24</td>
<td>4.53</td>
</tr>
<tr>
<td>s298</td>
<td>1.34</td>
<td>-2.86</td>
<td>-5.53</td>
<td>5.50</td>
<td>2.53</td>
<td>10.23</td>
<td>17.56</td>
<td>15.45</td>
</tr>
<tr>
<td>s38417</td>
<td>2.68</td>
<td>-5.07</td>
<td>-4.55</td>
<td>2.95</td>
<td>-2.75</td>
<td>0.37</td>
<td>0.96</td>
<td>-0.77</td>
</tr>
<tr>
<td>s38584</td>
<td>-1.86</td>
<td>-5.90</td>
<td>-3.88</td>
<td>-1.17</td>
<td>-3.72</td>
<td>-1.84</td>
<td>-1.35</td>
<td>-1.25</td>
</tr>
<tr>
<td>seq</td>
<td>-1.11</td>
<td>3.59</td>
<td>0.83</td>
<td>6.00</td>
<td>3.62</td>
<td>3.05</td>
<td>3.33</td>
<td>2.26</td>
</tr>
<tr>
<td>spla</td>
<td>-2.45</td>
<td>1.56</td>
<td>2.94</td>
<td>9.11</td>
<td>30.49</td>
<td>36.81</td>
<td>37.38</td>
<td>47.62</td>
</tr>
<tr>
<td>tseng</td>
<td>-0.77</td>
<td>-0.08</td>
<td>5.78</td>
<td>2.00</td>
<td>2.27</td>
<td>2.35</td>
<td>4.85</td>
<td>3.27</td>
</tr>
<tr>
<td>avg</td>
<td>-1.29</td>
<td>0.54</td>
<td>0.56</td>
<td>1.23</td>
<td>2.69</td>
<td>3.27</td>
<td>3.41</td>
<td>4.04</td>
</tr>
</tbody>
</table>

Table 5-1. Percentage Change in Routing Energy When Using the Activity Aware Cost Function and Memories with B = 512 bits

For completeness, Table 5-2 and Table 5-3 show the change in logic energy and the overall energy caused by the change in cost function. Intuitively, we would expect the logic energy to rise slightly, but again, the results are inconclusive due to noise in the place and route data. In this case, small changes in the place and route solution impact the critical path of the circuit, which affects the leakage energy dissipated by the circuit. The overall results are shown in Table 5-3. From the results in this table, we can confirm that the new cost function is not effective in decreasing the overall energy dissipated in these circuits. The experiment was repeated assuming \( B = 4096 \), and the conclusion was the same.
<table>
<thead>
<tr>
<th>Circuit</th>
<th>N=1</th>
<th>N=2</th>
<th>N=3</th>
<th>N=4</th>
<th>N=5</th>
<th>N=6</th>
<th>N=7</th>
<th>N=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>-2.12</td>
<td>-1.75</td>
<td>-2.44</td>
<td>-0.53</td>
<td>-0.84</td>
<td>1.75</td>
<td>-4.33</td>
<td>-3.16</td>
</tr>
<tr>
<td>apex2</td>
<td>0.80</td>
<td>3.20</td>
<td>-0.72</td>
<td>-0.77</td>
<td>1.83</td>
<td>-2.68</td>
<td>0.80</td>
<td>2.69</td>
</tr>
<tr>
<td>apex4</td>
<td>0.50</td>
<td>-5.04</td>
<td>-0.56</td>
<td>-2.22</td>
<td>-2.48</td>
<td>-3.85</td>
<td>-16.04</td>
<td>-19.94</td>
</tr>
<tr>
<td>bigkey</td>
<td>-3.67</td>
<td>-5.90</td>
<td>-1.90</td>
<td>15.52</td>
<td>-17.21</td>
<td>-16.32</td>
<td>-16.48</td>
<td>12.07</td>
</tr>
<tr>
<td>clma</td>
<td>-2.51</td>
<td>0.49</td>
<td>-2.23</td>
<td>2.35</td>
<td>-3.09</td>
<td>-0.93</td>
<td>-3.10</td>
<td>-7.58</td>
</tr>
<tr>
<td>des</td>
<td>-0.76</td>
<td>0.85</td>
<td>0.67</td>
<td>-0.14</td>
<td>1.12</td>
<td>-1.51</td>
<td>0.82</td>
<td>2.59</td>
</tr>
<tr>
<td>diffeq</td>
<td>-2.41</td>
<td>-12.81</td>
<td>-0.39</td>
<td>1.54</td>
<td>9.01</td>
<td>-4.61</td>
<td>-7.84</td>
<td>-2.42</td>
</tr>
<tr>
<td>dsip</td>
<td>-2.01</td>
<td>0.01</td>
<td>4.47</td>
<td>-6.06</td>
<td>-0.71</td>
<td>-4.67</td>
<td>1.51</td>
<td>-0.75</td>
</tr>
<tr>
<td>elliptic</td>
<td>2.37</td>
<td>14.28</td>
<td>0.22</td>
<td>2.27</td>
<td>0.32</td>
<td>-6.09</td>
<td>-5.69</td>
<td>-0.87</td>
</tr>
<tr>
<td>ex5p</td>
<td>-1.72</td>
<td>-7.30</td>
<td>-5.54</td>
<td>-1.30</td>
<td>-10.67</td>
<td>7.33</td>
<td>-11.95</td>
<td>-12.93</td>
</tr>
<tr>
<td>ex1010</td>
<td>0.20</td>
<td>4.32</td>
<td>0.13</td>
<td>0.75</td>
<td>-8.58</td>
<td>2.61</td>
<td>0.57</td>
<td>-3.12</td>
</tr>
<tr>
<td>frisc</td>
<td>-0.52</td>
<td>3.32</td>
<td>1.74</td>
<td>-4.37</td>
<td>-8.04</td>
<td>-4.24</td>
<td>-7.54</td>
<td>-22.09</td>
</tr>
<tr>
<td>misex3</td>
<td>-1.91</td>
<td>-1.53</td>
<td>1.48</td>
<td>-0.87</td>
<td>-2.33</td>
<td>-4.82</td>
<td>-4.63</td>
<td>0.00</td>
</tr>
<tr>
<td>pdc</td>
<td>0.00</td>
<td>2.06</td>
<td>-2.23</td>
<td>9.70</td>
<td>3.33</td>
<td>-1.07</td>
<td>-4.00</td>
<td>-0.59</td>
</tr>
<tr>
<td>s298</td>
<td>-1.13</td>
<td>-5.06</td>
<td>-5.29</td>
<td>-1.68</td>
<td>35.02</td>
<td>35.93</td>
<td>12.87</td>
<td></td>
</tr>
<tr>
<td>s38417</td>
<td>2.39</td>
<td>-5.83</td>
<td>-2.92</td>
<td>-0.25</td>
<td>-2.92</td>
<td>-3.37</td>
<td>2.78</td>
<td>-2.34</td>
</tr>
<tr>
<td>s38584</td>
<td>2.84</td>
<td>-0.99</td>
<td>0.59</td>
<td>0.19</td>
<td>-2.21</td>
<td>0.31</td>
<td>1.73</td>
<td>0.96</td>
</tr>
<tr>
<td>seq</td>
<td>-3.52</td>
<td>-0.89</td>
<td>1.94</td>
<td>-1.76</td>
<td>-10.16</td>
<td>-3.64</td>
<td>2.68</td>
<td>-0.54</td>
</tr>
<tr>
<td>spla</td>
<td>-0.94</td>
<td>-0.43</td>
<td>-0.27</td>
<td>1.80</td>
<td>20.83</td>
<td>18.86</td>
<td>19.56</td>
<td>30.05</td>
</tr>
<tr>
<td>tseng</td>
<td>-0.50</td>
<td>-6.17</td>
<td>2.10</td>
<td>-3.34</td>
<td>3.75</td>
<td>4.70</td>
<td>4.05</td>
<td>-3.19</td>
</tr>
<tr>
<td>avg</td>
<td>-0.73</td>
<td>-1.26</td>
<td>-0.56</td>
<td>0.54</td>
<td>-1.27</td>
<td>0.39</td>
<td>-0.56</td>
<td>-0.91</td>
</tr>
</tbody>
</table>

Table 5-2. Percentage Change in Logic Energy When Using the Activity Aware Cost Function and Memories with B= 512

<table>
<thead>
<tr>
<th>Circuit</th>
<th>N=1</th>
<th>N=2</th>
<th>N=3</th>
<th>N=4</th>
<th>N=5</th>
<th>N=6</th>
<th>N=7</th>
<th>N=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>-2.29</td>
<td>-2.67</td>
<td>-1.60</td>
<td>-0.83</td>
<td>-1.34</td>
<td>1.31</td>
<td>-1.92</td>
<td>-2.08</td>
</tr>
<tr>
<td>apex2</td>
<td>-2.52</td>
<td>4.94</td>
<td>1.02</td>
<td>-2.34</td>
<td>-0.67</td>
<td>-2.54</td>
<td>-0.22</td>
<td>0.96</td>
</tr>
<tr>
<td>apex4</td>
<td>-1.00</td>
<td>-2.97</td>
<td>-1.36</td>
<td>-2.05</td>
<td>-1.30</td>
<td>-2.26</td>
<td>-6.18</td>
<td>-6.47</td>
</tr>
<tr>
<td>bigkey</td>
<td>-5.12</td>
<td>-5.47</td>
<td>-2.02</td>
<td>4.86</td>
<td>-7.75</td>
<td>-8.61</td>
<td>-5.45</td>
<td>6.29</td>
</tr>
<tr>
<td>clma</td>
<td>-2.83</td>
<td>2.05</td>
<td>0.46</td>
<td>2.13</td>
<td>-0.38</td>
<td>-1.04</td>
<td>-1.60</td>
<td>-4.35</td>
</tr>
<tr>
<td>des</td>
<td>-0.71</td>
<td>-0.61</td>
<td>-3.49</td>
<td>-0.12</td>
<td>2.75</td>
<td>-2.12</td>
<td>1.39</td>
<td>1.71</td>
</tr>
<tr>
<td>diffeq</td>
<td>-0.55</td>
<td>-5.82</td>
<td>-1.41</td>
<td>-0.58</td>
<td>0.71</td>
<td>-7.22</td>
<td>-7.93</td>
<td>-4.98</td>
</tr>
<tr>
<td>dsip</td>
<td>-2.73</td>
<td>1.85</td>
<td>1.35</td>
<td>-5.44</td>
<td>-1.19</td>
<td>-0.42</td>
<td>0.53</td>
<td>-0.13</td>
</tr>
<tr>
<td>elliptic</td>
<td>0.66</td>
<td>10.39</td>
<td>1.01</td>
<td>-0.81</td>
<td>-1.63</td>
<td>-4.83</td>
<td>-3.06</td>
<td>-2.37</td>
</tr>
<tr>
<td>ex5p</td>
<td>-1.57</td>
<td>-1.84</td>
<td>-0.39</td>
<td>0.42</td>
<td>-1.90</td>
<td>1.10</td>
<td>-1.89</td>
<td>-1.80</td>
</tr>
<tr>
<td>ex1010</td>
<td>-0.86</td>
<td>0.76</td>
<td>-0.72</td>
<td>-0.76</td>
<td>3.96</td>
<td>0.07</td>
<td>-0.78</td>
<td>-1.65</td>
</tr>
<tr>
<td>frisc</td>
<td>-1.18</td>
<td>-0.68</td>
<td>-0.91</td>
<td>-1.94</td>
<td>-1.09</td>
<td>-0.93</td>
<td>-0.88</td>
<td>-0.84</td>
</tr>
<tr>
<td>misex3</td>
<td>-3.00</td>
<td>-1.02</td>
<td>2.03</td>
<td>0.28</td>
<td>1.15</td>
<td>-0.84</td>
<td>-0.80</td>
<td>0.19</td>
</tr>
<tr>
<td>pdc</td>
<td>-1.00</td>
<td>1.80</td>
<td>-0.32</td>
<td>6.23</td>
<td>2.99</td>
<td>0.65</td>
<td>-0.09</td>
<td>1.28</td>
</tr>
<tr>
<td>s298</td>
<td>-0.86</td>
<td>-3.39</td>
<td>-3.60</td>
<td>-0.16</td>
<td>0.46</td>
<td>5.40</td>
<td>6.35</td>
<td>2.71</td>
</tr>
<tr>
<td>s38417</td>
<td>1.47</td>
<td>-5.09</td>
<td>-3.52</td>
<td>0.27</td>
<td>-2.79</td>
<td>-1.89</td>
<td>0.71</td>
<td>-1.81</td>
</tr>
<tr>
<td>s38584</td>
<td>-0.07</td>
<td>-3.51</td>
<td>-1.94</td>
<td>-0.98</td>
<td>-3.02</td>
<td>-1.20</td>
<td>-0.43</td>
<td>-0.71</td>
</tr>
<tr>
<td>seq</td>
<td>-2.36</td>
<td>6.28</td>
<td>0.49</td>
<td>1.28</td>
<td>-2.42</td>
<td>-0.50</td>
<td>1.29</td>
<td>0.01</td>
</tr>
<tr>
<td>spla</td>
<td>-2.09</td>
<td>0.21</td>
<td>0.94</td>
<td>8.62</td>
<td>23.32</td>
<td>20.20</td>
<td>19.60</td>
<td>24.90</td>
</tr>
<tr>
<td>tseng</td>
<td>-1.12</td>
<td>-2.80</td>
<td>1.15</td>
<td>-1.21</td>
<td>0.44</td>
<td>0.49</td>
<td>0.62</td>
<td>-0.92</td>
</tr>
<tr>
<td>avg</td>
<td>-1.48</td>
<td>-0.38</td>
<td>-0.74</td>
<td>0.30</td>
<td>0.12</td>
<td>-0.26</td>
<td>-0.04</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Table 5-3. Percentage Change in Overall Energy When Using the Activity Aware Cost Function and Memories with B= 512
5.1.5 Summary for Activity-Aware Cost Function

In this section, we described our new activity-aware SMAP cost function. The goal of the cost function is to minimize the number of high activity connections in the mapped solution. We compared the results from the activity-aware SMAP against those from Section 4.2.1. In general, we found that the improvements (if any) were smaller than the noise produced by the place and route tool, and this could not be measured. We therefore conclude that this cost function is not effective at reducing the average power dissipated by the circuits.

As shown in the experiments from Chapter 4, the memory energy is the dominant component of the overall power dissipation. This means that in order to achieve significant energy savings when mapping logic to memories, the power dissipated by the memories needs to be reduced. In the next section, we describe a technique that targets the memory energy directly.

5.2 Power Efficient Super-Arrays

In this section, we describe a different approach to improving the energy-efficiency of logic implemented in memory arrays. The key idea in this technique is to combine two or more physical memory arrays into larger logical memories (also called super-arrays in [5]), and use SMAP to map logic to these larger logical memories. By doing this in a power-efficient way, we can achieve significant energy savings, compared to the original SMAP algorithm.
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-times. An example of this is shown in Figure 5-4a. In this example, two physical arrays with \( B=512 \) and \( w_{\text{eff}} = \{1, 2, 4, 8, 16\} \) are combined to implement a single logical memory with \( B=1024 \) and \( w_{\text{eff}} = \{2, 4, 8, 16, 32\} \). In this example, each physical array supplies one 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 5-4. Forming Logical Memories a)Area Efficient b)Power Efficient](image)

Figure 5-4b 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 \( w_{\text{eff}} = \{1, 2, 4, 8, 16\} \). 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 [65], however, they did not evaluate this idea in the context of heterogeneous technology mapping.

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 example of Figure 5-4b, $BF=2$.

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. Table 5-4 shows the number of extra logic elements required for several values of $BF$. For example, using $BF=4$ requires four LUTs for the memory enable control logic (one for each memory), and two LUTs for each output of the logical memory to perform 4:1 multiplexing. 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.
Table 5-4. Number of LUTs Needed For Power Efficient Logical Memories

<table>
<thead>
<tr>
<th>Blocking Factor</th>
<th>LUTs added</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ME Control</td>
<td>Per Output MUX</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Thus, in this section, we determine whether the proposed technique will actually reduce the power dissipation of implementations generated using SMAP, or whether the power increase due to the extra logic and reduced packing efficiency results in an overall power increase. In addition, we monitor the anticipated decrease in packing efficiency.

5.2.1 Experimental Methodology

Figure 5-5 shows our experimental methodology. Each benchmark circuit is mapped in two ways. In the first way, we use the original SMAP algorithm with $BF=1$, meaning that we do not combine physical memories in any way. This is the same method used in the experiments in Section 4.2.1. The second way is our proposed method and consists of two steps. In the first step, we use SMAP with a value of $BF>1$ (that is, $N$ physical memories are divided into $N/BF$ logical memories). 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 aspect ratios when the number of packed LUTs can overcome the overhead required for the output multiplexers. In the second step of this method, the support logic for combining the physical memories is added into the netlist.

Both implementations are then fed through VPR and the power model described in Chapter 3, and the power estimates compared. In addition, we compare the number of logic elements packed using each algorithm.
Figure 5-5. Methodology For Power-Efficient Super-Arrays

Table 5-5 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.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Logical Size</th>
<th>$N$</th>
<th></th>
<th>Experiment</th>
<th>Logical Size</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>512</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>Baseline</td>
</tr>
<tr>
<td>BF=2</td>
<td>1024</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>BF=2</td>
</tr>
<tr>
<td>BF=4</td>
<td>2048</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>2</td>
<td>BF=4</td>
</tr>
<tr>
<td>BF=8</td>
<td>4096</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>BF=8</td>
</tr>
</tbody>
</table>

Table 5-5. Summary of Experiments (left: B=512 bits, right: B=4096 bits)

5.2.2 Experimental Results

As discussed earlier, the technique proposed in this section will reduce the number of LUTs that can be removed, which we call the packing efficiency, but it will also reduce the overall power consumption of the circuit. When discussing the experimental results, we will look at these two impacts separately.
5.2.2.1 Packing Efficiency

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 5-6 (for $B=512$) and Table 5-7 (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 have been subtracted from the number of packed LUTs; if the result is negative, a "-" is shown in the table (this means that our version of SMAP was not able to find a solution that reduced the number of LUTs).

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

Table 5-6. LUTs Removed After Mapping (B=512)
As the table shows, 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 $BF>2$.

Although the packing-efficiency is worse for $BF=2$ than $BF=1$ for all circuits, the impact on packing-efficiency for some circuits is less severe than for other circuits. Figure 5-6 and Figure 5-7 show the results graphically. The vertical axis in these graphs is:

$$\frac{(LUTs \text{ removed when } BF > 1)}{(LUTs \text{ removed when } BF = 1)}$$

Therefore, 100% means that the packing efficiency in our new technique is just as good as original SMAP. As the graphs show, for some circuits, the impact on packing efficiency is small.

\textbf{Table 5-7. LUTs Removed After Mapping (B=4096)}
Figure 5-6. Distribution of How the Number of LUTs That Can be Removed Are Affected for 512Bit Memories When BF=2

Figure 5-7. Distribution of How the Number of LUTs That Can be Removed Are Affected for 4096Bit Memories When BF=2
5.2.2.2 Power Efficiency

Figure 5-8 and Figure 5-9 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, 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 do reduce the energy required to implement the benchmark circuits by an average of 19.79% and 32.93% for eight 512 bit and 4096 bit memories respectively when compared to the original SMAP algorithm.

![Figure 5-8. Impact on Energy When Increasing the Number of 512bit Memories](image)
To further understand these results, Table 5-8 breaks the overall energy improvement into logic energy (which is increased, since there are more LUTs required to implement each circuit), routing energy (which is increased for the same reason), and memory energy (which is reduced by 50% since BF=2, meaning one of the two physical arrays in each logical memory can be turned off on each cycle). The numbers in the table are the average percent change when using the power aware technique described in this section, compared to the original SMAP algorithm (in other words, the results from Section 4.2.1). These values are calculated by finding the percent change for each circuit, and then taking the average.
5.2.3 Summary for Power-Efficient Super-Arrays

In this section, we combined multiple physical memories into larger logical memories and mapped 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. We modified SMAP to account for the overhead of the support logic so that it chose mapping solutions that maximized the number of packed LUTs while minimizing the amount of support logic required. We found that when using a blocking factor that was greater two, a mapping solution that reduced the number of LUTs could not be found for many of the benchmark circuits.

When using eight memories and \( BF=2 \), we found an average reduction in overall energy of 19.79\% and 32.93\% for 512bit and 4096bit memories at the cost of a 55.47\% average reduction in the number of LUTs that could be removed. We also found that for some circuits, this penalty was as small as 5\%-20\% meaning that this technique is very good for those circuits.

<table>
<thead>
<tr>
<th></th>
<th>B=512, BF=2</th>
<th>B=4096, BF=2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( N=1 )</td>
<td>( N=2 )</td>
</tr>
<tr>
<td>Logic Energy</td>
<td>1.34</td>
<td>2.80</td>
</tr>
<tr>
<td>Routing Energy</td>
<td>0.70</td>
<td>0.65</td>
</tr>
<tr>
<td>Memory Energy</td>
<td>-50.00</td>
<td>-50.00</td>
</tr>
</tbody>
</table>

Table 5-8. Average Percent Change in Energy When Using BF=2
5.3 Summary

In this Chapter, we explored two modifications to SMAP to try and reduce the power penalty found in Chapter 4. In the first method, we changed the SMAP cost function to a new activity-aware cost function. Our experimental results showed that the impact of using this cost function was smaller than the noise generated by the place and route tools, and therefore conclude that modifying the cost function is not an effective approach. The results are important because they tell us that the only way to significantly reduce the power penalty is by reducing the power dissipated by the memory. The approach in the second method targets the memory power directly by mapping logic to larger logical memories in a way that allows for one or more physical memory to be disabled in each clock cycle. The results from this approach showed that although overall energy could be significantly reduced, the number of LUTs that could be removed from the circuit was also reduced. For some circuits, this reduction in packing efficiency is too severe. But for others, the packing efficiency was reduced by only 5-20% meaning that for some circuits, this technique is a very effective approach for reducing the power penalty when mapping logic to memories.
6 CONCLUSIONS

6.1 Summary of Contributions

In this thesis, we have described a power model for heterogeneous FPGAs that contain embedded memories. This model was implemented in two parts: an activity estimation part and a power estimation part. The activity estimation part was implemented as an enhancement to an existing FPGA activity estimation tool called ACE2.0. The power estimation part was implemented as an enhancement to the Poon Power Model which is incorporated into the VPR framework. This new tool provides the research community the ability to explore the impact of architectural and CAD modifications on power dissipation in heterogeneous FPGAs containing embedded memory blocks.

A second contribution was an investigation into the power implications of using memories configured as ROMs to implement logic. In this study, we showed that implementing logic in FPGA embedded memory arrays leads to an increase in power dissipation of the resulting circuit. Previous studies reported significant density increases when embedded memory is used in this way, but the results of this thesis show that this technique may be undesirable when power is a concern. In the situation where the density improvements of this technique warrants its use, we found that optimizing the
size and flexibility of the memory blocks to reduce this power penalty is important. We showed that for most array sizes, the arrays should be flexible but that increasing the pin count of the memory increases routing power regardless of whether the memories are used. We also showed that smaller memory arrays are more power efficient than large arrays, but when larger arrays are desired to increase density, memories with non-integral square roots and a larger depth-to-width ratio should be used.

To see whether we could reduce the power penalties of mapping logic to memories at the CAD level, we employed two techniques. The first technique was to make our heterogeneous technology mapping tool power-aware by changing its objective function. The new objective function attempts to minimize the number of high activity connections in the mapped solution. In the second method, we used a power-efficient technique to combine physical memories into larger logical memories, and mapped logic to these larger logical memories. In this method, some of the physical memories could be turned off to save power. The first method did not show any significant power savings. Any expected power savings would have been small since only a very small portion of the circuit is mapped to memories. Experiments suggested that the only way to make mapping logic to memories more power efficient was to reduce the power dissipated by the memories. In the second method, we aimed to do just that by using a power-efficient logical memory partitioning arrangement that allowed us to disable one or more of the memories in each access. The results showed that combining more than two memories together incurred too much support logic overhead. But when we combined two memories together, we found significant overall energy reductions. Although the
packing efficiency was still reduced, we found that for some circuits, this penalty was not very severe, meaning that this technique can be quite effective at reducing the power dissipation of some circuits implemented in LUTs and ROMs.

6.2 Future Work

This thesis focuses on two areas: the power modeling of embedded memories in FPGAs, and the power implications of mapping logic to memories using heterogeneous technology mapping algorithms. Suggestions for future research in both areas are discussed below.

6.2.1 Power Model

In modern FPGAs, the embedded memory blocks often have additional features that are not included in our power model. Signals such as byte-enables and asynchronous clears will affect the activities estimated at the outputs of the memories. Dedicated tracks and programmable connections are usually available to facilitate efficient implementation of larger logical memories and FIFOs. If these connections are required, our model assumes that they are routed through the general purpose interconnect. Many commercial FPGA embedded memory blocks also have dual-port RAM memory modes which isn’t currently supported by our power model. Although these components were not used in our studies, they need to be correctly modeled to further enable architectural and CAD studies.

6.2.2 Heterogeneous Technology Mapping

As the application domain of FPGAs increases, the memory resources on chip will expand to meet the needs of user circuits with larger storage requirements. The trend for
vendors has been to increase the size of their embedded memory blocks. The techniques employed by existing heterogeneous technology mapping algorithms were developed for relatively small memory arrays. These techniques may lose their efficiency when applied to large memory arrays and new techniques may need to be developed.

It is also unclear as to how the power consumption of logic implemented in traditional logic resources will scale compared to the power consumption of memories in the future. As we showed in a theoretical 0.18um FPGA and a real 0.13um FPGA, the power consumed by the memories is larger. But in future FPGAs this may change; more power efficient memory design, more exotic LE structures (such as the Altera StratixII ALUT), increasing interconnect power are all factors that may tip the scale.

In our technique that employed a power-efficient logical-to-physical memory mapping, we found that although power consumption of the circuit was reduced, the net change in packing efficiency was severely affected due to the need to insert support logic to combine the physical memories into the larger logical memories. Since this method of combining memories has been shown to be good for RAM applications [65] and our ROM applications, it would be interesting to explore a memory architecture where dedicated connections for combining memories in this fashion are available.
REFERENCES


