Leakage Power Reduction of Embedded Memories on FPGAs Through Location Assignment

Yan Meng Timothy Sherwood Ryan Kastner
University of California, Santa Barbara
Santa Barbara, CA 93106-9560
{yanmeng,kastner}@ece.ucsb.edu; sherwood@cs.ucsb.edu

ABSTRACT

Transistor leakage is poised to become the dominant source of power dissipation in digital systems, and reconfigurable devices are not immune to this problem. Modern FPGAs already have a significant amount of memory on the die, and with each generation the proportion of embedded memory to logic cells is growing. While assigning high V_{th} can limit the leakage power, embedded memory timing is critical to performance and will draw an increasingly significant amount of leakage current. However, unlike in many processor based systems, on-chip memory accesses are often fully deterministic and completely under the control of the scheduler. In this paper we explore a variety of techniques to battle the problem of leakage in FPGA embedded memories that range in complexity and effectiveness. Through the addition of sleep and drowsy modes, controlled by the scheduler, the amount of leakage power can be reduced by several orders of magnitude. We show how even very simple schemes offer large amounts of benefit, and that further reductions are possible through careful leakage-aware data placement.

Categories and Subject Descriptors

B.3.0 [**MEMORY STRUCTURES**]: General; J.6 [**Computer-Aided Engineering**]: Computer Aided Design

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

Embedded memory, leakage power, location assignment

1. INTRODUCTION

Transistor leakage is a growing problem in reconfigurable devices and will soon become the dominant source of power dissipation. FPGAs are an attractive option when implementing a variety of applications due to their high processing power, flexibility and non recurring engineering (NRE) cost. While there is some preliminary work on leakage power reduction in FPGAs, tackling the leakage problem requires solutions that consider the growing die area consumed by embedded memories, a problem which so far has been left unaddressed. In this paper, we argue that leakage in embedded memories will be of growing importance, and we propose a leakage-aware design flow with five power saving schemes to initiate the exploration.

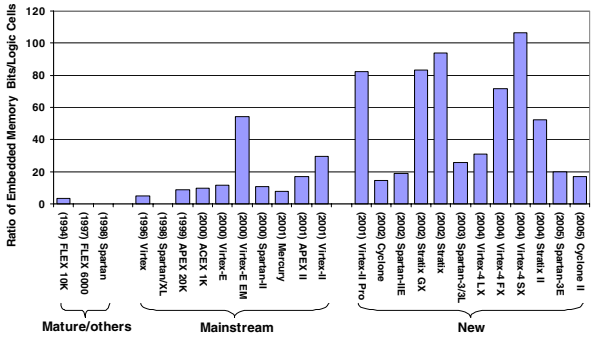

Figure 1: Ratio of embedded memory bits/logic cells on modern FPGAs. The number in the parentheses shows the release year of the device. New devices have 20 to 100 times more embedded memory bits than logic cells.

To justify the importance of this research area, we collected information on all Xilinx and Altera FPGA devices [1, 2] over the past 10+ years and grouped them into three categories — mature, mainstream, and new. Figure 1 plots the ratio of embedded memory bits to logic cells of the largest FPGA[1] for each family of devices. It clearly illustrates the growing importance of embedded memory as newer devices have increasingly larger amounts of embedded memory. For example, there are over 100 times more embedded memory bits in Virtex-4 SX than logic cells. This points to a pressing need for optimizations that target embedded memories of current and future generations of FPGA architectures.

As FPGA manufacturers move to advanced technology nodes[2], there are significant increases in leakage current due to the technology scaling of supplied voltage (V_{dd}), threshold voltage (V_{th}), channel length, and gate oxide thickness [10,

[1]The largest means that the chip has the largest number of logic cells, or logic elements, with each logic cell containing a 4-input LUT and a D-type flip-flop.

[2]90nm FPGAs are in production and 65nm is on the horizon.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

22]. These changes are making leakage power the dominant component of total power consumption, and new techniques are needed to address the leakage power concerns of FPGAs.

While dynamic power is dissipated only when transistors are switching, leakage power is consumed even if transistors are idle. Therefore, leakage power is proportional to the number of transistors [10]. An effective method in reducing leakage power is to put transistors into low power states. Since embedded memory blocks occupy an increasingly large area they are an ideal target for reducing overall power.

A number of low-leakage circuit techniques [13, 22] have been proposed that save power by putting memory bits into lower power states. Sleep transistors can be employed to shut off the power supply to the circuit and to put transistors into a *sleep* mode. While efficient in saving power, sleep mode does not retain data, and there is a large penalty to restore the data if it needs to be reaccessed [8]. Dual/multi-V_{dd} and dual/multi-V_{th} are other popular techniques that can be effectively used to limit dynamic power and to reduce leakage power. In these *drowsy* [10] schemes, data is preserved at a lower supply voltage and a small wakeup time is required to change supply voltage from low to high, which is necessary to access the data. Since drowsy mode does not fully turn off transistors, it does not reduce leakage power as much as sleep mode but preserves data.

In memory leakage power optimization, the above-illustrated techniques have been employed mainly in caches of microprocessors [8, 10]. Our research is specifically focused on studying leakage reduction control methods of FPGA embedded memories. While the central idea behind all leakage power saving techniques is to exploit temporal information to control the supply voltage of regions of memory, embedded memories have many fundamental differences from caches. First, FPGAs memory accesses are usually *statically scheduled* and cannot easily handle the variable latencies associated with the predictive methods used by processor caches. Second, the data in embedded memories are usually *placed statically* as opposed to the dynamic reshuffling that caches try to do. Finally, embedded memories are not necessarily part of an memory hierarchy with inclusion, and thus more care must be taken not to lose important data.

In this paper, we explore embedded-memory leakage power optimization in FPGAs and present an embedded memory leakage-aware design flow. We further propose a spectrum of leakage power management schemes for embedded memories. These schemes extract sleep and drowsy schedules from scheduled memory accesses and further reduce power through careful temporal control of, and data placement in, a given RAM. Through experimental evaluation of the schemes, we found that by simply turning off unused memory entries, 36.7% of the leakage power can be saved, while by carefully placing data in a leakage-aware manner, 94.7% of the memory leakage power can be eliminated.

The rest of the paper is organized as follows. We formulate the leakage power problem of embedded memories in Section 2. In Section 3, we propose different schemes for reducing leakage power. We report our experimental results in Section 4. After reviewing related work in Section 5, we draw our conclusions in Section 6.

2. PROBLEM FORMULATION

Considering that the embedded memory leakage problem is very important, and we are unaware of any currently available design flow that takes into account the location of tvariables within memory to optimize leakage power, our main contribution is the introduction of two components, *path-traversal* and *location assignment* into the design flow (Figure 2) to achieve the minimal leakage power consumption of embedded memory. In our flow, the intermediate representation (e.g., $CDFG$) of an application is first scheduled and its memory accesses intervals are then recorded through the path-traversal component to build an acyclic interval graph [16]. The interval graph, as exemplified by a real world example, radix-2 fft (fft-2), in Figure 3, consists of the temporal relationship of live and dead time of all memory access intervals, with each vertex representing a live interval and each edge representing a dead interval. The location assignment component is added to figure out the best power saving mode on each interval as well as the best placement of the variables within the memory in order to achieve the minimal leakage power consumption.

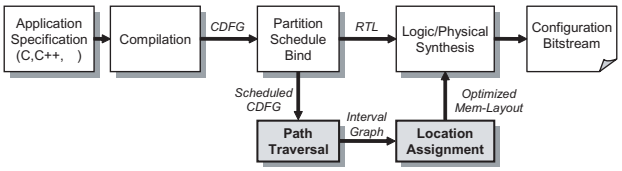

Figure 2: Design flow for leakage power reduction of embedded memory on FPGAs. Path traversal and location assignment are introduced components for deciding the best data layout within embedded memory to achieve the maximal power saving.

If an embedded memory has been configured based on the requirement of the bit-width, the number of memory entries, denoted as N, is known. Through traversing the scheduled intermediate representation of an application, a set of memory access intervals I ($|I| = n$) with precedence orders can be derived. Then, the memory leakage power optimizing problem can be formulated as the following.

Problem: Given a memory with N finite number of memory entries, and a set of memory access intervals I with temporal precedence orders, find the best layout of the variables within the memory so that the maximal leakage power saving can be achieved.

In our study, the leakage power saving problem of variables assigned in the bounded size (N) embedded memory is modeled by an Extended Directed Acyclic Graph (Extended DAG) $G(V, E)$, where V is a set of finite v ($v \in \{v_s, v_1, \ldots, v_n, v_e\}$) vertices and E is a set of finite e directed edges. A vertex v ($v \in V \backslash \{v_s, v_e\}$) in the DAG indicates that the variable v is in the embedded memory, and the weight on the vertex v shows the leakage power saving during the live time of the variable, which is denoted by $w(v_i)$. And edge, denoted as e_{ij}, represents the precedence order between two vertices v_i and v_j. Associated with the edge is a nonnegative weight $w(e_{ij})$ (the weight of an edge may be zeroed when the two incident vertices are in the same memory location), showing the leakage power saving during the time difference between assigning the two vertices into the memory, or the dead time of the vertex v_i. The number of edges is denoted by e. The source vertex of an edge is called the parent vertex while the sink vertex is called the child vertex. A vertex with no parent is called a starting vertex v_s, and a vertex with no child is called an ending

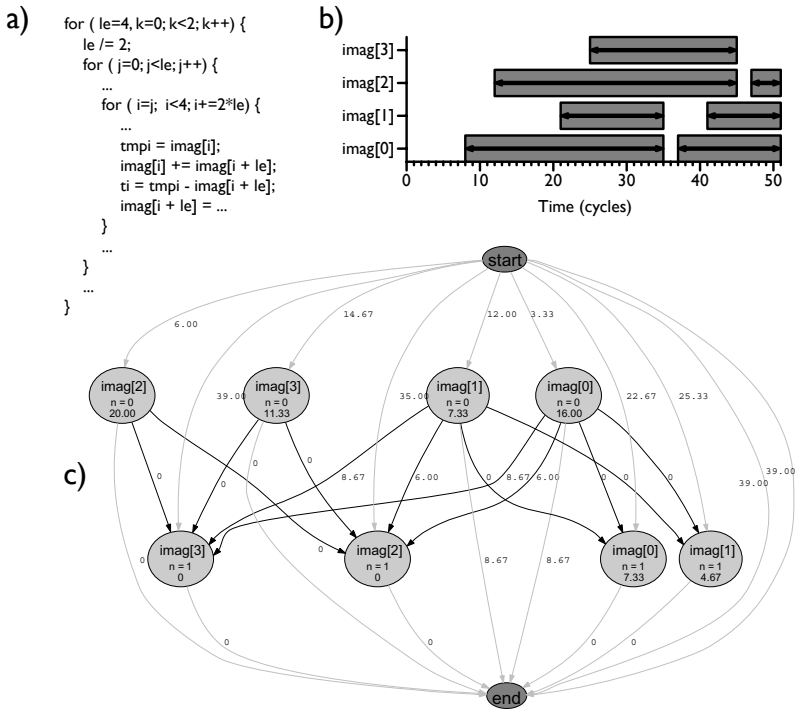

Figure 3: Problem formulation illustrated with the radix-2 fft example. The radix-2 fft example (a) is scheduled to extract memory access intervals (b), and an Extended DAG model is built by assigning all the intervals to $N = 10$ entries(c). The live intervals are indicated by the gray rectangles in (b) and the gray vertices in (c), and the dead intervals are depicted by the white space in (b) and edges in (c). A vertex includes the information of a variable name, its access number n and power saving. An edge shows the precedence order and the power saving between the adjacent vertices. The length of a path i, defined as the sum of all the weights on the vertices and edges along the path, indicates the leakage power saving of memory entry i.

vertex v_e. There is an edge from the starting vertex v_s to every vertex in $V\backslash\{v_s, v_e\}$, and similarly, there is an edge from the vertex v_i in $V\backslash\{v_s, v_e\}$ to the ending vertex v_e. The unused memory spaces, where no variables are assigned into, are represented as edges from the starting vertex v_s to the ending vertex v_e. The length of a path i is the sum of all the weights on the vertices and edges along the path, which shows the power saving in memory entry i.

Then, the memory leakage power problem is to assign n variables to N memory locations so that the maximal leakage power saving can be achieved by covering the n nodes $V\backslash\{v_s, v_e\}$ with N node-disjoint paths such that every node in $V\backslash\{v_s, v_e\}$ is included in exactly one path. Each path starts from the starting node and ends at the ending node.

According to the definition, the Extended DAG has the following properties:

Property 1. After path covering, the in-degree and the out-degree of the vertex v_i ($v_i \in V\backslash\{v_s, v_e\}$) are both equal to 1 to ensure that the paths have no duplicated vertices and edges assigned to the same entry.

Property 2. The number of edges from the starting vertex v_s to the ending vertex v_e is equal to $N - k$, where k is the number of paths that cover all the n vertices $\{v_1, \ldots, v_n\}$ and the corresponding edges.

The key to discovering the maximal leakage power saving is to choose the best operating mode on each interval, either active, drowsy, or sleep mode. This can be fulfilled by classifying an interval into one of the three categories: if an interval is very long then it would be beneficial to put that entry in sleep mode for the duration of that interval; if an interval is very short, it should be simply put into the active mode and powered with high-V_{dd} mode; if an interval is somewhere in the middle, the drowsy mode would be the best. For live intervals, only the active or drowsy operating modes are allowed. It is because that the sleep mode does not preserves data and once the data is lost, the system will have to go to off-chip memory to refetch the data back, which is too costly to get any gain in terms of power.

To classify intervals into those three categories, two inflection points are introduced in our study: the active-drowsy inflection point and the drowsy-sleep inflection point. Inflection points are defined as the interval length where the operating mode changes. The active-drowsy inflection point is the point between active and drowsy modes. It can be calculated as the sum of the durations within which the voltage changes either from V_{dd} to $V_{dd_{low}}$ or from $V_{dd_{low}}$ to V_{dd}. The drowsy-sleep inflection point is the point between drowsy and sleep modes. It is derived as the access interval length when the sleep and the drowsy modes consume the same amount of energy. If the interval is of a length less than the inflection point then drowsy mode would be optimal. If it is greater than the inflection point then sleep mode would be optimal. It has been proved that with perfect knowledge of the lengths of all intervals, the optimal leakage power saving can be achieved by applying the proper operating mode on each interval [17, 20].

3. EMBEDDED MEMORY LEAKAGE POWER REDUCTION SCHEMES

In this section, we will explore different leakage reduction schemes step-by-step to understand how the maximal leakage power saving can be achieved through carefully assigning the variables into memory entries. We start with keeping every entry active as our baseline, and move forward to the schemes that have different data layouts (see Figure 4).

Full-active. It assigns one variable per memory entry. All memory entries are kept active, and there is no leakage power saving.

Used-active. Similar to full-active, it assigns one variable per memory entry and powers on the memory entries that are used. But it turns off the rest of the entries that are not used. The power saving is the percentage of entries that are unused.

Min-entry. It assigns all variables to the minimal number of memory entries. Those entries that have been used are powered on and the rest of the unused entries are turned off. The power saving is also the percentage of the entries that are unused.

Sleep-dead. Similar to min-entry, it uses the minimal number of entries. But it also has power savings on the intervals that are dead. So the total power saving consists of two parts: savings in unused entries and savings in dead intervals of the used entries.

Drowsy-long. Similar to Sleep-dead, it uses the minimal number of entries and saves power on the dead intervals. But it also saves power in live intervals using the drowsy technique. So the total power saving consists of three parts: savings in unused entries, savings in dead intervals, and savings in the live intervals of the used entries.

Path-place. Different from the above schemes that use the least number of entries, path-place picks the N path-covers that can lead to the maximal power saving.

Figure 4 illustrates the above-mentioned schemes. From the figure, we can see that when the precedence orders of all the live and dead intervals are taken into account, different data layouts result in different power savings.

The path-place algorithm (Table 1) is a greedy approach that can find the N paths to achieve the maximal leakage power saving. It works by first sorting all the vertices in a topological order. Then a vertex v_i ($v_i \in V \backslash \{v_s, v_e\}$) is picked each time in the sorted list to calculate the maximal power saving from the starting vertex v_s up to v_i, or simply the length of the longest path reaching it. Note that the edges from the starting vertex v_s to the ending vertex v_e are the edges with the lowest priority to pick. In the end, the total power saving is computed as the sum of three components: the weights of all the final level vertices that have no child except the ending vertex v_e, the weights of their edges that connect to v_e, and the weights of the $(N-k)$ edges from the starting vertex v_s to the ending vertex v_e if k is less than N. The path(v_i) function is used to calculate the path ID of the vertex v_i. Each time it sets the path ID of the vertex v_i as the path ID of its parent that can lead to the largest power saving of the vertex v_i. The complexity of the algorithm is $O((n+e)*N)$.

While employing leakage control techniques at the entry level of embedded memory may cause the controller overhead, it decreases the cooling cost in package and increases circuit reliability [22]. Moreover, adding the components of path-traversal and location assignment does not affect current design flows for placement and routing in any way. It only gains additionally leakage power saving on embedded memory, which is a dominant portion on FPGAs.

```
ALGORITHM PATH_PLACE
      Input(G, N)
      Output(totalSaving, path)
      //G: the Extended DAG; N: the number of entries
      //path: the path for each vertex
Begin
1     Construct a list of all vertices V in topological order,
      call it Toplist
2     for each vertex v_i ∈ V\{v_s, v_e} in Toplist do
3       max = 0
4       for each parent v_p ∈ V of v_i do
5         if ( saving_level(v_p) + w(v_i) + w(e_pi) > max) then
6           max = saving_level(v_p) + w(v_i) + w(e_pi)
7           id = path(v_p)
8         endif
9       endfor
10      path(v_i) = id
11      saving_level(v_i) = max
12    endfor
13    totalSaving = 0
15    for each parent v_p ∈ V of v_e do
16      totalSaving += saving_level(v_p) + w(e_pe)
17    endfor
End
```

Table 1: The path-place algorithm.

4. EXPERIMENTS

In section 3, we have discussed different schemes for reducing leakage power of embedded memory. In this section, we report our experimental results gathered from several DSP applications [5]: *dft, idft, radix-2 fft (fft-2), radix-4 fft (fft-4), filter*, and *mp*, a real design for efficient wireless channel estimation [19]. We first derive inflection points for different configurations of the memory block. We then show the comparison results of applying different schemes on different applications.

Due to the lack of the detailed information of commercial embedded memories, in our study we use configuration schemes similar to dedicated blocks of on-chip memory, Block SelectRAM [2], of Xilinx Virtex family devices. That is to say, our targeted embedded memory is a true dual-read/write port synchronous RAM with 18Kb memory bits. Each port can be independently configured as a read/write port, a read port, or a write port. Each port can also be configured to have different bit-widths: 1 bit, 2 bits, 4 bits, 9 bits (including 1 parity bit), 18 bits (including 2 parity bits), and 36 bits (including 4 parity bits). A read or a write operation requires only one clock edge. Both ports can read the same memory cell simultaneously, but can not write to the same memory cell at the same time. Therefore, there is no write conflict. In our experiments, the bit-width of each entry is set to be 18 bits, which is reasonable in those DSP applications. So the number of entries N is equal to 1K.

4.1 Deriving Inflection Points

The active-drowsy and drowsy-sleep inflection points are used to categorize all the live and dead access intervals. They are also used to select the best operating mode on each interval. In our study, we use the parameters in [20]

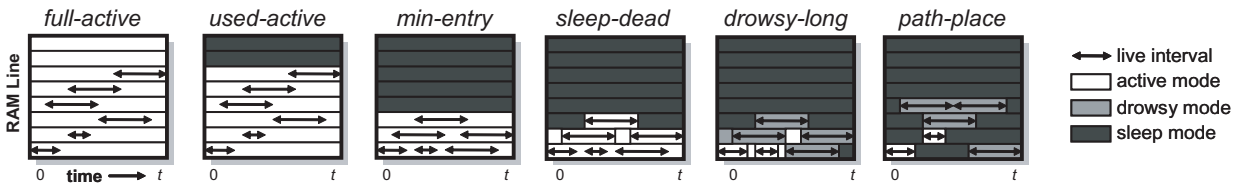

Figure 4: Different schemes to save leakage power of embedded memories on FPGAs. Full-active and used-active has one variable per entry. Min-entry, sleep-dead, and drowsy-long use the minimal number of entries, and apply power saving modes on unused entries, dead, and live intervals incrementally. Path-place layouts variables with leakage awareness, and uses power savings on all unused entries, dead, and live intervals.

to calculate inflection points, and assume that 3 cycles is needed to change the supply voltage from high to low and vice versa, and 30 cycles from high to off, and 3 cycles from off to high. So the active-drowsy inflection point can be calculated as 6 cycles. When calculating the drowsy-sleep inflection point, we simulated our target memory using modified eCACTI [18] to get both dynamic power and leakage power consumptions, and derived the point where drowsy and sleep modes consume the same amount of energy [20].

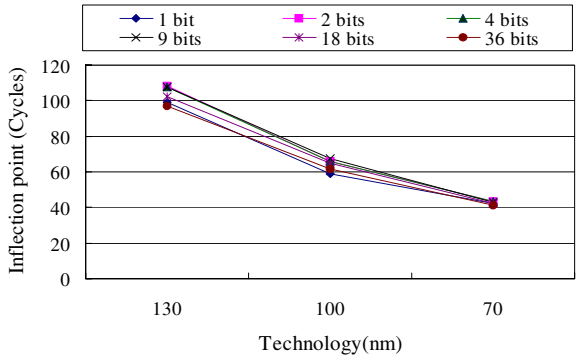

Figure 5: The drowsy-sleep inflection points are derived for different bit-width configurations of the embedded memory.

Figure 5 shows the inflection points for different configurations under different technologies. From the figure, we can see that under the same technology, drowsy-sleep inflection points for different configurations are the same; and when the technology scales down from 130nm to 70nm, the drowsy-sleep inflection point decreases from 102 to 43 cycles. Since 70nm is the most advanced technology available in eCACTI, and features the next generation FPGA design technology, we used the 70nm technology and picked 43 cycles as the drowsy-sleep inflection point in our study. Note that we also varied the drowsy-sleep inflection point from 43 to 640 cycles, and found the total leakage power savings are about the same. It is because that intervals that contribute to most of the saving are very long, and small changes of the drowsy-sleep inflection point will not limit the power saving from those long intervals.

4.2 Comparing Different Schemes

We have proposed five different schemes to reduce memory leakage power: used-active, min-entry, sleep-dead, drowsy-long and path-place. We now study the power savings of the five schemes are on DSP applications. To assign the variables to the minimal number of entries (for min-entry, sleep-dead, and drowsy-long), the left-edge algorithm [11] was implemented in our experiments, which colors the graph in $O(nlog(n))$ time. To evaluate the different schemes we compared the five schemes against the full-active scheme, which is used as the baseline and has no power saving. Figure 6 shows the comparison results for all the applications. From the figure, we can make the following observations:

(1). By simply putting a single interval per entry and turning off the rest, as used-active does, 36.7% of leakage power can be saved on-average.

(2). If the minimal number of entries are used, the leakage power savings is 75.6%, 77.3% and 86.0% for min-entry, sleep-dead, and drowsy-long, respectively, and the savings are increasing because more intervals are put into power saving modes. The reason that min-entry does well is that it packs the data very tightly (see Figure 4), and more entries that could be completely turned off to save power are left unused. Moreover, due to the effect of compact packing, the dead intervals that are in the used entries are very short, which leaves very little space for sleep-dead to save power. Consequently, sleep dead performs at a similar level in efficiency as min-entry.

(3). Among all, path-place achieves the best leakage power saving, 94.7%, which is about 8.7% better than the drowsy-long scheme. It is because path-place layouts the data in a way that the sleep mode can be explored to the largest extent on all the intervals, and among all three operating modes: active, drowsy, and sleep, the sleep mode can provide the maximal power saving.

(4). In terms of best schemes, both min-entry and path-place are favorable. Min-entry is very simple but effective. It only needs to use sleep techniques to turn off the unused entries after interval packing and can achieve a good amount of power saving. By contrast, path-place is very effective but a bit more costly in terms of running time to discover the best layout. If drowsy techniques are incorporated along with sleep techniques, path-place could accomplish the best leakage power saving.

(5). For *filter*, the simple used-active scheme does not save too much power. It is because that different from other applications, the number of its variables is close to the total number of entries, and only few entries that are unused can be put into sleep to save power. But when the variables are layout carefully, significant power saving can be achieved, as path-place does.

These provide us the answer that the layout of the data within memory entries has a significant impact on the leakage power optimization. Moreover, with available circuit techniques, careful placement of intervals within memory can reduce leakage power by a large magnitude.

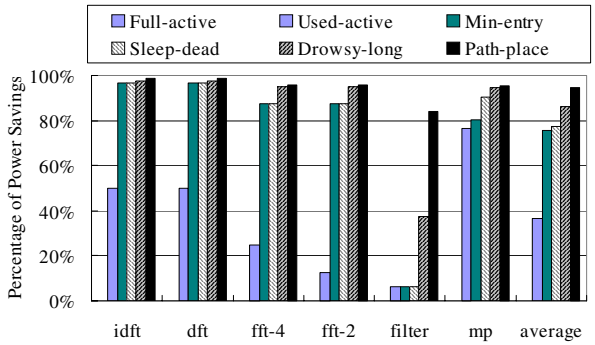

Figure 6: Comparison of the leakage power savings for different schemes.

5. RELATED WORK

We have reviewed low-leakage circuit techniques in Section 1 for optimizing leakage power of ASICs and microprocessors [6, 7, 8, 9, 10, 12, 22]. Now we see what are the different techniques that have been proposed to reduce leakage power for FPGAs, which has been in focus only recently [4, 3, 15, 23, 24]. Shang et al. [23] analyzed dynamic power consumption in Virtex-II FPGA family based on measurement and simulation. Tuan and Lai studied the leakage power of Xilinx architecture, and Li et al [14] proposed fpgaEVA-LP for power efficiency analysis of LUT based FPGA architectures. Several techniques for reducing leakage power on FPGAs have been proposed. Gayasen et al. [3] studied region constraint placement to disable unused portions by employing sleep transistors. Anderson et al. [4] considered selecting polarities for signals at the inputs of LUTs so they spend the majority of time in low leakage states. Li et al. [15] proposed to use pre-defined dual-V_{dd} and dual-V_t fabrics to reduce FPGA power. Rahman et al. [21] evaluated the trade-offs of different low-leakage design techniques for FPGAs. While there has been work in low power FPGAs and other work in architectural-level policies for controlling memory leakage, we believe this to be the first paper to address embedded memory leakage power in FPGAs.

6. CONCLUSIONS

In this paper we argue that embedded memory leakage power will be a large and growing concern for FPGAs and that design flows can be effective in reducing this power. We further present a leakage-aware design flow and proposed five schemes for reducing leakage power of embedded memory on FPGAs. The new flow takes into account the leakage-aware location assignment of variables within memory. The five proposed schemes employ sleep and drowsy techniques, and exploit the live and dead interval information of memory accesses to save power. They function by choosing the best operating mode, active, drowsy or sleep, on each interval. Through the experimental evaluation, we found that the simple scheme like used-active can provide a good amount of benefits, and by carefully placing data into memory entries, a significant amount of leakage power saving can be further achieved.

7. ACKNOWLEDGMENT

This work was supported by National Science Foundation Grants CNS-0411321 and CNS-0524771.

8. REFERENCES

[1] Altera press releases and device data sheets. http://www.altera.com.

[2] Xilinx press releases and device data sheets. http://www.xilinx.com.

[3] A.Gayasen, Y.Tsai, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, and T. Tuan. Reducing leakage energy in fpgas using region-constrained placement. In *FPGA*, 2004.

[4] J.H. Anderson, F.N. Najm, and T. Tuan. Active leakage power optimization for fpgas. In *FPGA*, Monterey, CA, 2004.

[5] P. M. Embree and B. Kimble. *C Language Algorithms for Digital Signal Processing*. Prentice Hall, Englewood Cliffs, 1991.

[6] P. Gupta, A. B. Kahng, P. Sharma, and D. Sylvester. Selective gate-length biasing for cost-effective runtime leakage control. In *DAC*, 2004.

[7] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu. Banked scratch-pad memory management for reducing leakage energy consumption. In *ICCAD*, San Jose, CA, 2004.

[8] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *the 28th ISCA*, Gőteborg, Sweden, June 2001.

[9] K. S. Khouri and N. K. Jha. Leakage power analysis and reduction during behavioral synthesis. *IEEE Trans. on VLSI*, 10(6), Dec. 2002.

[10] N. Kim, K. Flautner, D. Blaauw, and T. Mudge. Circuit and microarchitectural techniques for reducing cache leakage power. *IEEE Trans. VLSI*, 12(2):167–184, Feb. 2004.

[11] F. J. Kurdahi and A. C. Parker. Real: A program for register allocation. In *DAC*, 1987.

[12] E. Kusse and J. Rabaey. Low-energy embedded fpga structures. In *ISLPED*, 1998.

[13] D. Lee, D. Blaauw, and D. Sylvester. Gate oxide leakage current analysis and reduction for VLSI circuits. *IEEE Trans. on VLSI*, 12(2), Feb. 2004.

[14] F. Li and L. He. Power modeling and characteristics of field programmable gate arrays. *IEEE Trans. on Computer-aided design*, 24(11):1712–1724, Nov. 2005.

[15] F. Li, Y. Lin, L. He, and J. Cong. Low-power fpga using pre-defined dual-vdd/dual-vt fabrics. In *FPGA*, Monterey, CA, 2004.

[16] Y. D. Liang and G. K. Manacher. An $O(nlogn)$ algorithm for finding a minimal path cover in circular-arc graph. In *ACM Conference on Computer Science*, pages 390–397, 1993.

[17] J. Liu and P. Chou. Optimizing mode transition sequences in idle intervals for component-level and system-level energy minimization. In *ICCAD*, 2004.

[18] M. Mamidipaka and N. Dutt. ecacti: An enhanced power estimation model for on-chip caches. Technical Report Tech. Report TR-04-28, UC. Irvine, Sept. 2004.

[19] Y. Meng, A. Brown, R. Iltis, T. Sherwood, H. Lee, and R. Kastner. Mp core: Algorithm and design techniques for efficient channel estimation in wireless applications. In *DAC*, 2005.

[20] Y. Meng, T. Sherwood, and R. Kastner. On the limits of leakage power reduction in caches. In *HPCA*, 2005.

[21] A. Rahman and V. Polavarapuv. Evaluation of low-leakage design techniques for field programmable gate arrays. In *FPGA*, 2004.

[22] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE*, 91(2), Feb. 2003.

[23] L. Shang, A. S. Kaviani, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. In *FPGA*, 2002.

[24] T. Tuan and B. Lai. Leakage power analysis of a 90nm FPGA. In *CICC*, 2003.