Modeling and Automating Selection of Guarding Techniques for Datapath Elements

William E Dougherty and Donald E Thomas
Center for Electronic Design Automation
Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
(412) 268-{2476, 3545}

{wed, thomas}@ece.cmu.edu

ABSTRACT

While guarded evaluation has proven an effective energy saving technique in arithmetic circuits, good methodologies do not exist for determining when and how to guard for maximal savings. Three new internal guarding techniques are presented in adders that increase energy savings up to 38% over existing external guarding techniques. This allows guarded evaluation to be effective at duty cycles as much as 20% higher than are currently practical. A modeling methodology is presented defining the energy and energy delay of a unit in a generic application space. These models can easily be incorporated into an automated selection technique to determine the optimal guarded implementation. This technique is tested on a DSP ASIP, increasing overall energy savings by preventing unnecessary guarding. The data is generalized and it is observed that guarding is most beneficial when the ratio of guarding transistors to driven computational transistors is $^1/_{10}$ or lower.

Keywords

Guarded evaluation, low power design, datapath energy modeling.

1. INTRODUCTION

Guarded evaluation has proven to be a viable technique for the reduction of power consumption in arithmetic circuits, but no solid methodology exists to determine where and how to guard functional units to maximize energy savings. Previous work focused in two distinct directions. The first, dealing with large blocks of synthesized logic, focused on synthesizing guarding or precomputation logic in a way that minimizes the overhead of the technique[9]. The second, dealing more with arithmetic datapath blocks, has dealt with the insertion of guarded functionality as an afterthought to the design process[2][8]. Guards are inserted in front of idly transitioning components, meaning the full overhead of the guard and controlling logic must be incurred. This work seeks to marry both approaches by incorporating guarding functionality in datapath library components.

By incorporating guarding logic into the functional unit, it becomes possible to create more compact designs while reducing the overhead of guarding itself. Increasing energy savings allows guarded units to be used in situations where existing techniques prove ineffective. A methodology is developed to determine the best guarding technique in a generic application space defined by active and idle behaviors of a given functional unit. This methodology suggests an automated selection process, removing the where and how guesswork from the guard insertion process. Results of this modeling and selection process are measured in a DSP ASIP. While previous work has looked at determining when it is worthwhile to use guarding or precomputation in large blocks of random logic[9], we have seen no work in this area specifically targeting major datapath elements.

Finally, the data is generalized to other functional units by examining the tradeoffs between the cost of guarding and the combinational depth of the functional logic.

2. GUARDING METHODOLOGY

Two techniques for guarding functional units in datapaths are the insertion of latches or gates with controlling values (AND, NOR, etc.) onto the input wires. Latches provide a way to freeze current input values, preventing any spurious transitions from occurring. Gates allow the input values to be forced and held at some level, but one extra computation takes place as the inputs settle to the controlled value. In spite of the extra calculation, gates can be desirable in some cases, as their area and capacitance overhead is lower than a latch.

While latches and gates are typically placed externally to a functional unit, it is possible to use existing arithmetic circuitry to implement part of the guard functionality. This work devised three internal guard mechanisms, and built them into existing library cell adders.

The first method, called *internal latch*, locks the value on the adder output and severs all Vdd to GND paths inside the adder. The second method, called *float*, severs the internal Vdd to GND connections and uses gate capacitance to hold the outputs at the present value. The final method, called *internal gate*, forces the outputs to 0 and severs all internal Vdd to GND connections.

Each of the methods uses two techniques to reduce design power. First, holding the outputs at a constant value prevents downstream dynamic power consumption. Downstream logic is unable to tell

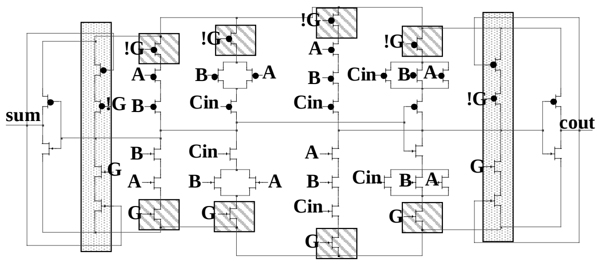

Figure 1. Internally guarded RCA. Dotted boxes show cutoff transistors of the *float* method. Adding transistors in dashed boxes as well creates the *internal latch* method.

if the unit is guarded internally or externally. The second part is the severing of Vdd to GND connections. While externally applied guarding techniques freeze the input logic values, this is impractical for internal techniques. Since input wires fan out to a large number of transistors, it becomes difficult to incorporate those same transistors in holding their gates at any particular level. Using pull up or pull down transistors to hold the inputs at a given logic level can be excluded, as the static current contribution is likely to outweigh the savings in reduced transitioning.

The most efficient solution appeared to be the removal of the Vdd to GND paths of transistors not used in the guarded state of the adder. This removes the dynamic power contribution of transistors and wires inside the guarded unit, although the gate capacitance of transistors controlled directly by the input wires remains unchanged.

2.1 Internal Guarding in a RCA

A ripple carry adder (RCA) can be used to demonstrate the changes made to basic computational logic by adding internal guarding techniques. This example is based on a 1-bit RCA as implemented in Cascade's hp14b library[4], which is based on a .6µm process from Hewlett Packard.

Figure 1 shows how the existing logic in a 1-bit RCA can be used to implement guarded functionality for both the *internal latch* and *float* techniques. The *float* method is implemented by adding cutoff transistors, shown in the figure with a striped gray background, to the existing logic. These transistors are controlled by guard and guard-bar signals, and prevent charge leakage on the wires that drive the two output inverters. Charge leakage may become a problem when using this method and this can severely effect energy savings.

To create the *internal latch* adder, all of the cutoffs are inserted as well as the transistors shown with the solid gray backgrounds. The cutoffs serve the same purpose as in the *float* implementation, while the remaining transistors form a latch to hold the outputs at the present level. Note that the output inverters form half of the latches' cross-coupled inverters.

The equivalent functionality implemented with external latches requires the addition of three latches on the first bit and two on every bit thereafter. The area cost of using one of these two internal techniques will be lower than the external latching. Depending on the implementation, all of the cut off transistors can be merged into single NMOS and PMOS transistors. The cutoff will effect the delay of the circuit, but the impact should be small and manageable through proper transistor sizing. The area and delay penalties for our RCA examples are shown in Table 1, but it should be noted that minimal effort was put forth optimizing either of

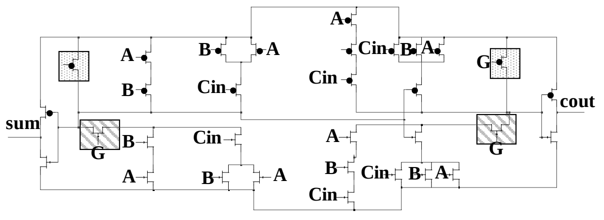

Figure 2. *Internal gate* guarded RCA. Transistors in the dotted boxes force the outputs to 0 while those in the dashed circles cut off internal GND paths.

these aspects in the internally guarded designs. The guarded functionality was inserted into the original library parts with no functional redesign or resizing of transistors. Even with this minimal redesign effort, the area penalties for the internal methods are significantly lower than the corresponding external implementations. Because the transistors were not resized however, the delay penalty is greater.

Figure 2 shows the implementation of the *internal gate* method. The transistors with the solid gray background force the outputs to 0, while the transistors with the striped gray background prevent charge from flowing to GND. The equivalent implementation using external gates requires the addition of AND gates on the inputs, three on the first bit and two on each bit thereafter. The impact on speed and area will be smaller here than in the previous two methods.

While this same functionality could be implemented with NAND gates, this would cause the adder inputs and outputs to go to all 1s, and would likely cause a larger number of transitions in the settling downstream logic, than an output of all 0s would. NOR gates could be used to force the adder outputs to all 0s, but require a different value for the guard signal than is used for latches in this library, meaning designers could not simply "swap in" the part without changing control logic.

Similar work was done on the other two adders in the hp14b library, a carry select and a carry lookahead adder. For all three types of adders, 8-bit versions of each of the discussed guarding possibilities were constructed from the baseline or unguarded adder: *internal latch, float, internal gate, external latch,* and *external gate*. For the RCA, 64-bit versions were created for each guarding possibility. The carry lookahead adder also had 4 64-bit versions built, the baseline, both external guarding techniques, and the *internal gate.* No 64-bit versions of the carry select adder were built since it consisted of RCA and mux cells, and behaved very similarly to the RCA adder.

3. ASSESSING ENERGY CONSUMPTION

The effectiveness of a guarding technique can be measured by comparing the positive and negative contributions. Positive contributions arise from the reduction of transitions in the guarded logic, which lowers switched capacitance in the design. Negative

Adder	Area Penalty	Delay Penalty
Unguarded	0%	0%
External gate	155%	0%
Internal gate	26%	19%
External Latch	182%	8%
Internal Latch	83%	69%
Float	31%	38%

Table 1. Area and Delay Penalties in a RCA

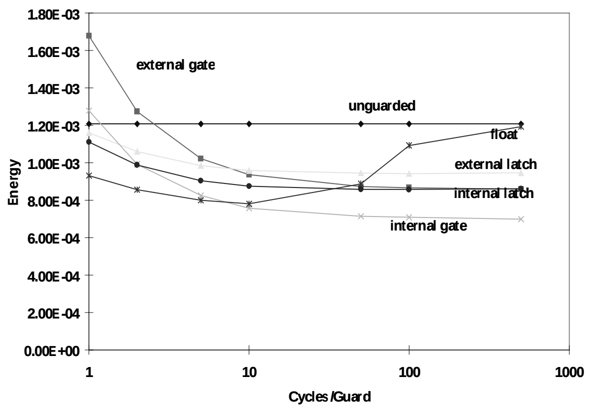

Figure 3. Guarding a single calculation in an 8-bit RCA.

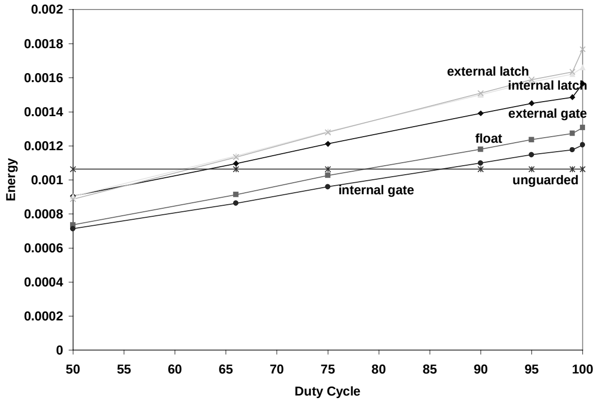

Figure 4. Energy consumption of 8-bit guarded RCAs with a guard cycle of 5.

contributions are due to additional interconnect and transistor requirements and the additional switched capacitance of the guards themselves during unguarded cycles. Controlling the guards also adds a negative contribution, but this is not measured in our experiments. Although some overheads vs. savings comparisons are made in [8], they are over a specific application space. Here, the measurements are designed to model a generic application space.

When guarding functional units, there are two main factors to consider. First is the duty cycle of the unit, or what percentage of time the unit does useful work. Second is the guard cycle of the unit, or how many consecutive cycles it remains idle. Just knowing the duty cycle is not enough, as a 50% duty cycle can represent a calculation performed every other cycle or 10 busy cycles followed by 10 idle ones. Various guarding techniques perform differently under each duty/guard cycle combination.

In our experiments, 1000 vectors were simulated using Star-Sim running in ACS mode [7] and the average energy measured over a variety of duty and guard cycle combinations. The first experiment was performed on the 8-bit adders, attempting to determine at what guard cycle each particular guarding technique became effective when performing only one useful computation. The second round of experiments measured the energy consumption of the 64-bit adders for duty cycles ranging from 50% to 100% with guard cycles of 1 to 5 consecutive idle cycles. The power lines on these adders were separated to allow measurements to be made on a variety of adder widths. These experiments determine which implementations perform best under various conditions, and create energy surfaces, allowing for automated selection of guarding techniques in datapaths.

3.1 Guarding Low Duty Cycles

The first set of experiments attempts to determine the number of consecutive idle cycles required before a guarded implementation becomes beneficial. Low duty cycle activities characterize the behavior of units such as rounders, which may only be used following a long period of calculation at some internal bit width, before sending the result to the outputs. Results were measured for the 8-bit RCA, carry select, and carry lookahead adders. The x-axis uses a log scale to show the guard cycle of the adder. Values on the x-axis can be translated to duty cycles by $^{1}/_{x+1}$.

For the RCA shown in Figure 3, all of the guarding schemes are beneficial beyond a guard cycle of 2, with only the gated methods consuming more power than the unguarded adder until that point. This occurs because of the additional computation performed when the gates force the adder inputs to the controlled value, causing a situation not unlike the unguarded adder. The curves for the two gated and two latched methods are almost identical in shape, but the internal methods provide additional savings of 25% and 8% respectively over their external counterparts. While the *float* method begins as the most beneficial method due to its low overhead, when guard cycles become sufficiently long, its power consumption increases as the wire capacitance can no longer hold the charge required to keep the inverters at full strength. Although not shown here, the carry select adder behaved in a similar manner, but the savings of the *internal gate* over the *external gate* method was 38%. In the carry lookahead adder, the internal methods have behavior much closer to the external methods. The *internal latch* actually had higher power consumption than the *external latch* because more transistors are required to latch the adder internally as opposed to externally. The *internal gate* savings were a few percentage points higher than the *external gate*.

One theme throughout these experiments is that gated methods consume less energy than latched methods as the guard cycle increases. When latches are used as guards, no spurious transitions are seen in the adder, while gates cause an additional computation. But gates have a smaller capacitive overhead, so less capacitance switches as the data toggles on the uncontrolled input. As the guard cycle increases, the additional computation becomes less important than the capacitance of the guard, and the gated methods prove more efficient.

3.2 Guarding High Duty Cycles

It is apparent that guarding a single calculation at duty cycles below 50% is generally beneficial, but information is still needed about more active units. This set of experiments examines the effects of guarding duty cycles ranging from 50% to 100% when holding the guard cycle constant at 5 cycles. Experiments were conducted on the 64-bit RCA and carry lookahead adders with separated power lines. Duty cycles appear on the horizontal axis. The unguarded adder creates a horizontal line since it performs the same number of additions regardless of the true duty cycle.

The data for an 8-bit RCA is shown in Figure 4. Even at higher duty cycles, gated techniques prove to be superior to the latched ones, with the *internal gate* method proving to be viable up to an 85% duty cycle. The performance of the *internal gate* is about 20% better than that of the *external gate*. While the two latched

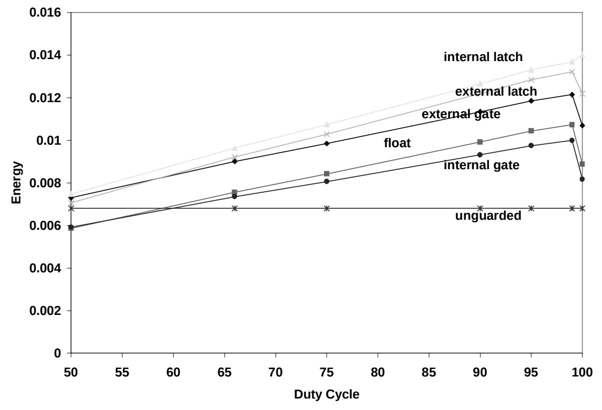

Figure 5. Energy consumption of 64-bit guarded RCAs with a guard cycle of 5.

methods have roughly the same performance, the float version provides identical behavior at 20% energy savings, keeping it a viable guarding mechanism beyond the 75% duty cycle. Both external methods and the *internal latch* only save energy for duty cycles below 60%.

Figure 5 plots the same data, but for a 64-bit RCA. The shape of the curves are similar, but are shifted upwards rather significantly. For the 8-bit adder, all of the guarding mechanisms are useful at the lower duty cycles, but at 64 bits, only two of the internal versions are ever useful, and even they become ineffective at duty cycles above 60%. The difference here is due to the distribution of the guard signal over the entire functional unit. This cost is most evident from the decrease in energy consumption going from a duty cycle of 99% to 100%. At the 100% point, the guard line never transitions.

The more complex logic of the carry lookahead adder yields very different results when examining the viability of the three implemented guarding schemes at the higher duty cycles. Whereas the gated techniques always outperformed latched ones in the RCA, the latched scheme proves slightly better at lower duty cycles for the carry lookahead adders. At this level of logic complexity and activity, the extra calculation cannot be amortized away and the unit is inactive enough that the higher capacitance of the latch is not detrimental. The gated schemes regain their superiority around the 60% duty cycle, but maintain only a slight edge until becoming useless at duty cycles above 85%. The *internal gate* method increases savings by a few percent over its external counterpart. There is little difference in the shape or crossover points on the curves when moving from 8 to 64 bits, so only the 64-bit data is shown in Figure 6.

4. CREATING ENERGY SURFACES TO AUTOMATE GUARD STYLE SELECTION

The previous set of experiments explored a range of duty cycles while keeping the guard cycle constant. In reality, guard cycles of adders in various applications cover a range of values, just like the duty cycle. By varying both duty cycles and guard cycles, an energy consumption surface, rather than line, can be derived for each adder implementation.

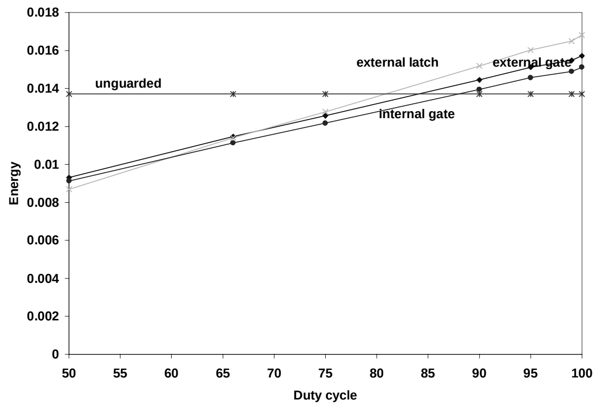

Figure 6. Energy Consumption of 64-bit guarded carry lookahead adders with a guard cycle of 5.

4.1 Energy Surfaces and Automating Guard Style Selection

Energy surfaces were created by experiments conducted on 64-bit adder implementations over duty cycles of 50% to 100% and guard cycles ranging from 1 to 5 consecutive cycles. These experiments create several planes in the experimental design space, one for each adder implementation. The unguarded adder plane cuts horizontally across this space, just as the unguarded adder line horizontally bisected the two dimensional space in Figure 5 and Figure 6. The various guarded energy planes each have some slope to them and cut across the unguarded plane at various points, creating a figure too complex to display here.

From these intersecting planes, a *minimum* energy surface can be created in that design space. At each coordinate point *(guard cycle, duty cycle)*, the Z value is set to be the lowest energy value seen across all the adder implementations. A table lookup associates each point *(guard cycle, duty cycle)* to the minimum energy implementation. This suggests an easy way to automate the selection of the best implementation. Guard and duty cycles can easily be determined by examination of the applications.

When inserting guards into a datapath, it is important to consider the delay introduced by the guards. In order to take this into account, minimum energy delay (ED) [5] surfaces, rather than minimum energy surfaces, are used. This tempers the energy savings of the guarding implementation with any increases in the delay of the critical path of the circuit.

5. EXAMPLE: A MOTOROLA DSP56K SUBSET

To demonstrate the guard selection process, an 11 instruction processor based on the Motorola DSP56000 ISA [3] was built. The design contains three adders, all of the carry lookahead type: a 16-bit address adder, a 56-bit accumulator, and a 33-bit rounder. The processor is capable of running three typical applications: a 4-tap FIR filter, 64-tap FIR filter and an adaptive LMS filter[1]. Each application places very different access patterns on the adders as shown in Table 2. Guard and duty cycles are easily derived from the assembly code or functional simulations of the DSP.

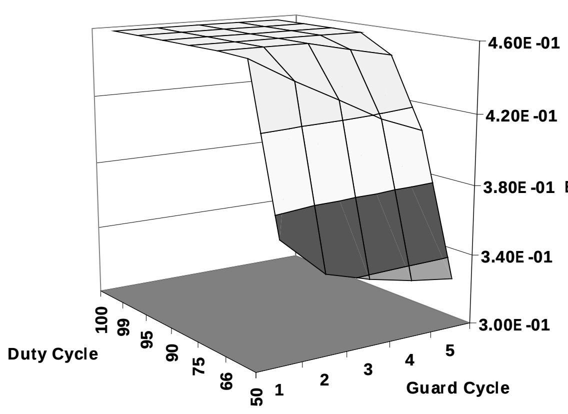

Figure 7. Minimum Energy Delay surface for the 16-bit carry lookahead adder.

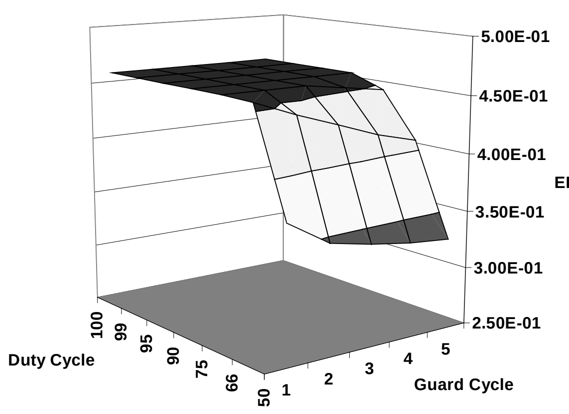

Figure 8. Minimum Energy Delay surface for the 56-bit carry lookahead adder.

Figure 7 shows the minimum ED surface used to select the best implementation for the 16-bit address adder. The usage patterns of the two FIR programs both lie in the flat region of the plane, which corresponds to the unguarded adder. This is not surprising since the adders are in almost constant use (duty cycles of 85% and 98%) and only idle for a single cycle. The 58% duty cycle and 2 clock period guard cycle data point of the LMS program lies on the sloped portion of the plane, at a point corresponding to the *external latch* method. Designers can apply knowledge as to the likelihood that each type of program will be executed to determine which of the two methods is likely to be best. In the absence of designer intervention, the decision can be made by which method is most frequently chosen as the best.

The same methodology can be applied to the 56-bit accumulator using the minimum ED surface in Figure 8. While the behavior in the 64-tap FIR lies at a point where the unguarded adder is the lowest ED choice, the 4-tap FIR and LMS both have behaviors that lie on the sloped region of the surface at points that correspond to the *external latch* method. The rounder will also benefit from guarding, but the points lie off the edge of the graph. By extrapolating the data, the *external latch* method is the best implementation for this adder.

The effectiveness of the predictions was measured by implementing two versions of the DSP ASIP, one with guarded adders and the other without. Both designs were built using a standard cell design flow through place and route. Back-annotated capacitance and design information was used with the gate level design to calculate energy. Estimations were made using an in house tool based on transition counting[6]. The results of this comparison are presented in Figure 9.

Program	Duty Cycle	Guard Cycle
LMS		
Accumulator	38%	3
Address adder	58%	2
Rounder	19%	4
4-tap FIR Filter		
Accumulator	57%	3
Address adder	85%	1
Rounder	14%	6
64-tap FIR Filter		
Accumulator	95%	3
Address adder	98%	1
Rounder	1%	66

Table 2. Duty and Idle Cycles of adders present in the DSP subset running various programs.

The topmost pair of columns shows the total energy consumption of all three adders over all three applications. Within the columns are breakdowns for each adder and any guarding latches if present. The remaining three pairs of columns show the results for each individual program.

Looking at the address adder energy, the technique correctly predicted the adder should not be guarded in either of the FIR programs. In the actual designs, guarding this adder increases energy consumption by 13% and 3% for the 4-tap and 64-tap FIR programs respectively. Only in the LMS application are guard latches beneficial, which matches the predictions. For the accumulator, the predictions were correct 2 out of 3 times. In the 64-tap FIR, the best adder was predicted to be the unguarded one, but there is an 8% saving when comparing the two designs. In the case of the rounder, the *external latch* implementation was picked to save energy in the design and the results confirm the prediction. Using the ED surfaces to select the best implementation would have prevented the energy losses from guarding the address adder.

6. GENERALIZING TO OTHER DATAPATH UNITS

The lessons learned working with adders can be extended to other datapath logic if the cost of guarding is thought of in terms of the

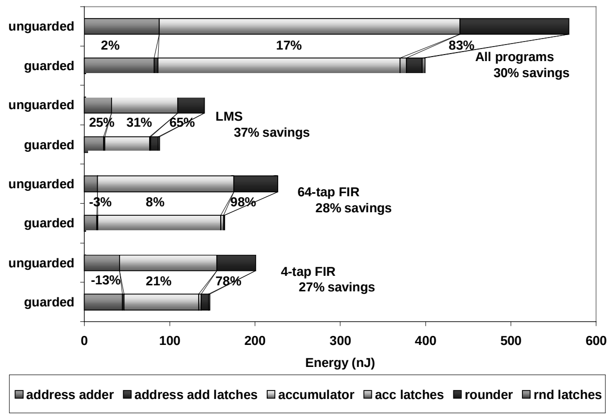

Figure 9. Comparison of adder energies in guarded and unguarded implementations of 11 instruction DSP

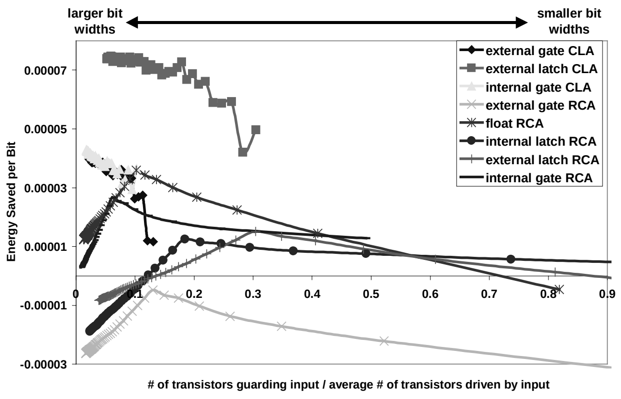

Figure 10. Normalized energy savings compared to the ratio of transistors needed to guard an input over the average number of transistors driven by an input.

number of transistors required to guard an input as compared to the number of transistors driven by the input. Figure 10 illustrates this point. It displays the energy savings for the modeled adders normalized by the bit width of the unit. The energy savings are an average over all measured guard cycles with a 50% duty cycle. The x-axis is the ratio of the number of transistors needed to guard an input to the average number of transistors driven by the inputs. The number of driven transistors includes those directly driven by the input lines as well as the transistors they drive, and so on. Each point on the line represents a different bit width adder.

There is a clear trend of decreasing savings as the overhead of the guarding logic increases. Two interesting features appear in the lines besides the overall trend. The peak in the curves of the RCAs is caused by the cost of distributing the guard signal over long distances. In the carry lookahead adders, this cost was masked by the more complex logic, but is quite apparent here. Around a ratio of .12 however, this effect begins eating away the savings from reduced transitioning, becoming more devastating as width increases. When moving to other functional units, the distribution cost must be weighed against the complexity of the arithmetic logic it is being delivered to.

The second feature, a sharp savings decrease at the right side of the curve, appears in the externally guarded carry lookahead implementations. This drop off corresponds with smaller bit widths, suggesting there is a minimum bit width necessary to get maximal benefits from guarding. The leveling off point here appears around ratios of .1 and .2, which corresponds to about 12 bits. The drop off is greatly reduced in the internally guarded method.

Similar graphs were created for other duty cycles, and while the shapes of the curves remain fairly constant, their positioning varied. Higher duty cycles caused the curves to shift down, since energy savings decreased. Relative positions also changed, with external techniques falling faster than their internal counterparts. Latched techniques tended to have a faster loss rate, and at higher duty cycles ended up with less savings or greater losses than the gated methods, having fallen prey to the higher capacitance of the latches.

Overall, the best results come when the ratio of guarding logic to computational logic is less than about 1 guard transistor for every 10 driven computational transistors.

7. CONCLUSIONS

This work has presented a new approach to implementing guarded evaluation in major datapath elements through the incorporation of the guarding functionality into the computational logic of library cell functional units. Three different internal implementations are presented and evaluated in adders. These techniques reduce the overhead of guarding by incorporating it into the functional logic. The reduction in overhead brings with it an overall energy savings of up to 38% over comparable external guarding methods. This allows guarded evaluation to remain a viable energy saving technique at duty cycles up to 20% higher than are practical for external guarding techniques.

A modeling methodology is presented that allows energy and ED characteristics of a generic application space to be defined in terms of functional unit usage and idle patterns. Based on models of the various guarding techniques in RCA, carry select, and carry lookahead adders, an automated technique for selecting the optimal guarding style is proposed. Selections are based on guard and duty cycle characteristics of target applications, removing the guesswork as to which style of guarding will maximize energy savings in the design. The selection technique was tested on a DSP ASIP, and proved to be effective 89% of the time, allowing overall energy savings to be increased.

The modeling is generalized to other functional units by looking at the overhead of guarding to functional logic depth. It was found that guarding is most beneficial when the ratio of guard transistors to driven computational transistors is below $^{1}/_{10.}$

8. ACKNOWLEDGEMENTS

This work was supported by the Defense Advanced research Projects Agency under order No. A564, the National Science Foundation under grant No. MIP9408457, and the Semiconductor Research Corporation under Task ID 068-073. The United States government has certain rights to this material.

9. REFERENCES

[1] *Archive Containing DSP56000 Related Files*, http://www.mot.com/SPS/DSP/software/dr_bub/56000.zip.

[2] Correale, Jr., A, "Overview of the Power Minimization Techniques in the IBM PowerPC 4xx Embedded Controllers," *Int. Symp. on Low Power Design,* 1995, pp. 75-80.

[3] DSP56000/56001 Digital Signal Processor User's Manual, Motorola Inc. 1990.

[4] *Epoch Users Manual*. Cascade Design Automation (Duet). Feb. 1997.

[5] Horowitz, M, et al., "Low Power Digital Design," *IEEE Symposium on Low Power Electronics*, 1994, pp. 8-11.

[6] Klass, et al., "Modeling Inter-Instruction Energy Effects in a Digital Signal Processor," *Power Driven Microarchitecture Workshop ISCA98*, 1998, pp. 25-30

[7] *Star-Sim User's Guide,* Avant! Corporation, 1997.

[8] Tiwari, V, et al., "Dynamic Power Management for Microprocessors: A Case Study," IEEE VLSI Design**,** 1997, pp. 185-92**.**

[9] Tiwari, V., et al., "Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design," *IEEE Trans. on CAD*, vol. 17, no. 10, Oct. 1998, pp. 1051-60