RTL Dynamic Power Optimization for FPGAs

David Howland and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003

***Abstract*— As FPGA devices grow in size and speed, power consumption is becoming a limiting problem. In this document, a dynamic power optimization CAD tool for FPGAs that is applied at the register transfer level (RTL) during design synthesis is described. The automatic design transformations implemented by this tool reduce dynamic power by eliminating unnecessary computation by functional units when unit results are not needed. Our new *guarded evaluation* algorithm locates design functional units that may perform unnecessary computation under specific unit input conditions. Once located, functional unit input logic is modified so that the function will remain dormant when the output of the functional unit is unused. This approach reduces unnecessary signal transitions, thus saving dynamic power. To locate the most desirable functional units for guarded evaluation, an area, depth, and switching activity estimation flow is presented that provides high-level estimates to aid in RTL design transformation. Our approach has been integrated into a design flow for Altera Cyclone II devices. Experiments show that, although not all designs benefit from our approach, an average 12% reduction in dynamic energy can be achieved for a subset of benchmark designs with suitable functional units.**

I. Introduction

Field programmable gate arrays (FPGAs) are critical components in a wide array of embedded and portable devices [1]. Their reconfigurability makes them desirable candidates for applications that require on-the-fly logic changes and specialization. Unfortunately, reconfigurability has a negative side effect. FPGAs are known to be power inefficient compared to other technologies, such as ASICs, due to the presence of abundant logic and routing switches. The resources consume significant dynamic power during circuit switching due to the charging and discharging of capacitance. Although FPGA dynamic power consumption can be addressed during low level circuit mapping, additional power optimization opportunities exist at the register-transfer level (RTL) [2], when the architectural design structure is still apparent.

To address this dynamic power issue, a computer-aided design tool which optimizes RTL designs prior to synthesis has been created. Our approach looks for specific opportunities to disable large functional blocks based on the consumption of their results later in the functional pipeline. If it is determined that the results of a unit will be unused on a specific clock cycle, its inputs are held constant, preventing logic and interconnect switching in the functional unit. Although a small amount of logic is inserted into the RTL design to suppress functional unit input switching, a net power reduction is achieved due to reduced switching in the internal unit logic and interconnect. In general, this type of design modification, called guarded evaluation, is only effective for specific types of designs and circuit structures. As a result, rapid area, performance, and switching estimation of RTL designs forms an important part of our CAD tool flow. A graph-based search algorithm is used to quickly assess candidate designs and assess their fitness for optimization.

To demonstrate the benefits of our approach, the new CAD tool was applied to fifteen randomly selected RTL designs. Following physical design with Altera Quartus tools, it was determined that six designs were able to achieve an average dynamic energy reduction of 12% with an average performance loss of about 2%. For the remaining designs, the original, unoptimized designs either displayed similar energy characteristics or significantly better performance than the optimized designs. In these cases, the functional unit structures in the designs were not suitable for our optimizations.

The rest of this document is organized as follows. Section II provides relevant background information on guarded evaluation. The guarded evaluation algorithm is presented in Section III. This algorithm requires a reverse-ordered traversal of the design which locates and evaluates candidate computational blocks. The results of our approach on designs mapped to Altera Cyclone II devices is presented in Section IV.

II. Background

The primary action of our CAD tool is the insertion of logic into an RTL design which amortizes circuit switching. This optimization is performed by modifying an existing FPGA synthesis flow with new low-power transforms. Full FPGA compilation then converts a design from an RTL specification to a low-level netlist through synthesis, tech-mapping, placement, and routing stages that prepare the design for the FPGA device. To optimize power, the synthesis stage has been augmented with several new steps. FPGA area, performance, and switching estimation provides preliminary information about the post-mapped circuit in order to assist with the optimizations. Logic insertion provides circuitry to hold the inputs to functional blocks constant if the block output is unneeded [3].

Figure 1 shows an example of guarded evaluation. A small portion of a design is shown before and after guard circuitry is inserted. Before modification, the circuit consists of logic that is continually evaluated regardless of whether its output is selected by the multiplexer. After guarding, the select signal controls both the multiplexer and the inputs of the logic group. No matter how the inputs change, the logic will only

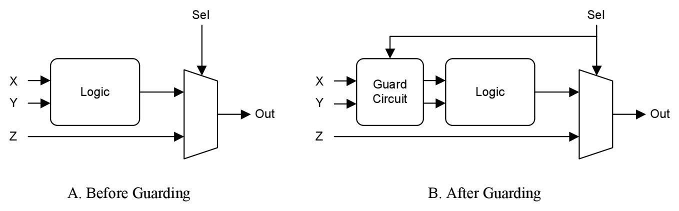

Fig. 1. Guarded Evaluation Example

perform computation if the results will be propagated. Larger portions of logic provide opportunities for more dynamic power savings.

The select input in the above example is the key to the guarded evaluation technique. These signals must isolate functional unit inputs and hold them at a steady state. The guard circuit may be implemented either with transparent latches or with AND gates [3]. The latch captures the inputs and holds them steady when it is deactivated. AND gates combine the functional unit inputs with a Boolean zero when inactive, which forces the functional unit inputs to zero. Since latches are not efficiently implemented in FPGAs, our approach uses AND gates.

Guarded evaluation is not necessarily effective for all designs. The addition of logic to a design may have an adverse effect on dynamic power. Depending on the function and programming style of a design, guarded evaluation may find plentiful opportunities for optimization or almost none. Our experiments, which will be presented in Section IV, show that an average 12% reduction in dynamic power is possible with guarded evaluation for a subset of our designs.

Guarded evaluation has been the subject of a number of recent research projects, although automated techniques targeting FPGAs have not been reported. In [4], the authors describe *precomputation*, an approach similar to guarded evaluation, for ASICs. Unlike guarded evaluation, logic is disabled by activating or deactivating register enables for registers already present in the circuit. In [2], the authors present an RTL power reduction method which identifies optimizations at the register transfer level, but postpones optimization commitment until gate level synthesis. The first phase of the approach involves the location and calculation of an activation signal. The second phase involves the evaluation of possible area and power changes due to circuit changes. In [3], the authors present guarded evaluation as a technique for gate level, rather than register transfer level, reduction. Their algorithm first identifies activation signals which can be combined to disable functional units. Although their approach is effective, it uses a time-consuming exhaustive search. A similar approach [5] also involves an exhaustive assessment of activation function candidates.

III. Approach

Before the power optimizations are performed, an estimation step is needed to collect information that is used during optimization. Logic area and depth information are used to draw attention to larger combinational sections of a design, as well as to locate critical paths. Many of the area and depth estimates in our system are determined from a custom library of blocks for functional units and simple LUT packing for random logic. When complex operations are encountered in an elaborated design, precomputed area, delay, and switching values for the unit are located in the library and used for estimation. The use of a block library greatly increases the speed and accuracy of the estimation process.

Switching estimation is used to locate the areas of a design that exhibit the greatest switching activity. These areas often exhibit the greatest dynamic power savings after optimization. The specific signal switching metric that is used in our work is transition density [6]. This metric represents switching activity as a probability that a given signal will make a transition to the opposite logic value at any given time. Significantly extending synthesis run time to perform optimization is not acceptable. Therefore, the time needed for estimation must be a reasonable part of the overall optimization algorithm run time. Transition density calculation can be completed without the need for computationally-expensive simulation.

Guarded evaluation circuitry is inserted into a design with the aid of estimation in four main steps:

- The activation functions of existing circuitry are calculated.
- The best locations in the design for optimization are determined.
- Guard logic is inserted into the design
- Any logic sections that are left only partially covered are tied off.

Algorithms have been created for each of these steps. The algorithms operate on a graph representation of the RTL design. Activation function calculation is the most time consuming of the four steps in the guarded evaluation process. We have created a graph traversal algorithm which examines the entire RTL design and calculates an activation function for every node. Activation functions are constructed using control signals that come from conditional branches due to multiplexers in the RTL design. The traversal algorithm is based on a breadth-first approach proposed by Münch et al [5].

The activation function of a node is based on the logic that it drives. In order to determine if a node may become idle, information must be gathered from the descendants of the node in reverse topological order. As a result, a node is not visited until all of its children have been visited and the search proceeds from design outputs and register inputs towards design inputs and register outputs.

Figure 2 shows an example circuit and the activation functions that is assigned to each node. The functionality of the nodes is not given because the activation function is not dependant on the function of the logic.

All activation functions begin with an *incorporation* operation. Consider multiplexer M3 in the example. When the algorithm visits M3, the control signal is incorporated into the activation function for each of the data inputs. The operation

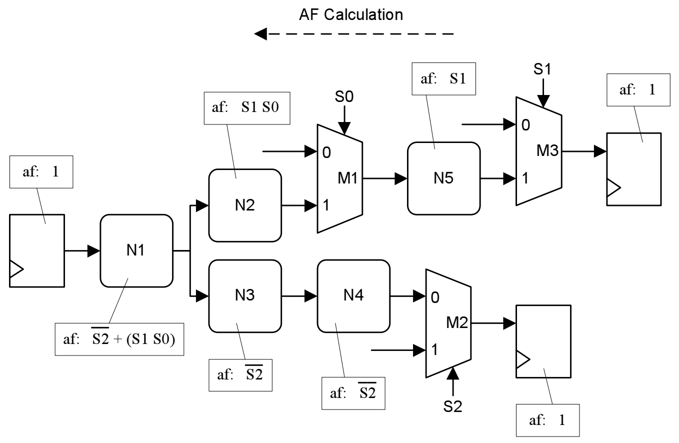

Fig. 2. Activation Function Example

is a logical AND. The empty activation function is a logical 1. Therefore, the activation function for data input 0 of M3 is $\overline{S1} \cdot 1 = \overline{S1}$ and the activation function for data input 1 is $S1 \cdot 1 = S1$.

The most basic operation is the *propagation* operation. This operation copies an activation function from a child node to a parent node. Consider nodes N3 and N4 in Figure 2. The activation function $\overline{S2}$ is inherited by N3 from N4 via propagation. Propagation can also be used in conjunction with incorporation, as is the case with multiplexer M1. The activation function $S1$ is propagated and then M1's own control signal, $S0$, is incorporated to produce $S0 \cdot S1$.

The final operation necessary for activation function calculation is *combination*. This action is necessary to handle the convergence of two paths with different activation functions. Consider nodes N1, N2, and N3 in Figure 2. When the algorithm visits node N1 during the traversal, the activation function from N3 may be propagated, but combination is required to take node N2 into account. Combination is a logical OR. The result is $\overline{S2} + S0 \cdot S1$.

The most important phase of the optimization process is the selection algorithm. Every node in a design that is assigned an activation function becomes a possible point of optimization. Guard logic insertion choices are made by examining the cost of nodes. The cost in this case is a metric that indicates the relative dynamic power consumption of logic rooted at a given node. The selection algorithm assigns a cost to every node in the graph based on area and switching activity. The guarded evaluation technique works better for some groups of logic than for others. In general, there are two requirements for a group of logic to be a favorable choice. First, it must experience periods of computation when its output data is not required in any subsequent logic. Second, it must be a high cost logic group (e.g. high switching and/or large area). Large groups of logic typically require many long interconnection wires. In addition, bulky groups of logic will experience a larger amount of glitching, which increases the amount of transitions which consume dynamic power.

The selection algorithm first calculates node cost functions, then examines several node properties (including cost) to determine the most desirable nodes for optimization. The cost function of a node is a multiplication of the area and activity, as shown in Eq. 1.

$$C(n) = A(n) \times D(n) \tag{1}$$

where $C(n)$ is the cost of node n, $A(n)$ is the area of node n in LUTs, and $D(n)$ is the output switching activity of node n as transition density.

The cost function in Eq. 1 is the *local cost*, the cost contribution of a single node. When guard logic is placed in a design, not only is the optimized node affected, but every descendant node in its fan-out is affected. Therefore, a *global cost* is needed to represent the cost of all logic descended from a given node. Like the activation function algorithm, the cost of a node is calculated using information from its children, thus a reverse topological ordering is needed. During the traversal, global cost is accumulated upwards through the design. The global cost metric is used by the selection algorithm. A sorted list of nodes ranked by cost is kept during the traversal of the data structure. As the cost is calculated for the current node, it is added to the list with a rank relative to the design-wide list of nodes. After the traversal, highest-cost nodes are selected for consideration first. Nodes continue to be selected until the pool of remaining nodes contains no nodes that have a cost above a threshold.

When implementing guarded evaluation, an effort must be made to minimize the impact on performance. We evaluate performance as the maximum operating clock frequency of the final mapped design, or F_{MAX}. F_{MAX} is directly influenced by the longest possible combinational path through the design from a register or an input to another register or an output. In order to avoid decreasing F_{MAX}, we avoid extending the critical path by using a depth metric.

The estimation flow provides depth information and assigns each node a slack value. Like area, depth is measured in terms of LUTs. The slack value tells the guarded evaluation algorithm how much room it has to implement the guard circuitry. Since guard circuitry will be added to the inputs of a logic group, it will effectively extend its depth. The activation function of selected nodes must not exceed the depth of their slack values. If the node does not pass this test, it is discarded and its cost is reset to zero. The depth of the activation function is calculated as the depth of the activation circuit plus the greatest depth of the control signals which contribute to the activation function. The depth of each of the control signals is determined using a recursive search. A node that passes the cost and critical path tests is selected for optimization. A guard circuit construction algorithm is called to instantiate the guard circuit. Once the selection algorithm has selected a node for optimization, the guard circuit must be constructed and inserted into the design. A subsequent algorithm builds the needed circuitry and rewires the guard logic onto the inputs of a given node.

The final algorithm in the guarded evaluation process is executed once the high-cost nodes have been targeted and guarded. For the guarded evaluation technique to be effective, the guard logic must effectively shut down a block of logic when its output is not needed. If all the inputs are not covered, it would still be possible to stimulate the logic and consume dynamic power. Therefore, an effort is made to tie off any leftover nodes that may be exposing groups of logic that have been targeted for guarding.

IV. Results

Our guarded evaluation algorithms were integrated into an existing FPGA synthesis flow. This flow includes a front-end parser and elaborator (Icarus) [7] and a back-end FPGA synthesizer (Odin) [8]. Odin is an academic synthesis tool for heterogeneous FPGAs, which provides a suitable interface to the Altera Quartus II tools (version 6.0). Quartus provides design synthesis, technology mapping, and place and route. Our estimation and optimization flows were directly coded into the Odin module. Altera PowerPlay was used to assess post-map power consumption at 50 MHz. Test vectors were created and provided to the Quartus software along with the benchmarks. This work targets the Altera Cyclone II family of devices. The Quartus tools were configured to optimize for performance, rather than power. The logic synthesis, technology mapping and fitting operations are configured to obtain the highest possible F_{MAX}. Our fifteen benchmarks were taken from the Quip benchmark suite [1]. Each benchmark includes Verilog source code and an input simulation vector file. Each design was targeted to the smallest Cyclone II FPGA which would fit it.

The results of our experiments are summarized in Tables I and II. As mentioned in Section III, not all designs benefit from our approach. As shown at the top of Table I, a total of 6 designs experienced an average dynamic energy savings of 12.8% with a cost of 1.6% in performance and an average increase of 100 LUTs. This savings includes the energy cost of the inserted logic. In the middle of the table it can be observed that although four designs achieved a small energy savings, each design suffered a substantial performance loss. In these cases, the original designs could be used for implementation, rather than the optimized designs. At the bottom of Table I, it is observed that five designs experienced no change during optimization since our CAD tool could not find any appropriate locations for guard logic.

Table II illustrates the importance of estimation in the guarding process. For the designs that benefit from guarded evaluation it can be seen that estimation helps ensure that guard logic is only inserted in locations which will not adversely impact design performance. If delay, area, and switching estimation is not used, resulting design performance and energy consumption suffers. Static energy is unaffected by the approach for the same sized FPGA since power to portions of the FPGA cannot be selectively shut down. For these 90 nm FPGAs, static power consumption is a fraction of total power consumption.

	LUTs		Max Freq. (MHz)		Energy nJ @ 50 MHz	
Benchmark	Orig.	Guard	Orig.	Guard	Orig.	Guard
Energy red.						
counter255	541	662	103.02	102.32	0.45	0.31
i2c	301	286	266.88	248.63	0.05	0.04
mac1	3030	2913	71.01	70.44	1.02	0.97
mem_ctl	4427	4458	73.82	74.62	0.63	0.60
framebuf	1094	1272	136.18	141.38	0.22	0.20
fir_scu_rtl	3162	3767	86.58	81.73	0.60	0.58
Geo. ave.	1401	1508	109.23	107.52	0.35	0.30
Perf red.						
alu	239	314	222.42	189.54	0.06	0.05
jacobi	8945	11041	173.73	144.67	2.79	2.67
minirisc	506	537	141.32	108.40	0.20	0.19
sdram16	311	336	256.61	178.18	0.08	0.07
No effect						
aes	5121	5121	142.39	142.39	3.25	3.25
barrell32	320	320	204.08	204.08	0.10	0.10
bubblesort	1904	1904	75.08	75.08	0.68	0.68
cf_fir	9998	9998	89.91	89.91	7.44	7.44
des	895	895	199.48	199.48	0.78	0.78

TABLE I

Experimental Results With and Without Guarded Evaluation

	Original		Guarded w/o Estimation		Guarded w/ Estimation	
Benchmark	Ener (nJ)	F_{MAX} (MHz)	Ener (nJ)	F_{MAX} (MHz)	Ener (nJ)	F_{MAX} (MHz)
counter255	0.45	103.02	0.32	100.00	0.31	102.32
i2c	0.05	266.88	0.05	161.21	0.05	248.63
mac1	1.02	71.01	1.10	72.89	0.97	70.44
mem_ctrl	0.63	73.82	0.66	60.50	0.60	74.62
framebuf	0.22	136.18	0.20	117.99	0.20	141.38

TABLE II

Impact of Estimation on Guarded Evaluation

V. Acknowlegments

This work was funded by a grant from Altera Corporation.

References

[1] Altera Corporation, "Altera web site," www.altera.com, March 2005.
[2] Q. Wang and S. Roy, "RTL power optimization with gate-level accuracy," in *ICCAD '03: Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Computer Society, 2003, p. 39.
[3] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: pushing power management to logic synthesis/design," in *ISLPED '95: Proceedings of the 1995 International Symposium on Low Power Design*. ACM Press, 1995, pp. 221–226.
[4] J. C. Monteiro, "Power optimization using dynamic power management," in *Proceedings of the 35th Design Automation Conference*, June 1998.
[5] M. Münch, B. Wurth, R. Mehra, J. Sproch, and N. Wehn, "Automating rt-level operand isolation to minimize power consumption in datapaths," in *DATE '00: Proceedings of the Conference on Design, Automation and Test in Europe*. ACM Press, 2000, pp. 624–633.
[6] K. K. Poon, S. J. Wilton, and A. Yan, "A detailed power model for field programmable gate arrays," in *Proceedings of the IEEE International Converence on Field-Programmable Technology*, 2002.
[7] S. Williams, "Icarus verilog," www.icarus.com/eda/verilog/, March 2005.
[8] P. Jamieson and J. Rose, "A verilog RTL synthesis tool for heterogeneous FPGAs," in *FPL '05: Proceedings of the 2005 International Conference on Field Programmable Logic and Applications*. IEEE Press, 2005, pp. 305–310.