Using Negative Edge Triggered FFs to Reduce Glitching Power in FPGA Circuits

Tomasz S. Czajkowski
Department of Electrical and Computer Engineering,
University of Toronto, Ontario, Canada
czajkow@eecg.toronto.edu

Stephen D. Brown
Department of Electrical and Computer Engineering,
University of Toronto, Ontario, Canada
brown@eecg.toronto.edu

ABSTRACT

This paper presents an algorithm for reducing dynamic power dissipated by Field-Programmable Gate Array (FPGA) circuits. The algorithm uses a fast probability based model to estimate glitches on wires in a circuit and then inserts negative edge triggered FFs at outputs of Lookup Tables (LUTs) that produce glitches. A negative edge triggered FF maintains the logic value produced by the LUT in the previous cycle for the first half of the clock period, filtering glitches that occur at the output of the LUT. The power dissipation is lowered by reducing the number of transitions that propagate to the general routing network.

We applied the algorithm to a set of benchmark circuits implemented on a commercial FPGA, Altera's Stratix II. The results obtained using Quartus II 5.1 CAD tool show a reduction in dynamic power dissipation by 7% on average and up to 25%.

Categories and Subject Descriptors

B.6.3 [**Design Aids**]: Optimization.

General Terms

Algorithms

Keywords

Dynamic Power, FPGAs, Glitches.

1. INTRODUCTION

Field-Programmable Gate Array (FPGA) devices are a popular choice for low to medium volume digital applications. They can implement various different circuits without a need to fabricate a new chip, reducing costs and time to market. This flexibility is paid for by increased power dissipation. The power consumed by a circuit implemented on an FPGA device in a 90nm process, or smaller, presents a problem, especially if an FPGA device is to be used in a mobile application.

The power dissipation of a 4-input LUT based FPGA consists 40% of static power and 60% of dynamic power, with static power increasing with shrinking feature sizes and increasing LUT sizes [1]. In this work we are concerned strictly with the dynamic power dissipated by a logic circuit.

Several techniques have been proposed to reduce dynamic power. At the circuit level, effective power reduction techniques include the insertion of control logic to synchronize inputs of logic blocks and force at most one transition to occur at the output of a logic block [2], and reducing the power supply voltage [3]. At the logic level, the power dissipation is addressed during synthesis and technology mapping. During synthesis, power dissipated by frequently toggling wires can be reduced through rewiring [4], local logic transformations [5], and addition of redundant gates [6]. During technology mapping, power can be reduced by covering connections with high toggle rates within a LUT and minimizing logic replication [7]. Also, proposing several local mappings and selecting the lowest power mapping [8], or trading off area and depth for lower toggle rates on wires [9], can be effective. Pipelining and retiming have also been used to balance the path delays and reduce wire toggle rates [10,17].

This work focuses on logic level techniques to reduce the power dissipated by glitches in a circuit. Glitches have been shown to be able to double the toggle rate of wires in the circuit, causing a substantial increase in power dissipation [11]. Their impact, however, is not well defined until all circuit delays are known.

We propose to address power dissipation due to glitches post-routing, where the delays between LUTs are known. To reduce power in a circuit our algorithm selects LUTs that produce glitches and inserts a negative edge triggered FF (nFF) at their outputs. An negative edge triggered FF will maintain the output wire state for the first half of a clock cycle, and when the clock signal toggles, the FF will assume a new logic value. Power is saved because glitches are not propagated to logic down stream.

This paper is organized as follows. Background information is given in Section 2. The power models we use are described in Section 3. Section 4 presents the idea of negative edge triggered FF insertion and discusses alternatives to using negative edge triggered FFs. The algorithm to implement the glitch reduction technique is described in Section 5, while experimental results, conclusion and acknowledgments are found in sections 6-8.

2. BACKGROUND

An FPGA can be thought of as a rectangular array of logic cells (LCs) connected by a general purpose routing network. Each LC consists of a k-input Lookup Table (k-LUT), typically a 4 or a 5-LUT, and a *Flip-Flop* (FF). A circuit can be implemented by programming logic functions into LCs and connecting them using the routing network.

A logic circuit is represented as a graph consisting of *nodes* (LUTs or FFs) and directed *edges* that form a logical connection between nodes. A *primary input* is an I/O pin configured to accept input from external devices. A *primary output* is an I/O pin configured to produce an output from a circuit.

Once each node is placed and connections between nodes are routed, each node and edge is assigned a delay. We determine the *minimum clock period* for a circuit by traversing it from outputs of FF nodes to inputs of FF nodes and computing the longest path delay to the output of each node, called the *arrival time*. A FF-to-FF path with the longest delay is called the *critical path*.

FPGA power dissipation consists of static, short-circuit and dynamic power. Static power dissipates when current flows through a transistor even when the transistor is off. The short-circuit power is dissipated by a CMOS gate during a short period of time when both pull-up and pull-down networks conduct current. The dynamic power is dissipated any time a capacitor in a circuit is charged or discharged. It can be computed as follows:

$$P_{avg} = \frac{1}{2} \sum_{i=1}^{number\ of\ nets} (C_i f_i V^2) \qquad \textbf{(1)}$$

where C_i is the capacitance of a net, f_i is the average toggle rate of a net and V is the power supply voltage.

3. POWER MODELS

The focus of this work is to minimize the number of glitches produced by a logic circuit, thereby reducing the dynamic power it dissipates. To do so a power model for our optimizations is needed. In particular, it is crucial to compute the average toggle rate for a net in each circuit as well as estimate the capacitance of that net. We also model the LUT and FF power dissipation.

3.1 Average Net Toggle Rate Computation

Several works that address the topic of toggle rate estimation and/or prediction already exist. The work in [11] looks at the transition density [12] and how this concept applies to circuits implemented on FPGAs. In [11] the transition density is represented as a weighted sum of transitions generated and propagated through a LUT. Generated transitions are a function of LUT depth and the number of paths to LUT inputs. Propagated transitions are due to glitches that propagate through a LUT and are based on the logic function the LUT implements. In [17] transitions generated by LUTs are computed using the arrival time of a signal at a gate. The propagation of glitches is consistent with [12], where the concept of Boolean Difference is used to determine the fraction of transitions that propagate through a gate.

While the above approaches predict overall toggle rate well, the accuracy of toggle rate prediction for individual wires is insufficient for the purpose of post-routing optimization. We found the problem to be how glitches are propagated through a LUT. In the above works the concept of Boolean Difference is used to determine the conditions under which a change in input x_i of a LUT y causes a change in the output of the LUT. The condition is expressed as a function of other signals that drive the LUT, denoted as dy/dx_i. When dy/dx_i is 1 then a change on input x_i causes output y to change. Then the fraction of glitches propagated from wire x_i to output y is $P[dy/dx_i=1]$.

This formulation does not account for path delays. It is possible that by the time dy/dx_i=1 condition is met, the input x_i has already stabilized. Thus, no glitches are passed to the output of the LUT. To avoid such overestimate, we impose a stricter constraint for glitch forwarding. We require that the $dy/dx_i=1$ condition be satisfied as well as that all inputs of the LUT y, except x_i, remain stable for the duration of the clock cycle.

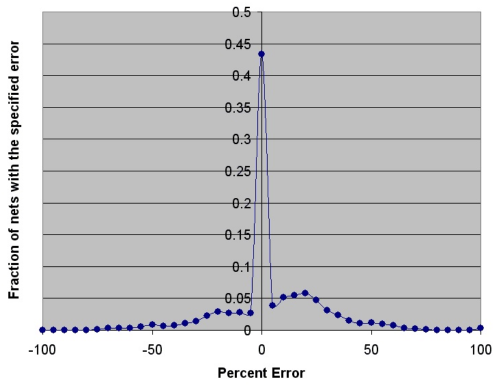

Figure 1: Transition Density Estimate Error

To compute transition density of any wire with reasonable accuracy we use five wire properties. The first property is the *static probability*. It indicates a probability that a signal assumes a logic 1 value. The second property is the *transition probability* ($P_t(y)$), which defines how often a signal changes state. This property accounts only for the functional behaviour of a signal and can be broken down into two components. The first is the *0→1 transition probability*, which indicates the probability a signal will change state from 0 to 1. The second is the *1→0 transition probability* that indicates a 1 to 0 transition probability. Both properties are considered conditional, since they only apply when a signal is at a particular state to begin with. For example, the 0→1 transition probability only applies when a signal is in a logic 0 state. The final property is the *transition density* (D(y)). It is similar to the transition probability, but it also includes transitions due to glitches. Thus, D(y) - $P_t(y)$ represents the number of glitches that occur on a per clock cycle basis on any given wire.

To estimate the toggle rates in a circuit we process nodes in topological order and compute the five wire properties for each node output. They are computed in three steps. First, we compute the wire properties under the assumption that inputs are glitch-free. We use the probability of each possible transition of inputs and determine the number of resulting transitions at the output, using the LUT function and the arrival time of its inputs. Glitches generated at the LUT output are accounted for, except those whose duration is shorter than the inertial delay [13] of the LUT. The second step accounts for glitches on LUT inputs as described earlier. Finally, to account for synchronous elements the transition density and probability values are updated once wire properties of each FF input are computed. We then repeat the process several times to account for the change in the toggle rate of FFs.

Figure 1 shows the accuracy of this method on the set of benchmarks listed in Section 6. The comparison is with respect to toggle rates obtained via timing simulation using ModelSIM-Altera 6.0c and Quartus II 5.1. The average error of our technique is 4% with 70% of the wires having transition density within ±20% of that obtained from simulation. This is better than in [11] where the average error was approximately 17%, though the results in [11] are obtained pre-placement and routing and hence are expected to be less accurate.

3.2 Net Capacitance Model

The second part of Equation 1 is the net capacitance, C_i. To estimate capacitance of each net we solved Equation 1 for C_i, using the value of 1.2V for V and the toggle rate obtained from

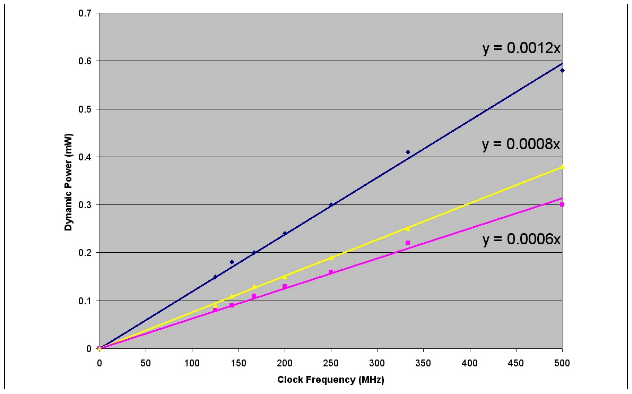

Figure 2: Power dissipation of a gated D latch (top), a FF (middle), and a gated LUT (bottom)

simulation for f_i. The power dissipated by each net was obtained from Quartus II 5.1. We used this data to model the capacitance of a net (C_i) based on its fanout and the average connection delay (d). The best-fit approximations of net capacitances were:

- for fanout 1 nets, $C_i = 1.4633*d$,
- for fanout 2 nets, $C_i = 2.4892*d$,
- for fanout 3 nets, $C_i = 3.1716*d$, and
- for fanout 4 nets, $C_i = 3.6234*d - 0.6789$

To obtain a net capacitance estimate for a net with fanout $n \geq 5$, the following formula is used:

$$C_{est}(n) = \left\lfloor \frac{n}{4} \right\rfloor \cdot C_{est}(4) + C_{est}(n \bmod 4) \quad \textbf{(2)}$$

This simplified recursive estimate only applies to nets with fanout greater than 4, which constituted 5% of nets in our benchmark set.

3.3 LUT Power Model

The LUT Power Model accounts for charging and discharging of capacitors inside of a LUT. While the model is consistent with Equation 1, we estimate LUT power in terms of the transition density of LUT output. This is because we have insufficient data to recreate the exact capacitance values for each wire in the hardware implementation of a Stratix II LUT. By using a model related to transition density we can determine the change in power dissipation of a LUT if some of its inputs have their transition densities altered.

To model the power dissipation of a LUT we used ModelSIM-Altera 6.0c to obtain transition density values in transitions per second and performed power analysis using Quartus II 5.1. Using its power analyzer we determined the power dissipated inside each individual LUT (P_{LUTy}) and graphed it versus the transition density measured in millions of transitions per second (x). A second order approximation of the average dynamic power dissipated by a LUT is given by the following equation:

$$P_{LUTy}(x) = 6.0*10^{-6}x^2 - 9.0*10^{-5}x + 0.0091 \quad \textbf{(3)}$$

3.4 FF Power Model

The final power model accounts for the power dissipated by a FF, which we will insert into the circuit to filter out glitches. It includes the power dissipated by the logic components used to create the FF as well as the power dissipated by the clock network due to the increase in capacitive load. The power dissipated by a FF will have to be outweighed by the power saved due to the removal of glitches for the optimization to be effective.

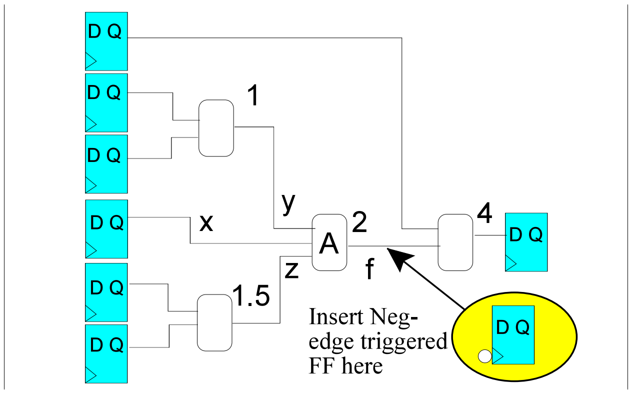

Figure 3: Negative Edge Triggered FF Insertion Example

To account for FF power we related the power dissipated by the FF to the clock frequency used to trigger it. We ran experiments to determine how much power FFs use. In addition, we measured the power consumed by two alternate glitch filters - a Gated LUT and a gated D latch. A Gated LUT is a LUT that has its output either ANDed or ORed with the clock signal. In an FPGA it is implemented inside of a single LUT by altering the function a LUT implements. A gated D latch on the other hand is a level sensitive storage element, such that when clock input is 1 it is transparent, while when clock is 0 it maintains its logic state.

To estimate the power dissipated by each of the above glitch filters, we implemented them on a Stratix II FPGA and measured the power they dissipate using the Power Play Power Analyzer in Quartus II 5.1. The comparison of power dissipated by these filters is shown in Figure 2. The graph shows power dissipation as a function of clock frequency. A gated D latch takes the most power, because we needed to use a LUT with feedback to create a gated D latch, thus its implementation is far from optimal in terms of power dissipation. An nFF and a gated LUT dissipated 33% and 50% less power than a gated D latch, respectively.

4. GLITCH REDUCTION

Glitches occur because input signals to a LUT arrive at different times, causing intermediate transitions to occur before the LUT output stabilizes. We propose to insert an nFF at an output of a LUT that produces glitches to prevent these glitches from propagating to the routing network and causing subsequent glitches down stream. This technique ensures that during the first half of a clock cycle the output net retains its old value and is updated during the latter half of the clock cycle.

Consider for example a LUT network in Figure 3. The clock period is 5 and the arrival times of each LUT output are denoted on the right hand side of each LUT. Now suppose that the routing delays are such that input signals x, y and z arrive at the inputs of LUT A in order *x*, *y* and *z*. Now suppose that wires *x*, *y* and *z* have values 0, 1 and 0 respectively, and they will all toggle in the current clock cycle. Because a change in signal *x* will be visible to the LUT before a change in signals *y* and *z*, the LUT will initially perceive the input to have changed from 010 to 110 and produce a new output value for wire *f*. Subsequently, a change in the value of wire *y* will occur, causing the LUT to again evaluate a new output value for the input sequence 100. Lastly, the wire *z* will change value and a final output value will be generated for input sequence 101. Thus, the output of LUT A changes value three times, two of which are unnecessary.

To remedy the problem, an nFF can be inserted at the output of LUT A. During the first half of the clock cycle the output of the nFF remains unchanged, and only changes once in the second half of the clock cycle when the output wire assumes that value of the output of LUT A. The wire *f* now retains its previous value until the clock signal becomes zero and only changes once thereafter. This operation reduces the number of times the output wire toggles by two, thereby reducing power dissipation.

Alternatively we can use a gated LUT or a gated D latch to achieve a similar effect. A gated D LUT can be used because we can AND (or OR) the LUT function with the clock signal, inverted if using an AND gate. It will force the output of the LUT to 0 (with AND) or 1 (with OR) for the first half of the clock cycle, masking the glitches present there. The downside is that if we use an AND gate and the output of the LUT would remain at logic 1 (or 0 if using an OR gate) for two consecutive clock cycles, this technique would cause a glitch. We found that even though a gated LUT uses least power (section 3.4) the additional glitches it creates outweighed the power savings it provided.

A gated D latch was also considered. Rather than latching data at the negative edge of the clock, it would become transparent during the latter half of the clock cycle. The principle of its operation in this context would be the same as for the nFF. However, according to our analysis in Section 3.4, a gated D latch implementation on a Stratix II device takes more power than a nFF. This option is therefore not as attractive as using an nFF.

The cost of the nFF insertion is the increase in power dissipation due to the presence of an additional FF, including the increase in the capacitive load imposed on the clock network, and a possible increase in the critical path delay. The area penalty was found to be negligible (see Section 6), because an LC already contains a FF, though it may be unused. Thus, choosing to insert an nFF required the use of few additional ALMs on a Stratix II device.

5. OPTIMIZATION ALGORITHM

In the previous two sections we described how glitches can be detected and how their presence in a logic circuit can be reduced. We use the aforementioned data to create an algorithm to reduce power dissipated due to glitches. The algorithm is rather straightforward:

1. Scan all nets in a logic circuit to determine if any of the optimizations from Section 4 can be applied
2. Analyze the resulting set of nets to determine the benefit of applying the optimization to each net
3. Apply the optimization to a net on which the most power could be saved

The effort to find wires suitable for optimization is guided by a cost function that determines if glitch reduction on a particular net is beneficial. It consists of three components: the power saving, the power cost, and the delay penalty.

The power saving component determines how much power can be saved on a particular net, and its transitive fanout, when the optimization is applied. The power saving for a given net is:

$$P_{save}(y) = \frac{V^2}{2} * C_y * Trans_{save}(y) + P_{transitive}(y) \quad \textbf{(4)}$$

where C_y is the capacitance of net *y*, $Trans_{save}(y)$ is the average number of transitions per cycle saved by the optimization, $P_{transitive}(y)$ is the power saved in the transitive fanout of net *y*, and V is the power supply voltage (1.2V). C_y is computed as described in Section 3.2. The $Trans_{save}(y)$ component depends on the applied optimization. For the nFF insertion it is equal to:

$$Trans_{save}(y) = D(y) - P_t(y) \quad \textbf{(5)}$$

and indicates that all glitches are removed from the net.

The $P_{transitive}(y)$ component is computed by calculating how the power dissipated in the transitive fanout of net *y* is affected by removing glitches from wire *y*. It is done by summing the change in power dissipation of each LUT and its output net using models described in Section 3.

For the optimization to be effective the power saving associated with the transformation must outweigh the power consumed by the added circuit elements ($P_{cost}(y)$). Two factors contribute to the power cost: their power dissipation of a FF as described in Section 3.4, and the need for translocation of existing FFs. The second factor requires further explanation.

In an FPGA an LC can be configured so that the LUT inside it drives a FF, or vice-versa. In the latter case it is necessary to move the FF to another LC to accommodate the nFF. This causes an increase in capacitance on the net driven by the moved FF as the signal source will be placed farther away from its destination. The expected increase in the power dissipation for this net is therefore added to the $P_{cost}(y)$ term.

Using the definitions of $P_{save}(y)$ and $P_{cost}(y)$ we define the power saving component of the cost function as $P_{save}(y)$-$P_{cost}(y)$. It represents the expected change in power dissipation if a nFF is inserted at the output of LUT y. Note that the static power is not included in this computation. This is because inserting a FF in the FPGA context means activating it, as it already exists on the silicon fabric. Because it is activated in an LC (or ALM on a Stratix II FPGA) that is already powered up, the FF itself is being powered and hence dissipates static power, even when not used.

The final component of the cost function is the delay penalty. The delay penalty arises when we apply an optimization and it changes the delay profile of the circuit. For example, inserting an nFF requires that paths, which either start or terminate at that FF, need to compute in less than half of a cycle. While we can compute the delay penalty for each net, we allow the user to specify how much delay penalty, Δt, is acceptable to save power.

Using the above components we can fully formulate the cost function. In its simplest form it is defined as:

$$\Delta C = [1 - u(\Delta t_e - \Delta t)] * [P_{save}(y) - P_{cost}(y)] \quad \textbf{(6)}$$

where Δt_e is the expected increase in the minimum clock period for the circuit, u(t) is the unit step function, $P_{save}(y)$ is the power saving component, and $P_{cost}(y)$ is the cost of applying the optimization. The ΔC function is 0 when the expected delay penalty exceeds Δt. Otherwise, the function returns the expected change in power dissipation that will be observed if the optimization is applied. The optimization is applied when $\Delta C > 0$.

6. EXPERIMENTAL RESULTS

To evaluate the effectiveness of our glitch reduction technique, we applied it to several benchmark circuits distributed with the QUIP 5.0 [14] and from OpenCores.org. We now describe the experimental setup, methodology and the results we obtained.

Table 1: Benchmark circuits before and after optimization

Circuit Name	Sim. Freq. (MHz)	Area(# of ALMs)			Critical Path Delay (ns)			Dynamic Power (mW)		
		Initial	Final	%Change	Initial	Final	%Change	Initial	Final	%Change
barrel64*	200	337	342	1.46	4.386	4.806	8.74	229.94	189.7	-17.50
mux64_16bit	275	608	608	0	3.052	3.052	0	389.24	389.2	0.00
fip_cordic_rca	125	182	182	0	7.551	7.851	3.82	43.28	39.49	-8.76
oc_des_perf_opt	290	1343	1356	0.96	2.989	3.07	2.64	1058.8	796.7	-24.75
oc_video_comp_sys_huffman_enc	260	212	214	0.93	3.626	3.626	0	94.88	95.19	0.33
cf_fir_24_8_8	170	1401	1401	0	5.375	5.71	5.87	290.41	292.9	0.84
aes128_fast	140	2458	2514	2.23	6.251	6.569	4.84	879.24	870.6	-0.99
rsacypher	140	419	419	0	6.376	6.563	2.85	50.73	48.22	-4.95
Average				**0.7**			**3.6**			**-7.00**

6.1 Setup

The experimental setup consists of 8 circuits synthesized, placed and routed with Quartus II 5.1. For each circuit the target clock frequency was set to 1 GHz and the optimization technique was set to *balanced*. The power optimization option was set to *Normal*, indicating that power optimization is performed as long as the design performance is maintained. All circuits compiled this way were then simulated using ModelSIM-Altera 6.0c. The simulation was performed using a random set of 10000 input vectors over a period of 10000 clock cycles, and the clock frequency was set to approximately 15% below maximum clock frequency. The simulation step was set to 10ps. The simulation result, in the form of a Value Change Dump File (.vcd), was then used by the Quartus II 5.1 to determine the dynamic power dissipated by each circuit. The *Enable Glitch Filtering* option was turned ON to enable the power analyzer to account for inertial delay of logic components in a circuit.

In Table 1 the name of each circuit in column 1, the clock frequency used to simulate it in column 2, and the area each circuit occupies in terms of Arithmetic Logic Modules (ALMs) [15] in column 3. The critical path delay and power dissipation are listed in columns 6 and 9 respectively. The power dissipation does not include I/O power, because most of the circuits are small. Most of these circuits would be sub-circuits in a larger design and hence their I/O pins would become internal wires.

6.2 Methodology

For the purposes of power optimization each circuit was first synthesized, placed and routed using Quartus II 5.1. The timing analysis was then performed to compute arrival time for all nodes, which is subsequently used to compute toggle rates of all circuit wires. The algorithm from Section 5 was then applied to each circuit, allowing only a 5% critical path delay penalty. Once the algorithm was applied any modified net connections were rerouted. The resulting circuit was then timing analyzed and simulated using ModelSIM-Altera 6.0c. In each case we used the same simulation clock frequency as before the optimization, and ensured that the clock frequency is lower than the maximum clock frequency for the original and the modified circuit. The simulation results were used to obtain power estimate using Quartus II 5.1 Power Play Power Analyzer.

6.3 Results

We applied the glitch reduction optimization to 8 benchmark circuits. The results are summarized in Table 1. Table 1 shows the name of each circuit in column 1, the final size of the circuit in terms of ALMs in column 4, the critical path delay in column 7, and the dynamic power dissipated by the circuit in column 10. The change in area, delay and power is shown in columns 5, 8 and 11, respectively. The percentage change in these columns was computed using the following formula:

$$\%difference = 100\frac{final - initial}{\max(final, initial)} \qquad (7)$$

Table 1 shows that inserting negative edge triggered FFs into a circuit is an effective way of reducing the impact of glitches. We see a 7% reduction in power dissipation, verified by a commercial tool through simulation, at a cost of 3.6% in critical path delay.

6.4 Discussion

In general, the insertion of an nFF is successful in reducing the number of glitches in a circuit at a cost of increased critical path delay. We obtained good power reduction results in most of the circuits tested, ranging from 1% to 25% in power reduction.

The power savings come at a cost of circuit delay. The delay increase is due to the delay introduced by inserting an nFF. Another factor is the majority of glitches were found on near-critical paths because they generally consisted of the largest number of LUTs. However, for the nFF insertion to be effective, a target LUT must have the output arrival time less than half a clock period and the required time above half a clock period. For near-critical paths this margin is small, thus it is difficult to find candidates for optimization without incurring a delay penalty.

The largest power reduction was observed in the *oc_des_perf_opt* circuit. This circuits is an implementation of a Data Encryption System optimized for performance on Altera devices. The circuit contains a large number of XOR gates with a large number of unbalanced paths. Because an XOR gate allows all glitches to propagate through it, removing glitches on one net causes a large number of glitches to disappear from nets in its transitive fanout.

The second largest power reduction was observed in the *barrel64* circuit. In this circuit 56 nFFs were added to produce a 12.7% power reduction. A reduction of 17.5% in power was also possible in this circuit, when a delay penalty of 8.7% was allowed. Given that most FPGA circuits run at speeds under 200 MHz, sacrificing more speed in this circuit may be acceptable. Other circuits that operated at above 200MHz did not produce larger power savings when a 10% delay penalty was allowed.

The power reduction of 1% was observed on the *aes128_fast* circuit, which is another cryptographic circuit. A detailed analysis of this circuit revealed that we saved 34.14mW (9.2%) of power in the routing alone. However, the saving came at a cost of 24.6mW for adding 173 negative edge triggered FFs and 1.86mW for routing the clock signal to Logic Array Blocks (LABs) in which a clock signal was not needed previously. While the optimization was successful in reducing the toggle rate of each wire to which it was applied by 50-70%, the capacitance on most of these nets was too low to significantly affect the core power dissipation. In fact, most of the net connections in this circuit were realized through short wires inside of a LAB. This is an excellent example of how power dissipation was reduced by using lower capacitance wires, rather than through toggle rate reduction.

For *cf_fir_24_8_8* and *oc_video_comp_sys_huffman_enc*, the power dissipation increased slightly. In these circuits the toggle rate was overestimated, however not significantly enough to cause a large power dissipation increase. Finally, the *mux64_16bit* circuit saw no power, area or speed change. The reason for this rests in the very low presence of glitches. We were able to estimate the toggle rate of all wires almost perfectly and since the circuit consisted primarily of short wires, the cost function used in the algorithm did not recommend the insertion of an nFF.

6.5 Related Works

The works in [10] and [16] are the closest to our approach. The problem of glitches is discussed in both papers and the research presented there indicates that retiming is a good way to rebalance path delays and at the same time reduce the appearance of glitches. While the general idea is similar, that is to insert FFs at glitchy nodes, the process of FF insertion is quite different.

A retiming algorithm rebalances path delays and maintains circuit latency. This is accomplished by pushing FFs from the output of a LUT to its inputs (push-back), or vice-versa (push-forward). The push-back operation removes a single FF from the output of a LUT and adds a FF on each net driving the LUT to preserve circuit latency. However, if one of these nets has fanout greater than 1, then it is necessary to apply push-back operation to all LUTs that net drives. This can potentially affect an entire circuit. A similar effect occurs in a push-forward operation.

On the other hand, the algorithm presented here only needs to ensure that any FF-to-FF path contains at most one negative edge triggered FF. The process of insertion therefore becomes a local operation and need not affect the rest of the circuit, and the cost of inserting an nFF is limited to the power dissipated by that FF.

7. CONCLUSION

This paper addressed glitches post-routing, where it is possible to use post-routing delay data to identify nets in the circuit prone to glitching. We presented an algorithm for reducing glitches by inserting a negative edge triggered FFs at the output of LUTs that produce glitches thereby preventing the unnecessary logic transitions from propagating to the general routing network and subsequent LUTs.

The algorithm is successful in reducing power by an average of 7%, and up to 25%, on a commercial FPGA device, the Altera Stratix II. The algorithm has an advantage over pipelining in that the latency of the circuit remains unchanged. Also, in contrast to retiming our algorithm only affects a small portion of a circuit at a time, specifically the source of a net, whereas retiming can cause a cascade of push-back/forward operations. Finally, because the insertion of a negative edge triggered FF does not affect the logical operation of the circuit, this technique can be used in tandem with other power optimization techniques.

8. ACKNOWLEDGEMENTS

The authors thank Altera Corporation for assistance during the course of this work and funding this research.

9. REFERENCES

[1] F. Li, D. Chen, L. He, J. Cong, "Architecture Evaluation for Power-Efficient FPGAs," ACM/SIGDA Int. Conf. On FPGAs, 2003, pp.175-184.

[2] N.R. Mahapatra, S.V. Garimella, A. Takeen, "Efficient techniques based on gate triggering for designing static CMOS ICs with very low glitch power dissipation", Proc. Of the Int. Symp. On Circ. & Sys., 2000, vol. 2, pp. 537-540.

[3] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey and R. Brodersen, "Optimizing Power Using Transformations," IEEE Trans. on CAD of Int. Circ. and Sys., Vol. 14, No. 1, Jan. 1995, pp. 12-31.

[4] A. Shen, A. Ghosh, S. Devadas and K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks," Proc. Of Conf. On CAD, 1992, pp. 402-407.

[5] KIW Kim, T Kim, TT Hwang, SM Kang, CL Liu, "Logic Transformation for Low-Power Synthesis," ACM Trans. on Des. Auto. of Electronic Systems, 2002, pp. 265-283.

[6] D. K. Pradhan, M. Chatterjee, M. V. Swarna and W. Kunz, "Gate-Level Synthesis for Low-Power Using New Transformations," Proc. Of Int. Symp. On Low Power Electronics and Design, California, 1996, pp. 297-300.

[7] J. H. Anderson and F. N. Najm, "Power-Aware Technology Mapping for LUT-based FPGAs," Proc. Of Field-Prog. Tech. Conf., Hong Kong, 2002, pp. 211 - 218.

[8] C. Yeh, C.-C. Chang and J.-S. Wang, "Power-driven technology mapping using pattern-oriented power modelling," IEE Proc.-Comp. Digit. Tech., Vol. 146, No. 2, March 1999, pp. 83-89.

[9] A.H. Farrahi and J. Sarrafzadeh, "FPGA Technology Mapping for Power Minimization," Int. work. on Field-Programmable Logic, 1994, pp. 66-77.

[10] J. Leijten, J. van Meerbergen, J. Jess, "Analysis and reduction of glitches in synchronous networks," Proc. of the European Design and Test Conf., Paris, 1995, pp. 398-403.

[11] J. H. Anderson and F. N. Najm, "Switching Activity Analysis and Pre-Layout Activity Prediction for FPGAs," ACM/IEEE Workshop on SLIP, 2003, pp. 15 - 21.

[12] F. Najm, "Transition Density: a new measure of activity in digital circuits," IEEE Transactions on CAD of Integrated Circuits and Systems, February 1993, Vol. 12, pp. 310-323.

[13] C. Tsui, M. Pedram and A. M. Despain, "Efficient Estimation of Dynamic Power Consumption under Real Delay Model," IEEE Int. Conf. On CAD, 1993, pp. 224-228.

[14] Altera Quartus University Interface Program 5.0 package. URL: http://www.altera.com/education/univ/research/unv-quip.html. Last Accessed March 1, 2006.

[15] Altera Stratix II datasheet, URL: http://www.altera.com/literature/lit-stx2.jsp

[16] J. Monteiro, S. Devadas and A. Ghosh, "Retiming Sequential Circuits for Low Power," Proc. of ICCAD, 1993, pp. 398-402.

[17] M. Hashimoto, H. Onodera and K. Tamaru, "A Practical Gate Resizing Technique Considering Glitch Reduction for Low Power Design," Proc. Of the 36th ACM/IEEE Conf. On Design Automation, 1999, pp. 446-451.