Constrained Clock Shifting for Field Programmable Gate Arrays

Deshanand P. Singh
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, Canada
singhd@eecg.toronto.edu

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

ABSTRACT

Circuits implemented in FPGAs have delays that are dominated by its programmable interconnect. This interconnect provides the ability to implement arbitrary connections. However, it contains both highly capacitive and resistive elements. The delay encountered by any connection depends strongly on the number of interconnect elements used to route the connection. These delays are only completely known after the place and route phase of the CAD flow. We propose the use of Clock Shifting optimization techniques to improve the clock frequency as a post place and route step.

Clock Shifting Optimization is a technique first formalized in [4]. It is a *cycle-stealing* algorithm that allows one to reduce the critical path delay of a synchronous circuit by shifting the clock signals at each register. This technique allows late arriving signals to be sampled at a later point in time by intentionally introducing a skew on the clock input of the sampling register. Typical FPGAs contain a number of special purpose global clock networks that distribute clock signals to every register in the chip. Unused global clock lines in FPGAs can be used to distribute a finite set of clock skews to the entire circuit. We propose an efficient integer programming method to find the optimal circuit improvement for a finite set of clock skews. This technique is modified to consider inherent uncertainties present in the timing models. The uncertainty controls the aggressiveness of the optimizations as we must take great care in ensuring functionality for any range of possible timing characteristics.

Our results confirm intuition that more aggressive speed optimizations can be performed as timing models become more accurate. We also show that providing 4 skewed versions of the nominal clock signal results in the best delay–area tradeoff. This result is evocative as it may suggest future FPGA architectures that contain greater numbers of global clock lines, as we tradeoff gains in speed for greater power requirements from increased clock network flexibility.

FPGA'02, February 24-26, 2002, Monterey, California, USA.

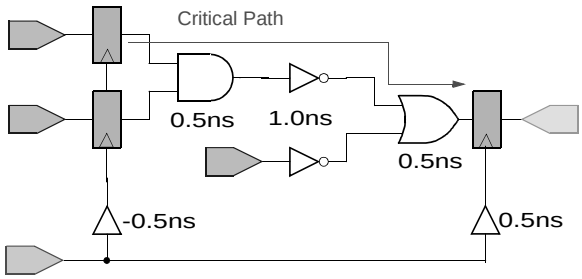

Figure 1: Clock Shifting Example.

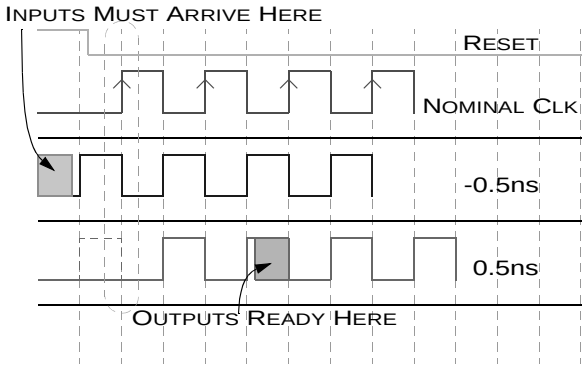

Figure 2: Clock Shifting Waveform.

1. INTRODUCTION

FPGA designers face the dilemma that their designs are dominated by connection delays routed along programmable interconnect elements. For many designs, these delays are impossible to predict until the actual placement and routing steps are complete. If a design does not meet timing because of these interconnect delays, many iterations of adding placement constraints or re-synthesizing logic may be necessary to reduce critical delays. In this paper, we propose the use of a clock shifting circuit optimization technique that can be automatically applied after the circuit has been completely placed and routed.

In a conventional FPGA architecture [1] [9], the length of the longest stretch of interconnect used can be a significant factor in determining the maximum operating frequency. Strategies to cope with these delays include the use of rescheduling techniques for data flow so that signals

are allowed one or more clock cycles to traverse these long stretches of interconnect. One method of rescheduling these operations is through the use of sequential retiming along with registers within the routing fabric of the FPGA. Studies such as [8] proposed methods for moving registers around the FPGA after the place and route phases of the FPGA CAD flow.

Another approach of rescheduling is that of clock shifting optimization. This optimization moves clock signals in time rather than moving registers in the space of the circuit netlist. Figures 1 and 2 show an example of this optimization on a simple synchronous sequential circuit. The circuit in Figure 1 has a critical path delay of 2ns. However, the circuit can be clocked with a period of 1ns if we shift the clock backwards by 0.5ns at the leftmost registers and shift the clock forward by 0.5ns at the rightmost registers. For the circuit to function correctly, the inputs must arrive early enough so that we may steal the extra 0.5ns of time on the input side. Similarly, the external circuitry must have a small setup time so that we are allowed to wait an additional 0.5ns before clocking the rightmost register. Thus, the ability to reduce the clock period does not come for free. We must steal bits of slack from inputs and outputs and then propagate it through the circuit.

FPGAs seem like a natural environment to implement this type of optimization since they contains several prefabricated global clock nets that connect to every register in the circuit. This paper seeks to explore the feasibility of applying clock shifting for circuits implemented in FPGAs.

The rest of this paper is organized as follows: Section 2 briefly provides background information on synchronous clocking in the presence of clock skew. Section 3 describes how clock shifts can be introduced in FPGAs. Section 4 describes the fundamental uncertainties in the timing models for various components within FPGAs. Sections 5 and 6 describe an algorithm to find a set of clock shifts compatible with the FPGA architecture such that the clock period of the circuit is optimized. Section 7 details our experimental methodology along with a discussion of the results. Section 8 presents our conclusions and plans for future work.

2. SYNCHRONOUS OPERATION WITH CLOCK SKEW

In this section, we briefly review the fundamentals of synchronous circuits operating in the presence of clock skew. Consider two registers r_i and r_j that are connected by some combinational path. Let the clock skew at r_i be designated by SK_i and the clock skew at register j be designated by SK_j. In addition, let $MAX(i,j)$ represent the maximum combinational path delay from r_i to r_j. Similarly, let $MIN(i,j)$ represent the minimum delay from r_i to r_j.

The **Zero-Clocking** constraint is represented by Eq. 1.

$$SK_i + MAX(i,j) \leq C_p + SK_j \quad (1)$$

This equation expresses that the signal from r_i propagates to r_j at $SK_i + MAX(i,j)$ relative to the active clock edge. The signal is sampled one clock period C_p later plus the skew SK_j at r_j. Clearly the arrival time must be less than or equal to the sampling time for correct synchronous operation.

The **Double-Clocking** constraint is represented by Eq. 2.

$$SK_i + MIN(i,j) \geq SK_j \quad (2)$$

In a zero skew environment, synchronous circuits operate correctly because they rely on the combinational elements between registers to hold its value for a small amount of time after the clock arrives. This allows the register to latch its input value reliably on the clock edge. In the presence of clock skew, a situation may occur where we wait too long and the input value that was to be sampled has disappeared. Eq. 2 formalizes this constraint. The time where the input value could disappear is the time at which r_i changes its value plus the minimum propagation time to get from r_i to r_j or $SK_i + MIN(i,j)$. For correct operation, the register r_j must sample its input before this time.

3. APPLICATION TO FPGAS

FPGAs are unique architectures in that they have several prefabricated global clock networks that can distribute signals to every register in the circuit. Several networks are necessary so that multiple clock domain circuits could be efficiently mapped to the FPGA. These clock networks can also be used to distribute skewed versions of a single clock to the various registers in the circuit. This concept is illustrated in Figure 3. However the number of distinct skewed clock signals is limited by the number of clock lines that are available in the target architecture. This number is denoted by L throughout this paper.

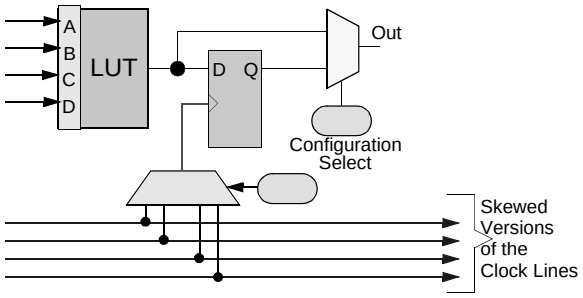

Figure 3: Typical FPGA Logic Block.

The circuitry that would provide the clock shifts is shown in Figure 4. This circuit provides only positive clock shifts (skewed versions of the clock arrive only after the nominal clock). The PLL along with the divider is a commonly used structure to multiply the frequency of the clock. The multiplied version of the clock is used to drive a simple shift register that samples a clock operating at the input clock frequency. In this way, we can tap various shifted versions of the clock by sampling outputs of the shift register. Variable taps are implemented by the multiplexer circuitry connected to the shift register. The system shown is very similar to the internals of many digital PLL implementations. It is unclear whether the extra shifting circuitry could be operated reliably at high speeds when implemented in programmable logic. Specialized hardware may be required. Notice also that the asynchronous reset line for the circuit is also connected to the phase shifting mechanism. Whenever the reset signal is activated, the phase shifting flip-flops are also reset. Resetting these flip-flops ensure that once the reset signal is de-asserted, the registers in the circuit are clocked in the correct order. Active edges of the shifted clocks are only asserted after the first active edge from the nominal clock.

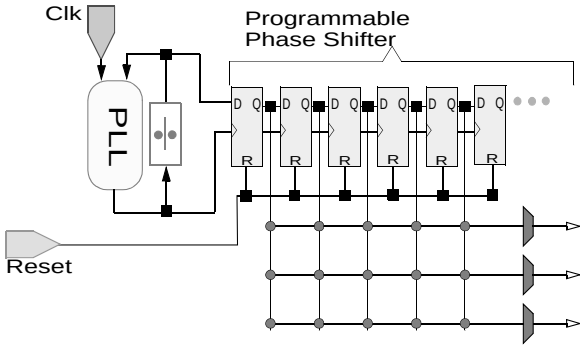

Figure 4: Phase Shifting Circuitry.

4. TIMING UNCERTAINTIES

One of the great problems in utilizing Clock Shifting Optimization for FPGAs are the uncertainties present in the timing models. The exact timing of the chip fabricated in silicon is determined by a number of factors. Among these are operating temperature, supply voltage, and the variance of parameters in the fabrication process. For example consider the situation shown in Figure 5. Although any stretch of metal interconnect has a value for its width, height and thickness, there can be significant deviations during the fabrication process. These deviations affect the equivalent resistance and capacitance for this stretch of metal, and hence the signal propagation time. Similar variations also affect gate delays as transistor drive strengths and parasitic capacitances also vary; however, these metal lines have a larger impact on the final delay.

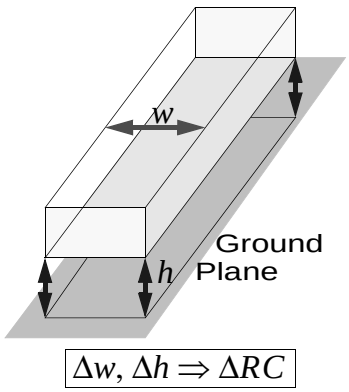

Figure 5: Fabrication Uncertainties.

We propose the use of a simplistic lumped uncertainty model for timing in FPGAs. The delay of any path $T(P)$ can be characterized using average or expected parameters. However because of the various effects discussed above, it is possible for the actual delay to deviate significantly from this nominal value. We introduce two parameters p and q that govern the uncertainty bound for the timing on any given path. The best case timing for a particular path is given by $p \cdot T(P)$, while the worst case timing is given by $q \cdot T(P)$. This relationship is expressed in Eq. 3.

$$(T_{min} = p \cdot T(P)) \leq T(P) \leq (T_{max} = q \cdot T(P)) \quad (3)$$

The difference between p and q provides an indication of the timing model accuracy. If p and q are both *close* to 1, then there is little deviation from the delay predicted by the timing model. We will define the timing uncertainty U as

$$U = 1 - \frac{p}{q} \quad (4)$$

This quantity simply represents the percent difference between the worst case bound and the best case for any given combinational path. We choose this parameter because FPGA vendors often present us with conservative values for this parameter.

5. FPGA CLOCK SHIFT DECISION PROBLEM

This section introduces the decision problem that can be utilized to optimize the maximum operating frequency of a circuit implemented in a FPGA. The parameters passed to the decision problem are:

- The timing uncertainty parameters p, and q.
- The target clock period C_p.
- The number of global clock lines available L.
- An array containing the clock shifts on each of these clock lines $V[0] \ldots V[L-1]$

Given these parameters, the decision function will return a single boolean value. The value is **true** if the target clock period can be realized given the uncertainties, clock lines and shifts available. First we will show how this decision problem can be expressed, and then show an efficient algorithm for solving it.

5.1 Decision Problem Definition

For every register i, we associate an integer shift identifier $s(i)$. This shift identifier indicates which clock line the register will be clocked from. This also implies that valid solutions of $s(i)$ lie between 0 and $L-1$. For example, a shift identifier $s(i) = 2$ indicates that the register i will be clock from global clock line number 2. From our previous definitions, the actual clock shift at register i will be $V[s(i)]$. The value of $s(i)$ actually provides us the select configuration bits for the clock multiplexer shown in Figure 3.

The circuit must obey a number of clocking constraints if it is to operate properly. These constraints are simply modified versions of those formalized by Fishburn [4]:

$$V[s(i)] + q \cdot MAX(i,j) \leq V[s(j)] + C_p \quad (5)$$

$$V[s(i)] + p \cdot MIN(i,j) \geq V[s(j)] \quad (6)$$

Equations 5, 6 are the new zero and double clocking constraints. They are identical to the original formulation except that the individual skews can now take on only certain discrete values. If we can find a labeling for each $s(i)$ such that it satisfies the clocking constraints, then the decision function is true. More formally:

Problem CSDP

- *Let p,q, and C_p be positive real numbers that represent the timing uncertainties, and target clock period.*
- *Let R be a set of registers in the synchronous circuit under consideration.*

- *Let V be a clock shift mapping function from integers to positive real numbers.*

Find a mapping function s from R to integers, such that Eq. 5, 6 are satisfied, or determine that no such mapping function exists.

5.2 Solving the Decision Problem

For a given set of clocking constraints, the decision problem can be solved as a special case integer program based on a discrete version of the Bellman-Ford algorithm. It uses an iterative relaxation based approach. Notice that constraint Equations 5, 6 both have the following general form:

$$V[s(i)] \geq V[s(j)] + K_{ij} \tag{7}$$

We shall denote this constraint in the form $C(i, j, K_{ij})$. The core algorithm is shown in Figure 6. The parameters p,q, and C_p are used to create a set of clocking constraints S.

function *CSDPcore*(**ConstraintSet** S, **Skews** V)
begin
  Order the skews in non-decreasing order
    $V[0] \leq V[1] \leq \cdots \leq V[L-1]$
1: $\forall i, s_{lb}(i) = 0$
  do
2: $C(i, j, K_{ij}) =$
      {**any unsatisfied constraint in S**$|s(i) = s_{lb}(i)$}
3: $rhs_{lb} = V[s_{lb}(j)] + K_{ij}$
4: **Increment** $s_{lb}(i)$ **until** $V[s_{lb}(i)] \geq rhs_{lb}$
  until *all constraints are satisfied* **OR** $s_{lb}(i) \geq L$

  if *all constraints are satisfied*
  then
    $\forall i, s(i) = s_{lb}(i)$
    return SOLUTION EXISTS
  else
    return NO SOLUTION
  end if.
end function.

Figure 6: Clock Shift Decision Problem

Definition 1. Let $s_{min}(i)$ denote the minimum value of $s(i)$ among values of $s(i)$ which participate in feasible solutions of the CSDP.

Theorem 1. If the CSDP has a solution, then every iteration of the CSDPcore algorithm maintains the loop invariant that $\forall i, s_{lb}(i) \leq s_{min}(i)$.

PROOF. First assume that CSDP has a solution, then we may use an inductive argument to prove the invariant. The base case is trivially true as all values of s_{lb} are initialized to 0 in line 1 of CSDPcore. To prove the inductive hypothesis, assume that bound condition holds on the previous iteration of the loop. This assumption implies that the rhs_{lb} variable on line 3 is a lower bound on the value of the right-hand side of the chosen constraint since $s_{lb}(j)$ is a lower bound on any feasible value of $s(j)$ and V is ordered in non-decreasing value. Suppose that incrementing the $s_{lb}(i)$ variable on line 4 violates the loop invariant. This implies that $s_{min}(i) < s_{lb}(i)$. However the exit condition in line 4 enforces that any value of $s(i) < s_{lb}(i)$ has a value that violates the condition that $V[s(i)] \geq rhs_{lb}$ (i.e., $V[s(i)]$ is lower than the lower bound on the right-hand side of the constraint C). Thus this value of $s(i)$ could not possibly participate in any feasible solution to the CSDP. This contradicts our proposal that line 4 violates the loop invariant. □

Theorem 2. If the CSDP has a solution, then the algorithm CSDPcore returns a feasible solution. If the CSDP has no solution, then the algorithm CSDPcore returns no solution.

PROOF. **Case I:** *CSDP has a solution implies CSDPcore returns a feasible solution.* Assume that CSDP has a feasible solution, and that CSDPcore returns no solution. This is only possible for some value $s_{lb}(i) \geq L$. However from Theorem 1, we know that $s_{lb}(i)$ is a lower bound on all feasible $s(i)$ values. A value $s_{lb}(i) \geq L$ contradicts the fact that feasible solutions range from $0 \leq s(i) \leq L-1$.
Case II: *CSDP has no solution implies CSDPcore returns no solution.* If the CSDP has no solution then it is impossible that the **all conditions satisfied** condition can be satisfied in CSDPcore. Hence after at most, $|R| \cdot L$ iterations, some value of $s_{lb}(i)$ will be greater than or equal to L. This will cause CSDPcore to return no solution. □

The CSDPcore algorithm runs in $O(|R||S| \cdot L)$ time. This computation includes $|R| \cdot L$ iterations along with the fact that finding an unsatisfied constraint (line 2) may require a search of all the skew constraints S. For conventional FPGA architectures the value of L is fixed, so the execution time is $O(|R||S|)$. We have implemented our algorithm in such a way that searching for an unsatisfied constraint takes at most $O(|R|)$ (and typically $O(1)$). Thus the worst-case total run time is proportional to the square of the number of registers in the circuit.

6. FPGA CLOCK SHIFT OPTIMIZATION PROBLEM

In this section, we use the CSPDcore algorithm to find the set of skews that find the optimal clock period achievable with L available global clock lines along with the skews that would realize this period.

6.1 Binary Search for Best Clock Period

Figure 7 shows how the optimal clock period for a given set of skews V can be found. The algorithm performs a binary search for the best clock period C_p in the range $(0 \ldots T_{crit})$. T_{crit} is the post place and route critical path delay. The CSDPcore algorithm is used to check if a feasible solution exists for the given skews and the constraint set generated from the uncertainties and the clock period. Each result cuts the following search space in half.

6.2 Finding The Optimal Skew Set

To find the optimal skew set, we simply exhaustively enumerate all possible skew combinations for a given number of available clock lines L. The binary search function described above is used to evaluate each set of skews. To make the search space reasonable (and account for hardware limitations), we impose a minimum granularity on the skew set.

```
function CSOBinarySearch( p, q, T_crit, V )
begin
    ubound = T_crit
    lbound = 0
    while(ubound − lbound ≥ searchthreshold)
        C_p = (ubound + lbound)/2
        create constraint set S using p,q,C_p
        if CSDPcore(S,V)
        then
            ubound = C_p
        else
            lbound = C_p
        end if.
    end loop.
    return ubound
end function.
```

Figure 7: Binary Search for Best Clock Period

The only skews examined are those that are multiples of the minimum granularity. Our studies have used a granularity of $\frac{1}{32}$ of the target clock period. Since a small set of clock lines is usually available, this algorithm is extremely efficient. For example 4 clock lines means the examination of $\binom{32}{4} = 35960$ skew sets. Clearly this algorithm does not scale well, but our experiments show diminishing benefit in using extra global clock lines. In addition, these lines are extremely power and area hungry so FPGA manufacturers would be reluctant to provide architectures with an abundant set of global clock lines.

7. EXPERIMENTAL RESULTS

Our experiments used the circuits described in Table 1. Each of these circuits was mapped into a netlist of LUTs using FlowMap [3]. The resulting netlists were placed and routed using VPR [2]. The delay models were produced from SPICE simulations of a 0.18 micron fabrication process. Each netlist was routed assuming a *low-stress* environment. This term indicates that chip has 20% percent more tracks than the absolute minimum required to route the circuit. The clock shifting optimizations were applied directly to the post-routing netlist.

Figure 8 plots average speedups for the circuits shown in Table 1. The average speedup is plotted against two axes. The X-axis represents the timing model uncertainty $K_{uncertain}$. The Y-axis represents the total number of clock lines available to use, L. The average speedup is plotted on the Z-axis. Also shown, is an estimate of the area penalty from adding extra lines to a FPGA in comparison to the base case of FPGAs with 1 clock-line.

This plot confirms that lower values of timing model uncertainty increase the achievable speedup. As well it shows that increasing the number of clock lines available allows for larger speedup. However, it seems that going beyond $3-4$ lines provides a diminishing return considering the extra area and power that they would consume and that typical uncertainty values are in the $60-70$ percent range [10]. Given these 4 extra lines with an uncertainty of 60%, the average speedup obtained is approximately 14%.

Table 1: Circuit Characteristics

Circuit	Size (LUTs)
bigkey-mcnc	1707
dsip-mcnc	1370
daio-rec	315
diffeq-mcnc	1497
ecc-mcnc	331
elliptic-mcnc	3604
frisc-mcnc	3556
s38417-mcnc	6406
s9234.1-mcnc	505
tseng-mcnc	1047
cordic-oc	1513
hc11-oc	3877
des-fip	15509
fir-filter	921
mult10x10	1162
sisc8	1434

8. CONCLUSIONS

We have shown that clock shifting can be an effective technique to optimize synchronous circuits implemented in FPGAs. In extremely speed-intensive applications, users routinely use logic cell delays to manually shift clock signals at registers with late arriving signals. Our method can automate this procedure using more reliable clock shifts provided by PLL circuitry.

The largest barrier in implementing this technique for FPGAs is that few FPGA manufacturers provide access to the minimum delays through their various components. If numbers are provided, they are usually only conservative estimates. This situation is understandable as there have been few optimization techniques that actually require minimum delay information.

An open question is deciding when clock shifts should be used and when *sequential retiming* [5] [6] should be applied. The disadvantage of retiming is that it may introduce extra registers in order to reduce the clock period. The granularity of a retiming solution is also determined by the largest delay element in a circuit. For example, it is impossible in conventional architectures to physically move a register in between a long segment of routing. However, clock shifting may be able to deal with this situation because we can flexibly change the register's sample point.

In the context of post placement FPGA optimization, a number of factors must be considered. For example if the chip is almost fully utilized, shifts on the global clock lines is perhaps the only form of retiming that will be effective. If the clock lines are all used in a particular design but there are number of unused LUTs, then sequential retiming is probably our only viable rescheduling alternative. Perhaps some combination of these techniques can be applied. Sequential retiming provides a coarse rescheduling of the circuit, while clock shifting methods perform the fine tuning. Our future experiments will focus on the combination of these two techniques.

In addition, we must consider the FPGA architecture itself when determining whether to use retiming or clock shifting. If routing delays can be accurately estimated from placement parameters, then retiming and incremental place-

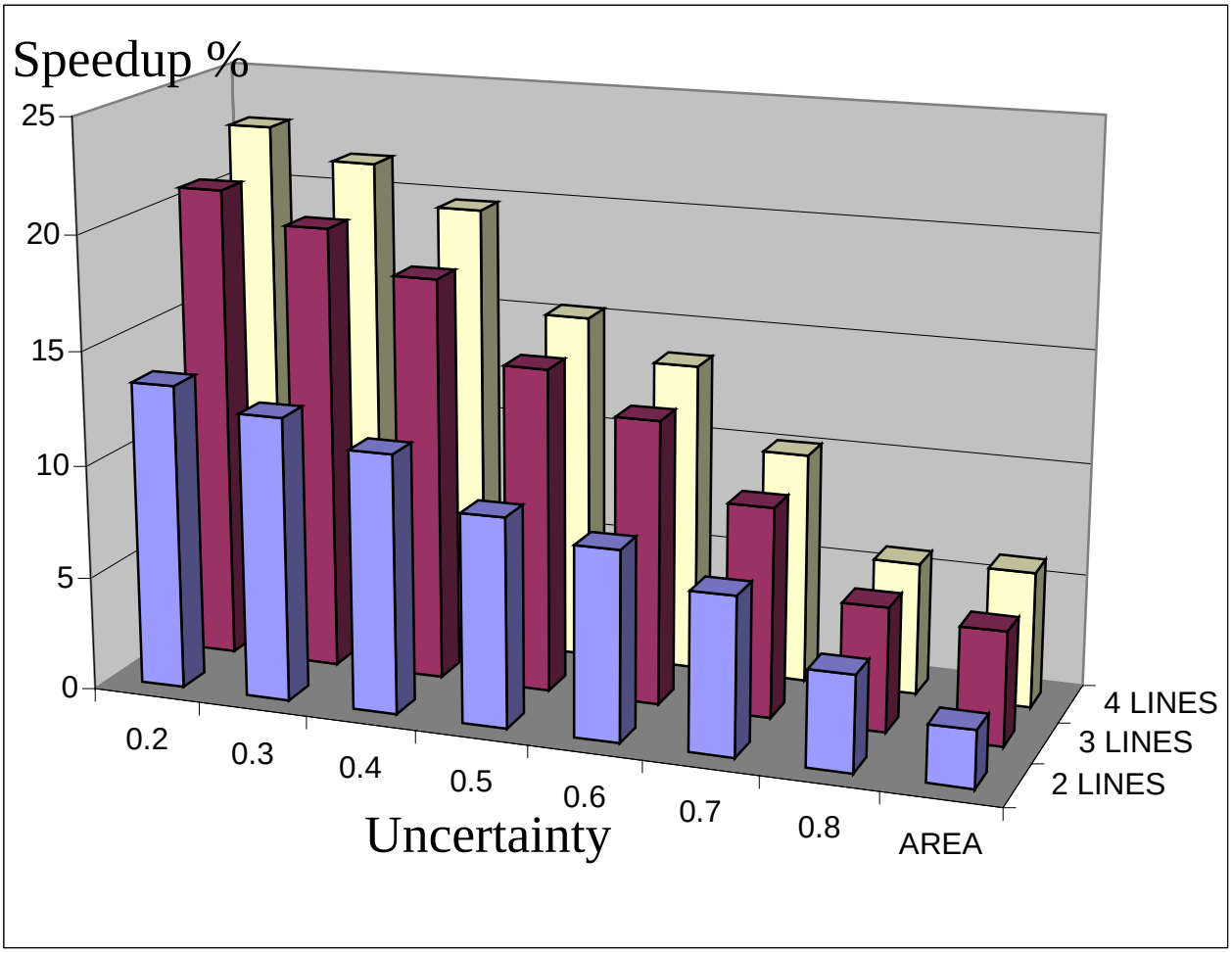

Figure 8: Speedup as a function of Clock Lines and Uncertainty.

ment solutions [7] can be effective. However, if the routing delays cannot be accurately predicted (perhaps because of congestion) then clock shifting is one of the few algorithms that can be effective after the entire place and route flow has been completed.

Finally, the timing model presented in this paper is extremely simple and intended only to show the effects of timing-uncertainty. All of the techniques described can be easily extended to handle upper and lower timing bounds derived from simulation. Effects such as clock jitter and the small amounts of skew present on the global clock lines have also been neglected.

9. REFERENCES

[1] Altera. *Altera 2000 Databook*. Available from: http://www.altera.com/html/literature/lds.html.

[2] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[3] J. Cong and Y. Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on CAD*, pages 1–12, Jan 1994.

[4] John P. Fishburn. Clock Skew Optimization. *IEEE Trans. Computer*, Vol. 39, No. 8, pp. 945–951, July 1990.

[5] C. Leiserson, F. Rose, and J. Saxe. Optimizing synchronous circuitry. *Journal of VLSI and Computer Systems*, pages 41–67, 1983.

[6] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[7] D. Singh and S. Brown. Integrated Retiming and Placement for Field Programmable Gate Arrays. *to appear in* the Tenth International Symposium on Field Programmable Gate Arrays, 2002.

[8] D. Singh and S. Brown. The Case for Registered Routing Switches in FPGAs In *FPGA 2001*.

[9] Xilinx. *Xilinx 2000 Databook*. Available from: http://www.xilinx.com/partinfo/databook.htm.

[10] Xilinx. *A Look at "Minimum" Delays*. Available from: http://support.xilinx.com/xcell/xl21/xl21-40.pdf.