A Comprehensive Approach to Modeling, Characterizing and Optimizing for Metastability in FPGAs

Doris Chen, Deshanand Singh, Jeffrey Chromczak,
David Lewis, Ryan Fung, David Neto, Vaughn Betz
Altera Corporation
151 Bloor Street West, Toronto, Canada
dochen|dsingh|jchromcz|dlewis|rfung|dneto|vbetz@altera.com

ABSTRACT

Metastability is a phenomenon that can cause system failures in digital circuits. It may occur whenever signals are being transmitted across asynchronous or unrelated clock domains. The impact of metastability is increasing as process geometries shrink and supply voltages drop faster than transistor V_ts. FPGA technologies are significantly affected since leading edge FPGAs are amongst the first devices to adopt the most recent process nodes. In this paper, we present a comprehensive suite of techniques for modeling, characterizing and optimizing metastability effects in FPGAs. We first discuss a theoretical model of metastability, and verify the predictions using both circuit level simulations and board measurements. Next we show how designers have traditionally dealt with metastability problems and contrast that with the automatic CAD algorithms described in this paper that both analyze and optimize metastability-related issues. Through our detailed experimental results, we show that we can improve the metastability characteristics of a large suite of industrial benchmarks by an average of 268,000 times with our optimization techniques.

Categories and Subject Descriptors

B.7.3 [**Hardware**]: Integrated Circuits—*Reliability and Test*

General Terms

Algorithms, Measurement, Reliability

1. INTRODUCTION

Metastability is a phenomenon that can occur whenever the setup or hold time of a flip-flop is violated. This can happen when there are either asynchronous signals in the design, or multiple phase-unrelated clock domains which must communicate. Since phase-unrelated clocks have no fixed phase relationship, no static timing analysis constraints can be created for data transfers between these clock domains. Whenever the setup or hold relationship between the source and the destination register is violated, the output of the register has a chance to become *metastable*, a condition where the output voltage is neither high nor low, but hovers at some intermediate voltage for an indefinite period of time. Eventually, the metastable value will resolve to a state of logic-**0** or logic-**1**. This situation is illustrated graphically in Fig. 1. It shows that the input signal to the flip-flop changes during the window defined by t_{su} (FF setup time) and t_h (FF hold time) where no transitions should occur for normal behavior. In this situation, the output value may become metastable and eventually resolve to its new logic value (FF Output (a)) or revert to its old logic value (FF Output (b)). The resolution time happens sometime after the nominally specified t_{co} (clock-transition to output delay) of the flip-flop. If this extra time isn't accounted for with extra timing slack, then failures may occur.

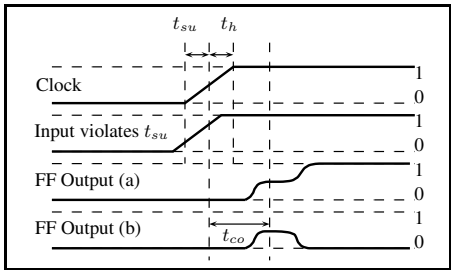


Figure 1: Metastable FF behaviour.

Entering a metastable state is a probabilistic function related to the clock frequency, the transition frequency of the asynchronous data signal and a constant that defines the window in which a transition can cause metastability. Once in a metastable state, the value to which the flip-flop resolves cannot be determined. The amount of time required for resolution is also a probabilistic function whose distribution is defined by a metastability "time constant" for the flip-flop.

The problem with metastable events is not merely their occurrence, but when the event causes inconsistent values to be latched into subsequent flip-flops as shown in Fig. 2(a). In this example, if one flip-flop latches a value of **1** while another latches a value of **0**, then the design can become unpredictable and may fail. This situation may occur because the two paths shown could have different routing delays and it isn't necessarily the case that the resolved value will reach both flip-flops before the next clock sample.

To reduce the chance of metastable events propagating through the design, designers often use a sequence of back-to-back flip-flops, called a synchronizer chain or a synchro-

FPGA'10, February 21–23, 2010, Monterey, California, USA.

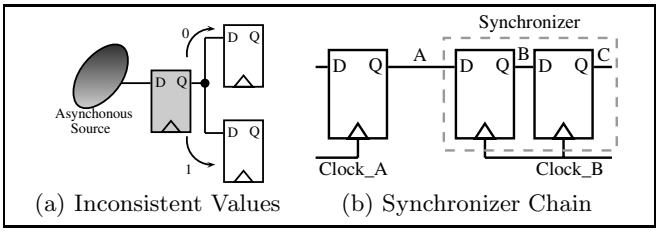


Figure 2: Metastability Propagation and Mitigation.

nizer, whenever data is transferred between unrelated clock domains. An example of a synchronizer is shown in Fig. 2(b). Each flip-flop in the chain, except the last, feeds only one flip-flop so there is no chance of a metastable output being resolved differently by two fanout flip-flops, and the failure mechanism of Fig. 2(a) does not occur within the chain.

Should the output of the first synchronization flip-flop become metastable, it still needs to propagate through the rest of the chain before its value will be used by the rest of the design. The extra amount of time provided by the additional synchronizer flip-flops increases the probability that the metastable value will resolve, and lowers the possibility that the design will fail. The cost of the synchronizer chain is that it increases the design latency.

Synchronizer chains reduce the chance that metastable events cause system failures; however, the metastability problem can never be completely avoided in any system that is not purely synchronous. We typically quantify the likelihood of system failure due to metastability using the metric of Mean Time Between Failures (MTBF) [11, 10, 9].

Since the discovery of metastability, numerous studies have been undertaken to analyze this phenomenon both theoretically and empirically. These improvements have led to circuit level optimizations that can substantially improve the metastability robustness of systems.

However, these techniques cannot eliminate metastability-related failures, and such failures are a significant portion of total system soft errors. They are extremely difficult to debug in-system, as they occur at random time intervals and are hard to distinguish from soft errors due to other effects such as radiation-induced flip-flop or configuration RAM upsets. As we show in Section 2.1, the fundamental metastability robustness of flip-flops is rapidly diminishing with process scaling, so without new design and CAD techniques metastability will become an ever larger source of system failures. This paper presents novel CAD algorithms to analyze and optimize metastability, making it easier for FPGA designers to achieve metastability robustness without compromising other system metrics.

The rest of this paper is organized as follows. First we review and derive a theoretical model for metastability and use this model to highlight methods by which metastability can be optimized. Next, we perform SPICE circuit simulations to generate the metastability model parameters for several commercial FPGA devices. To validate our models, we then develop a hardware characterization methodology and compare the simulated results against device measurements. Given this accurate model, we describe a CAD tool that can automatically analyze a design for metastability issues and we describe optimizations that the designer can perform to improve the metastability characteristics of his or her design. We conclude by presenting a new automatic CAD tool that can optimize a design without any user intervention to achieve an orders of magnitude improvement in MTBF on a suite of industrial benchmarks, with no impact on system latency, and with minimal impact on design operating speed.

2. METASTABILITY MODELLING

Although we have described metastability in terms of flip-flops, the analysis of a single latch simplifies many issues. The single latch analysis can then be easily extended to a cascade of latches such as found in a master-slave flip-flop or a synchronizer chain.

Consider the simple CMOS latch in Fig. 3(a). In a noiseless system, it can be viewed as a deterministic device, with a discrete sampling instant somewhere in time closely related to the clock. The particular time that the data transitions at is referred to as T_{dc} (the transition of the data signal with reference to the sampling clock transition). When T_{dc} is a small number close to 0, the clock-to-output delay, T_{co}, of the latch is increased as described in [7, 6]. Note that a small positive T_{dc} corresponds to a FF hold violation, while a small negative T_{dc} corresponds to a FF setup violation.

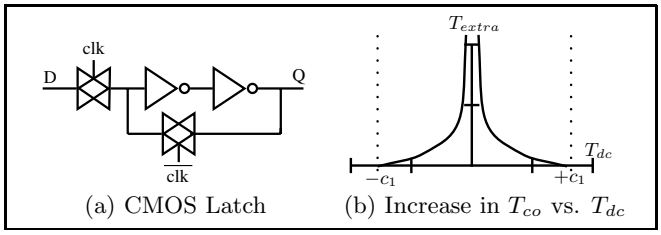


Figure 3: Latch Circuit and Behaviour.

The increase in T_{co} is denoted as T_{extra} and varies in the way shown in Fig. 3(b). It is 0 for normal (non-metastable) operation of the latch. However if T_{dc} is allowed to approach 0, and goes below some constant c_1, T_{extra} increases from 0 with a logarithmic behaviour and a slope of $-c_2$. The constant c_1 defines a window in which a data transition will cause a metastable event while the constant c_2 is related to the extra time required to resolve the metastable state. The equation relating these values can be expressed in the following manner:

$$T_{extra} = \begin{cases} -c_2 \log \frac{|T_{dc}|}{c_1} & |T_{dc}| \leq c_1 \\ 0 & otherwise \end{cases} \quad (1)$$

To gain a physical understanding of this relationship, we can perform a simplified analytic examination of the CMOS latch. The transfer curve relating V_{in} to V_{out} for a pair of back-to-back inverters is shown in Fig. 4. During normal op-

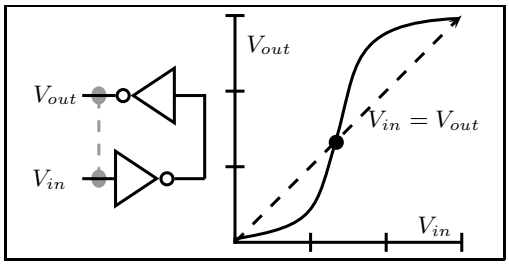


Figure 4: Transfer Function.

eration of the latch, this loop propagates either a logic-**0** or logic-**1** to store a particular state. However, in metastable

operation the loop propagates a signal that somewhere in between these two extremes. The inverters act as amplifiers

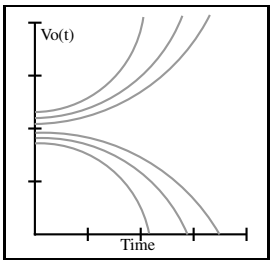


Figure 5: Latch V_o vs. time for different $V_{initial}$.

and positive feedback causes the loop to eventually settle to one of the two logic values as shown in Fig. 5. We can use standard techniques to analyze this situation. First the DC bias point can be calculated as the voltage at which the transfer curve intersects $V_{in} = V_{out}$. Given that the transfer function behaves approximately linearly around the bias point, we can perform transient analysis using the small signal model as shown in Fig. 6. The g_m values represent the

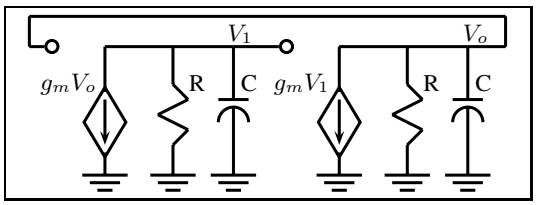


Figure 6: Small Signal Model.

total transconductance contribution from the N and P transistors in the inverters. Similarly, R and C are lumped values accounting for contributions from various sources. The system of equations describing this system can be written as follows:

$$\begin{aligned} g_m V_o + \tfrac{V_1}{R} + C\tfrac{dV_1}{dt} = 0 \\ g_m V_1 + \tfrac{V_o}{R} + C\tfrac{dV_o}{dt} = 0 \end{aligned} \tag{2}$$

Assuming solutions of the form Ve^{st} for both V_o and V_1, we find that:

$$V_o = V_o(0)e^{\left(\frac{g_m - \frac{1}{R}}{C}\right)t} \tag{3}$$

Rearranging and solving for a particular $V_o(t) = V$ (corresponding to a value for logic-**0** or **1**), we can find the amount of time required for metastability resolution:

$$t = -\frac{C}{g_m - \frac{1}{R}} log \frac{V_o(0)}{V} \tag{4}$$

Eq. 4 has the same fundamental structure as Eq. 1, where $c_2 = \frac{C}{g_m - \frac{1}{R}}$. Similarly, the inital value $V_o(0)$ is related to the sampling time T_{dc} and a constant c_1. A more complete analysis of the small signal model, including second order effects such as the Miller capacitance, can be found in [8].

Given the relationship between T_{dc} and T_{extra}, we can derive the equations for the failure rate of a latch [7, 6] in the presence of a data source whose transition times are uncorrelated to the clock input. We denote the slack on the latch output under consideration as T_{met}, the amount of time that the latch has to resolve a metastable state. If we exceed T_{met}, it is possible that errors will propagate to other "down-stream" components and cause the system to enter an incorrect state. Thus the probability of failure can be expressed as $p(failure) = p(T_{extra} \geq T_{met})$. Subsituting Eq. 1, we can derive the following:

$$p(failure) = p\left(|T_{dc}| < c_1 e^{\frac{-T_{met}}{c_2}}\right) \tag{5}$$

Thus the probability of failure is equal to the probability of a transition occuring between $T_{dc} = -c_1 e^{\frac{-T_{met}}{c_2}}$ and $T_{dc} = +c_1 e^{\frac{-T_{met}}{c_2}}$. Assuming the data signal is uniformly distributed with a frequency of F_d, then the probability of failure can simply be expressed as:

$$p(failure) = 2 * F_d * c_1 * e^{\frac{-T_{met}}{c_2}} \tag{6}$$

This equation can be interpreted as the probability of failure given a single clock transition. Assuming that the latch is operating at a frequency of F_c, then the total number of failures per second can be given as:

$$2 * F_d * F_c * c_1 * e^{\frac{-T_{met}}{c_2}} \tag{7}$$

The MTBF is simply the reciprocal of this value:

$$MTBF = \frac{e^{\frac{T_{met}}{c_2}}}{2 * F_d * F_c * c_1} \tag{8}$$

Flip-flops typically consist of a cascaded latch structure consisting of a master and a slave which do not necessarily have the same c_1 and c_2 constants as shown in Fig. 7. The analysis of such a multi-stage structure can be derived in a similar manner as the single latch analysis shown above. We seek

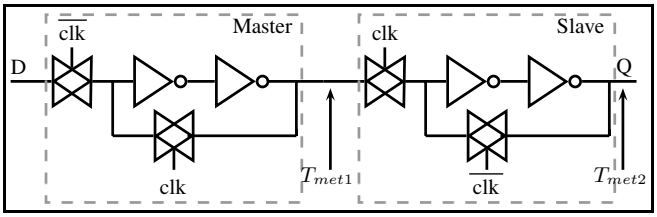


Figure 7: Master-Slave flip-flop.

to derive the conditions under which the second latch in the cascade becomes metastable. As described above, transitions that occur between $-c_1 \leq T_{dc} \leq c_1$ on the input of the second latch can cause metastability. We assume that such transitions are solely the result of a metastability event on the first latch. For a system that meets all its timing constraints, all synchronous transitions happen outside this window. Thus, we need to consider when metastable transitions from the first latch occur. Using Eq. 6, we can express the probability of a transition occurring after some time t as:

$$p(T_{extra} > t) = 2 * F_d * c_1 * e^{\frac{-t}{c_2}} \tag{9}$$

The cumulative distribution function $CDF(T_{extra})$ is:

$$CDF(T_{extra}) = p(T_{extra} < t) = 1 - 2 * F_d * c_1 * e^{\frac{-t}{c_2}} \tag{10}$$

Differentiating the CDF produces a probability distribution for the transitions at the input of the second latch:

$$p(T_{extra} = t) = 2 * F_d * \frac{c_1}{c_2} * e^{\frac{-t}{c_2}} \tag{11}$$

In the case of a master-slave flip-flop, the second latch samples the output of the first at time $T_{clk}/2$ before becoming opaque. The amount of timing slack available is $T_{clk}/2$ minus the nominal T_{co} of the first latch. We refer to this quantity as T_{met1}. Similarly, T_{met2} is defined as the amount of slack available between when the second latch becomes opaque and when this value is latched downstream. We are interested in the window where T_{extra} from the first latch causes transitions in the window defined by c_1 on the second latch. The PDF of transitions during this window happen between $T_{met1} - c_1 \leq t \leq T_{met1} + c_1$ and are shown in Fig. 8. In the analysis of the single latch case, we assumed

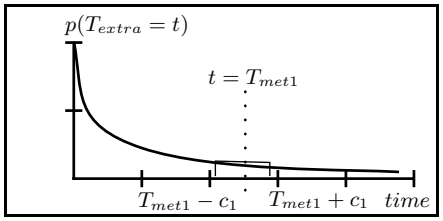


Figure 8: $p(T_{extra} = t)$ at the output of the first latch.

that the transition probability was uniformly distributed. From Eq. 11, we find that this is no longer true. However, to simplify the analysis, Eq. 11 can be approximated as a uniformly distributed function in the area of interest:

$$p(T_{extra} = t) \approx 2 * F_d * \frac{c_1}{c_2} * e^{\frac{-T_{met1}}{c_2}} \quad (12)$$

In a similar manner to Eq. 6, we can express the probability of failure of the second latch as:

$$p(failure, 2nd) = 2*(2*F_d*\frac{c_1}{c_2}*e^{\frac{-T_{met1}}{c_2}})*c_1*e^{\frac{-T_{met2}}{c_2}} \quad (13)$$

Simplifying, this leads to:

$$p(failure, 2nd) = 4 * F_d * \frac{c_1^2}{c_2} * e^{\frac{-T_{met1}-T_{met2}}{c_2}} \quad (14)$$

One implicit assumption that we have made in this derivation is that the slack T_{met2} from the second latch is a positive value (the maximum delay from the output of the second latch to its destination flip-flops is less than $T_{clk}/2$). In the case where this is not true, we need to do a more complex analysis where we consider metastable events from the first latch "flowing through" the second latch in its transparent state. This analysis is complex, but does not change the fundamental formulas that we are describing.

Note Eq. 14 is identical in form to the equation for the single latch with a few exceptions. The extra factor of 2 can easily be removed by incorporating it into c_1. In addition, note the extra factor of $\frac{c_1}{c_2}$ in comparison to the failure rate for a single latch. This factor is attained because the second latch "filters out" the tail of the probability distribution shown in Fig. 8. Our simple analysis above made some assumptions that caused this calculation to be somewhat optimistic. For example, it was assumed that the output of the first latch would only transition once before it resolved from a metastable state. This is not necessarily true as some latch structures may oscillate before resolution. Logic between synchronizer stages may also cause transitions which are not modelled by our analysis. To account for these factors, we use a slightly more conservative formulation where we assume that subsequent latches in the cascade do no filtering, but only "passes through" the output of the first latch. In this case, the cascade of latches only increases the effective T_{met}. Thus, our formula for the MTBF of multi-stage synchronizers can be expressed as:

$$MTBF = \frac{e^{\frac{\Sigma_i T_{met,i}}{c_2}}}{F_d * F_c * c_1} \quad (15)$$

Although this is clearly not the case for sequences of master-slave flip-flops, we use it as a worst-case bound.

Simply, this formula indicates that the MTBF of a chain is determined by the "chain slack" which is the sum of the output slacks of each latch along the chain. For simplicity, we have assumed that the c_1 and c_2 constants are the same for each latch in the cascade. This is not necessarily true and the implementation of this model used in this paper does not make this assumption.

The analysis above assumed a noiseless system. In a real system having conditions close to the metastable point, the resolution to **0** or **1** will be a stochastic process. However the presence of noise has been shown to have no effect on the overall circuit reliability due to metastability [11, 5].

Eq. 15 provides a formula for the MTBF of a single synchronizer chain. A design may consist of chains denoted as $S_1, S_2 \ldots S_n$. The total number of failures per second of the entire system can be given as:

$$\frac{1}{MTBF(S_1)} + \frac{1}{MTBF(S_2)} + \cdots + \frac{1}{MTBF(S_n)} \quad (16)$$

The MTBF of the entire system is simply the reciprocal of this failure value:

$$\frac{1}{\frac{1}{MTBF(S_1)} + \frac{1}{MTBF(S_2)} + \cdots + \frac{1}{MTBF(S_n)}} \quad (17)$$

This equation corresponds to the $\frac{1}{n}$th of the harmonic mean of all synchronizer chain MTBFs and indicates that the performance of the system is heavily influenced by the performance of the worst chain in the design.

2.1 Effects of Process Technology

Technology scaling has yielded faster transistors and improved FPGA density. Unfortunately, as processes are scaled the operating voltage (V_{dd}) decreases and the probability of metastability failures increases. The theoretical model derived above can provide us with insight into this phenomena. From Eq. 4, we found that c_2 was approximately inversely proportional to the transconductance g_m and from Eq. 15 it is evident that larger values of c_2 lead to exponential reductions in MTBF. The value of g_m is given by:

$$g_m = k\frac{W}{L}(V_{GS} - V_t) \quad (18)$$

where k is a transistor constant, W,L define the dimensions of the transistor, V_t is the transistor threshold voltage, and V_{GS} is the DC-bias point shown in Fig. 4. This voltage is approximately $V_{dd}/2$. Therefore, the value of g_m is proportional to the value of $V_{dd}/2 - V_t$. Fig. 9 shows how V_{dd} and V_t have scaled over time, according to the NTRS semiconductor roadmap [12]. This graph shows that the value of V_{dd} is decreasing faster than V_t. It is evident that the value of $V_{dd}/2$ is getting closer to the value of V_t. Given this trend, the effective value of g_m is decreasing and the value of c_2 is increasing. Fig. 10 shows the measured increase in

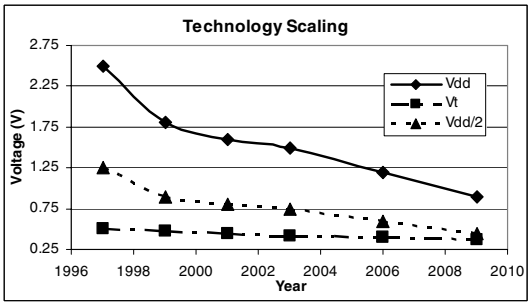


Figure 9: Technology Scaling.

c_2 as we decrease the operating voltage on a Stratix III device [2]. One might expect that the value of T_{met} would increase to compensate for the shrinking c_2; however, frequency requirements also increase as the process shrinks so the value of T_{met} does not grow with process scaling.

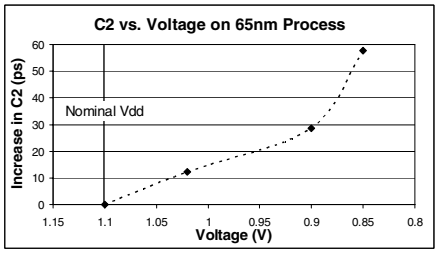


Figure 10: Change in c_2 vs. Voltage.

To combat the effect of process scaling on c_2, we need to make g_m as large as possible while minimizing C, through circuit design and layout techniques such as increasing $\frac{W}{L}$ and using lower V_t (LVT) transistors. The sizing ratios of the N and P transistors can be varied to find a DC bias point that increases g_m without compromising delay.

3. SIMULATING METASTABILITY

As discussed in the previous section, metastability failure probability is a function of two register parameters, c_1 and c_2. The values of these parameters can be extracted from SPICE simulations by measuring the increase in output delay, T_{extra}, as a function of data arrival time.

The ports of the register are loaded to provide the correct waveform shapes at all inputs and the proper capacitive load at all outputs. Transitions are applied at the D and CLK input ports at times T_d and T_c as shown in Fig. 11.

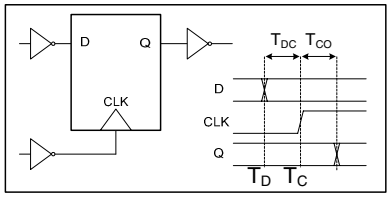


Figure 11: Register simulation testbench.

The relative timing of the data and clock transitions, T_{dc}, is swept by varying T_d with T_c fixed. At each T_{dc} point, the output transition time is measured and T_{extra} calculated by subtracting the nominal T_{co}. As T_{dc} is advanced it eventually reaches a point where the register no longer captures the new data value. The last data arrival time where new data is captured and the first where it is not bound is T_{crit}. If an input transition occurs at exactly T_{crit} the register theoretically has an infinite resolution time. Additional data arrival times in the T_{crit} window are simulated to tighten the bound on T_{crit} and we measure T_{extra} as input transitions get closer to the decision point. The process is repeated and each successive T_{dc} sweep produces a smaller T_{crit} window which can be further subdivided for the next sweep. The T_{crit} window is refined until T_{dc} increments reach the minimum time difference that the simulator can resolve.

Fig. 12 shows a plot of T_{extra} values extracted using this simulation methodology. T_{extra} is plotted as a function of $log(T_{dc} - T_{crit})$. The plot is divided into two distinct re-

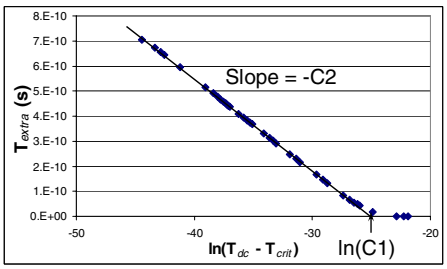


Figure 12: Example simulation data.

gions. When the input transition satisfies the setup and hold times of the register, T_{extra} is zero. Once setup time is violated, T_{extra} increases linearly as the input transition approaches T_{crit} logarithmically. The breakpoint between these two regions is c_1, the width of T_{crit} window near clock edges where data transitions cause some increase in output delay. The slope of the linear region is $-c_2$, the regeneration time constant of the latch feedback loop.

In reality, the transition between the two regions is not completely abrupt. There exists an intermediate region where the two lines are smoothed together. In this region input arrival times are close enough to the clock edge to cause some pushout. This pushout is due to the latch internal nodes not swinging fully rail-to-rail before the latch closes. Until the positive feedback of the latch restores these nodes to full-rail values, they drive downstream gates with less-than nominal voltages, which increases their switching delay. Data arrival times in this region are not close enough to the decision point to drive the latch into "deep" metastability where the small-signal response described earlier better models latch behavior. A two-piece linear model can underpredict T_{extra} in the transition region, but pushout in this region is generally small. As long as register outputs have a moderate amount of slack, only events in the deep metastability region will add enough T_{extra} to cause failures. As long as c_2 is extracted from the deep metastability region, the two-piece linear model will accurately model real cases.

The approach described above characterizes the metastable behavior of a single latch, but an edge-triggered flip-flop is actually a cascade of two latches. Data transitions near the rising edge of the clock can cause the master latch to go metastable, but if the slave latch sees an input transition near the falling clock edge it can go metastable as well. The slave latch does not necessarily have the same characteristics as the master so it must be characterized separately.

Metastability propagates from the master to the slave if the master latch resolves near the falling clock edge. In most flip-flops the master and slave latches are tightly cou-

pled, with little or no buffering between them. As a result it is not possible to toggle the D input to the slave latch directly as the input waveform will not have the correct shape. Instead it is necessary to trigger metastability in the slave by properly timing the resolution of the master latch near the falling clock edge. This requires precise control over master latch resolution time. It was shown earlier that very small differences in input arrival time create large differences in output transition time, so it is impractical to vary slave latch input transition time by sweeping data arrival times at the input to the master. Fig. 13 shows typical master latch output waveforms as it resolves from two slightly different initial loop voltages. Since these waveforms have identical

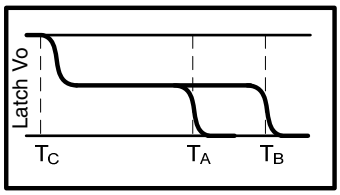


Figure 13: Output resolution waveforms.

shapes but are transposed in time, we can use the master latch only to produce the proper input waveform shape and vary the timing of the falling edge to characterize the slave.

4. METASTABILITY CORRELATION

Thus far, we have described a model for metastability and a method for simulating the master and slave latches to obtain the c_1 and c_2 constants that characterize these devices. To verify the accuracy of our simulated model we "correlate" the simulation to actual hardware measurements. This is a necessary step in ensuring that our models are accurate. The task of performing meaningful correlation for metastability is difficult because metastability is a statistical event. Unlike timing or power correlation, where absolute numbers (such as exact delays of paths and current drawn from the power supply) can be obtained, metastability samples must be collected over a period of time where it consistently occurs before we have sufficient statistical data to compute the MTBF. Since typical MTBFs are often expressed in years, it is infeasible to measure real designs in their natural environment. An additional complication of metastability correlation is that it may not be possible to obtain a part that matches up exactly with the device process and operating parameters used in the simulation. Because the c_2 constant is very sensitive to these parameters, it can vary widely. Due to the exponential nature of the MTBF, we can get results that differ by orders of magnitude. Therefore, our correlation is successful as long as we can obtain MTBF measurements that are bounded by the MTBFs obtained from simulation of the device at the worst-case (slow) process and operating condition corner.

4.1 Test Circuit

As described in the Introduction, it is impossible to detect that a flip-flop/latch has entered a metastable state in all cases; however, it is possible to detect the occurrence in some situations. The failures that we do detect can provide us with a scaled estimate of the number of true failures.

To obtain meaningful metastability measurements in a reasonable period of time, we have constructed a test circuit that has a very small T_{met}, so that its MTBF is on

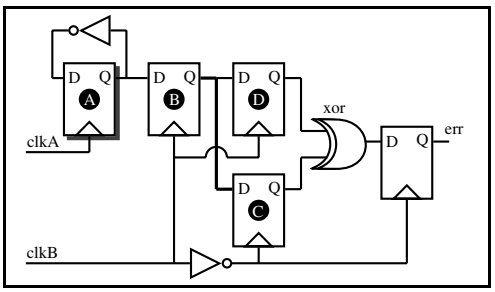


Figure 14: Correlation Test Structure.

the order of seconds, not years. This allows us to sample many occurrences of metastability within a short time frame. A sample test circuit is shown in Fig. 14. It is constructed to have a data transfer between 2 unrelated clocks, *clkA* and *clkB*. The flip-flop under test is *B*. Specifically, we are trying to identify metastable events in the master latch of flip-flop *B*. As shown in Fig. 15, the output value from the master latch is sampled twice. One sample, is taken at

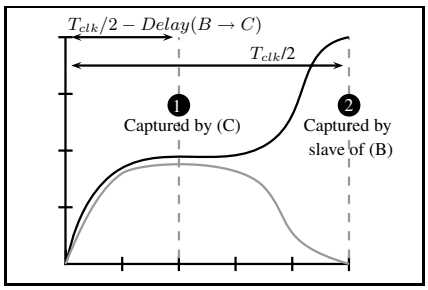


Figure 15: Sampling the Master Latch.

$T_{clk}/2 - Delay(B \rightarrow C)$ by the falling edge triggered flip-flop *C*. Due to the construction of the circuit, the path from *B* to *C* is the most critical path emanating from *B* and thus $T_{met} = T_{clk}/2 - Delay(B \rightarrow C)$. The other sample is taken at $T_{clk}/2$. This sample corresponds to the time at which the slave latch of *B* captures its value. If the two samples differ, then a metastable event was detected. Fig. 16 illustrates the detection of the metastable event with the output resolved after the first sample at T_{met}. Thus our circuit tries to quantify the number of transitions that occur after T_{met}.

We note that this test circuit does not capture all possible metastability events, because our first sample captures a signal in a metastable state and this state has to be discretized. Although only a subset of metastability failures are captured, this is enough to obtain sufficient data for the correlation of c_2. The accuracy of our measurement is ex-

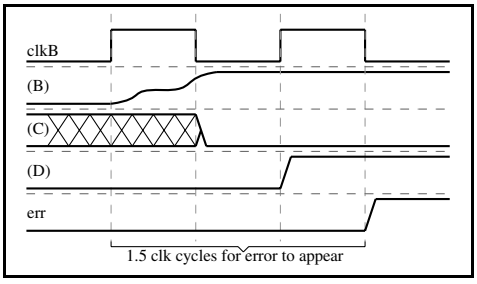


Figure 16: Timing Diagram.

tremely dependent on the delays of every path because we require small T_{met} values coupled with high operating fre-

quencies to create an observable MTBF. The delays between B and its destination flip-flops are minimized by limiting them to the same Logic Array Block (LAB). Our test circuit is run for 1 minute, and counts the number of errors detected. The measured MTBF can then be calculated using $60/NumErrors$. This test is conducted at various clock frequencies F_c while keeping the data frequency F_d constant. The results, for a Stratix III device [2], are plotted on a log-scale graph as shown in Fig. 17. On the y-axis, we plot the

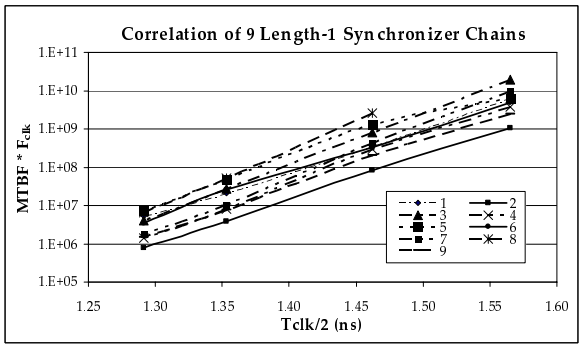


Figure 17: Correlation of 9 length-1 chains

quantity $MTBF * F_c$ and the x-axis represents the value of $T_{clk}/2$. From Eq. 15, we can rearrange to find:

$$\begin{aligned} log(MTBF * F_c) = & \frac{T_{met}}{c_2} - log(c_1 * F_d) \\ = & \frac{T_{clk}/2}{c_2} - \frac{Delay(B \rightarrow C)}{c_2} - log(c_1 * F_d) \end{aligned} \quad (19)$$

Thus the slope of the log-graph graph provides us with a measure of c_2 (specifically $\frac{1}{c_2}$). To capture the effects of local variation, this test structure is replicated 9 times across the chip. Through experimentation, the absolute number of errors differ greatly depending on the location of the test structure, but the slope tends to be the same.

The c_1 constant is less important since it has an inversely proportional relationship to MTBF while c_2 has an exponential relationship. To determine its value, we need to construct a circuit to measure the on-board value of $Delay(B \rightarrow C)$ to isolate c_1 from Eq. 19. We omit these details for brevity.

4.2 Worst-Case MTBF vs. Typical MTBF

As described in the Metastability Modeling section, the MTBF of a design has an exponential relationship to the c_2 constant. This constant can vary widely depending on the process, operating temperature and voltage of the device. Although we can simulate constants by assuming the worst set of all parameters, this does not necessarily reflect actual device behaviour. It may not be physically possible to have all parameters simultaneously be at the worst point for metastability. To have a more meaningful representation of design reliability, we compute two sets of MTBFs: the **Worst-Case MTBF**, and the **Typical MTBF**. The worst-case MTBF is obtained by performing SPICE simulation at the worst possible set of process and operating conditions. This metric is useful when predicting the expected overall product MTBF in the field, since the overall MTBF will be dominated by the worst-performing device. In contrast, the typical MTBF uses nominal process and operating conditions. This metric is useful when the designer is trying to match a lab measurement to the predicted MTBF. The designer should aim to achieve a worst-case MTBF that ensures high product reliability.

To provide a sense of how well the simulation model lines up with reality, we perform correlation experiments for both the worst-case and typical MTBF on Stratix III. For the correlation of the typical MTBF, we choose a typical process device and conduct our test under nominal operating conditions ($V_{dd} = 1.1V, T = 25C$). To correlate the worst-case MTBF, we look for the slowest device available and use the worst set of possible operating conditions. This usually occurs at the lowest operating voltage and the lowest temperature (eg. $V_{dd} = 1.02V, T = 0C$).

In Fig. 18, we show an example of the correlation for a design containing 2000 length-1 synchronizers. In this experiment, we vary the supply voltage to demonstrate how the change in condition impacts the measured MTBF. The outermost lines are the predicted MTBFs, whereas the inner lines are the measurements obtained when the voltage is varied. Notice that as we lower the voltage, the measured MTBFs tend to be closer to the predicted worst-case values. Our measurements are well correlated with the predicted MTBFs, and are consistent across frequencies. These results provide strong validation of our simulated models.

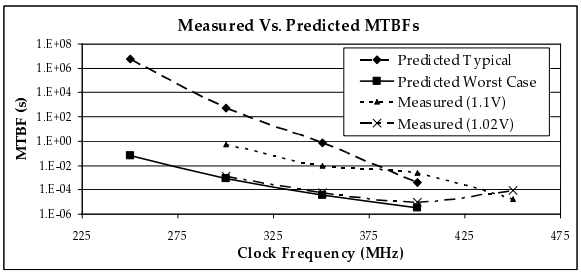


Figure 18: MTBF Comparison for StratixIII

5. MITIGATING METASTABILITY

In previous sections, we have established a model for metastability and validated it using hardware measurements. The fundamental question remains: What can a designer do to mitigate the problems caused by metastability?

As shown in Eq. 15, the MTBF of any chain has an exponential relationship with the T_{met} of the chain and the c_2 constant. Therefore, to improve MTBF one should focus on either increasing the T_{met} or decreasing c_2.

There are several things the designer can do to improve the T_{met} of synchronizer chains on a design level. The best way to improve the MTBF is to increase the number of stages in each of the synchronizer chains. The designer can also increase the available output setup slack by using location constraints. By constraining registers of the same synchronizer chain to the same LAB, the slack between synchronizer stages is maximized. The designer must also ensure that synchronization registers are not merged or optimized away by synthesis optimizations.

If there is a large number of synchronizer chains in the design, the designer should prioritize optimizations to improve the chain with the highest clock and data frequency, and the lowest T_{met} because the MTBF is dominated by the chain with the worst performance. The total number of asynchronous signals can also be reduced by using suitable handshaking protocols and signals, so that fewer synchronizer chains are needed.

To improve c_2, the designer should implement their design on a faster speed grade device if possible. A faster speed grade device usually means that metastable events will resolve faster. Also, in devices where programmable power technologies exist, synchronizers should be implemented using the highest speed setting, since this usually means the transistor will have a lower effective V_t. From Eq. 18, a lower V_t means that the flip-flop will have a higher gain, and metastable events will resolve faster.

6. METASTABILITY-AWARE CAD FLOW

Many of the strategies outlined in the previous section can be cumbersome and tedious to apply manually if there are many synchronizer chains in the design. In addition, modern CAD tools execute numerous optimizations to improve traditional metrics such as speed, area and power. In doing so, the CAD tool may actually hurt the MTBF of a design since typical flows are completely unaware of metastability. To address these issues, we have developed a CAD flow, integrated within Altera's Quartus II v9.0 [1], that is fully aware of metastability, and can place and route the design to optimize MTBF. The goals of this CAD flow are:

- To identify synchronizer chains automatically.
- To prevent optimizations that may reduce the MTBFs of synchronizer chains.
- To optimize the design to improve MTBF with no degradation in performance or area.

Before describing the details of our metastability-aware CAD flow, it is useful to review the conventional FPGA CAD flow. This is illustrated in Fig. 19. The first step is **Synthesis** which transforms a design into a netlist of logic cells, RAMs, DSPs and other specialized blocks found in modern FPGAs. **Placement** then assigns a physical location on the device to each element in the netlist with the objective of minimizing wirelength and improving performance. **Routing** attempts to create connections between elements in the netlist using the FPGA's programmable routing network. The router's goal is to successfully route all signals and ensure that performance is maximized. Notice that both placement and routing are tightly coupled with **Timing Analysis**. Given delays for each element and connection in the netlist, timing analysis can determine the operating speed of the circuit and determine timing slacks on each connection. As we progress through the steps of placement and routing, timing analysis is repeatedly executed using the best possible delay approximations available at these points. It can then provide both placement and routing with a measure of how **critical** each connection is in the netlist. Critical connections are generally those that have the least amount of timing slack. Both placement and routing can use this notion of criticality in their cost functions to ensure that critical connections are placed close to each other and get the fastest possible routing resources. After placement and routing (P&R) are complete, a bitstream can be generated to configure a device.

We have added metastability-"awareness" to all relevant sections of the CAD flow (synthesis, placement and routing). We refer to optimizations as metastability-aware when they recognize if a potential transformation may hurt metastability, but these optimizations do not actively try to improve metastability characteristics. Specifically synthesis optimizations such as register duplication and retiming fall into this category.

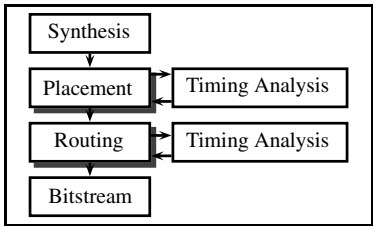


Figure 19: FPGA CAD flow.

Techniques that improve design MTBF have also been added to both the placement and routing steps of the CAD flow. This is achieved by enhancing timing analysis to make edges which heavily influence the MTBF of a design seem more critical than they would normally appear when only considering timing performance optimization.

6.1 Automatic Synchronizer Identification

Our first step is to perform a traversal of the netlist to identify synchronizer chains. We define a synchronizer chain using the following criteria:

- Each flip-flop in the chain must transitively fanout to a single flip-flop. There may be reconvergent fanout as in Fig. 20 in the synchronizer chain.

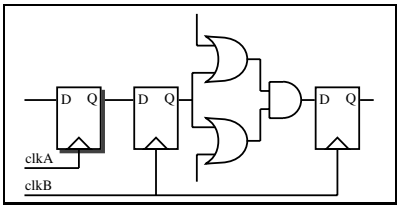


Figure 20: Reconvergent Logic.

- The input to the first flip-flop is driven by a flip-flop in an unrelated clock domain, or by an asynchronous signal. The flip-flop chain shown in Fig. 21 will *not* have flip-flop B identified as the head of a synchronizer – while flip-flops A and B have different clocks, there is still a relationship between the clock frequencies and phases which allows the creation of static timing constraints for the data transfer from A to B.

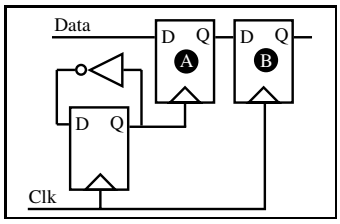


Figure 21: Related Clock Domains.

- The last flip-flop in the chain has more than 1 flip-flop in its transitive fanout cone as illustrated in Fig. 22.
- All flip-flops in the synchronizer chain are driven by the same clock.

To automatically identify synchronizer chains in the circuit, we first identify all locations of asynchronous data transfers.

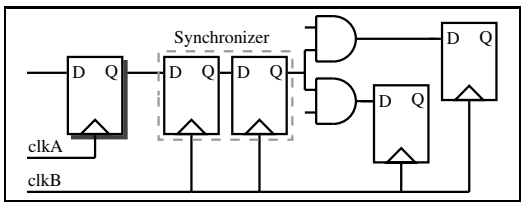


Figure 22: Chains end at multi-fanout points.

Each receiving flip-flop of the asynchronous data transfer is marked as the head of a potential synchronizer chain. We then explore the output paths of each flip-flop to see if it satisfies the requirements of a synchronizer chain listed above. By using both structural and timing information, we can reduce mis-identifications of synchronizer chains due to related clock domains. For each chain, we compute the MTBF using the formulae described earlier. Given the MTBF for each chain, we can then automatically provide a design MTBF.

Once synchronizer flip-flops are identified, it becomes a simple task to allow conventional optimization techniques to become metastability-aware. The techniques that can hurt the chain MTBF the most are the synthesis optimizations of register duplication and retiming. In retiming, registers can be moved across logic causing a decrease in the available output slack of the chain. While this optimization may be good for performance, it is not necessarily a good thing for metastability. Similarly, if any flip-flop in the synchronizer chain gets duplicated, its source flip-flop now has multiple flip-flops in its fanout and therefore the synchronizer chain terminates sooner. This reduces the T_{met} of the chain by removing entire synchronization stages.

6.2 Metastability Optimization

Once we have obtained a list of all synchronizer chains and have ensured that synthesis optimizations are metastability-aware, we can then modify the place-and-route engine to take metastability into account when optimizing critical paths.

Consider the synchronizer chain in Fig. 2(b) as it may appear during placement in a conventional CAD flow. The T_{met} of the synchronizer chain is the sum of the slacks on edge B and C. The delay on edge B might be increased significantly by the placement engine to optimize the wirelength of A and C. While this is good for placement, it would have a detrimental effect on the chain MTBF. Thus, the focus of metastability optimization is to make sure that the synchronizer edges, such as edge B, receive more attention during P&R.

Fig. 23 shows our general optimization approach. We add an additional step at the end of timing analysis to update the slacks and criticalities of synchronizer edges so that P&R can understand that they are "critical" for a good MTBF. A

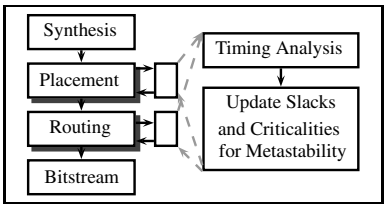


Figure 23: Metastability optimization approach.

simple metric of criticality can be defined as:

$$criticality(connection) = 1 - \frac{slack(connection)}{TimingConstraint} \quad (20)$$

Generally, this equation attempts to normalize slack values so that connections with small amounts of slack relative to the timing constraint, have high criticalities and vice-versa. This value can be clipped so that it is in the range of 0 to 1.

As is the case with many timing-driven P&R algorithms [4], both the placer and router use cost functions. These cost functions use connection criticalities so that connections with higher criticalities are placed closer to each other and connected with the fastest routes. The delay of non-critical connections can be increased to optimize for other metrics. If we modify the slacks and criticalities of synchronizer edges, we can easily instruct the P&R algorithms to optimize for metastability. The difficulty in doing this is that we do not wish to compromise performance in this process.

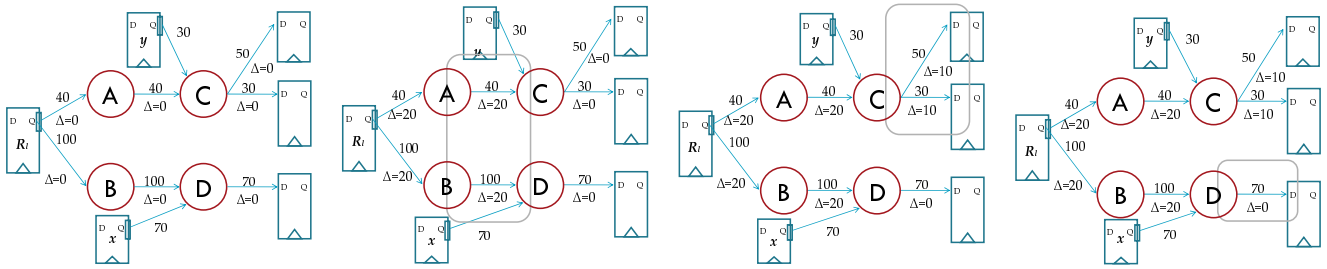
Our algorithm essentially involves reducing the slack of paths involving synchronizer flip-flops by some amount Δ. This increases the criticality and encourages the P&R algorithms to improve the T_{met} values for the chains in the design. For paths that are internal to a synchronizer, often a single edge paths from a source flip-flop R_i to a destination flip-flop R_j, reducing the slack by Δ is a trivial exercise.

The value of Δ is set to be the following:

$$\Delta = max(-\alpha * TimingConstraint + slack(R_i), 0) \quad (21)$$

where $slack(R_i)$ is defined to be the minimum slack on any edge on the immediate fanout of R_i. This effectively ensures that the criticality of the worst path starting from R_i is at least $1 - \alpha$. The Δ values are never negative so we do not increase the slack of any connection. Thus small values of α cause synchronizer edges to become increasingly critical.

The last flip-flop in the synchronizer has transitive fanout to more than one destination flip-flop. Let us denote the last flip-flop as R_i. We can then reduce the slacks of all paths emanating from R_i by Δ. This optimization is significantly more complicated than the single edge case because we cannot simply reduce the slacks of all edges in the transitive fanout cone of R_i by Δ since the slacks on these edges may involve paths that are unrelated to R_i. This concept is best illustrated with an example as shown in Fig. 24. Suppose that we choose the value $TimingConstraint = 200$, $\alpha = 0.1$ and the original slack for every edge is shown in Fig. 24(a). Given that the worst-case slack at the output of $R_i = 40$, the value of $\Delta = 20$. We first start by reducing the slack on the immediate fanout edges of R_i. We then propagate Δ forward to ensure that the worst-case input slack of every intermediate node is equal to the worst-case output slack. It is fairly straightforward to propagate through single-fanout nodes such as nodes A and B in Fig. 24(b). However, for node C, although one of its input edge slacks is reduced by 20, the worst-case input slack only changes by 10 since the limiting factor is now the edge A to C, instead of y to C. Therefore, we only need to reduce the slack of all its outputs by 10 (Fig. 24(c)). For node D, since its worst-case input slack remains unchanged, we do not need to update the slacks of its output edge (Fig. 24(d)). This process is iterated until the destination flip-flops of paths starting from R_i are reached. Note that the final slacks can be obtained by taking the original slacks and subtracting the Δs shown on each edge. This propagation scheme ensures that the worst-



(a) Original slacks (b) Nodes A and B updated (c) Node C updated (d) Node D updated

Figure 24: Local slack adjustment cases.

case path from R_i has a criticality of at least $1-\alpha$, while all other paths starting from R_i are scaled appropriately.

To evaluate the performance of this algorithm, we require designs that are fully timing-constrained. Our synchronizer identification algorithm requires proper I/O and clock constraints. Since the MTBF is irrelevant if the design does not meet its timing constraints, we measure our algorithm on a set of 100 industrial circuits targeted to Stratix III [2] and Stratix IV [3] with realistic timing constraints. The average size of these circuits is 35,000 logic elements (LEs) with a range from 1020–245,000 LEs. Since the design MTBF is limited by the worst-performing synchronizer chain, we compare the worst-case T_{met} in each circuit with and without our proposed optimization and note the improvement. Conventional metrics such as the maximum operating frequency of the circuit (F_{max}) and runtime penalty are also noted. To provide context for the improvement in T_{met}, we define an MTBF multiplier as:

$$MTBF_{multiplier} = e^{\frac{\Delta T_{met}}{c_2}} \quad (22)$$

Given an increase in T_{met}, this is the factor with which we have improved the design MTBF. In all cases, we average these metrics using the geometric mean across all circuits.

In Fig. 25, the performance of our algorithm is shown for different values of α. The baseline is Quartus II v9.0, with no metastability optimization and it is compared to our version of Quartus II with metastability optimizations enabled. As α increases, the improvement in T_{met} gets smaller while the performance (F_{max}) penalty is also decreased. We choose a value of α that provides consistently good results across many different circuit sets and architectures. The best tradeoff between performance and metastability for Stratix III and IV is shown in Table 1. On our benchmark set, we can achieve 600-760ps improvement in T_{met} without a significant drop in circuit performance. Depending on the process technology, we have improved the average design MTBF by 22,000 to 268,000 times. The runtime of our metastability-aware CAD flow is negligible.

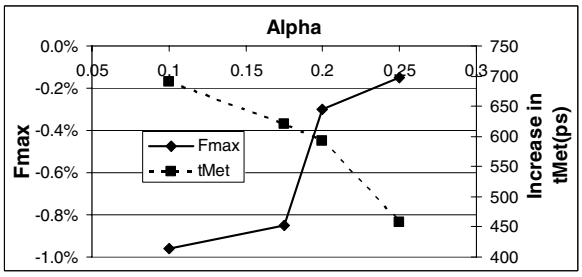


Figure 25: α value tradeoff.

Table 1: Optimization Results

Metric	StratixIII	StratixIV
F_{max} Improvement	-0.3%	-0.5%
ΔT_{met}	+600ps	+760ps
MTBF Multiplier	**268,000**	**22,000**
Runtime penalty	0%	0%

7. CONCLUSIONS

In this paper, we have derived the theoretical model for metastability from first principles, and have explained in detail how to obtain the constants necessary for this model through SPICE simulations. We have also validated our simulation methodology by correlating our model to silicon. To ensure that metastability is optimized through the FPGA CAD flow, we have introduced a technique to guide placement and routing to improve the chain slacks of synchronizers. Using this approach, our results indicate that we can improve the average design MTBF by up to 268,000 times.

8. ACKNOWLEDGEMENTS

We would like to thank Colman Cheung, Jason Govig and Bill Davis for their contributions to this project.

9. REFERENCES

[1] Altera. *Quartus II Handbook v9.0*, 2009.
[2] Altera. *Stratix III Device Handbook*. v1.9, Jul. 2009.
[3] Altera. *Stratix IV Device Handbook*. v3.3, Jun. 2009.
[4] V. Betz, J. Rose and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.
[5] G. R. Couranz and D. F. Wann. Theoretical and experimental behavior of synchronizers operating in the metastable region. In *IEEE Trans on Computers*, vol. C-24, June 1975, pp. 604-616.
[6] T. J. Gabara, G. J. Cyr, and C. E. Stroud. Metastability of CMOS Master/Slave Flip-Flops. In *IEEE Transactions on Circuits and Systems*, Vol. 39, No. 10, Oct. 1992, pp. 734–740.
[7] J. Horstmann, H. Eichel, R. Coates. Metastability Behavior of CMOS ASIC Flip-Flops in Theory and Test. In *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 1, Feb. 1989, pp. 146-157.
[8] L.-S. Kim, and R. Dutton. Metastability of CMOS Latch/Flip-Flop. In *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 4, August 1990, pp. 942–951.
[9] L. Kleeman and A. Cantoni. Metastable Behavior in Digital Systems. In *IEEE Design and Test of Computers*, Vol. 5, NO. 6, 1987, pp. 4-19.
[10] P. A. Stoll. How to Avoid Synchronization Problems. In *VLSI Design*, Vol. 3, No. 6, 1982, pp. 56-59.
[11] H. J. Veendrick. The behavior of flip-flops used as synchronizers and prediction of their failure rate. In *IEEE Journal of Solid-State Circuits*, vol. SC-15, April 1980, pp. 169-176.
[12] L. Wilson, ed. The National Technology Roadmap for Semiconductors: 1997 Edition, *Semiconductor Industry Association*, San Jose, California.