Towards Reliable 5Gbps Wave-pipelined and 3Gbps Surfing Interconnect in 65nm FPGAs

Paul Teehan
Dept. of ECE
University of British Columbia
paul.teehan@gmail.com

Guy G. F. Lemieux
Dept. of ECE
University of British Columbia
lemieux@ece.ubc.ca

Mark R. Greenstreet
Dept. of Computer Science
University of British Columbia
mrg@cs.ubc.ca

ABSTRACT

FPGA user clocks are slow enough that only a fraction of the interconnect's bandwidth is actually used. There may be an opportunity to use throughput-oriented interconnect to decrease routing congestion and wire area using on-chip serial signaling, especially for datapath designs which operate on words instead of bits. To do so, these links must operate reliably at very high bit rates. We compare wave pipelining and surfing source-synchronous schemes in the presence of power supply and crosstalk noise. In particular, supply noise is a critical modeling challenge; better models are needed for FPGA power grids. Our results show that wave pipelining can operate at rates as high as 5Gbps for short links, but it is very sensitive to noise in longer links and must run much slower to be reliable. In contrast, surfing achieves a stable operating bit rate of 3Gbps and is relatively insensitive to noise.

Categories and Subject Descriptors: B.8.0 [Hardware]: Performance and Reliability (General), C.5.4 [Computer Systems Organization]: Computer System Implementation (VLSI Systems)

General Terms: Design, Performance, Reliability.

Keywords: FPGA, network-on-chip, programmable, interconnect, bit-serial, on-chip SERDES, reliable, surfing, wave pipelining.

1. INTRODUCTION

The traditional design objectives for FPGA interconnect are to minimize latency (delay), area and power. FPGA interconnect is not well suited for highly connected, bus-oriented designs that require high bandwidth and throughput for two primary reasons. First, wide buses require a large amount of routing resources; for example, a completely connected chip-multiprocessor was found to be routing constrained after 22 nodes [1]. Second, the bandwidth of a link is the inverse of its latency, which degrades as the link length increases. Thus, long links have very low bandwidth.

Throughput can be greatly improved by pipelining the interconnect [2, 3]. The traditional use of FPGA interconnect propagates a bit end-to-end before the next bit is sent, decreasing bandwidth with length. In contrast, wave pipelining sends several bits in rapid succession down the combinational path in one clock cycle. This can be implemented by adding serializer (SER) and deserializer (DES) circuitry at data sources and sinks. The key is to keep communication reliable with relatively low area overhead. Figure 1 shows that a 1-bit wave-pipelined link offers bandwidth comparable to a non-pipelined 16-bit parallel bus when the signal travels farther than 25 interconnect stages.[1] An *interconnect stage* is a unidirectional routing wire that spans 4 CLB tiles (length 4). A *link* is a sequence of stages, forming a routed net that connects source and sink logic with no intermediate logic. All delay results in this paper are obtained from HSPICE simulations in 65nm technology.

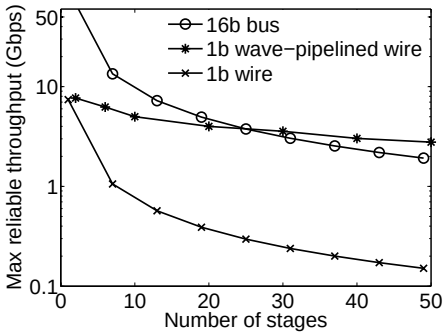


Figure 1: Throughput advantage of wave-pipelined links

The bandwidth of a wave-pipelined link is set by the minimum safe time between consecutive bits. If the bits are too close together, intersymbol interference (ISI) may cause incorrect data to be sampled at the receiver. Noise tolerance is thus an important design criteria in wave-pipelined links. A recent technique called surfing [4, 5, 6], which actively compensates waveforms to remove jitter and skew, offers greatly improved reliability and noise tolerance with modest latency, area, and power overheads. In this paper, the throughput and latency of wave pipelining and surfing for bit-serial FPGA interconnect are evaluated and compared in the presence of crosstalk and power, voltage, and temperature (PVT) variation. The contributions are as follows:

1. Two novel source-synchronous, bit-serial interconnect designs for FPGAs, using wave pipelining and surfing, are proposed in Sections 2 and 3;
2. The effects of crosstalk and PVT variation on delay variation are presented for both designs in Section 4;
3. HSPICE-based analysis and reliability estimates based upon statistical timing concepts are presented in Section 5;
4. Area details and HSPICE simulations for latency, throughput and energy are provided in Section 6; and
5. A third design that adds asynchronous FIFOs to wave pipelining is proposed in Section 7.

[1]The design assumptions to produce Figure 1 are described later.

FPGA'09, February 22–24, 2009, Monterey, California, USA.

We find that voltage supply noise is the biggest concern for both designs. Wave pipelining provides superior latency and higher bandwidth than surfing, but surfing is more robust against supply noise, and is a promising scheme if stable bandwidth performance is needed or voltage noise is largely unknown.

1.1 Related work

A number of recent papers have explored wave pipelining in Networks-on-Chip, achieving simulated throughput as high as 67 Gbps in a 65nm process [7]. NoC research is not directly applicable to FPGAs because the interconnect is not programmable.

Wave-pipelined interconnect for FPGAs was studied in [8, 9]. That work, which does not include a reliability assessment or attempt to customize FPGA interconnect circuitry, predicts a throughput of 1.4Gbps at a length of 75 tiles (about 18 stages using our terminology) in a 65nm process. In [9], the authors implemented a wave-pipelined connection in a Xilinx FPGA, achieving roughly 0.25Gbps per wire. In this paper, reliability estimates and Monte Carlo HSPICE simulations predict roughly 3Gbps in the presence of noise with optimized mux and driver circuitry.

2. SYSTEM DESIGN

2.1 Motivation and requirements

High-bandwidth interconnect can be useful in two ways. First, in throughput-oriented designs, it moves a large amount of data across the chip in as little time as possible. For example, communication between large memory blocks, hard core processors, and high-speed IOs (e.g., PCIe, QuickPath, HyperTransport, 10/100Gb Ethernet) can benefit from high throughput. These hard blocks have fixed locations and must communicate over a fairly long distance.

Second, high-bandwidth, bit-serial wires can save area in low-cost devices like Cyclone and Spartan which have a restricted amount of interconnect available. In particular, it is easy to generate large designs using EDK and SOPCBuilder; these tools generate complex systems interconnected by a word-oriented crossbar-like fabric. Adding bit-serial interconnect to these devices can greatly alleviate routing congestion and allow them to implement much larger systems while still remaining low-cost.

Figure 2 presents estimates of total *interconnect area* as 512 length-4 channel wires are gradually replaced with 8-to-1 bit-serial wires.[2] Although area is needed for the bit-serial wiring and SER/DES circuits, area is saved as traditional wiring is removed. In particular, note the input connection block area shrinks significantly because semi-random switch patterns can be replaced with more orderly word-wide inputs [10]. Hence, low-cost devices can have greater datapath interconnect capacity and use less area with bit-serial interconnect.

From these two scenarios, we see that low-area and high-bandwidth are the two main objectives for bit-serial interconnect. Bit-serial links can either be exposed to the user as a new resource, or hidden from the user to transparently implement datapath connections. An FPGA environment also adds several further requirements:

- The links must be reconfigurable, meaning input multiplexers and rebuffering are required at each stage.
- The length and topology of links vary by application, but very long connections with fanout should be supported.

[2]The figure assumes a 15% sparse input connection pattern with 77:1 input muxes, surfing interconnect, and 8 DES and 4 SER circuits for every CLB location. In determining the number of SER and DES circuits, we pessimistically assume CLBs have 64 inputs and 32 outputs. For a detailed area breakdown, please see [27].

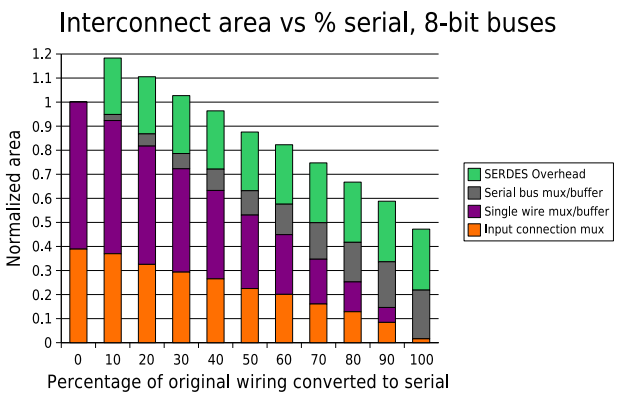


Figure 2: Interconnect area estimate

- Serial transfers should finish with a minimum latency, ideally in less than one user clock cycle; a user clock operates on the order of 100 to 200MHz.
- To save power, serial data should be sent in a burst when ready, not continuously.
- Custom driver circuitry is possible, but should be minimized; custom receiver circuitry (at each mux input) must be avoided due to area overhead.
- Ideally, regular FPGA interconnect circuitry should be reused to carry bit-serial signals.
- Some power overhead is expected, but should be kept low.

2.2 Source-synchronous architecture

The architecture of our system, shown in Figure 3, adds dedicated serializer (SER) and deserializer (DES) blocks to each CLB, multiplier and memory block. The serializer inputs (deserializer outputs) capture a word of data, allowing very sparse, organized output (input) switch patterns. The precise word size, number of SER and DES blocks, and switch patterns need to be studied in future architecture experiments. In this paper, we assume 8 or 16 bit words, but other word sizes are possible.

The user clock edge triggers the SER to capture a word of data from the block outputs. It also starts a programmable clock generator based on a pausible ring oscillator [11, 12]. To save power, this serial clock automatically stops after the word is sent. Both edges of the clock are used to clock data. No pilot bit or stop bit is needed. The scheme produces a waveform similar to Figure 4. We assume one word is transmitted each user clock cycle.

The clock and data are both forwarded in source-synchronous fashion along two parallel interconnect wires. For double the bandwidth and lower latency, a second data wire can share the same timing wire. The interconnect layout and/or CAD tools must keep these signals together to reduce skew. If needed, the interconnect can fan out the clock and data to multiple sinks. After traversing the interconnect, the clock shifts the data into the deserializer. At this point, it is ready to be processed in bit-parallel form as a set of combinational signals. For simplicity, we assume all further combinational paths from this point will use bit-parallel wires. That is, the serializer only takes data at the start of a user clock. In the future, more flexible start triggers can be made using programmable delays or data-ready signals.

In our circuit designs, the DES circuit emits the first bit as soon as possible, allowing downstream logic to begin early computation. Also, the SER circuit captures the last bit as late as possible to allow early triggering of the data transmission.

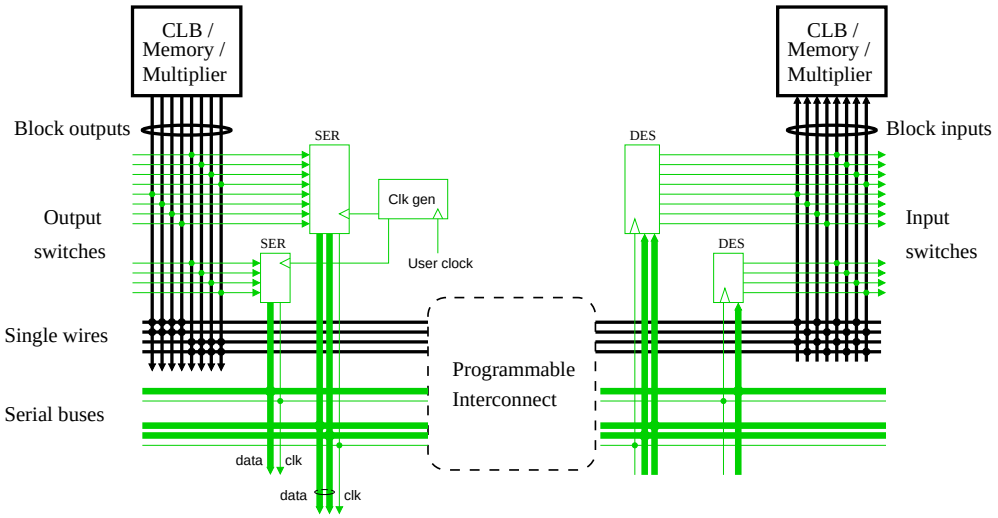


Figure 3: High-bandwidth, bit-serial interconnect buses

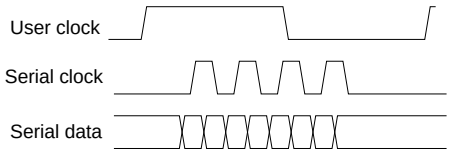


Figure 4: High-level timing diagram

2.3 Alternative signaling schemes

This section explains why source-synchronous signaling was selected in preference to several alternatives.

2.3.1 Global clock

We first considered the "safe" and obvious fully synchronous approach, which requires distributing a high-speed global clock to control SER and DES blocks. In this case, data validity is needed, e.g. by sending a pilot bit. This scheme sends data on just one wire using either wave pipelining or register pipelining. Wave pipelining needs additional circuitry at the receiver to phase-align the received data to the global clock. Register pipelining adds registers to the interconnect to keep data aligned to the clock.

With a global DDR clock, a low-skew 2.5GHz clock is needed to reach the 5Gbps data rates in this paper. CLBs that do not use the SER/DES blocks can locally gate the global clock to save power. However, CLBs that use the SER/DES blocks, or that route data from other CLBs using register pipelining, must be clocked continuously. Hence, they do not have "quiet" periods like our source-synchronous scheme — power scales with the number of sources, sinks, and total wirelength, not with the amount of data delivered.

This scheme was abandoned because distributing a global clock is expected to consume enormous power. Register pipelining adds even more power overhead.

2.3.2 Asynchronous

Many asynchronous signaling schemes exist and would be suitable for a bit-serial link in an ASIC [13]. However, in FPGAs, links must be programmable, which requires that the req/ack handshake signals travel over a programmable path. To reach high bandwidth, fine-grained handshaking is required between the interconnect stages; the latency of a long connection would make end-to-end handshaking much slower.

Adding support for asynchronous handshaking requires several specialized modifications to FPGA interconnect. Asynchronous handshake signals must operate glitch-free to avoid unintentional state transitions. In a two-wire handshake, the ack must be reverse-routed back to the source, doubling the area and power relative to a single source-synchronous timing signal. Supporting fanout is a bit more complex — after a req at the splitting point, the sender needs to wait for all fanouts to ack before sending the next datum.

In one-wire (aka bidirectional-wire) handshake schemes [14], req pulls one end low and ack pulls the other end high in response. We cannot see how to support this without reverting to inferior bidirectional interconnect [15] to support the backwards-routed ack.

2.3.3 Clock embedded in data

A typical off-chip serial link embeds timing information into the data signal using specialized encodings; the receiver recovers the clock from the data using a clock-data recovery (CDR) circuit. This technique is not of interest here for three main reasons. First, it relies on analog signaling techniques which are not compatible with the mux-buffer topology in FPGAs. Second, the transmitter and receiver are relatively complex, leading to high area overhead and unwanted design complexity. Finally, CDRs typically use phase-locked-loops, which work well for streams, but not bursts, because they take a long time to lock (for example, 60ns in [16]).

2.3.4 Two-wire pulsing

Pulses can be used to send data down a link. A simple scheme introduced in [17] sends a 0 by pulsing one wire, or sends a 1 by pulsing the other wire. It also uses arbiter-like logic to avoid skew between wires. Pulses as atomic units use twice as much power and require twice the latency compared to edges and are less likely to be competitive. Also, the FPGA interconnect multiplexers require wider pulses than fixed ASIC interconnect, further slowing the bit rate. This multiplexer impact is further discussed in Section 3.1.

3. PHYSICAL DESIGN

The programmable interconnect is assumed to consist of a variable number of identical cascaded stages, each of which includes a 16:1 multiplexer, a driving buffer, and a wire which spans four

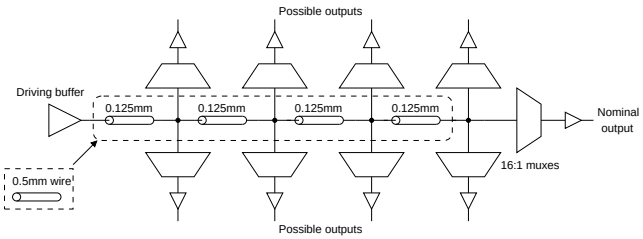


Figure 5: Detail of 4-tile wire

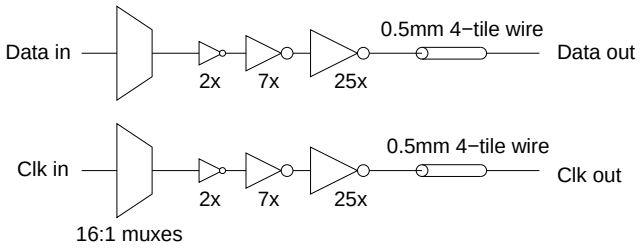


Figure 6: Wave pipeline interconnect stage

tiles [18]. Figure 5 shows a circuit schematic of a four-tile wire. FPGAs often have longer wires, e.g. 20 tiles, but for simplicity these are not considered. Further assumptions are: each tile is 0.125mm wide, making each stage 0.5mm long; wires are routed on a middle layer with twice-minimum width and twice-minimum spacing; multiplexer taps for turns appear every 0.125mm; and the data travels the full length of the wire (all four tiles).

HSPICE simulations in a CMOS 65nm process suggest that the latency from mux input to mux input, through a driving buffer and past four CLB tiles, is on the order of 150ps. Circuit modifications to better support wave pipelining and surfing are presented below.

3.1 Wave pipelining

Figure 6 shows the circuit diagram for a wave pipelined interconnect stage with one data wire and one timing wire. To limit skew, both signals must be routed side-by-side. To save area, the muxes for the data and timing wires may share one set of configuration bits because they follow the same route. More data wires can be added to improve bandwidth and latency, and to amortize the area and power overhead of the timing wire.

Each interconnect stage consists of an odd number of inversions to average out any risetime/falltime mismatch over long links. To achieve high throughput, drivers should be sized to produce sharp edges because the width of the narrowest pulse that can be safely propagated depends primarily on risetime [19]. The multiplexers use full CMOS transmission gates rather than only NMOS transistors to improve risetime, which unfortunately conveys a significant area penalty. The multiplexer design is the hybrid style used in [18] which contains two series transmission gates in the signal path. Throughput tends to be limited by the mux risetime.

Notice that the circuits are open-loop, allowing jitter and skew to each accumulate through cascaded stages. Surfing interconnect, as described in the next section, attenuates both jitter and skew.

3.2 Surfing

Surfing interconnect, shown in Figure 7, uses the same basic structure as wave pipelining with a few key additions:

- The middle buffer in both data and clock lines is variable-strength; a weak inverter is connected in parallel with a strong tri-state inverter.
- The data line middle buffer is controlled by a pulse (approx. 80ps) generated after each clock edge by a special circuit.
- The clock line middle buffer is controlled by a delayed version of the clock. The delay is half of the clock period; since it is a DDR clock, this is equivalent to one full bit period.

Figure 8 illustrates timing with surfing. The extra circuitry serves two functions. First, the variable-strength buffer on the data line serves to keep data signals aligned with the corresponding clock edges [5]. The mechanism is similar to a pulsed latch, except that late data edges can be sped up relative to the clock. Second, the variable-strength buffer and delay loop on the timing line serve to preserve the spacing between incoming clock edges so that the output period is stable [6]. The mechanism is similar to a delay-locked-loop. There is no inversion on the delayed line — as surf_in rises, the surf_en signal falls because it is an echo from half a clock earlier, helping surf_out to rise faster and remain on-schedule.

The key concept behind these mechanisms is that an edge leading its control signal sees a weak buffer and is slowed down, while an edge trailing its control signal sees a strong buffer and goes at full speed. Edges tend to be pulled towards a stable point during the control signal transition in which the buffer strength is midway between the weak and strong modes. Certain timing constraints must be satisfied for the surfing mechanism to work; for example, the latency through the clock stage must be slower than the data path when the data buffer is strong, yet faster when the data buffer is weak. Hence, two extra inverters are added to the clock path. Timing constraints are described in more detail in [6]. The robustness of surfing has been demonstrated in a working prototype chip [20].

4. TIMING UNCERTAINTY

Sources of timing uncertainty can be broadly classified by their relative time scales. Fixed effects, such as process variation, or extremely slow-varying effects, such as electromigration, affect static timing. In contrast, fast transient effects, the most significant of which are crosstalk and voltage variation [21], affect the timing from cycle to cycle. In this section, we consider all of these effects and their impact on the bit-serial design.

4.1 Tolerating uncertainty

With wave pipelining and surfing, static effects can be accounted for at the system level by adding timing margin, i.e. operating at a slower bit rate. As will be shown in Section 5.1, wave pipelining has an unstable region where pulse width deteriorates from stage to stage, so the bit rate must be selected slow enough to keep the circuit above this critical operating point. In contrast, surfing can restore narrow pulse widths and make them wider. However, this cannot be sustained if the incoming rate exceeds the outgoing rate. Hence, the surfing bit rate must also be slow enough to keep the circuit operate above this critical operating point.

In contrast, dynamic effects threaten the safe transfer of data. In particular, transient noise can push data and timing edges apart and lead to incorrect sampling (excessive skew), or it can push two consecutive edges on the same wire sufficiently close to cause inter-symbol interference. Wave pipelining needs additional timing margin to tolerate this, hence must run slower as a connection lengthens. However, surfing is more stable and needs significantly less margin — it can better tolerate pulse width narrowing by restoring short pulses into wider ones, and it limits skew by using the slower clock path to modulate the data path speed.

4.2 Process and temperature variation

The minimum reliable pulse width is a function of transistor speed, which can vary by up to $\pm 30\%$ [7] with respect to the typical speed. Estimates of within-die variation vary but often fix it at about half the full range [22], up to $\pm 15\%$ from the average, such that a variation from SS to TT is possible within one die (or from

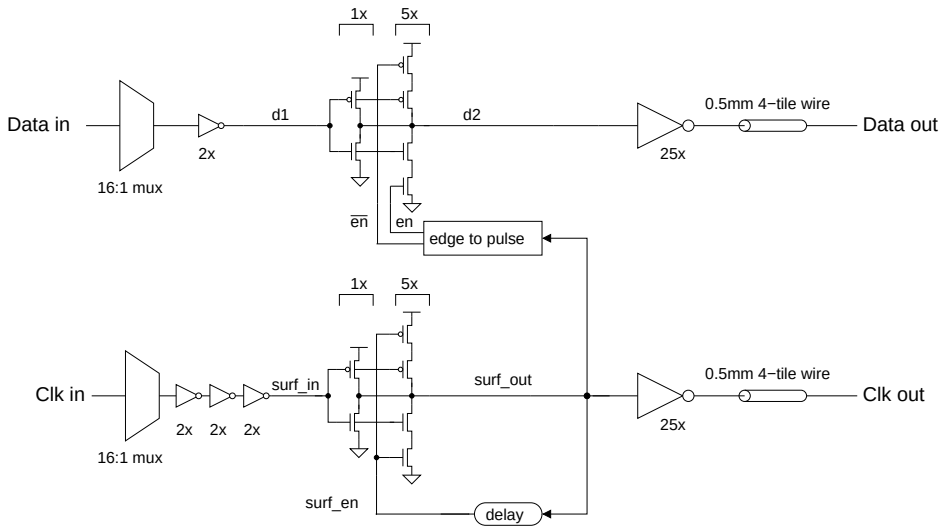


Figure 7: Surfing interconnect stage

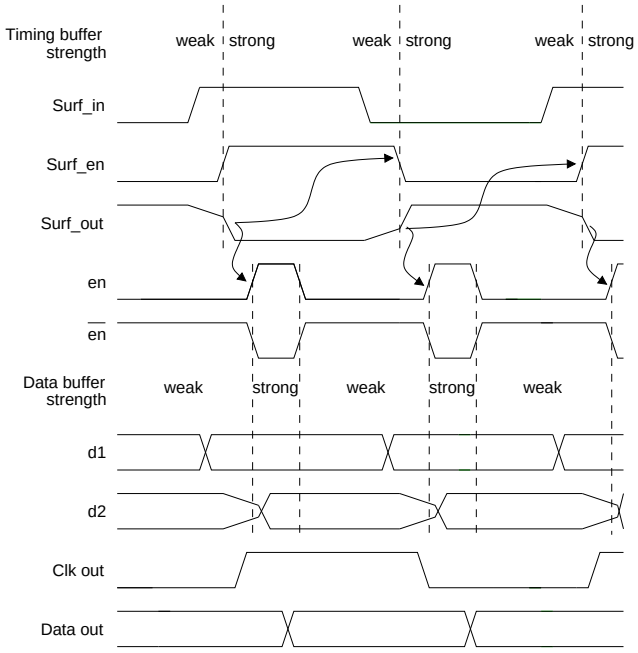


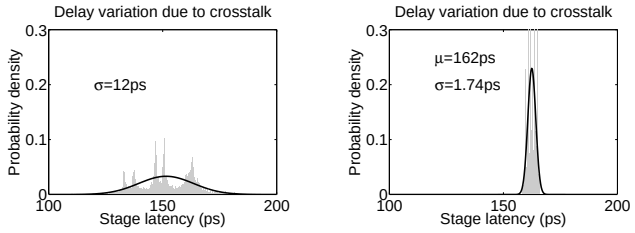
Figure 8: Surfing timing diagram

TT to FF, or somewhere in between). Because the slower transistor speeds tend to set the minimum pulse widths, all circuits in this paper are tested in the SS corner, at high temperature (125^oC). As a result, we will choose an operating point (bit rate or pulse width) with enough margin to function reliably under worst-case process and temperature conditions.

Speed mismatch between data and timing paths can also cause skew. This is particularly harmful to wave-pipelining because its effect is cumulative. As long as the transistors and wires are close together, systematic variation in process and temperature should cause negligible skew effects [7]. Skew still suffers from random uncorrelated variation — we model path delay mismatch as delay variation with a standard deviation (σ) equal to 2% of stage latency.

4.3 Crosstalk

Crosstalk is a fast noise source contributing to skew and ISI. We expect the fast edges and close proximity of bit-serial interconnect



(a) No shielding (b) Full shielding

Figure 9: Delay variation due to crosstalk

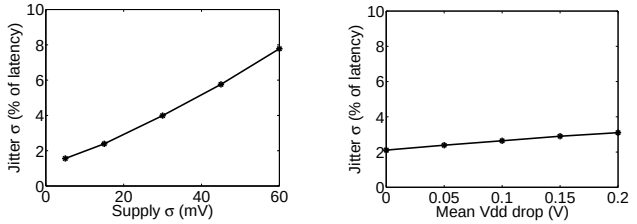
to be the primary source of crosstalk noise. To simulate worst-case conditions, aggressor data sources with random binary sequences were added on both sides of a serial link. New data tokens were chosen every 50ps, corresponding to a 20Gbps bit rate, which is extremely pessimistic. We performed Monte Carlo simulations and measured the delay of one wave-pipelined stage, both shielded and unshielded. The shields are minimum width and minimum spacing; the unshielded wires are twice-minimum spacing. In all cases, the clock and data wires are twice-minimum width. Second-order coupling capacitances (i.e. from one signal wire through a shield to the next signal wire) accounted for about 3% of total capacitance.

The resulting delay histograms for one interconnect stage are shown in Figure 9. The histograms do not follow an ideal normal distribution because of deterministic coupling between the data and timing wires. Also, a slight mismatch between rising and falling edges leads to double-peaked behavior. A normal curve fit to the data provides a crude first-order estimate of delay variation: standard deviation of the unshielded case is $\sigma = 12$ps and the shielded case is $\sigma = 1.74$ps. In comparison, Section 5.3 shows about $\sigma = 11$ps jitter delay variation from supply noise. Although the impact of crosstalk is significant, we mitigate it by shielding.

4.4 Voltage variation (supply noise)

There are many ways to model supply noise. The typical industry rule of thumb of a uniform $\pm 10\%$ [23] provides reasonable DC bounds but gives no information about the frequency content of the noise. The DC level will vary, but transient voltage spikes will be present as well. Both effects need to be considered.

A number of papers have studied power supply noise in ASICs. One study suggests that decoupling capacitors remove high fre-



(a) Transient noise (μ=0.95V, σ variable)

(b) DC noise (μ variable, σ=15mV)

Figure 10: Delay variation due to supply noise.

quency noise, so the supply can be considered a constant DC voltage [24]. Other studies model the supply as a slow sinusoid in the 100-500MHz range [25] to model resonance in the grid with a $\pm 5\%$ peak-to-peak swing. Other work measured power supply noise and found a mixture of deterministic noise caused by the clock signal and its harmonics, random cyclostationary noise caused by switching logic, and random high frequency white noise [26].

Compared to ASICs, FPGAs have a slower user clock, larger interconnect capacitance driven by strong buffers, and more disperse logic. We do not know of any study modeling power supply noise in FPGAs, so the net effect of this is unknown.

In this paper, we model supply noise as a memoryless random process that is normally distributed and changes value every 100ps. The mean DC level, μ, is nominally 1.0V, but analysis focuses on mean DC voltage levels below the nominal because lower voltages limit performance more than higher voltages. The standard deviation, σ, is left as an unknown ranging from 0 to 60mV.

Transient high-frequency voltage spikes should affect cycle-to-cycle latency and thus threaten safe bit transfer, while changes in the DC level affect mean latency but should have little affect on cycle-to-cycle latency. To confirm this assumption, Monte Carlo simulations were performed with random supply noise waveforms with varying DC levels and transient noise of varying magnitude. Latency of one interconnect stage was plotted in histograms and fit to normal curves (not shown; see [27]). The trends are shown in Figure 10, which plots the standard deviation of the change in cycle-to-cycle latency (or jitter) against the voltage noise. As expected, more transient noise leads to higher jitter, whereas more DC noise has little effect on jitter.

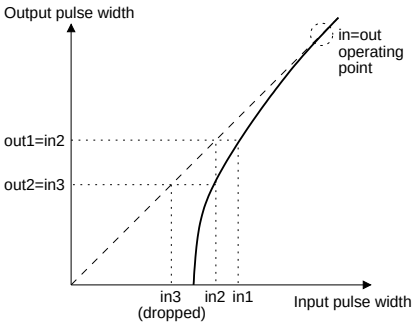
5. ANALYSIS

This section first investigates the noise sensitivity using HSPICE, then formulates a reliability estimate using statistical timing.

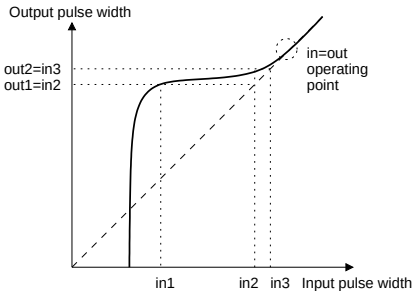
5.1 Minimum safe pulse widths

Prior work in wave pipelining defines the minimum pulse width as the smallest pulse at the link input that produces a 90% swing at the link output [19]. This pulse width is usually determined analytically, but it may also be found from simulation by measuring a stage's response to pulses of varying width, and plotting output pulse width against input pulse width. Because the output of one stage is the input to the following stage, this pulse width transfer curve can predict the behavior of a link.

Figure 11 illustrates pulse transfer curves for wave pipelining and surfing, in which the width of a pulse at the output of a stage is plotted against the corresponding input pulse width. If the curve is below the diagonal, as is the case for wave pipelining, then a pulse will become progressively narrower as it propagates stage-by-stage through the link, until it either reaches the end of the link, or is dropped. If the curve is above the diagonal, as is the case for surfing, then a pulse will become progressively wider, until it settles

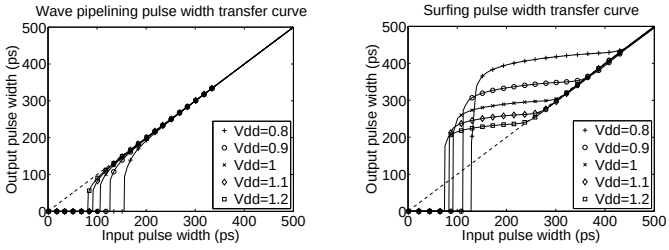


(a) Wave pipelined (unstable)



(b) Surfing (stable)

Figure 11: Illustration of pulse transfer behavior



(a) Wave pipelined, SS

(b) Surfing, SS

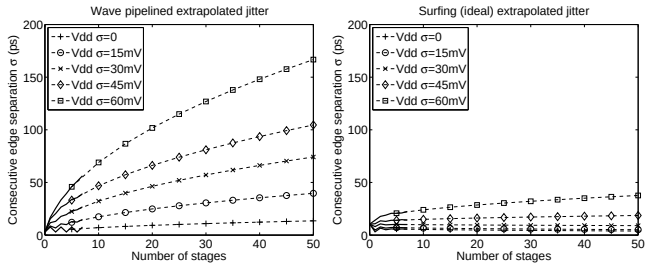
Figure 12: Pulse width transfer curves, from simulation

at the operating point. The latter is the desired behavior: random noise and timing uncertainty as previously discussed will create jitter, causing the input pulse width to vary from the nominal. These variations will be amplified in a wave pipelined link, but surfing restores narrow pulses to the desired width, preventing accumulation of jitter. This behavior is demonstrated in Section 5.2.

Figure 12 shows pulse transfer curves measured from HSPICE simulations at the SS process corner and a variety of DC supply voltages. For wave pipelining, the minimum pulse width is about 200ps (5Gbps) for supply droop alone; adding margin to accommodate dynamic timing uncertainty leads to slower bit rates. For surfing, operating points, which are set by the delay element in the loop, range from about 250ps to 450ps (4Gbps to 2.2Gbps). In surfing, the margin is also a function of the bit rate; margin can be added by increasing the delay and operating at a slower rate.

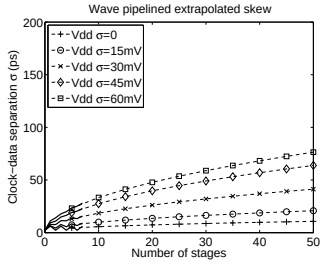
5.2 Jitter and skew propagation

This section uses HSPICE simulations to demonstrate that wave pipelining amplifies timing uncertainty while surfing attenuates it. Two forms of timing uncertainty are examined: jitter, the change in consecutive edges on the clock line, and skew, the time difference

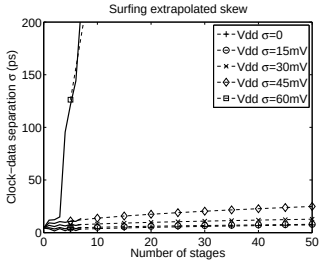


(a) Wave pipelining jitter

(b) Surfing (ideal) jitter



(c) Wave pipelining skew



(d) Surfing (ideal) skew

Figure 13: Jitter and skew propagation (simulation in bold)

between data and clock edges. Jitter and skew can each be modeled as normally distributed random variables with standard deviation σ. In a wave-pipelined link, where we expect jitter and skew to accumulate over a number of stages, the standard deviation at stage N should be $\sqrt{N} \cdot \sigma$. In a surfing link, the jitter and skew should be constant across all stages.

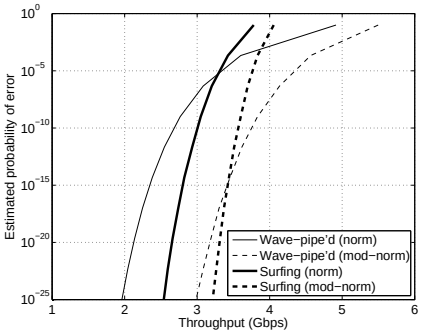
Figure 13 shows the jitter propagation behavior of wave pipelining and surfing taken from Monte Carlo HSPICE simulations. Links of up to eight stages were simulated with a supply of $V_{DD} = 0.95V$ with transient noise applied with varying standard deviations as marked on the graph. To show jitter, the standard deviation of the time separation of consecutive edges on the timing path is plotted; to show skew, the standard deviation of the data-clock separation is shown. The standard deviations were fitted to square-root curves and extrapolated out to 50 stages. The behavior is mostly as expected; jitter and skew accumulates in wave pipelining but is attenuated in surfing. Jitter tends to have larger magnitude than skew. Note these surfing simulations use an *ideal* delay element, rather than a delay circuit where voltage noise affects the delay.

Surfing could not tolerate the largest amount of supply noise simulated, $\sigma = 60mV$. The large spike in skew indicates that data bits are dropped. This particular design uses a delay loop of 250ps; more noise could be tolerated simply by increasing this delay.

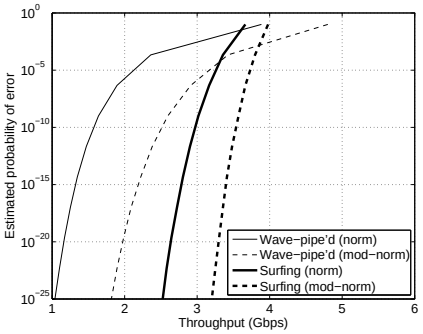
5.3 Reliability estimate

The results in Sections 5.1 and 5.2 can be used to estimate link reliability. By modeling jitter and skew as random variables with the means and standard deviations taken from the data in Figure 13, we can estimate the probability of error. For this analysis, a supply noise level of $\sigma = 30mV$ was chosen.

To prevent a pulse from collapsing, consecutive edges must be wider than the worst-case cutoff pulse widths evident in Figure 12. For wave pipelining, this simply means the nominal bit separation minus accumulated random jitter must be greater than the cutoff. For surfing, ideally the distance between the operating point and cutoff determines the amount of jitter than can be tolerated; in practice, we will assume 70% of this margin can actually be realized. For wave pipelining, the jitter at stage i was measured to be approximately $\sigma = 10.6ps * \sqrt{i}$. For surfing, it is constant at $\sigma = 10.8ps$ regardless of the stage number.



(a) 10-stage link



(b) 50-stage link

Figure 14: Probability of error estimates. Due to uncertainty regarding the noise models, these results should be considered for illustration only.

To prevent incorrect sampling, the total of skew and jitter must be less than one half of a bit period. An additional skew term of $\sigma = 2\%$ of the stage latency is applied to account for random process skew. The overall skew standard deviation is determined from the sum of the random process skew and the random supply noise skew. For wave pipelining, the overall skew at stage i is $\sigma = \sqrt{(5.8ps * \sqrt{i})^2 + (l * 0.02 * \sqrt{i})^2}$, where l is the average latency of the stage. For surfing, it is again constant, at $\sigma = \sqrt{(4.6ps)^2 + (l * 0.02)^2}$.

The probability of a successful transfer is the joint probability that pulses are sufficiently wide and skew is sufficiently small. For simplicity, we can assume these events are independent. To be practical, these error rates should be on the order of 10^{-20} to 10^{-25} so the mean time to failure is much longer than the device lifetime.

Skew and jitter are sensitive to supply variation. Because we do not know the nature of the supply variation, or the true distributions of skew and jitter that would result, we assume they are normally distributed. These are shown as solid lines in Figure 14.

Next, we wish to consider the change in reliability when the distribution assumption is changed. We note the Gaussian tails are unbounded, so the random variables will assign a non-zero probability for large skew values that represent physically impossible events. Rather than arbitrarily bounding the Gaussian, we chose to attenuate these tails more quickly by raising the normal distribution to the fourth power and renormalizing; mathematically, this is the same as cutting the standard deviation in half. These are shown as dashed lines in Figure 14 and labeled "mod-normal".

The graphs show a clear tradeoff between throughput and reliability: to decrease error rate, the circuits must operate at a slower

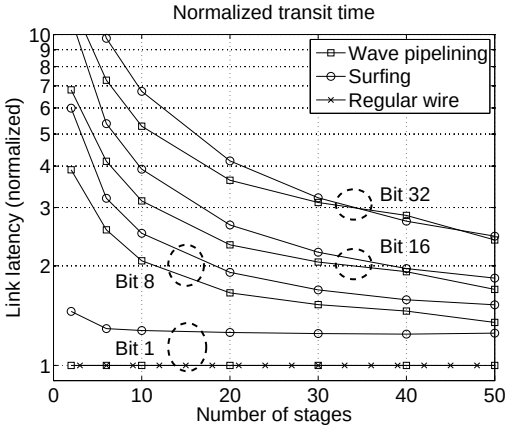


Figure 15: Latency normalized to a regular wire (Vdd $\mu = 0.95$V, $\sigma = 30$mV)

rate. Notice, however, that surfing is much more reliable than wave pipelining in several respects: it is largely insensitive to changes in link length, less sensitive to changes in the underlying noise model, and has a much smaller range in throughput with respect to changes in error rate assumptions.

6. SIMULATION

This section investigates latency, throughput, area, and energy performance. Both wave-pipelined and surfing schemes were simulated in HSPICE using random supply noise and a sixteen-bit data pattern.[3] In this section, surfing uses a *practical* delay element, which is a real transistor circuit subject to voltage noise.

6.1 Latency

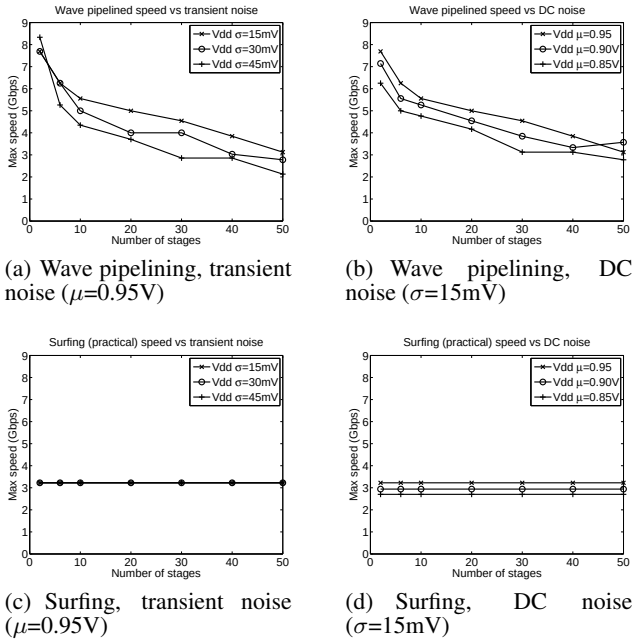
Serial communication suffers from an obvious latency penalty compared to parallel communication, simply because serial bits arrive sequentially while parallel bits arrive all at once. Figure 15 examines this penalty, showing the arrival time of the first bit as well as bits 8, 16, and 32 relative to a regular interconnect wire. The regular interconnect wire follows the same circuit design as the wave-pipelined interconnect (shielded, 25x final driver, CMOS multiplexer), so it should be considered very fast. Latency penalties for a 16-bit word vary from 3-4X for a ten-stage link down to just under 2X for a fifty-stage link. Wave pipelining tends to have lower latency than surfing, but both are significantly higher than a parallel bus on regular interconnect wires.

6.2 Throughput

Throughput was measured by increasing the bit rate with a 10ps step size (50ps for large circuits) until a transmission failure was detected. A Monte Carlo approach was used, where a different random supply noise waveform was applied for each sweep or trial. One sweep takes a CPU day, so only three Monte Carlo trials could be done for large circuits; the maximum bit rate reported is the minimum of all trials. Successive trials almost never produced a different result. Figure 16 shows the results. More trials and a finer step size might smooth the curves, but runtime was prohibitive. Also, due to limited trials, the throughput results in this section may be more optimistic than the previous section.

Wave pipelining throughput exceeds 5Gbps for short links (ten stages or less), but throughput degrades with link length due to accumulated skew and jitter. For a 50-stage link, the throughput

[3]The data pattern [0 1 0 1 0 0 0 1 0 0 1 1 1 0 1 1] was chosen because it includes positive and negative pulses, high frequency switching, and runs of three zeros and ones.



(a) Wave pipelining, transient noise (μ=0.95V)

(b) Wave pipelining, DC noise (σ=15mV)

(c) Surfing, transient noise (μ=0.95V)

(d) Surfing, DC noise (σ=15mV)

Figure 16: Throughput measurements from HSPICE

ranges from 2 to 4 Gbps. Surfing is slower, around 3Gbps, but does not degrade with link length and is insensitive to high-frequency supply noise. Notice, however, that surfing is vulnerable to changes in DC supply noise.

Similar to Figure 1 earlier, Figure 17 shows throughput results compared against a 16-bit parallel bus. Wave pipelining and surfing curves at $\mu = 0.95V, \sigma = 30mV$ are plotted, as well as ideal curves which assume a steady 1V supply with no noise. All three practical curves have a crossover at roughly thirty stages. For very long links, surfing offers superior bandwidth. For very short links, a parallel bus does better. Note that wave pipelining also has high throughput for short links. We will propose a scheme to take advantage of this in the future work section.

6.3 Area and power estimates

Area estimates are shown in Table 1. Transistor counts of each relevant block are provided as well as area in minimum-transistor-widths. The interconnect area includes routing muxes (full CMOS muxes for wave pipelining and surfing, and NMOS-only muxes for regular wires), configuration bits, buffers, and any additional logic. The table shows only transistor area, not wire area. Wave pipelining and surfing wires are assumed to be twice-minimum width with minimum-width/minimum-spacing shields. Assuming minimum spacing is equal to minimum width, a shielded serial bus with one data and one clock wire requires 2.5x the metal area of regular wire with twice-minimum width/twice-minimum spacing.

System-level interconnect area was previously estimated in Figure 2; the result is sensitive to a number of architectural decisions that have yet to be explored. For a complete description of area estimate methodology and results, please see [27].

Energy estimates are shown in Table 2. For a parallel bus, 12.5% data activity is assumed. Wave pipelining and surfing both suffer a significant penalty because they include a timing strobe which has 100% activity, and because serialization increases data activity to 50% on average. Surfing has a further penalty due to the extra logic it has to drive. The penalty, 12-15x, can be reduced to 8-9x by sharing a timing strobe across two data lines. For wave pipelining, the penalty can be further reduced to 6-8x using LEDR encoding [28],

Table 1: Area tabulation

	# transistors	Transistor area (min. transistor widths)
Wave pipelining		
One data wire	140	262
Two data wires	186	369
Surfing		
One data wire	207	402
Two data wires	259	532
Regular wire		
8 bit bus	592	1216
16 bit bus	1184	2432

Table 2: Energy estimates

	Per 8-bit transfer		Per 16-bit transfer	
	Energy(fJ)	Ratio	Energy(fJ)	Ratio
Parallel bus (12.5% activity)	62	1 (base)	124	1 (base)
Wave	744	12.0	992	8.0
Wave+LEDR	496	8.0	744	6.0
Surfing	912	14.7	1168	9.4

in which there is exactly one transition per bit across both data and timing wires. (Surfing cannot use LEDR encoding because it requires a timing strobe with 100% activity.)

7. CONCLUSIONS

This paper proposes adding bit-serial interconnect for word-oriented applications to achieve high-throughput in high-end FPGAs and reduced routing area in low-cost FPGAs. It investigates two of the biggest concerns with bit-serial interconnect, reliability and performance. The next concern, power, is left as future work.

Two bit-serial circuit designs are suggested: wave pipelining and surfing. In both cases, a source-synchronous approach is proposed for improved performance and reliability. In particular, this work has presented the first in-depth study of noise on wave-pipelined interconnect in FPGAs, and proposed surfing as a potential solution.

Wave-pipelined interconnect was shown to have the potential for higher bandwidth while incurring a smaller latency penalty than surfing. However, due to reliability concerns, wave pipelining bandwidth drops as connections are made longer.

Surfing interconnect was shown to be more reliable than wave pipelining: it exhibits less jitter and skew, and the probability of error is less sensitive to voltage noise. Furthermore, surfing bandwidth is insensitive to connection length.

Noise sources are either static or dynamic. Static noise, such as systematic process and temperature variation, results in lower bit rates. Random process variation results in a small skew component that affects wave-pipelining more than surfing. Also, low-frequency supply voltage variation was found to affect wave pipelining and surfing throughput, but not significantly affect reliability as it does not contribute to jitter or skew.

Dynamic or high-frequency variation is the greatest concern for reliability. Crosstalk can be tolerated by shielding the wires. In general, however, tolerating dynamic variation requires extra timing margin, resulting in lower performance. In particular, high-frequency voltage variation inducing cycle-to-cycle jitter is the greatest concern, and jitter is a larger concern than skew. Given the high sensitivity of wave pipelining to supply noise, it is very important to have accurate noise models for the power grid; without them, it is difficult to accurately assess its reliability.

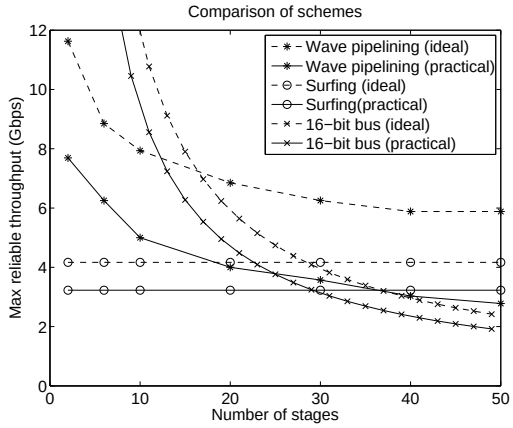


Figure 17: Throughput comparison for all schemes

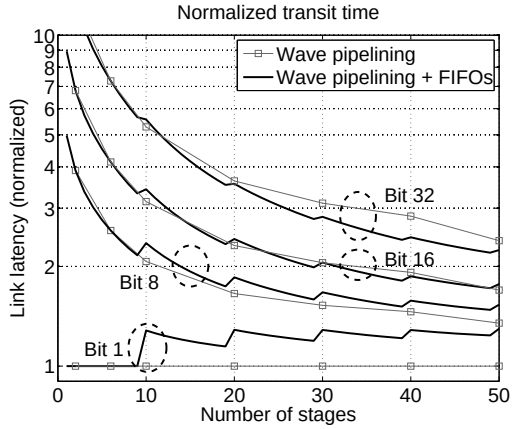


Figure 18: Latency at 5Gbps with 400ps FIFOs

7.1 Future work

Bit-serial interconnect introduces many exciting issues for future work. Power is likely to be the most immediate challenge; accordingly, the first priority for future work is investigating low-power techniques such as low-swing signaling.

For improved throughput, wave pipelining can be 'rebuffered' by placing an asynchronous FIFO roughly every 10 stages. This would attenuate accumulated jitter and would allow higher sustained bandwidths of 5Gbps on long links. Each FIFO would be small, only 1b wide and about 8b deep for 32b words, adding minimal area and latency. Asynchronous FIFOs are simple to design and very fast [13]. Figure 18 shows that an additional 400ps FIFO latency has almost no impact on the total latency of long links, because the higher bit rates attained with FIFOs improves latency.

Once data is transmitted serially, it is possible to replace some parallel computation with serial computation. For example, a serial adder becomes a single BLE, a bit-parallel multiplier becomes a fast adder, and wide multiplexers in the user datapath become a single 1-bit multiplexer. This may save considerable area.

The data transfer window is smaller than the user clock (e.g., 8*300ps=2.4ns versus 10ns), so it may be possible to time-multiplex two serial nets onto one physical wire if they arrive in a routing channel at non-overlapping times. The interconnect would merge (logical OR) the two signals onto a single bit-serial wire and broadcast both bursts to all downstream sinks for both nets. The DES block at each sink accepts either the first or second burst. This may save wiring.

For improved area utilization, the clock and data wires in wave pipelining can be dual-purpose and used to transport regular signals (not bit-serial signals). These wires are fast and shielded, making them highly suitable for clocks, asynchronous signals, or timing-critical nets [29].

The analysis in this paper considered only length 4 wires, but longer wires should be considered. For example, wave pipelining and surfing may achieve optimum performance at different lengths. Architectural experiments also need to determine how many SER and DES blocks are needed, whether SERDES blocks should be in every CLB, how much bit-serial interconnect is required, and what connection patterns should be used. Furthermore, datapath-preserving CAD such as [30, 31] is needed.

Finally, our analysis and simulation shows that properly accounting for noise is essential for designing reliable serial interconnect for FPGAs, and power supply noise has the largest impact of the noise sources we considered. Unfortunately, existing power noise models tend to be simplistic and don't provide the information needed to design robust serial interconnect. Thus, we see better supply noise modeling as a critical area for future work. This is especially important in the context of FPGA designs where noise depends on the functionality and placement of blocks in the FPGA.

8. ACKNOWLEDGMENTS

The authors would like to thank Suwen Yang, Andy Ye, Terrence Mak, Alastair Smith, the Actel architecture team including Jonathan Greene and Sinan Kaptanoglu, and the anonymous reviewers for valuable feedback. This work was funded by NSERC.

9. REFERENCES

[1] M. Saldana, L. Shannon, J.S. Yue, S. Bian, J. Craig, and P. Chow, "Routability prediction of network topologies in FPGAs," *T-VLSI*, vol. 15, no. 8, pp. 948–951, August 2007.

[2] A. Singh, A. Mukherjee, and M. Marek-Sadowska, "Interconnect pipelining in a throughput-intensive FPGA architecture," *FPGA*, pp. 153–160, 2001.

[3] D.P. Singh and S.D. Brown, "The case for registered routing switches in field programmable gate arrays," *FPGA*, pp. 161–169, 2001.

[4] B.D. Winters and M.R. Greenstreet, "A negative-overhead, self-timed pipeline," *ASYNC*, pp. 37–46, 2002.

[5] M.R. Greenstreet and Jihong Ren, "Surfing interconnect," *ASYNC*, 2006.

[6] S. Yang, M.R. Greenstreet, and J. Ren, "A jitter attenuating timing chain," *ASYNC*, pp. 25–38, 2007.

[7] R.R. Dobkin, A. Morgenshtein, A. Kolodny, and R. Ginosar, "Parallel vs. serial on-chip communication," *SLIP*, pp. 43–50, 2008.

[8] T. Mak, C. D'Alessandro, P. Sedcole, P.Y.K. Cheung, A. Yakovlev, and W. Luk. "Implementation of wave-pipelined interconnects in FPGAs," *NoCs*, pp. 213–214, 2008.

[9] T. Mak, P. Sedcole, P. Y. K. Cheung and W. Luk, "Wave-pipelined signalling for on-FPGA communication," *FPT*, pp. 9–16, December 2008.

[10] A.G. Ye and J. Rose, "Using bus-based connections to improve field-programmable gate-array density for implementing datapath circuits," *T-VLSI*, vol. 14, no. 5, pp. 462–473, 2006.

[11] K. Y Yun and R. P Donohue, "Pausible clocking: a first step toward heterogeneous systems," *ICCD*, pp. 118–123, October 1996.

[12] P. Teehan, M.R. Greenstreet, and G. Lemieux, "A survey and taxonomy of GALS design styles," *IEEE Design and Test*, vol. 24, no. 5, pp. 418–428, 2007.

[13] J. Sparsø, *Asynchronous circuit design - a tutorial*, Kluwer Academic Publishers, Boston / Dordrecht / London, December 2001.

[14] I. Sutherland and S. Fairbanks, "GasP: a minimal FIFO control," *ASYNC*, pp. 46–53, 2001.

[15] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," *FPT*, pp. 41–48, December 2004.

[16] C. Yingmei, W. Zhigong, and Z. Li, "A 5GHz 0.18-μm CMOS technology PLL with a symmetry PFD," *ICMMT*, vol. 2, pp. 562–565, 2008.

[17] M. Miller, G. Hoover, and F. Brewer, "Pulse-mode link for robust, high speed communications," *ISCAS*, pp. 3073–3077, 2008.

[18] E. Lee, G. Lemieux, and S. Mirabbasi, "Interconnect driver design for long wires in field-programmable gate arrays," *Journal of Signal Processing Systems*, Springer, 51(1), 2008.

[19] V.V. Deodhar, "Throughput-centric wave-pipelined interconnect circuits for gigascale integration," PhD dissertation, Georgia Institute of Technology, 2005.

[20] S. Yang, B.D. Winters, and M.R. Greenstreet, "Surfing pipelines: Theory and implementation," *JSSC*, vol. 42, no. 6, pp. 1405–1414, 2007.

[21] S. Nassif, K. Bernstein, D.J. Frank, A. Gattiker, W. Haensch, B.L. Ji, E. Nowak, D. Pearson, and N.J. Rohrer, "High performance CMOS variability in the 65nm regime and beyond," *IEDM*, pp. 569–571, 2007.

[22] P. Sedcole and P.Y.K. Cheung, "Within-die delay variability in 90nm FPGAs and beyond," *FPT*, pp. 97–104, December 2006.

[23] N.E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison Wesley, 2005.

[24] S. Kirolos, Y. Massoud, and Y. Ismail, "Power-supply-variation-aware timing analysis of synchronous systems." *ISCAS*, pp. 2418–2421, 2008.

[25] J. Jang, S. Xu, and W. Burleson, "Jitter in deep sub-micron interconnect," *Proc. Symp. VLSI*, pp. 84–89, 2005.

[26] E. Alon, V. Stojanovic, and M.A. Horowitz, "Circuits and techniques for high-resolution measurement of on-chip power supply noise," *JSSC*, vol. 40, no. 4, pp. 820–828, 2005.

[27] P. Teehan, "Reliable high-throughput FPGA interconnect using source-synchronous surfing and wave pipelining," MASc thesis, Department of Electrical and Computer Engineering, University of British Columbia, 2008.

[28] M.E. Dean, T.E. Williams, and D.L. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail (LEDR)," *Proc. Conf. Advanced Research in VLSI*, pp. 55–70, 1991.

[29] M. Hutton, V. Chan, P. Kazarian, V. Maruri, T. Ngai, J. Park, R. Patel, B. Pedersen, J. Schleicher, and S. Shumarayev, "Interconnect enhancements for a high-speed PLD architecture," *FPGA*, pp. 3–10, 2002.

[30] A.G. Ye, J. Rose, and D. Lewis, "Synthesizing datapath circuits for FPGAs with emphasis on area minimization," *FPT*, pp. 219–226, December 2002.

[31] A.G. Ye and J. Rose. "Using multi-bit logic blocks and automated packing to improve field-programmable gate array density for implementing datapath circuits," *FPT*, pp. 129–136, December 2004.