An Asynchronous FPGA with Two-Phase Enable-Scaled Routing

Christopher LaFrieda, Benjamin Hill, and Rajit Manohar
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, USA
{ccl28,ben,rajit}@csl.cornell.edu

***Abstract*—The configurable routing in asynchronous FPGAs accounts for 80-90% of the total area and consumes 80-90% of the total power. This paper presents an asynchronous FPGA that applies two techniques to reduce power consumption. First, the routing is altered to use two-phase logic rather than four-phase logic. Second, enable (acknowledge) signals are voltage scaled such that the overall FPGA performance is not affected. The resulting FPGA is evaluated across eight of the MCNC LGSynth93 benchmarks. This FPGA consumes up to 60% less power than a conventional asynchronous FPGA. In addition, the extra slack provided by two-phase routing increases the throughput of some benchmarks by up to 70%. The additional hardware required to implement the low-power techniques increases the total area by only 12%.**

I. Introduction

A field-programmable gate array (FPGA) is a reprogrammable integrated circuit composed of configurable logic elements surrounded by configurable interconnect. Such a system is attractive because it provides a high-speed solution when compared to software and a low-cost solution when compared to dedicated hardware. Synchronous FPGAs contain a relatively small amount of pipelining. Typically, flip-flops are placed adjacent to the configurable logic elements and the configurable interconnect is not pipelined. As a result, synchronous FPGAs operate at a fraction of the speed of other types of integrated circuits, e.g., application-specific integrated circuits (ASIC).

FPGAs have been shown to be amenable to the fine-grain pipelining of high-speed asynchronous design [1]. Asynchronous FPGAs can run at speeds up to three times faster than their synchronous counterparts [2], [3]. However, this speed advantage comes at a significant increase in both area and power for the configurable interconnect. Whereas the synchronous interconnect contains single-rail data and multiplexers, the asynchronous interconnect contains full asynchronous buffer stages, multiplexers, and programmable c-elements. Bundled-data is not applicable because the FPGA interconnect needs to support bitwise routing. The interconnect in an asynchronous FPGA accounts for 80-90% of the total power consumption.

We propose two techniques to reduce power in asynchronous FGPA interconnect: i) use two-phase logic [4] and ii) apply voltage scaling to the enable (acknowledge) signals. Two-phase logic has been previously proposed for system-level interconnect with four-phase logic used for computation [5]. We show that a similar scheme is feasible for a realistic FPGA architecture. The additional challenges here are: i) the two-phase interconnect is buffered and ii) the protocol converters are more expensive due to the bitwise routing. The resulting interconnect has double the slack, which improves performance in some designs.

We present a novel scheme for applying voltage scaling to the enable signals in the interconnect. Most user designs do not operate near peak throughput because they lack the necessary pipelining to take full advantage of the asynchronous FPGA. In these cases, it is possible to apply voltage scaling to certain regions of the design to save power without impacting performance. We derive a series of equations that determine the amount of voltage scaling possible in these structures without reducing the throughput. Some of the supporting circuitry required is based upon our previous work [6].

This paper is organized as follows. Section 2 presents an overview of the asynchronous FPGA architecture. Section 3 describes the circuits needed for two-phase conversion and enable scaling. Section 4 details the timing analysis needed to properly apply enable scaling. Section 5 discusses the setup details of our experiments. Section 6 presents our results and we conclude in Section 7.

II. FPGA Architecture

A. Overview

A high-level overview of the asynchronous FPGA is shown in Figure 1. The FPGA is composed of configurable logic blocks (CLB) connected together through an array of switch boxes (SB). In a synchronous FPGA, the SB is built out of simple multiplexers. In an asynchronous FPGA architecture, the SB is made of full asynchronous buffer stages, multiplexers, and programmable c-elements. This buffering in the interconnect allows asynchronous FPGAs to run at higher frequencies than synchronous FPGAs. However, this buffering also greatly increases the size of the SB and increases its power consumption. The FPGA is programmed through a chip-wide distributed memory (not shown).

B. Routing

Each switch box has 32 inputs and 32 outputs in each direction, as shown in Figure 2. The switch box is disjoint, i.e., horizontal and vertical tracks connect only to tracks of the same number. The switchpoint is implemented as 32 switch

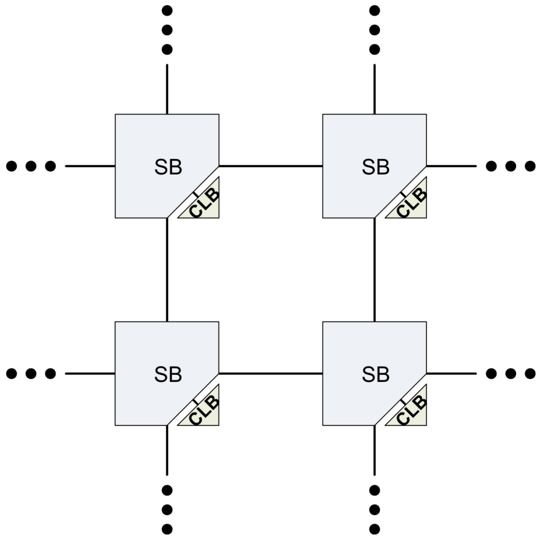

Fig. 1. An asynchronous FPGA fabric composed of switch boxes (SB) and configurable logic blocks (CLB).

points along the diagonal. At each switch point, an input may change directions or enter the CLB. In order for an input to change tracks, it must enter and exit the CLB, burning CLB resources. The switch point can handle up to two different inputs and copy them out to any combination of outputs. Each switch point contains two four-input to four-output (4:4) switches.

It is often the case that outputs of a CLB are inputs to another CLB that is several tiles away, rather than adjacent to their tile. Due to this, it is unnecessary for each track to stop at every SB. To take advantage of these cases, different length wire segments are often used to reduce the area of SBs and decrease the latency through a route. Figure 3 shows the three types of wire segments used in the target architecture. For instance, a hex segment connects SBs that are six tiles apart. In the target architecture, there are 12 singles, 12 doubles and 8 hexes.

C. Configurable Logic Block

The CLB for the target FPGA architecture is shown in Figure 4. The CLB contains a logic core surrounded by input

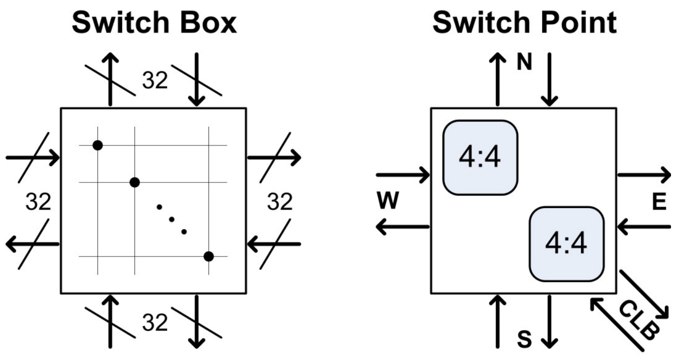

Fig. 2. A 32 x 32 disjoint switch box made from 32 switch points.

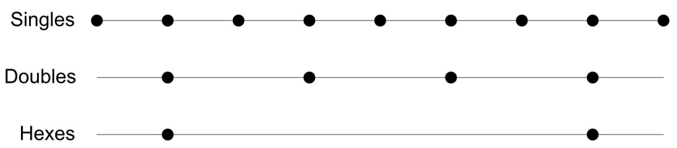

Fig. 3. The three types of routing segments used in this FPGA.

and output connection boxes. The connection boxes allow any input or output of the logic core to connect to any switch point in the SB. Inside the logic core are four four-input lookup tables (LUT4). The LUT4 outputs one of 16 preprogrammed values based upon the four inputs. There are four LUT4s, therefore the logic core has a total of 16 inputs and 4 outputs. To support two-phase routing, the only alteration needed is the addition of phase converters for each input and output of the logic core.

III. Supporting Circuitry

Two types of supporting circuits are needed to implement two-phase switching and enable scaling in the interconnect. Protocol converters are needed to convert between the two-phase logic in the switch boxes and the four-phase logic in the configurable logic blocks. In addition, we require a special two-phase 4:4 switch that can be configured to use a low-voltage enable signal.

A. Protocol Converters

There are two basic protocols used for two-phase handshakes. We will refer to the first protocol as the rail transition (RT) protocol, as shown on the left of Figure 5. With the RT protocol, you simply transition the rail that you want to send as data. For example, we will send a ’1’ from state ’00’. Both the true rail (the first digit) and the false rail (the second digit) are logic low in state ’00’. To send a ’1’, we set the true rail high, which makes the current state ’10’. To send another ’1’, we set the true rail low and the current state returns to ’00’.

The second basic protocol is the level-encoded dual-rail (LEDR) protocol [4], as shown on the right side of Figure 5. Rather than two data rails, the LEDR protocol encodes a data signal and a repeat signal. The data rail is always set to the value of the current token. The repeat rail is toggled when the current token is the same value as the previous token. Unlike the RT protocol, the value of the most recently sent token can

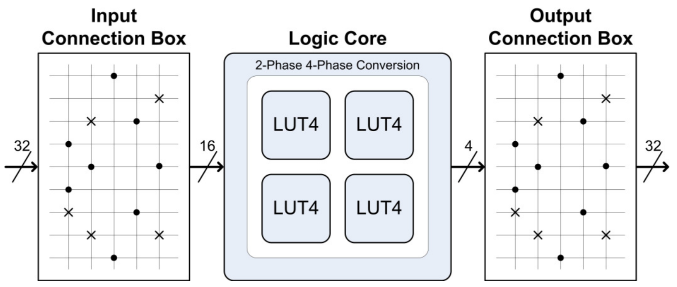

Fig. 4. The CLB contains input/output connection boxes, four LUTs, and phase converters.

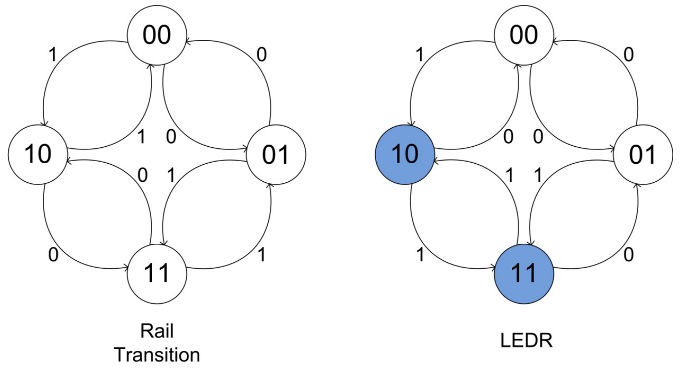

Fig. 5. The rail transition and LEDR two-phase protocols.

be inferred from the current state. For example, the two shaded states, '10' and '11', have most recently sent a '1'.

Which protocol is superior? The two-phase buffer presented in [6] works for both protocols. Single bit protocol converters are roughly the same size for both protocols. However, multi-bit conversions are cheaper with LEDR because the current state can be determined without examining the previous state. In addition, readily knowing the current token value makes debugging two-phase circuits easier. For these reasons, we choose the LEDR protocol for our circuits.

The four-phase to two-phase LEDR converter is shown in Figure 6. The input channel, L, is a standard dual rail channel composed of a true rail (L.t), false rail(L.f), and an enable signal(L.e). The output channel, R, is an LEDR based channel composed of a data rail(R.d), repeat rail(R.r), and an enable signal(R.e). The en variable is implemented as a dual rail variable to avoid using back-to-back latches in the converter. Back-to-back latches introduce some tricky timing constraints because of the need to satisfy their setup and hold times. The latency of the converter is three transitions in the worst case because of the need to invert L^f and L^t. The four-phase to two-phase converter is about $3x$ larger than a simple four-phase buffer.

The two-phase LEDR to four-phase converter takes an LEDR input, L, and produces a dual rail output, R. The converter is shown in Figure 7. Similar to the four-phase to two-phase converter, the enable signal is implemented as a dual rail variable to avoid the use of back-to-back latches. This converter also has a forward latency of three transitions due to inverting the input data rails and is roughly $3x$ larger than a simple four-phase buffer.

B. Enhanced 4:4 Switch

Two-phase circuits have the potential to reduce power by up to 50% compared to four-phase circuits. Unfortunately, two-phase buffers are significantly larger than four-phase buffers. There are two ways to implement two-phase routing for a minimal area impact: i) replace two four-phase stages with a single two-phase stage, and ii) replace a single four-phase stage with a single two-phase stage, but undersize the logic on the backward path to mitigate the area overhead. Typically, the second option would have a larger area impact, but it provides

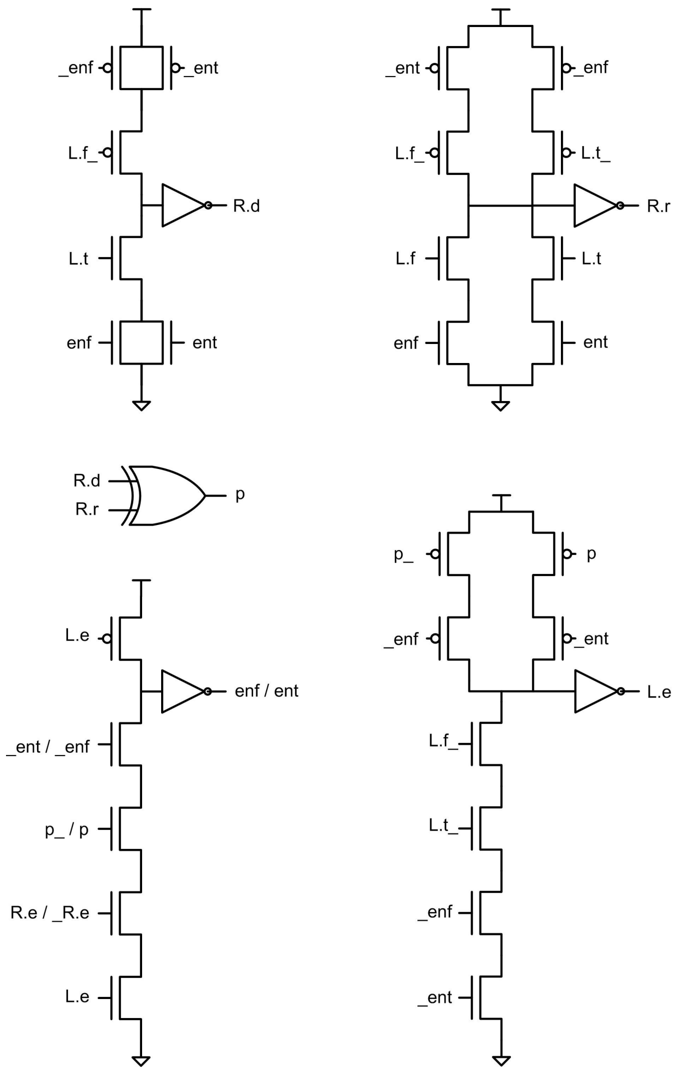

Fig. 6. A four-phase to two-phase LEDR converter.

a compelling performance advantage.

When four-phase half-buffers are replaced with two-phase full-buffers, the amount slack in the pipeline is doubled. This alters the throughput curve [7] of the pipeline, as shown in Figure 8. For the hole-limited domain (the right leg of the triangle), the throughput of the pipeline is greater when using two-phase circuits. This is a very important result because it mitigates the negative performance impact of certain throughput limiting structures, e.g., unbalanced reconvergent paths and hole-limited loops. The throughput of common such structures in an FPGA is discussed in the next section.

In addition to being two-phase, each switch has additional logic to support enable scaling (voltage scaling on enable signals). Figure 9 shows the resulting two-phase low-power switch point. The shaded logic can be configured to use the nominal voltage, V_{dd}, or a lower voltage , V_{ddl}. This is done through a virtual V_{dd} [8]. The programmable c-element labeled PC2V has a built-in voltage converter. Although it may appear that this switch point would be much larger than a typical four-

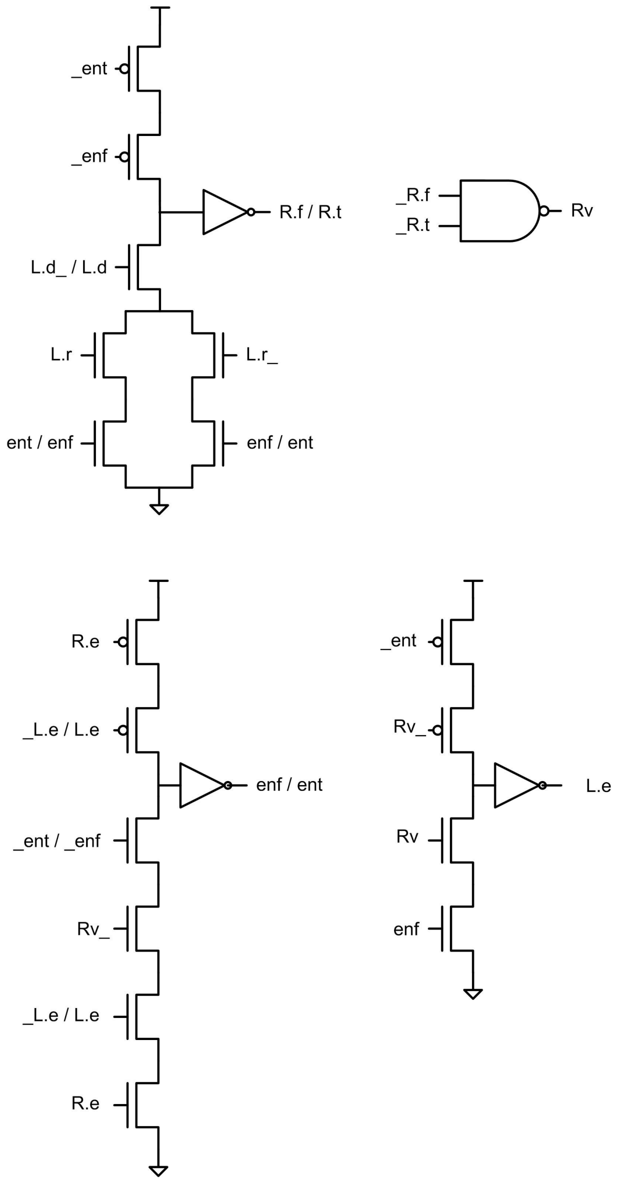

Fig. 7. A two-phase LEDR to four-phase converter.

phase 4:4 switch, it is less than 10% larger. The reason is that all of the logic on the backward path, namely the XOR and PC gates, can be downsized by more than 50% and the circuit will still run at the same frequency of a four-phase 4:4 switch. This style of enable scaling hardware allows each 4:4 switch to be enable-scaled individually. There is a single global low voltage V_{ddl} available on the chip, but each 4:4 switch can be programmed to use V_{ddl} or the nominal V_{dd}.

Only a small amount of logic in the two-phase 4:4 switch runs at a lower voltage. However, most of the capacitance in the circuit resides on the external wires (the two data rails and the enable). The capacitance on each external wire (for a 65 nm process) is 20 fF, 40 fF, and 120 fF for singles, doubles, and hexes, respectively. This dwarfs the less than 5 fF capacitances seen on the internal nodes. The lower-voltage enable rail and one of the nominal-voltage data rails switch

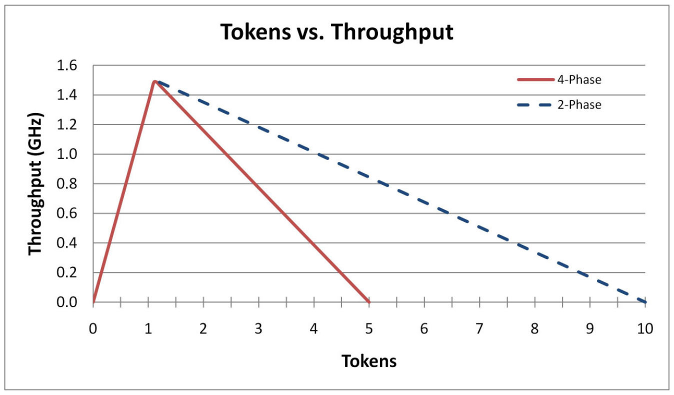

Fig. 8. Throughput improvement in hole-limited domain from using two-phase routing.

each cycle. Therefore, the amount of capacitance that runs at a lower voltage is nearly 50%.

IV. Enable Scaling

In the previous section, we described hardware that allows enable scaling in the FPGA interconnect. Every switch in the switch boxes may be configured to use a global low-voltage source, V_{ddl}, or the nominal voltage source, V_{dd}, for its enable signal. In this section we present the timing analysis needed to determine which circuits may use the low-voltage enable without impacting performance.

A. Voltage Scaling

A high level discussion on voltage scaling in asynchronous architectures can be found in [9]. We focus on voltage scaling in pipelines where the throughput is limited by the architecture. Specifically, we consider token limited loops and reconvergent paths. The goal is to scale voltage in places where there is no impact on performance.

Typically, the equation for dynamic power consumption is given as follows:

$$P_{dynamic} = C{V_{dd}}^2 F \qquad (1)$$

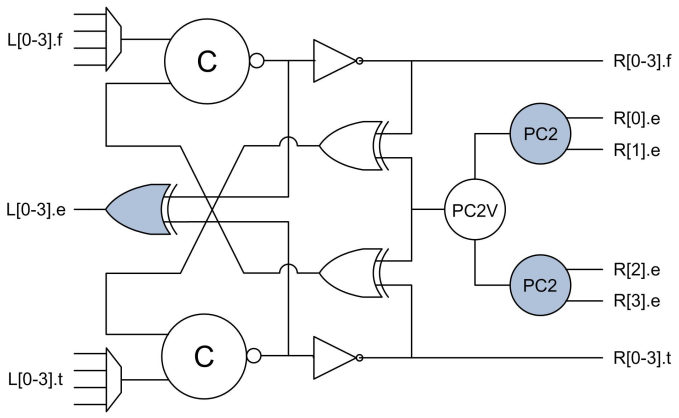

Fig. 9. The 4:4 low-power switch used in the low-power switch point. The shaded logic can be configured to use a lower V_{dd}.

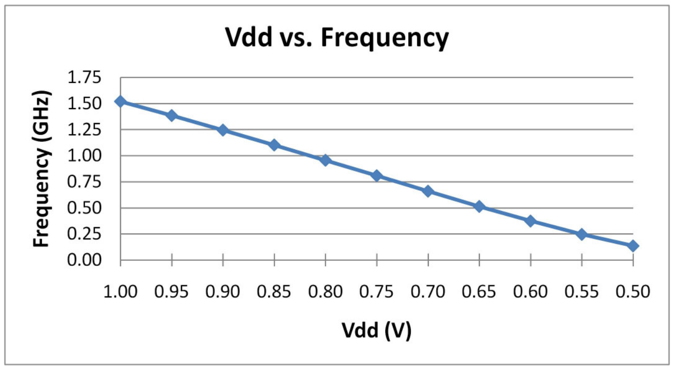

Fig. 10. The operating frequency of a typical asynchronous circuit in a 65 nm process as its supply voltage is scaled.

In this equation, C is the load capacitance, V_{dd} is the supply voltage, and F is the operating frequency of the circuit. We can simplify the relationship between V_{dd} and dynamic power via the following:

$$F \propto V_{dd} \tag{2}$$

$$P_{dynamic} \propto {V_{dd}}^3 \tag{3}$$

The time it takes to charge a capacitor is proportional to $1/V_{dd}$. Therefore, the frequency is proportional to V_{dd}. The resulting operating frequency for V_{dd} scaling in a 65 nm process for a typical asynchronous circuit is shown in Figure 10. We avoid scaling past .5 V (roughly twice the threshold voltage) in this technology because it results in extremely slow circuits. In addition, the power reduction at .5 V is already greater than 90%. Due to the linear relationship of V_{dd} and frequency, dynamic power scales proportionally to ${V_{dd}}^3$.

When circuits are operating at peak throughput, a cubic reduction in power is possible at the cost of a linear reduction in frequency. However, when circuits are not operating at peak throughput, it is possible to reduce power in certain portions of the circuit with negligible impact on frequency. In these cases, the frequency has already been reduced due to some limitation in the architecture and as a result there is a linear reduction in power. That leaves a possible ${V_{dd}}^2$ reduction in power when reducing the supply voltage to match the already limited frequency. Figure 11 shows this additional reduction in power from voltage scaling for a typical asynchronous circuit in a 65 nm technology. Two common throughput limiting structures we will examine are token limited loops and reconvergent paths.

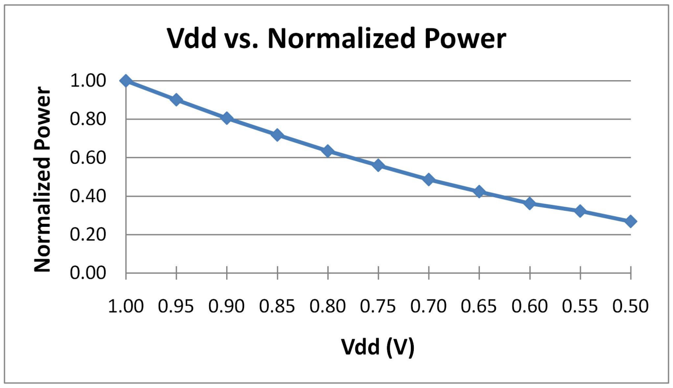

Fig. 11. Normalized power reduction resulting from V_{dd} scaling in a typical asynchronous circuit in a 65 nm technology. Note that this power reduction is in addition to the power already saved from operating at a reduced frequency.

B. Voltage Scaling and Throughput

The throughput of a pipeline, γ, is often defined as a function of slack per token, σ, with dynamic slack, r, and peak throughput, T [7], [10]:

$$\gamma(\sigma) = \begin{cases} \dfrac{T}{\sigma r} & \sigma r \geq 1 \\ \dfrac{T(\sigma - 1)}{\sigma(1-r)} & \sigma r \leq 1 \end{cases}$$

The $\sigma r \geq 1$ case occurs when the pipeline is token-limited. In other words, there aren't enough tokens to keep the pipeline running at full throughput. The throughput of this type of pipeline is limited by the total forward latency divided by the number of tokens. The $\sigma r \leq 1$ case occurs when the pipeline is hole-limited (a hole is a buffer absent a token). The throughput of this type of pipeline is limited by the total backward latency divided by the number of holes.

We consider pipelines composed of one of two types of buffers. For logic functions, we consider pipelines made from four-phase half-buffers. For routing, we consider pipelines made from two-phase full-buffers (all two-phase buffers are full-buffers). The throughput of four-phase(4h) and two-phase(2f) logic is computed as follows:

$$\begin{aligned} 4h\colon T &= \min\left(\frac{k}{nl_f}, \frac{n-2k}{2nl_b}\right) \\ 2f\colon T &= \min\left(\frac{k}{nl_f}, \frac{n-k}{nl_b}\right) \end{aligned} \tag{4}$$

In the above, k is the number of tokens, n is the number of pipeline stages, l_f is the forward latency, and l_b is the backward latency. Note that the token-limited case is the same in both full-buffers and half-buffers.

We envision two possible ways to scale voltage in asynchronous circuits. The first method is to simply scale V_{dd}. This increases both the forward and backward latencies of each stage in a pipeline. The second method is to scale the voltage of only the enable (acknowledge) signals and their associated logic. This method keeps the forward latency fixed, but increases the backward latency of each stage.

Figure 12 shows the impact on throughput when applying each method of voltage scaling in a ten-stage half-buffer pipeline. The outermost triangle (solid line) is the throughput of the pipeline without any scaling (the left leg corresponds to the token-limited domain and the right leg corresponds to the hole-limited domain). The innermost triangle formed by the dotted and dashed lines represents the throughput when V_{dd} is reduced by 40%. All points off of the base of the triangle have worse throughput than the nominal V_{dd} triangle. The triangle with a dashed right leg and solid left leg corresponds to the throughput of the pipeline when the enables are scaled by

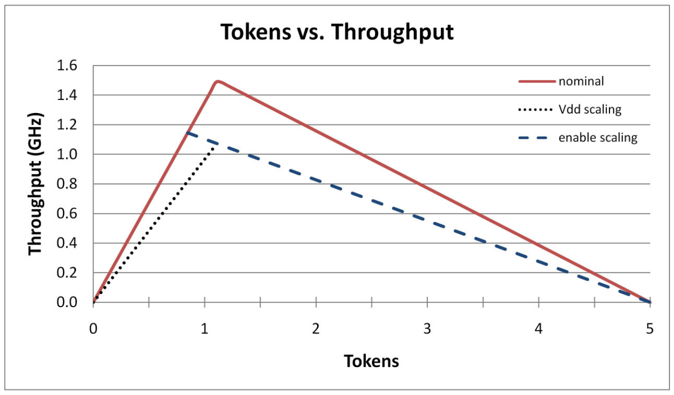

Fig. 12. Impact of V_{dd} and enable scaling on throughput of a ten-stage half-buffer pipeline.

40%. Much of the left leg of this triangle is shared with the nominal V_{dd} case. This makes enable scaling ideal in cases where the pipeline is always token-limited. By and large, user designs that run on an asynchronous FPGA are token-limited. Therefore, enable scaling is our preferred method of voltage scaling.

1) Loops: Loops are ubiquitous in any reasonably complex design. The key to running loops at peak throughput is to have just enough tokens in the loop so that no stage is ever waiting for a token. This happens when $n = k(2l_f + 2l_b)/l_f$ for four-phase buffers and $n = k(l_f + l_b)/l_f$ for two-phase buffers. Three examples of loops are shown in Figure 13. Loops are usually token-limited, rather than hole-limited. This is because adding initial tokens to a loop changes the computation, while adding buffers does not. As a result, loops will often operate on the left leg of the solid triangle in Figure 12. This is an ideal case for enable scaling because the enable voltage can be reduced to some degree without any impact on the throughput.

In order to apply enable scaling to loops, we need to choose the appropriate voltage for the enable signal that saves the most

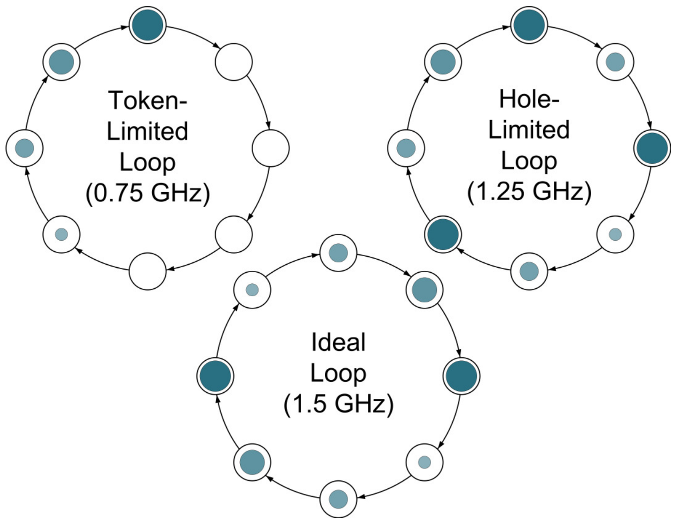

Fig. 13. Three examples of loops. The head of a token is represented by the largest darkest circle. The tail of the token is the smallest lightest circle.

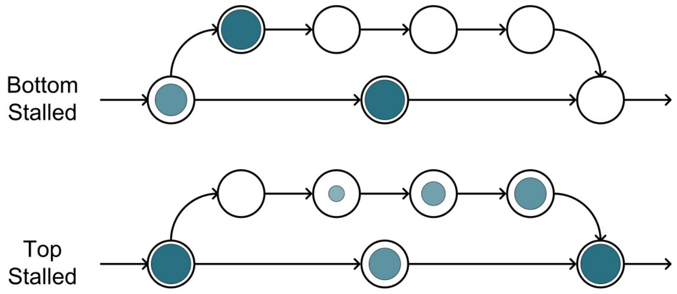

Fig. 14. An example of a reconvergent path. On some cycles the bottom pipeline is stalled and on some cycles the top pipeline is stalled.

power without reducing throughput. This is achieved through a two-step process. The first step is to find the largest backward latency that will not impact the throughput of a token-limited loop. This occurs when the hole-limited domain intersects the loop's operating point on the token-limited domain. For four-phase half-buffers, we solve the following for $l_b{}'$, the increased backward latency:

$$\frac{k}{nl_f} = \frac{n-2k}{2nl_b{}'}$$

The $l_b{}'$ for four-phase(4h) and two-phase(2f) logic is as follows:

$$4h: l_b{}' = \frac{l_f(n-2k)}{2k} \qquad 2f: l_b{}' = \frac{l_f(n-k)}{k} \tag{5}$$

The second step is to characterize the relationship between enable scaling and the backward latency of a specific buffer stage through analog circuit simulation. The optimal value of $l_b{}'$ is then cross-referenced against these results to select the best voltage for the enable signal.

2) Reconvergent Paths: Reconvergent paths are another common structure found in asynchronous architectures. This type of structure is formed whenever a copy is made of a token and both the original and the copy are later used together in some computation. An example of a reconvergent path is shown in Figure 14. When the paths are unbalanced, each can stall the other at different cycles. A reconvergent path will run at full throughput if the amount of slack (buffering) on each pipeline is matched. One of the main goals of an asynchronous designer is to make sure that slack is matched across parallel pipelines in a reconvergent path. However, this is a difficult goal to achieve in reconfigurable architectures where the length and composition of each pipeline are not known at design-time.

The throughput of a reconvergent path is the minimum of each pipeline's throughput [10]:

$$\gamma_{||}(\sigma_a, \sigma_b) \quad = \quad \min(\gamma_a(\sigma_a), \gamma_b(\sigma_b))$$

The throughput of short, ten-stage, and long, 20-stage, half-buffer pipelines are shown in Figure 15. The throughput of the composition of these pipelines is the overlapping triangle with a solid line on its left leg and a dotted line on its right leg. In the steady-state, the number of tokens in this structure is determined by the intersection of the token-limited domain

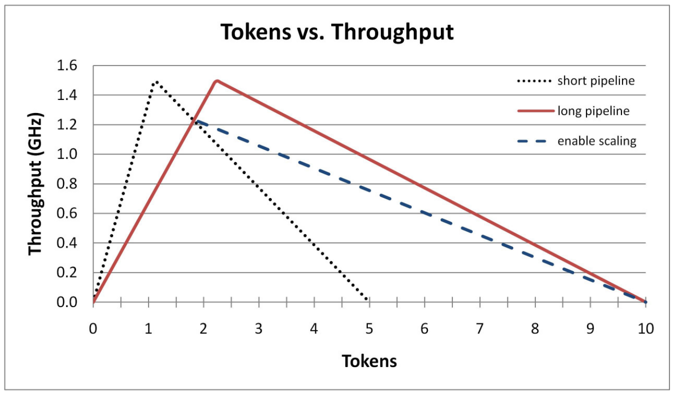

Fig. 15. The throughput of the composition of short, ten-stage, and long, 20-stage, half-buffer pipelines. Enable scaling is shown for the long pipeline.

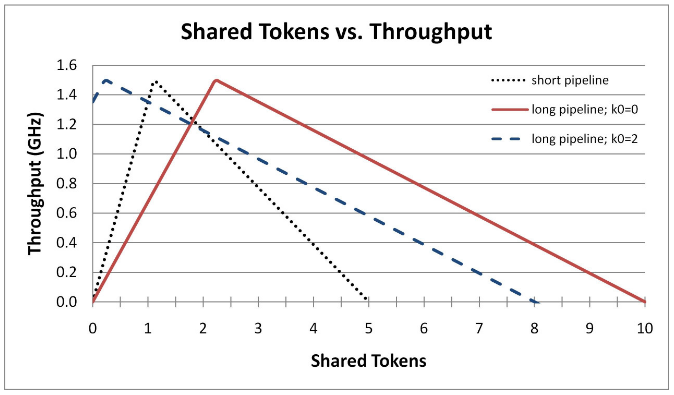

Fig. 16. The throughput of the composition of short, ten-stage, and long, 20-stage, half-buffer pipelines. The throughput plot for the long pipeline is shifted left when two initial tokens are added (dashed line).

of the long pipeline and the hole-limited domain of the short pipeline. Solving for the number of tokens, k, in four-phase and two-phase buffer pipelines results in the following:

$$4h\colon k = \frac{n_l n_s l_f}{2(n_s l_b + n_l l_f)} \qquad 2f\colon k = \frac{n_l n_s l_f}{n_s l_b + n_l l_f} \tag{6}$$

The variables n_l and n_s represent the number of stages in the long and short pipelines, respectively. Substituting these values back into Equation 4 results in the following throughput equations for half-buffers and full-buffers:

$$4h\colon T = \frac{n_s}{2(n_s l_b + n_l l_f)} \qquad 2f\colon T = \frac{n_s}{n_s l_b + n_l l_f} \tag{7}$$

Based on Figure 15, reconvergent paths are another ideal case for enable scaling. However, we would like to scale the enable on the long path only. Scaling the enable on both paths would hurt the throughput because it drops the intersection point on the left leg (solid line) of the throughput triangle.

As we did with loops, we would like to find the largest backward latency (for the long pipeline) that will not reduce throughput. For reconvergent paths, this occurs when the hole-limited domain of the long pipeline intersects with the throughput. For four-phase half-buffer pipelines, we solve the following for ${l_b}'$:

$$T = \frac{n_l - 2k}{2 n_l {l_b}'}$$

Substituting k with Equation 6 and T with Equation 7, yields the following equations for ${l_b}'$:

$$4h\colon {l_b}' = \frac{n_l l_f}{n_s} + l_b - l_f \qquad 2f\colon {l_b}' = \frac{n_l l_f}{n_s} + l_b - l_f \tag{8}$$

3) Reconvergent Paths with Initial Tokens: When there aren't any initial tokens, parallel pipelines in a reconvergent path each contain an equal number of tokens. Tokens enter/exit each pipeline simultaneously. However, if one of the pipelines is initialized with k_0 initial tokens, then it will always contain k_0 more tokens than the other pipeline. Initializing both pipelines with the same number of tokens is equivalent to not having any initial tokens. Approaching the steady-state, tokens will be added or removed from the pipelines until they contain an optimal number of tokens. Therefore, we are only concerned with the difference between of the number of initial tokens in each pipeline. We define k_0 as difference between the number of initial tokens in each pipeline. We define k as the number of *shared tokens* between each pipeline (the number of tokens in the pipeline with fewer initial tokens). P_f and P_m are the pipelines with fewer initial tokens and more initial tokens, respectively. The number of stages in P_f and P_m are n_f and n_m.

In relation to P_f, the throughput curve of P_m is shifted left as k_0 increases. The throughput for the pipeline with more initial tokens is:

$$\begin{aligned} 4h\colon T_m &= \min\left(\frac{k+k_0}{n_m l_f}, \frac{n_m - 2(k+k_0)}{2 n_m l_b}\right) \\ 2f\colon T_m &= \min\left(\frac{k+k_0}{n_m l_f}, \frac{n_m - (k+k_0)}{n_m l_b}\right) \end{aligned} \tag{9}$$

In Figure 16, the longer pipeline is P_m. At $k_0 = 0$ the peak of long pipeline is to the right of the peak of the short pipeline and at $k_0 = 2$ the peak of the long pipeline is to the left of the peak of the short pipeline. This is significant because the ordering of their peaks changes which legs of each throughput curve intersect to form the composite throughput curve. When $k_0 = 0$, we can compare n_f and n_m directly to determine which has an earlier peak. However, at $k_0 \neq 0$ the position of the peak corresponds to a pipeline with ${n_m}' = n_m - k_0\tau/l_f$ stages. Therefore, we compare n_f with ${n_m}'$ to determine the relative ordering of the peaks.

When $n_f \leq {n_m}'$ the throughput is limited by the intersection of the hole-limited domain of P_f and the token-limited domain of P_m. When $n_f \geq {n_m}'$ the throughput is limited by the intersection of the hole-limited domain of P_m and the token-limited domain of P_f. We solve for k at the intersection for each case:

$$4h\colon k = \begin{cases} \dfrac{n_f(n_m l_f - 2k_0 l_b)}{2(n_f l_b + n_m l_f)} & n_f \leq {n_m}' \\ \dfrac{n_f l_f (n_m - 2k_0)}{2(n_m l_b + n_f l_f)} & n_f \geq {n_m}' \end{cases}$$

$$2f: k \quad = \quad \begin{cases} \dfrac{n_f(n_m l_f - k_0 l_b)}{n_f l_b + n_m l_f} & n_f \le n_m{}' \\ \dfrac{n_f l_f(n_m - k_0)}{n_m l_b + n_f l_f} & n_f \ge n_m{}' \end{cases} \tag{10}$$

Substituting the above for k in Equation 9 yields the following equations for throughput:

$$4h: T \quad = \quad \begin{cases} \dfrac{n_f + 2k_0}{2(n_f l_b + n_m l_f)} & n_f \le n_m{}' \\ \dfrac{n_m - 2k_0}{2(n_m l_b + n_f l_f)} & n_f \ge n_m{}' \end{cases}$$

$$2f: T \quad = \quad \begin{cases} \dfrac{n_f + k_0}{n_f l_b + n_m l_f} & n_f \le n_m{}' \\ \dfrac{n_m - k_0}{n_m l_b + n_f l_f} & n_f \ge n_m{}' \end{cases} \tag{11}$$

As we did in the previous subsection, we need to find the target $l_b{}'$ for enable scaling. When $n_f \le n_m{}'$ we intersect the hole-limited domain of n_m with point (k, T) on the throughput graph. When $n_f \ge n_m{}'$ we intersect the hole-limited domain of n_f with point (k, T) on the throughput graph. This yields the following:

$$4h: l_b{}' \quad = \quad \begin{cases} \dfrac{n_f l_b + n_m l_f}{(n_f + 2k_0)} - l_f & n_f \le n_m{}' \\ \dfrac{n_m l_b + n_f l_f}{(n_m - 2k_0)} - l_f & n_f \ge n_m{}' \end{cases}$$

$$2f: l_b{}' \quad = \quad \begin{cases} \dfrac{n_f l_b + n_m l_f}{(n_f + k_0)} - l_f & n_f \le n_m{}' \\ \dfrac{n_m l_b + n_f l_f}{(n_m - k_0)} - l_f & n_f \ge n_m{}' \end{cases} \tag{12}$$

4) Determining V_{ddl}: In order to choose the correct voltage for enable scaling, we characterize the backward latency for a 4:4 enhanced switch in the target process (65 nm) using Hspice. We assume that we can accurately adjust the voltage by increments of $.05V$. Figure 17 shows the backward latency, $l_b{}'$, (left y-axis) as we scale the enable signal from $1V$ to $.5V$. It also shows the power reduction (right y-axis) resulting from enable scaling from $1V$ to $.5V$. Note that the power reduction displayed is in addition to the reduction that results from operating at the reduced frequency.

Although we have reversed the x-axis, $l_b{}'$ follows $f(x) = 1/x$. This is as expected because the delay is proportional to $1/V_{dd}$. As a result, $l_b{}'$ increases slowly through about $.75V$

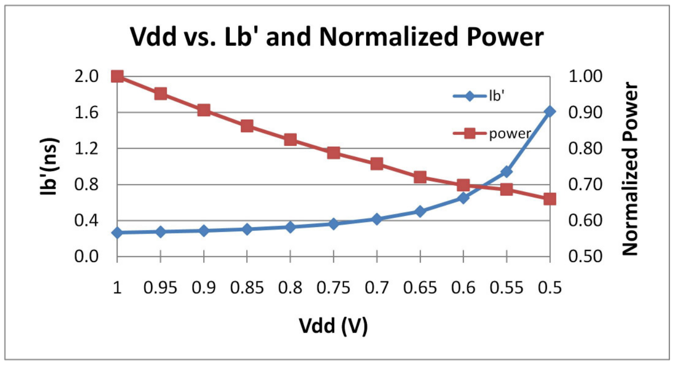

Fig. 17. Relationship of $l_b{}'$ to normalized power during enable scaling for an enhanced 4:4 switch.

and then more quickly as we approach $.5V$. This means that small increases in $l_b{}'$ initially result in large voltage drops for the enable signal, but after about $.75V$ it takes larger increases in $l_b{}'$ to see addition drops in voltage.

We can choose the appropriate V_{ddl} by first finding $l_b{}'$ from Equations 5 and 12, depending on the situation, and cross-referencing that value with Figure 17. For example, suppose the target switch is in a token-limited loop that can tolerate a backward latency, $l_b{}'$, of 400 ps. From this graph, we see that we can scale V_{ddl} down to $.75V$ without exceeding a 400 ps backward latency. This graph also shows that we can expect a more than 20% drop in power at this voltage.

5) Operating vs. Potential Throughput: For the purposes of timing analysis, we can construct a graph representation of user designs that have been mapped to the FPGA. Each node in the graph represents a buffer stage in the FPGA, e.g., LUTs and switches. From this graph, we can identify throughput limiting loops and reconvergent paths. The operating throughput, T_O, of all the nodes in the graph is limited by:

1) the least throughput node anywhere in the graph
2) the least throughput loop anywhere in the graph
3) the least throughput reconvergent path anywhere in the graph

Although each node is limited by T_O, they may have a much higher potential throughput, T_P. Figure 18 shows a simple graph with three potential throughput limiting structures, i.e., Loop 1, Loop 2, and RCP 1. The T_O of the graph is limited by the structure with the least throughput. The T_P of the shaded node may be higher than T_O because it is only affected by the throughput of Loop 1. If $T_P > T_O$, then some amount of enable scaling is possible. The precise amount of enable scaling can be determined by methods described in this section. Specifically, we find the backward latency, $l_b{}'$, that makes $T_P = T_O$, then cross-reference $l_b{}'$ with a characterization of the underlying circuit implementation, similar to that shown in Figure 17. Note, the interactions between loops and reconvergent paths may be more complex than what is depicted in Figure 18. In some cases, more sophisticated analysis was needed to approximate the throughput.

6) Determining Global V_{ddl}: Once we know the minimal enable voltage for each node, V_{ddli}, determining the global V_{ddl} is straightforward. V_{ddl} can be anywhere from 1 V to .5 V at increments of 50 mV. Therefore, there are only 11 possible values for V_{ddl}. For each possible V_{ddl}, we compare V_{ddl} to the V_{ddli} of each node. If $V_{ddl} < V_{ddli}$, then no power savings are possible in this node. If $V_{ddl} \ge V_{ddli}$, then we can lookup the associated power reduction for the underlying circuit with an enable operating at V_{ddl} (similar to Figure 17). We choose a global V_{ddl} that maximizes the overall power reduction.

V. Evaluation Setup

A. FPGA Simulation Environment

Table I highlights the most important architectural parameters of the target asynchronous FPGA. This architecture is designed to support all of the pipelined MCNC LGSynth93 benchmarks. Simulations are performed by generating a netlist

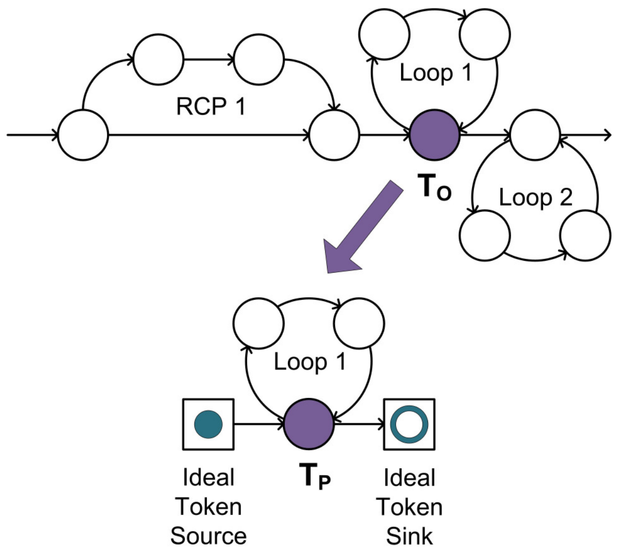

Fig. 18. The operating frequency of the shaded node is limited by the least throughput structure of the two loops and reconvergent path.

for each design and running them in our digital simulator, prsim. The simulator is back-annotated with delays measured in HSpice. Power numbers are determined by simulating each distinct circuit in HSpice at the appropriate frequencies and voltages. We sum the power of each individual circuit to determine the overall power consumption of the design.

B. Benchmarks

The benchmarks used in our evaluations are listed in Table II. These are 8 of the 20 MCNC LGSynth93 benchmarks [11]. Only ten of the MCNC LGSynth93 benchmarks are pipelined and two were excluded because they ran at less than 100 MHz.

Each benchmark was mapped to our CLB design using T-VPack, then VPR was used to place and route the benchmark on our FPGA architecture [12]. The synchronous benchmarks were mapped to the asynchronous architecture by simply converting all flops in the design into initial tokens. The only additional hardware needed to support this is a configurable initial token on the output of each LUT.

VI. Results

A. Area Estimates

Table III lists the area estimates for main components for the baseline FPGA and the low-power version. These area estimates are determined by comparing the total diffusion area of the sized netlist for each component against the post-layout area of similar circuits in this technology. Overall, the low-power FPGA is only about 12% larger than the baseline FPGA.

The largest area increase occurs in the low-power CLB, which is about 36% larger than the baseline CLB. However, in practice this area increase would be much less. Typically, the logic core would contain a number of full-adders which would help to amortize the cost of the phase converters. In addition, the logic core could be altered to use two-phase bundled-data, which would be more compatible with the two-phase routing. Converting from LEDR to two-phase bundled-data is much cheaper than converting from LEDR to QDI.

B. Power and Performance

Figure 19 shows the operating frequency of each benchmark with four-phase (baseline), two-phase, and two-phase enable-scaled routing. None of the benchmarks come within 40% of the peak frequency of the underlying architecture. This is partly due to the fact that a synchronous place and route tool, VPR, was used to map the designs to the FPGA. (An academic asynchronous place and route tool does not exist yet.) However, even if an asynchronous place and route were used, these benchmarks do not contain enough pipelining to run near the peak frequency of the technology. Synthesis at a much higher level would be required to take full advantage of the high-speed asynchronous FPGA. Even at these speeds, some of these benchmarks may be 2-3 x faster than they would be running on a synchronous FPGA.

Moving from four-phase routing to two-phase routing results in a 40% performance improvement in bigkey and dsip, and a 70% performance improvement in elliptic. These designs were limited by either a reconvergent path or a hole-limited loop. Two-phase circuits double the slack in the routing, which drastically improves the throughput in these structures. Due this performance increase, the power reduction in these benchmarks from two-phase routing is much less than the

TABLE I
TARGET FPGA ARCHITECTURAL PARAMETERS.

FPGA	
Fabric Maximum Frequency	1.5 GHz
Process Technology	65 nm
Switch Box	32 x 32 Disjoint Network
Wire Segments	12 Singles, 12 Doubles, and 8 Hexes
Logic Core	4 4-input LUTs
Array Size	48 x 48
Place and Route	VPR

TABLE II
THE EIGHT MCNC LGSYNTH93 BENCHMARK CIRCUITS USED IN EVALUATIONS.

Name	Array Size	LUT Count
bigkey	36 x 36	1707
clma	47 x 47	8383
diffeq	20 x 20	1497
dsip	36 x 36	1370
elliptic	31 x 31	3604
frisc	30 x 30	3556
s38584.1	41 x 41	6447
tseng	17 x 17	1047

TABLE III
ESTIMATED AREA OVERHEADS FOR THE LOW-POWER FPGA CIRCUITS.

Circuit	Area Overhead
Low-Power Switch Point	8%
Low-Power Switch Box	8%
Low-Power CLB	36%
Low-Power FPGA	**12%**

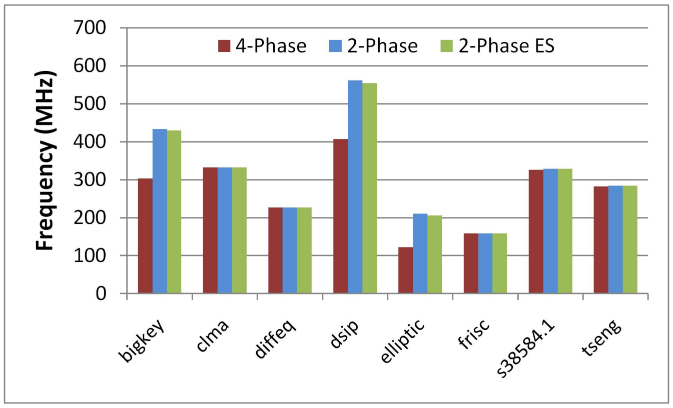

Fig. 19. Operating frequency of each benchmark for four-phase, two-phase, and two-phase enable-scaled routing.

other five benchmarks, as shown in Figure 20. The bigkey benchmark has a 15% power decrease, dsip has a 30% power decrease, and elliptic has a 3% increase. There is a 40% power decrease in the remaining benchmarks. The full 50% power decrease is never seen because the slight area increase from using two-phase circuits makes the wires in the routing a bit longer and more capacitive. However, even a 40% power reduction is quite large.

Enable scaling provides an additional 28% power reduction across all benchmarks. The choice of V_{ddl} for clma and s38584.1 is .55 V and .5 V is used for all the other benchmarks. The power reductions are close to the theoretical 35% power reduction possible from enable scaling down to these operating frequencies. This occurs because a high percentage of switches can be enable scaled. The structures that prevent enable scaling, such as being on the short path of a reconvergent path or on a hole-limited loop, are rare. In addition, two-phase routing fixes some of these structures and prevents them from limiting enable scaling. Although elliptic sees a total power reduction of only 25% because of its 70% performance increase, the other benchmarks experience a power reduction of 40% - 60%.

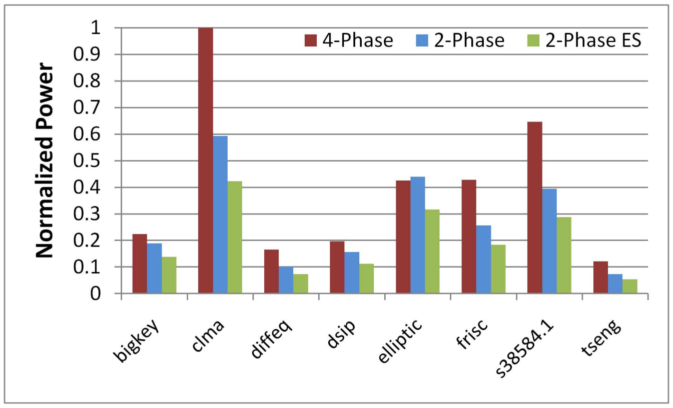

Fig. 20. Normalized power consumption of each benchmark for four-phase, two-phase, and two-phase enable-scaled routing. All benchmarks are normalized to the clma benchmark.

VII. Conclusions

The interconnect in asynchronous FPGAs consumes a large amount of power compared to synchronous FPGA interconnect. Roughly 80-90% of total power consumption in an asynchronous FPGA is attributed to its buffered interconnect. We presented two techniques to reduce power consumption in asynchronous FPGA interconnect: i) two-phase logic, and ii) enable scaling. We designed hardware to support these techniques and presented the analysis to determine where and when to apply enable scaling. For eight of the MCNC LGSynth93 benchmarks, two-phase interconnect provides up to a 70% performance improvement and up to a 40% power reduction. The enable scaling circuits provide an additional 30% power reduction across these benchmarks. The total power reduction for applying both these methods is up to 60%.

VIII. Acknowledgements

This research was supported in part by the Air Force Rome Laboratories under contract no. FA8750-07-2-0191, and in part by the FCRP/DARPA Center on Circuits and Systems Solutions (C2S2).

References

[1] J. Teifel and R. Manohar, "Highly pipelined asynchronous fpgas," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2004, pp. 133–142.

[2] C. Maxfield. (2008, September) New 1.5 ghz fpgas shipping now! [Online]. Available: http://www.pldesignline.com/210601830

[3] K. Morris. (2008, September) Fast. very, very fast. [Online]. Available: http://www.fpgajournal.com/articles_2008/20080916_fast.htm

[4] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail (ledr)," in *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*. Cambridge, MA, USA: MIT Press, 1991, pp. 55–70.

[5] A. Mitra, W. F. McLaughlin, and S. M. Nowick, "Efficient asynchronous protocol converters for two-phase delay-insensitive global communication," in *ASYNC '07: Proceedings of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 186–195.

[6] C. LaFrieda and R. Manohar, "Reducing power consumption with relaxed quasi delay-insensitive circuits," in *ASYNC '09: Proceedings of the 2009 15th IEEE Symposium on Asynchronous Circuits and Systems (async 2009)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 217–226.

[7] A. M. Lines, "Pipelined asynchronous circuits," Master's thesis, California Institute of Technology, 1996.

[8] S. Mutoh, T. Douseki, Y. Matsuya, S. Shigematsu, and J. Yamada, "1-v power supply high-speed digital circuit technology with multithreshold-voltage cmos," in *IEEE J. Solid-State Circuits*. IEEE Computer Society, 1995, pp. 847–854.

[9] R. Manohar and M. Nystrom, "Implications of voltage scaling in asynchronous architectures," Cornell Computer Systems Lab CSL-TR-2001-1013, Tech. Rep., April 2001.

[10] J. Teifel, R. Manohar, D. Fang, C. Kelly, and D. Biermann, "Energy-efficient pipelines," in *ASYNC '02: Proceedings of the 8th International Symposium on Asynchronous Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2002, p. 23.

[11] S. Yang, "Logic synthesis and optimization benchmarks, version 3.0," Microelectronics Centre of North Carolina, Tech. Rep., 1991.

[12] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. Fang, and J. Rose, "Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," *FPGA '09*, Feb 2009.