Energy-Efficient Pipelines

John Teifel, David Fang, David Biermann, Clint Kelly, and Rajit Manohar
Computer Systems Laboratory
Electrical and Computer Engineering
Cornell University
Ithaca, NY 14853, U.S.A.

Abstract

We discuss the design of energy-efficient pipelines for asynchronous VLSI architectures. To maximize throughput in asynchronous pipelines it is often necessary to insert buffer stages, increasing the energy overhead. Instead of optimizing pipelines for minimum energy or maximum throughput, we consider a joint energy-time metric of the form $E\tau^\alpha$, where E is the energy per operation and τ is the time per operation. We show that pipelines optimized for the $E\tau^\alpha$ energy-time metric may need fewer buffer stages and we give bounds when such stages can be removed. We present several common asynchronous pipeline structures and their energy-time optimized solutions.

1. Introduction

Designing systems for the joint energy-time metric, $E\tau^\alpha$, involves a non-trivial tradeoff between performance and power consumption. Given a fixed α, which determines the desired optimization point between time and energy, it is not immediately apparent how much cycle time (τ) we should sacrifice for energy (E) and vice-versa. In this paper, we analyze the energy-time optimization problem for asynchronous pipelines and show how our pipeline results can be applied to micro-architectural design issues.

The dynamics of asynchronous pipelines are well known and it is straightforward to optimize systems for steady-state throughput [1, 4, 15]. Peak pipeline throughput occurs when the handshakes of each pipeline stage operate at their maximum rate. To achieve this, it is usually necessary to space tokens physically across the pipeline by adding buffer stages in between computation stages (this procedure is called *slack matching* [10]). The number of additional buffers (called *slack-matching* buffers) required depends on the circuit characteristics of the pipeline stages. For instance, Quasi-Delay Insensitive (QDI) asynchronous circuits typically need either 8 half-buffers per token or 4 full-buffers per token to operate at full throughput [10].

In systems with deep pipelines and wide datapaths, slack-matching buffers can consume excessive energy. Since these buffers are added only for performance reasons, we are free to vary their number without changing the correctness of the system.[1] For energy-efficient designs the number of slack-matching buffers will be less than or equal to the throughput optimal case, since adding more buffers can only increase the energy without any improvement in throughput. In addition to the circuit implementation details, the number of slack-matching buffers for optimal $E\tau^\alpha$ designs depends on the computation energy of the pipeline, the energy of the buffers, and on α.

To optimize a pipeline for energy-efficiency, we minimize a design's $E\tau^\alpha$ value. E and τ are functions of the number of slack-matching buffers and α is fixed for a given metric. To optimize for energy-efficiency we take the derivative of $E\tau^\alpha$ with respect to the number of slack-matching buffers and calculate a local minimum. A global $E\tau^\alpha$-minimum can be obtained by choosing the smaller of the local minimum and the throughput-optimal number of slack-matching buffers.

We target the metric, $E\tau^2$, because it is approximately independent of the operating voltage, V [10]. Over transistor operating regions of interest to digital circuits, E is proportional to V^2 and τ is proportional to $1/V$. So to first approximation, $E\tau^2$ will be independent of voltage. Thus given two designs we can compare their energy efficiencies, regardless of their individual operating voltages. For asynchronous circuits, voltage independent metrics are desirable so that a circuit will have a single metric value when the circuit is running at low voltages (for low power applications) and at high voltages (for high performance applications).

Common energy-time metrics include the Power-delay product (equivalent to E; $\alpha = 0$) [2], the Energy-delay product (equivalent to $E\tau$; $\alpha = 1$) [3], and MIPS/Watt (equivalent to $1/E$). The above metrics suffer because

[1] We restrict our attention to slack-elastic pipelines [5].

they are not voltage independent and are too energy dominated ($\alpha \leq 1$), resulting in slow circuits. Nowka, Hofstee, and Carpenter discuss more complicated energy-time models that account for subthreshold and velocity saturation effects, and show that $E\tau^2$ is an appropriate approximation for practical designs [12].

While we focus on fine-grained pipelines constructed from QDI circuits using the synthesis methodology found in [9], our approach is generally applicable to all pipelines designed using asynchronous handshake circuits. We simply choose QDI circuits for ease of illustrating our analysis of energy-efficient pipelines. Any other asynchronous circuit design style only changes the values obtained for $E\tau^2$ and does not invalidate our conclusions. Our approach also covers coarse-grained pipeline designs, although these designs will have lower throughput and often lead to poorer $E\tau^2$ values.

Existing work in designing for joint energy-time metrics has either focused on high-level architectural optimizations [7, 14] or has been limited to the circuit level [2, 3, 12]. However, between these two extremes there is a large degree of flexibility in choosing how to optimize for $E\tau^\alpha$. This is especially true in asynchronous systems, where logical and physical pipelines are decoupled. Martin provides an $E\tau^2$ comparison between sequential and linear pipelines, but does not address the slack-matching design problem or consider more complex pipeline structures [11]. Our goal is to provide insight into how to design asynchronous pipelines that are energy efficient and use an energy-time optimal number of slack-matching buffers.

This paper is organized as follows. Section 2 gives an analytic framework that we use in the ensuing sections. In Sections 3, 4, and 5 we analyze energy efficient solutions for typical asynchronous pipeline topologies: ring, parallel, and interleaved pipelines. Section 6 discusses composing $E\tau^2$ pipelines and Section 7 presents an $E\tau^2$ optimal slack-matching solution for array accesses. In Section 8, we discuss micro-architectural choices for designing an energy efficient exception handling mechanism. Concluding remarks are in Section 9.

2. Pipeline Concepts

Much of the performance optimization notation and terminology used in this paper is adopted from Lines [4]. We define a *token* to be a data element in a pipeline stage, and a *hole* to be the absence of a data element in a pipeline stage. Tokens and holes travel in opposite directions through a pipeline. In four-phase handshake circuits, tokens (and holes) can be distributed across multiple pipeline stages.

The static slack of a pipeline, s, is the upper bound on the difference between the number of communication cycles on sender and receiver ends of a pipeline, and represents the maximum number of tokens that a pipeline can hold. Static slack is most easily implemented with *buffers*, which are circuits that carry and propagate tokens. Two such classes of QDI buffer circuits are described in [4]: *half-buffers* which have a static slack of $\frac{1}{2}$ and *full-buffers* which have a static slack of 1. Dynamic slack, *d*, is the number of tokens at which maximum steady-state throughput is achieved [4]. Williams defines *dynamic wavelength* to capture the notion that a token can be spread over multiple pipeline stages [15]. When a pipeline operates at full steady-state throughput the dynamic wavelength is the inverse of the dynamic slack.

The forward latency, l_f, is the delay of a token traveling forwards through a pipeline stage and the backward latency, l_b, is the delay of a hole traveling backwards through a pipeline stage [15]. The forward and backward latencies of a buffer determine its *local cycle time*, which is the time between successive token receives on a buffer assuming it is surrounded by an infinitely fast environment. The local cycle time is an ideal time, and may never be achieved in practice because of handshaking overheads when buffers are connected together to form a pipeline. The *pipeline cycle time* or simply *cycle time*, τ, is the time between successive token receives on a buffer stage in a pipeline. The total forward latency of a pipeline is simply the sum of the forward latencies for each stage, and total backward latency is analogously defined. We define the *steady-state throughput* as the throughput of a pipeline operating with a fixed number of tokens. Steady-state throughput is useful for analyzing the performance of pipelines where the number of tokens is constant, such as in many closed systems, where there is a cyclic dependency among pipeline stages.

The steady-state throughput of a linear pipeline can be classified into a *token-limited* or *hole-limited* domain.[2] Token-limited throughput is limited by the total forward latency of a pipeline divided by the number of tokens. A large, nearly empty pipeline ring with a single token racing around is an example of token-limited operation because the throughput is limited by how fast a single token can move forward in the pipeline. Hole-limited throughput is limited by the total backward latency divided by the number of holes. A large, nearly full pipeline ring with a single hole is limited by how fast that hole can move backwards around the pipeline. The token- and hole-limited domains of throughput (as a function of the number of tokens) define two lines, which form a triangle function, so the dynamic slack of a linear pipeline is unique.[3]

The throughput-optimal number of tokens in a linear pipeline is proportional to the dynamic slack of the pipeline

[2] Williams refers to these regions as "data-limited" and "bubble-limited," respectively.

[3] We will assume, for simplicity, that the local cycle time of a pipeline stage does not limit the total pipeline throughput. In practice, the peak of the triangular throughput function is deformed due to this second order "handshake-limited" constraint [15].

stage, r, which is equal to the forward latency over the local cycle time for full-buffers.[4] Typical values for r in QDI full-buffer pipelines are between $\frac{1}{8}$ and $\frac{1}{4}$. The dynamic and static slacks of the pipeline are related by the equation $d = sr$. Thus a linear pipeline, supporting k tokens, will require a total of k/r buffers to run at peak throughput. *Slack matching* is the process of adjusting the slack of a pipeline in order to optimize its throughput.[5]

In our analysis, we define a normalized static slack per token, $\sigma = s/k$, with $0 \le k \le s$. We define the throughput γ as a function σ, with peak throughput T [4]:

$$\begin{aligned} \gamma(\sigma) &= \begin{cases} \frac{T}{\sigma r} & 1 \le \sigma r \\ \frac{T(\sigma-1)}{\sigma(1-r)} & \sigma r \le 1 \end{cases} \\ T &= \max_{\sigma} \gamma(\sigma) \end{aligned}$$

The domain where $1 \le \sigma r$ represents token-limited throughput, and the domain where $\sigma r \le 1$ represents hole-limited throughput. Figure 1 shows a contour plot of the throughput γ as a function of static slack s and the number of tokens k independently (from simple substitution of $\sigma = s/k$ in $\gamma(\sigma)$), and also shows profiles of the throughput function for constant number of tokens $\gamma_k(s)$ and of constant static slack $\gamma_s(k)$. The continuously differentiable pieces of the throughput function under a constant number of tokens $\gamma_k(s)$ are inverse linear with respect to static slack. Globally maximum throughput $\gamma = T$ occurs along the entire ridge $\sigma = s/k = r$.

Throughout this paper, we refer to the hole-limited region of the throughput function as $\gamma^+(\sigma)$ and the token-limited region as $\gamma^-(\sigma)$. By definition, the throughput region that is increasing with respect to k is decreasing with respect to s. In the energy-efficiency metric, $E\tau^{\alpha}$, τ is the cycle time which is the reciprocal of throughput, so $\tau^+(\sigma) \equiv \frac{1}{\gamma^-(\sigma)}$ and $\tau^-(\sigma) \equiv \frac{1}{\gamma^+(\sigma)}$.

3. Rings of Pipeline Stages

Token rings appear in several places in concurrent VLSI architectures. Examples of token rings include iterative divider architectures [15] as well as data-dependent instruction execution and program counter computations [10].

A token ring consists of n concurrent pipeline stages connected in a ring, with the output of process i connected to the input of process $(i+1) \bmod n$. In this section we assume each stage in the ring is an idealized half-buffer with latencies l_f and l_b, and takes time $2l_f + 2l_b$ before it is ready to accept the next input token.[6] For a ring with k tokens constructed from half-buffer stages,[7] the cycle time is given by the expression

$$\tau = \frac{n}{k} \max\left(l_f, \frac{l_b}{\frac{n}{2k} - 1}\right)$$

and the optimal operating point occurs when $n = k(2l_f + 2l_b)/l_f$ [15]. At this choice of n, a data token moves around the ring and back to the input of a computation stage just when the stage is ready to accept its next input. Normally, buffer stages are inserted into the ring to ensure that n is equal to this optimal value. We now consider the impact of taking energy into account for this optimization problem.

The total energy required will be proportional to the amount of energy required by the stages that perform computation plus the energy required by the additional buffer stages. The total energy is given by $E_f + nE_b$, where E_f is the energy expended in functional computation, and E_b is the energy required per buffer.

Since we are considering the case of adding buffering, we know that without any buffer stages $n < k(2l_f + 2l_b)/l_f$. If we minimize the expression $(E_f + nE_b)((n/k)\max(2l_b/(n/k-2), l_f))^{\alpha}$ with respect to n,

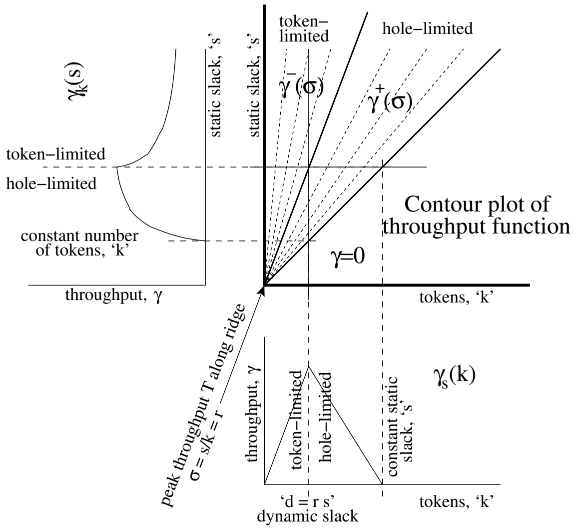

Figure 1. Throughput of a linear pipeline as a function of static slack and number of tokens, and profiles of the function for constant number of tokens and constant static slack.

[4] For half-buffers, r is equal to twice the forward latency over the cycle time.

[5] Slack matching applies more generally to all pipeline topologies and not just linear pipelines [10].

[6] This time is a typical time and depends on the exact reshuffling of the buffer stage [4].

[7] If full-buffers are used, $\tau = n/k \max\left(l_f, l_b/(n/k-1)\right)$.

we obtain:

$$n_{opt} = k(\alpha+1)\left(1+\sqrt{1+\frac{2\alpha(E_f/E_b)}{(1+\alpha)^2}}\right).$$

If n_{opt} satisfies constraint $n_{opt} < k(2l_f + 2l_b)/l_f$, then it corresponds to the optimal $E\tau^\alpha$ choice for the number of stages in the ring.

In prior state-of-the-art QDI designs, $(2l_f + 2l_b)/l_f \approx 8$ [10]. For our motivating metric of $E\tau^2$, $\alpha = 2$. The result of our optimization will result in a different operating point when $n_{opt} < k(2l_f + 2l_b)/f$. Using the equation above, this occurs just when $E_f/E_b < 4$. Therefore, if the total energy required for computation is less than four times that for a single stage of buffering, $E\tau^2$-optimized rings will have fewer number of stages than those optimized just for performance.

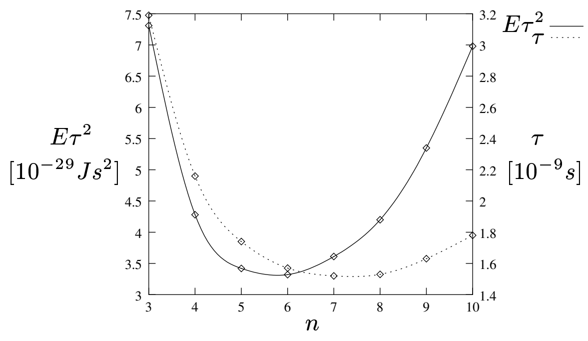

Figure 2. $E\tau^2$ and τ metrics for a ring of n QDI half-buffers in 0.25μm CMOS ($k = 1$, $E_f = 0$).

Example: Consider a half-buffer token ring with $k = 1$ and $E_f = 0$. For the handshake reshuffling used in these buffer circuits $E_b \approx 2^{-12}J$, $l_f \approx 0.28ns$, and $l_b \approx 0.64ns$. For $\alpha = 2$, $n_{opt} = 6$ and $(2l_f + 2l_b)/l_f \approx 7$. Figure 2 shows HSPICE results for various sized rings designed in a 0.25μm CMOS process (TSMC). A seven stage ring is optimal for the τ metric and a six stage ring is optimal for the $E\tau^2$ metric, confirming our analytical calculations. In this example, with both rings running at the same voltage, we can save 14% in energy consumption (by eliminating one stage from the τ-optimized ring) if we can tolerate a 3% reduction in performance. However with voltage scaling, and if the energy consumption of the six and seven stage pipelines are equalized, the six stage design will actually have a higher throughput because of its lower $E\tau^2$ value and hence is a better design than the seven stage ring.

4. Parallel Composition of Pipelines

The parallel composition of pipelines is a familiar structure in asynchronous architectures. They appear whenever data or control tokens are sent or copied to multiple processes, and re-synchronized at some later process. A schematic representation is shown in Figure 3. In a typical processor the instruction decoder issues control tokens to execution units and the register file which are then re-synchronized at the writeback unit.

We are given the following characteristics of the two buffer pipelines (that do not necessarily use the same types of buffers): peak throughputs T, static slack s, dynamic slack sr, and cycle energies per buffer E. We also assume that the system operates at a steady-state, thus having a constant number of tokens. The goal is to optimize the static slacks per token, σ_a and σ_b, (and hence the number of buffers) in both branches of a parallel composition of two pipelines for a general energy-time metric.

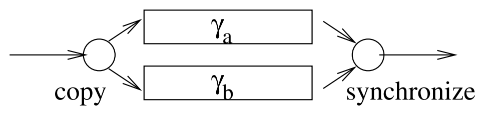

Figure 3. Schematic representation of two parallel pipelines.

Optimizing for τ. First we look at the throughput-optimal solution to the parallel pipeline problem. For fixed static slacks s_a, s_b, the parallel throughput $\gamma_{||}$ of two pipelines is the functional minimum of the two individual throughput functions, $\gamma_a(\sigma_a)$ and $\gamma_b(\sigma_b)$, assuming that the throughput is not constrained by the environment or the copy and synchronize processes.

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

Figure 4 illustrates a typical contour plot of a parallel throughput function $\gamma_{||}(\sigma_a, \sigma_b)$ for $T_a < T_b$. The four continuously differentiable regions of the function represent where the throughput is token- or hole-limited by the A or B pipeline. The global maximum of throughput occurs along ridge C at throughput T_a. The complete mathematical analysis can be found in the technical report by the same authors [13].

Optimizing for $E\tau^\alpha$. Using the $E\tau^\alpha$ metric, our goal is now to optimize the slack in the parallel pipeline for energy-efficiency and speed. Like in the token-ring analysis, the energy scales linearly with the number of buffers (or static slack), plus some computational overhead. Without loss of generality, we assume that the buffers are full-buffers,[8] so the number of buffers n = the static slack $s = \sigma k$. We choose to order the pipelines such that $T_a \leq T_b$; the solution for $T_a \geq T_b$ is symmetric. Since $E_{||}$ is monotonically increasing and continuously differentiable everywhere, $E_{||}\tau_{||}^\alpha$

[8] Whereas for half-buffers, $n = 2s = 2\sigma k$.

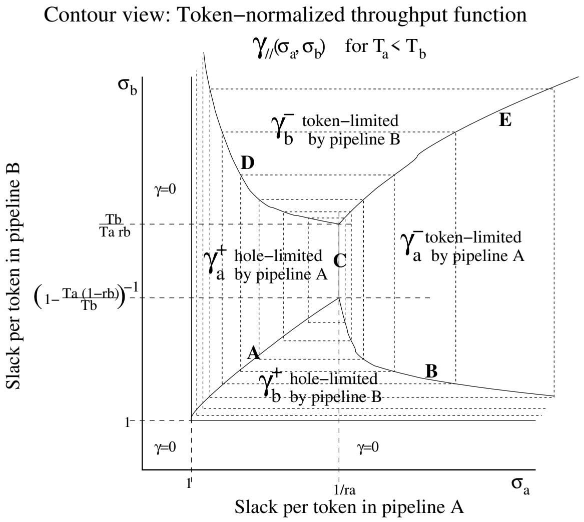

Figure 4. Plot of parallel throughput function $\gamma_{\|}(\sigma_a, \sigma_b)$ regions and boundaries for $T_a < T_b$. (not drawn to scale) Ridges A-E mark the boundaries of the continuously differentiable pieces of the function.

has the same piecewise differentiable boundaries as $\tau_{\|}$.

$$\begin{aligned} E_{\|}(\sigma_a, \sigma_b) &= E_f + \sigma_a k E_a + \sigma_b k E_b \\ \tau_{\|}(\sigma_a, \sigma_b) &= \max\left(\frac{\sigma_a(1-r_a)}{(\sigma_a-1)T_a}, \frac{\sigma_b(1-r_b)}{(\sigma_b-1)T_b}\right) \end{aligned}$$

Since we are minimizing $E_{\|}\tau_{\|}^{\alpha}$, we need not consider the domains where cycle time increases with respect to static slack σ_a, σ_b. For the domains where τ is constant, we only need to consider the lower bound on the static slack, because adding more slack will only increase the energy. Thus we define the upper bound of the solution to be the intersection of ridges A, B, C in Figure 4. The global minimum cycle time for $T_a < T_b$ is found at:

$$\begin{aligned} \sigma_{a_{opt}} &= \min\left(\sigma_a^*, 1/r_a\right) \\ \sigma_{b_{opt}} &= \min\left(\sigma_b^*, (1 - T_a(1-r_b)/T_b)^{-1}\right) \end{aligned}$$

where the σ^*'s are the solutions for the static slacks where the gradient, $\nabla(E_{\|}\tau_{\|}^{\alpha})$ with respect to σ_a, σ_b, is zero. The complete derivation of this solution can be found in the technical report [13]. The solutions for the locally optimal static slacks σ^* satisfy:

$$\begin{aligned} s_a^* = \sigma_a^* k &= u\left(1 + \sqrt{1 + v\frac{E_f + s_b^* E_b}{E_a}}\right) \\ s_b^* = \sigma_b^* k &= u\left(1 + \sqrt{1 + v\frac{E_f + s_a^* E_a}{E_b}}\right) \end{aligned}$$

$$\text{where} \quad u = \tfrac{(\alpha+1)k}{2} \quad \text{and} \quad v = \tfrac{4\alpha}{(\alpha+1)^2 k}$$

This local minimum only depends on the relative ratios of the energy constants, and not on the individual throughput properties of the buffers, because the throughput parameters factor out completely in the gradient. However, the global minimum will still depend on these buffer parameters.

The solutions to the above coupled equations of s_a^*, s_b^* that are of interest are the positive roots of a quartic polynomial. For example, for $E_f = 0$, and equal energy buffers in each pipeline $E_b = E_a$, we can deduce from symmetry that $s_a^* = s_b^* = s^*$. This simplified example is equivalent to the analysis of a full-buffer token ring. The coupled system reduces to a single quadratic in s^*, whose solution is in terms of α and k:

$$\begin{aligned} s^* &= k(2\alpha + 1) \\ \sigma^* &= 2\alpha + 1 \end{aligned}$$

For $\alpha = 2$, a local minimum of $E\tau^2$ occurs at $s^* = 5k$, or $\sigma^* = 5$. With identical buffers $T_a = T_b$ and $r_a = r_b$, the global minimum of $E\tau^2$ occurs before the $\sigma^* \leq 1/r$ boundary only for $r \geq 1/5$. Otherwise, for $r \leq 1/5$, the minimal $E\tau^2$ occurs at the throughput-optimal boundary.

This general solution is still valid in the limiting extremes of the $E\tau^{\alpha}$ metric. As $\alpha \to \infty$, minimizing the metric becomes equivalent to or maximizing throughput. Though the local minimum $s_a^*(\alpha)$ increases without bound, the *global* minimum is still bounded by throughput-optimal static slacks per token, which depend on buffer parameters r, T. The global minimum of the metric will occur at the throughput piecewise-differentiability boundary, and is independent of E_f, E_a, E_b.

As $\alpha \to 0$, the minimizing metric becomes equivalent to minimizing energy. The local minimum of static slack per token σ_a^* and σ_b^* approaches 1. This means that the pipeline should have only the minimum amount of static slack necessary to support operation, regardless of how slow the pipeline operates, which is the only logical conclusion.

5. Interleaving Composition of Pipelines

Another common pipeline structure is to compose two pipelines in parallel with a split-interleaver in place of the copier, and a merge-interleaver in place of the synchronizer in Figure 3. The interleaved pipeline structure could be used in an application where duplicate execution units are available resources of a processing unit. An alternative to further pipelining a particular datapath or execution unit is to duplicate existing units, and having the instruction decoder dispatch control and data tokens alternately to one of several identical execution units. This is particularly useful when the execution units are expected to be highly occupied.

Tokens that enter the split-interleaver are alternately routed to two different pipelines, and a merge-interleaver sequentializes tokens alternately from two incoming pipelines. The analysis is similar to that of the parallel token pipelines in the previous section, only in the interleaved case, the tokens in each pipeline are distinct. While here we will only consider the case of two interleaved pipelines with strict alternation, one could perform a similar analysis for a larger collection of pipelines with different token distributions.

Given the same buffer characteristics E, T, and r (which may be different for each pipeline), we want to find a throughput-optimal relationship between s_a and s_b, for steady-state operation at $2k$ tokens in the pipeline.

In an interleaved pipeline, the static slack is equal to twice the static slack of the shorter pipeline. This is because once the shorter pipeline is full, the split-interleaver cannot issue any more tokens. The throughput of this system can be no more than double the throughput of the slower pipeline, since we can think of the tokens as being issued in pairs, one token per pipeline, every cycle.

The interleaved throughput function of two interleaved pipelines $\gamma_{\bowtie}$ has the same form as the parallel throughput function $\gamma_{\|}$ analyzed in Section 4, but with double the static slack and double the peak throughput. Therefore the analysis for $2k$ tokens in interleaved composition is the virtually the same as for k tokens in parallel composition. This is a close approximation since the difference in number of tokens in each pipeline in the interleaved case is bounded by two instead of always being zero.

Given an equation for throughput, we now need the energy of the composition in order to evaluate different designs for the general energy-time metric $E\tau^{\alpha}$. Since the behavior of the interleaving pipelines can be modeled as pairs of tokens in parallel, we can also use the same energy equations as for the parallel case.

Since the energy and time equations are the same, optimizing interleaving pipelines for $2k$ tokens is equivalent to optimizing parallel pipelines for k tokens. Therefore the optimal values for $E\tau^{\alpha}$ are scaled versions of the results from Section 4, but the optima occur at the same σ's.

6. Composing $E\tau^2$-Optimized Pipelines

Composing pipelines that have been independently optimized for $E\tau^2$ requires careful consideration to avoid degradation in the composed energy efficiency. While the total energy of a composed system is the sum of each individual pipeline, the composed cycle time depends on the type of composition. If the cycle times of the individual pipelines are different then the composed cycle time will degrade upon composition in the manner described by Lines [4]. Any resulting degradation in the cycle time will cause a degradation in $E\tau^2$.

An analysis of the general composition problem where different pipeline stages can operate at different voltages can be found in [8]. For both pipelined and parallel compositions, the energy-time optimal composition occurs when the cycle time of all pipeline stages are equal. This is intuitively obvious, because the cycle time is governed by the slowest stage in the computation. In the case of sequential composition, the time for the computation is the sum of the times of the components. In this case, the optimal configuration occurs when the E/τ ratios for each component are equal [8], i.e., the different components that are composed require the same power.

If the operating voltage can be individually adjusted for different pipelines then the cycle time can be equalized for most pipelines of interest. Usually, however, the operating voltage is uniform across large sections of a design and is difficult to locally adjust for individual pipelines both in terms of distributing multiple power supplies and using voltage conversion circuits at the interfaces.

In the following sections, we consider examples where we compute the energy and cycle times for more complex pipeline structures as functions of parameters that are then chosen to optimize our energy-time metric.

7. 1D-Array Pipelines

One-dimensional register arrays are common in hardware for computations and in memory circuits. One canonical way of implementing an array of registers is to use binary trees of buffering split cells to address one element of a particular array, and similar binary trees for routing data into and out of the array. (Other designs may use larger N-way splits and merges, or multidimensional arrays to reduce depth.)

On a write access, data and address bits are concurrently split along the binary route determined by the address bits, such that one fewer address bit is used after each stage. The writing throughput is only limited by the throughput of the splits, therefore no slack-matching is necessary to improve the throughput.

However on a read access, the control for a merge element arrives from the corresponding split element on the address side, and is independent of the data width. Without additional slack, the first split could not accept another token until the previous token completed at the last merge, thus hindering throughput. If we assume each split, merge, and register process has unit slack, then optimizing the array read-access for throughput would require adding buffers along the merge-control channels to equalize the slack along all paths between the first split and the last merge. Figure 5 shows a throughput-optimized array of 8 registers ($b = 3$). For an array of 2^b elements, the to-

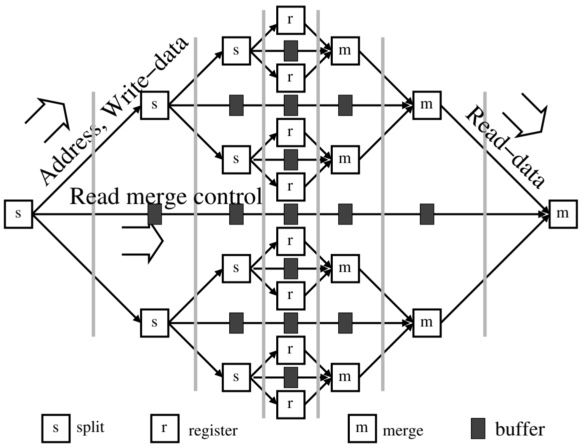

Figure 5. Pipeline schematic of a binary tree 1D Array for depth b=3, slack-matched for optimal throughput.

tal number of buffers in a throughput-optimized network, $N_{buf} = 3(2^b - 1) - 2b$. This exponential number of buffers needed makes slack-matching a 1D-array for throughput expensive in area.

To optimize $E\tau^\alpha$ for the read-access of a 1D array, we might expect that fewer buffers would be used in slack-matching, to trade off throughput for energy (or area) savings. We assume that the array is operating at a fixed number of tokens k, so we can analyze the steady-state throughput of the closed system. With full buffers, the total static slack in the throughput-optimized array is $2b + 1$, and the dynamic slack is $(2b + 1)r$.

Removing buffers from the pipelined array reduces the static slack of the system to the minimum path slack, because the pipeline cannot hold more tokens than its shortest pipeline path. Once one buffer is removed, this creates one path that is shorter than all of the other buffer paths, so subsequent buffers may be removed without further decreasing throughput, until all buffer paths again have equal slack depth. In Figure 5, removing an entire layer of buffers is the same as removing any one buffer from each horizontal buffer path, if any remain in that path. It is convenient to visualize removing a layer by deleting all buffers in the pipeline column with the most buffers. It only makes sense to start removing buffers from the leaf (innermost) level out, for the greatest amount of energy savings per unit slack removed.

Starting with a throughput-optimized array for $(2b + 1)r > k$, the system is operating at fewer tokens than the dynamic slack, thus the throughput of the system is token-limited. Removing buffers will not decrease the system throughput because it is limited by the forward latency of the longest path in the pipeline, which is that of the split-register-merge network. Without sacrificing throughput, energy is reduced by removing levels of buffers from the leaf-level out, until the slack of the shortest path is equal to the throughput-optimal slack.

From the case where $(2b+1)r = k$, we want to minimize $E\tau^\alpha$ by removing layers of buffers. The throughput will be a function of the system's static slack per token $\sigma = \frac{2b+1-v}{k}$, where v is the number of buffer layers removed from the throughput-optimal system. We define $n[v]$ as the number of buffers used per token. The new throughput is hole-limited because the dynamic slack of the pipeline will be less than the number of tokens.

Using an analysis similar to that of the token rings, we define the total energy per token $E(v)$, and throughput $\tau(v)$ as a function of the number of buffer levels removed. The total number of buffers in the array and the buffers used per token $n[v]$ are summarized for the first few values of v in Table 1.

For analysis purposes, we may choose to use a close approximation function in place of $n[v]$, $n(v) = (b - v/2)^2$, which exact for v even, only off by $1/4$ for v odd. The "computation energy overhead" E_f is the amount of energy in the split-register-merge network of the 1d array in terms of energies per bit $E_{split}, E_{register}$, and E_{merge}. E_f depends on the depth of the array b, and data width w. The number of splits used per token is quadratic in b because the ith stage of the address split network must split the $b - i$ other bits to the $i + 1$ level. Since splits, registers, and merges can be constructed from slightly modified buffers [4], we approximate their energies per bit $E_{s,r,m}$ to that of a buffer E_b, so $E_f/E_b \approx b^2/2 + b(w - 1/2) + 1$. We ignore the overhead energy required to copy and amplify signals.

$$\begin{aligned}
E(v) &= E_f + E_b n[v] \\
&\approx E_f + E_b(b - v/2)^2 \\
E_f &= \frac{1}{2}b(b-1)E_{split} + E_{reg} + bwE_{merge} \\
&\approx \left(b^2/2 + b(w - 1/2) + 1\right) E_{s,r,m} \\
\tau(v) &= \frac{(s-v)(1-r)}{T(s-v-k)} \\
s &= 2b + 1
\end{aligned}$$

Using $n(v)$, the minimum of $E\tau^\alpha$ occurs at:

$$(2b - v)(s - v - k)(s - v) = \alpha k \left(\frac{E_f}{E_b} + \left(b - \frac{v}{2}\right)^2 \right)$$

We verify that this solution makes sense in the limit as $\alpha \to 0$, and as $\alpha \to \infty$. As $\alpha \to 0$, the metric is dependent on only energy.

$$\begin{aligned}
0 &= -(2b - v)(s - v - k)(s - v) \\
v^* &= \min(2b, s - k, s) \\
&= s - k = 2b + 1 - k
\end{aligned}$$

v	total buffers N_{buf}	$n[v]$
0	$3(2^b-1)-2b$	b^2
1	$2(2^b-b-1)$	b^2-b
2	$3(2^{b-1}-1)-2(b-1)$	$(b-1)^2$
3	$2(2^{b-1}-(b-1)-1)$	$(b-1)^2-(b-1)$
4	$3(2^{b-2}-1)-2(b-2)$	$(b-2)^2$
$\vdots$	$\vdots$	$\vdots$

Table 1. Total number of buffers in array and number of buffers used per token for each level of buffers removed. $v = 0$ corresponds to the throughput-optimized array pipeline.

For any number of tokens $k \geq 1$, $s - k = 2b + 1 - k \leq 2b$, which means that all buffer levels should be removed to minimize energy, which reverts back to the un-buffered design. As $\alpha \rightarrow \infty$, the metric depends only on the cycle time.

$$\begin{aligned} 0 &= \alpha k \left(E_f/E_b + (b - v/2)^2\right) \\ 0 &= \left(b^2/2 + b(w - 1/2) + 1\right) + (b - v/2)^2 \end{aligned}$$

This quadratic is always positive and thus, has no real roots. On the domain $v \in [0, s - k]$, the cycle time τ is monotonically increasing because the pipeline is operating in the backward latency limited region, therefore τ is minimal on the boundary, $v = 0$, where no buffers are removed.

For all other intermediate α, we are interested in only the positive roots of v of the cubic equation that lie in $[0, s-k]$. Table 2 provides a list of roots of the above polynomial for $\alpha = 2$, and several values of b, k, w. All negative roots are reported as "< 0", which implies that no buffers should be removed, i.e. the τ- and $E\tau^2$-optimal solutions are the same. In practice, one would check the $E\tau^2$ above and below the non-integer solutions and choose whichever yielded the minimum.

From this table we see some interesting results. As the data width increases, the merge-energy term of the energy overhead increases proportionally. With larger overhead energy, the optimal number of buffer levels to remove decreases. For a 1D array of 32 32-bit registers running at 2-token occupancy ($w = 32, b = 5, k = 2$), $v^* = .80$, which means only the single innermost level of buffers is removed to minimize $E\tau^2$.

8. Energy-Efficient Exception Design

In previous sections we discussed finding the $E\tau^2$-optimal number of slack-matching buffers for a given pipeline topology. An important and more general optimization problem is to determine the $E\tau^2$-optimal number of slack-matching buffers for a design *without* restricting its pipeline topology. To illustrate this concept we choose a practical example of designing an energy-efficient exception mechanism for pipelined processors. We derive energy equations for various exception handling pipeline topologies and discuss how they compare with the $E\tau^2$ metric.

v^*: $E\tau^2$-optimal number of buffer levels to *remove* from τ-optimal slack-matched 1D array of registers					
depth b	number of tokens k in steady state: 1	2	3	4	5
data width $w = 8$					
2	0.85	< 0	< 0	< 0	< 0
3	2.33	0.75	< 0	< 0	< 0
4	3.89	2.20	0.76	< 0	< 0
5	5.49	3.70	2.20	0.82	< 0
6	7.12	5.24	3.69	2.28	0.92
7	8.78	6.82	5.21	3.75	2.37
8	10.45	8.41	6.76	5.26	3.86
data width $w = 32$					
2	< 0	< 0	< 0	< 0	< 0
3	0.31	< 0	< 0	< 0	< 0
4	1.70	< 0	< 0	< 0	< 0
5	3.17	0.80	< 0	< 0	< 0
6	4.70	2.21	0.27	< 0	< 0
7	6.27	3.67	1.65	< 0	< 0
8	7.87	5.18	3.09	1.28	< 0

Table 2. Values of v^* for several array sizes b, data widths w, and number of tokens k, at which $E\tau^2$ is minimal. The system static slack is $s - v^* = 2b + 1 - v^*$.

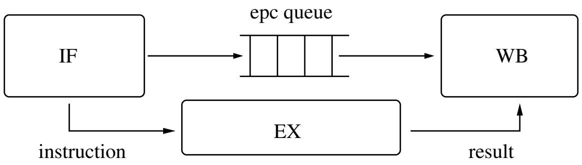

Figure 6. Processor with EPC queue.

In a typical microprocessor design, implementation of precise exceptions is handled by the writeback (WB) unit. An exception may be generated by the instruction fetch (IF), an execution unit (EX) or an external interrupt. When an exception occurs, the writeback unit notifies the fetch loop to stop fetching new instructions and branch to the exception handler. The writeback unit needs access to the program counter value which raised the exception (hereafter referred to as the EPC). This EPC value is stored until the exception handler has finished and then execution resumes at the EPC value. As shown in Figure 6, the fetch sends the EPC value to the writeback for each instruction that it issues. In the MiniMIPS, this is implemented as a linear FIFO queue that contains the EPCs of all instructions that are currently executing within the processor [6]. Because an EPC value will be sent on every cycle, this FIFO must be slack-matched with the execution pipeline to run at full throughput.

The average number of tokens, k, in the EPC queue is equivalent to the number of outstanding (non-retired) instructions executing in the processor. For pipelined processors, the number of full-buffers ($\frac{k}{r}$) required to slack-match the EPC queue for k tokens can be a significant energy waste (with r as defined in Section 2). For EPC values w bits wide, an EPC queue constructed from full-buffers will have an energy given by the expression

$$E_1 = (w)\frac{k}{r}E_b$$

where E_b is the buffer energy per bit of a FIFO stage.

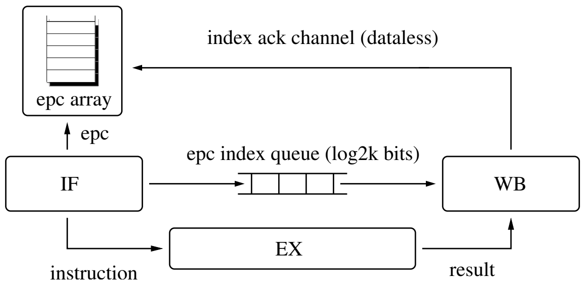

Figure 7. Processor with EPC index queue and EPC array.

EPC index queue. We observe that, though the writeback must receive some information from the fetch on every cycle, the writeback only needs the actual EPC value when exceptions occur (a rare event!). For a more energy efficient design (Figure 7) we eliminate sending the entire EPC on every cycle. Instead we store the EPC value in an array and send an index[9] into this array through a slack-matched FIFO to the writeback [13]. The size of this array determines the maximum number of outstanding instructions. To support an average of k outstanding instructions, we will assume the array is of size $2k$ and the index is $\log_2(2k)$ bits wide.

Before an entry in the EPC array can be overwritten, the fetch unit must know that the EPC value in the entry has been safely retired and that no exception has been raised. For this reason an acknowledge channel to the EPC array is added so the writeback unit can signal that it has retired an instruction. Since EPCs are sent from the fetch and retired by the writeback in program order, the acknowledge channel does not need to carry any data beyond the acknowledgment itself.

It is important that sending EPC array indices from the fetch unit, and the receipt of acknowledgments from the writeback are decoupled processes. This means that the fetch unit can write EPCs into the array, while an independent process receives acknowledgments from the writeback and then marks the EPC entries that have been retired. To accommodate this, the EPC array must include a extra valid bit in every entry that indicates whether the stored EPC has been retired by the writeback. The array must also be dual ported to allow it to be simultaneously accessed by the two processes. If the index send and acknowledge receive were linked, the index queue and index acknowledge channel would form a ring. This means the total number of tokens in the ring would be fixed. Since the number of instructions that are "in flight" in the processor can vary, we want the number of EPCs indices in the FIFO to be flexible as well. Additionally, if the EPC index FIFO and the acknowledge FIFO form a ring then the acknowledge FIFO would need to be slack-matched for the same number of tokens as the index queue (greatly increasing the energy overhead). In the decoupled scheme there is only minimal energy overhead since the acknowledge channel needs only to be slack-matched for a single token.

The total energy of the EPC index queue design, E_2, is the sum of the energy in the EPC index queue (E_{Index}), the acknowledge queue (E_{Ack}), the EPC array (E_{Array}) and the energy required to maintain pointers into the array ($E_{Pointers}$).

$$\begin{aligned}
E_2 &= E_{Index} + E_{Ack} + E_{Array} + E_{Pointers} \\
E_{Array} &= (w + 2\log_2(2k))(\log_2(2k))E_s \\
&\quad +(w+2)E_r \\
E_{Index} &= (\log_2(2k))\frac{k}{r}E_b \\
E_{Ack} &= \frac{1}{r}E_b \\
E_{Pointers} &= 2\log_2(2k)E_{cntbit}
\end{aligned}$$

We do not consider the energy required to read an EPC array entry because this occurs rarely, only after an exception happens. E_{Index} and E_{Ack} are computed in a similar manner as our previous computation of E_1. $E_{Pointers}$ is the energy of the number of bits required to encode the two array pointers, where E_{cntbit} is the counter energy per bit. On each cycle (in steady-state) there will be two accesses to the array.[10] Both accesses must go through $\log_2(2k)$ stages of splits (E_s is the split energy per bit) and write into the array (E_r is the array write energy per bit). When the fetch unit writes the EPC into the array, it sends both EPC (width w) and the index (width $\log_2(2k)$) along the split network mentioned above. The acknowledge channel, however, only sends the index (width $\log_2(2k)$) when it accesses the array. The fetch unit writes the EPC (w bits wide) and the valid bit, whereas the acknowledge channel only writes the valid bit

[9] A more aggressive design can eliminate sending this index entirely by keeping a duplicate array index counter in the WB, and the EPC index queue would then turn into a dataless channel.

[10] The array is implemented such that the EPC data values do not move and so no passive energy is consumed.

into the array.

Since the designs above are both slack-matched identically along their critical paths, they will have identical throughputs. The design with the optimal $E\tau^{\alpha}$ will then be the one with the lower energy. The condition for $E_1 > E_2$ is given by the inequality:

$$w > \frac{2\log_2(4k)E_s + E_{Index} + E_{Ack} + E_{Pointers} + 2E_r}{\frac{k}{r}E_b - \log_2(2k)E_s - E_r}$$

If we make the approximation that the energies per bit are equal, then Figure 8 shows that the above inequality is virtually constant for reasonable values of k. Thus for large EPC bit-widths ($w > 8$), the EPC index array design will use less energy and have a lower $E\tau^2$ value than the original EPC queue design.

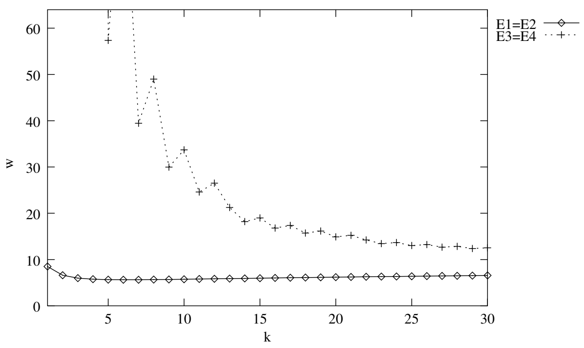

Figure 8. Bit-width of EPC (w) versus tokens (k) for $E1 = E2$ and $E3 = E4$ (assuming the energies/bit are equal and $r = 0.25$).

Effect of other FIFO styles. Here we examine the energy efficiency of replacing the linear FIFOs of the previous designs with binary tree FIFOs. We adopt the binary tree FIFO description and analysis introduced by Lines [4]. A "binary tree FIFO" is constructed by recursively composing split-interleavers at the input to the FIFO, placing FIFOs at each leaf of the tree, and then using a complementary merge-interleaver tree for the output of the FIFO. The latency of this FIFO design is proportional to the log of the total static slack. Since binary tree FIFOs require fewer slack-matching buffers than linear FIFOs to run at full throughput, designs using tree FIFOs may have lower $E\tau^2$ values.

We assume our tree FIFOs, as shown in Figure 9, are composed of two levels of binary splits, which feed four parallel internal FIFOs, followed by two levels of binary merges. It is also assumed that if a linear FIFO supports an average of k tokens, the equivalent tree FIFO should support a maximum of $2k$ tokens at full throughput. It can be shown that a binary tree FIFO supporting $2k$ tokens at full

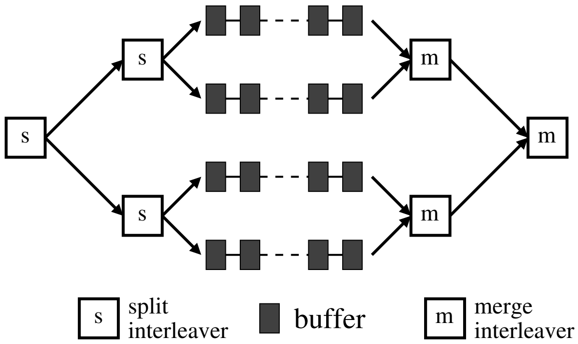

Figure 9. Binary tree FIFO topology.

throughput will require n total buffers for its internal FIFO stages, where n is given by the expression: [13].

$$n = 4\lceil\frac{4(k-3r-4)}{r+7}\rceil.$$

The energy, E_3, for a design that uses a tree FIFO instead of a linear EPC queue FIFO is

$$E_3 = w(2E_s + 2E_m + \lceil\frac{4(k-3r-4)}{r+7}\rceil E_b),$$

where E_s is the split energy per bit, E_m is the merge energy per bit, and E_b is the buffer energy per bit.

If we replace the linear index queue with a binary tree FIFO in the array EPC design, the energy of the modified design, E_4, will be equivalent to E_2 with the following modification to E_{Index}:

$$E_{Index} = (\log_2(2k))(2E_s + 2E_m + \lceil\frac{4(k-3r-4)}{r+7}\rceil E_b)$$

Since both of the designs (E_3 and E_4) which use tree FIFOs will have identical throughputs, we need only compare their energies to determine their relative energy efficiency. The condition for $E_3 > E_4$ is given by the inequality:

$$w > \frac{2\log_2(4k)E_s + E_{Index} + E_{Ack} + E_{Pointers} + 2E_r}{(2E_s + 2E_m + \lceil\frac{4(k-3r-4)}{r+7}\rceil E_b) - \log_2(2k)E_s - E_r}$$

Again if we make the assumption that the energies per bit are equal, then the above equation depends strongly on the values of w and k. As shown in Figure 8, if the number of tokens is large enough ($k > 10$) the inequality will hold for normal EPC bit-widths ($w = 32$). So when binary tree FIFOs are used, the EPC queue design (E_3) will be more energy efficient than the EPC index array design (E_4), for small numbers of tokens ($k \leq 10$).

Because the cycle time of the processor in all of these designs is fixed and determined by the IF to EX to WB pipeline, the $E\tau^2$ values for the designs are proportional

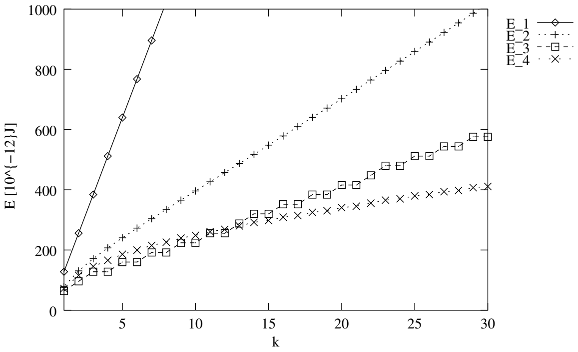

Figure 10. Energy functions for various EPC queue designs which support an average of k tokens (assuming $\frac{Energy}{bit} = 1^{-12}J$, $w = 32$, and $r = 0.25$).

to their energies. Figure 10 shows an energy summary of the four designs presented in this section. We make the assumption that the cell energies per bit are approximately equal. From this figure we can see that the designs using the EPC array index technique (E_2, E_4) require less energy than the EPC FIFO queue designs (E_1, E_3). As expected the designs using binary tree FIFOs (E_3, E_4) use less energy than the linear FIFO designs (E_1, E_2), since they have fewer slack-matching buffers. In practice, a more detailed circuit analysis may be required to determine the best exception handling design, especially if the number of tokens (k) is small.

9. Summary

Starting with a model for the steady-state throughput behavior of an asnychronous buffer, we constructed $E\tau^{\alpha}$-optimal analytic solutions for a number of asynchronous pipelines, and defined conditions when slack-matching buffers may be removed to improve energy efficiency. We also showed an example of micro-architectural design choices that can be used to reduce the number of slack-matching buffers in pipelined systems.

With a small set of simplifying assumptions, we have outlined a practical method of pipelining asynchronous circuits for a general energy-time metric. To design arbitrary energy efficient pipelines, we began with a system that has been slack-matched for throughput and remove (if possible) slack-matching buffers until $E\tau^{\alpha}$ is minimized. This energy-time optimization approach, via varying the number of slack-matching buffers, provides the asynchronous designer with a tractable method for designing energy efficient pipelines.

Acknowledgments

The research described in this paper was supported in part by the Multidisciplinary University Research Initiative (MURI) under the Office of Naval Research Contract N00014-00-1-0564, and in part by a National Science Foundation CAREER award under contract CCR 9984299. David Fang was supported in part by a National Defense Science and Engineering Fellowship. John Teifel was supported in part by a National Science Foundation Fellowship.

References

[1] S. M. Burns and A. J. Martin. Performance analysis and optimization of asynchronous circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, pp. 71–86, 1991.

[2] A. P. Chadrakasan, S. Sheng, R. W. Brodersen. Low Power CMOS Digital Design. *Journal of Solid-State Circuits*, V27, N4, pp 473–484, April 1992.

[3] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-Power Digital Design. *Symposium on Low Power Electronics*, 1994.

[4] A. M. Lines. *Pipelined Asynchronous Circuits*. M.S. Thesis, California Institute of Technology, 1996.

[5] R. Manohar and A. J. Martin. Slack Elasticity in Concurrent Computing. *Proceedings of the Fourth International Conference on the Mathematics of Program Construction*, Lecture Notes in Computer Science 1422, pp. 272-285, Springer-Verlag 1998.

[6] R. Manohar, M. Nystrom, and A. J. Martin. Precise Exceptions in Asynchronous Processors. *Proceedings of the 19th Conference on Advanced Research in VLSI*, March 2001.

[7] R. Manohar. Width-Adaptive Data Word Architectures. *Proceedings of the 19th Conference on Advanced Research in VLSI*, Salt Lake City, Utah, March 2001.

[8] R. Manohar and M. Nystrom. *Implications of Voltage Scaling in Asynchronous Architectures*. Cornell Computer Systems Lab Technical Report CSL-TR-2001-1013, April 2001.

[9] A. J. Martin. Compiling Communicating Processes into Delay-insensitive VLSI circuits. *Distributed Computing*, **1**(4), 1986.

[10] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. V. Cummings, and T.K. Lee. The Design of an Asynchronous MIPS R3000. *Proceedings of the 17th Conference on Advanced Research in VLSI*, September 1997.

[11] A. J. Martin. Towards an Energy Complexity of Computation. *Information Processing Letters*, V77, pp 181-187, 2001.

[12] K. Nowka, P. Hofstee, G. Carpenter. Accurate power efficiency metrics and their application to voltage scalable CMOS VLSI design. Preprint.

[13] J. Teifel, D. Fang, D. Biermann, C. Kelly, and R. Manohar. *The Derivation of Energy-Efficient Pipelines*. Cornell Computer Systems Laboratory Technical Report CSL-TR-2001-1015, October 2001.

[14] J. A. Tierno. *An Energy-Complexity model for VLSI computations*. Ph.D. thesis, California Institute of Technology, 1995.

[15] T. E. Williams. *Self-Timed Rings and their Application to Division*. Ph.D. thesis, Stanford University, May 1991.