Impact of Parallelism and Memory Architecture on FPGA Communication Energy

EDIN KADRIC, DAVID LAKATA, and ANDRÉ DEHON, University of Pennsylvania

The energy in FPGA computations is dominated by data communication energy, either in the form of memory references or data movement on interconnect. In this article, we explore how to use data placement and parallelism to reduce communication energy. We show that parallelism can reduce energy and that the optimal level of parallelism increases with the problem size. We further explore how FPGA memory architecture (memory block size(s), memory banking, and spacing between memory banks) can impact communication energy, and determine how to organize the memory architecture to guarantee that the energy overhead compared to the optimally matched architecture for the design is never more than 60%. We specifically show that an architecture with 32 bit wide, 16Kb internally banked memories placed every 8 columns of 10 4-LUT logic blocks is within 61% of the optimally matched architecture across the VTR 7 benchmark set and a set of parallelism-tunable benchmarks. Without internal banking, the worst-case overhead is 98%, achieved with an architecture with 32 bit wide, 8Kb memories placed every 9 columns, roughly comparable to the memory organization on the Cyclone V (where memories are placed about every 10 columns). Monolithic 32 bit wide, 16Kb memories placed every 10 columns (comparable to 18Kb and 20Kb memories used in Virtex 4 and Stratix V FPGAs) have a 180% worst-case energy overhead. Furthermore, we show practical cases where designs mapped for optimal parallelism use 4.7× less energy than designs using a single processing element.

CCS Concepts: • **Computer systems organization → Reconfigurable computing;**

Additional Key Words and Phrases: FPGA, energy, power, memory, architecture, banking, communication

ACM Reference Format:
Edin Kadric, David Lakata, and André DeHon. 2016. Impact of parallelism and memory architecture on FPGA communication energy. ACM Trans. Reconfigurable Technol. Syst. 9, 4, Article 30 (August 2016), 23 pages.
DOI: http://dx.doi.org/10.1145/2857057

1. INTRODUCTION

Energy consumption is a key design limiter in many of today's systems. Mobile devices must make the most of limited energy storage in batteries. Limits on voltage scaling mean that even wired systems are often limited by power density. In these situations, reducing energy per operation can increase the performance delivered within a limited power envelope.

Within FPGAs, for many applications, data movement energy—the energy for moving data from one physical location on the FPGA to another—can dominate computational energy. Data movement includes both energy for accessing memory and energy for

This research was funded in part by DARPA/CMO contract HR0011-13-C-0005. David Lakata was supported by the VIPER program at the University of Pennsylvania. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. government.
Authors' address: E. Kadric, D. Lakata, and A. DeHon, Department of Electrical and Systems Engineering, University of Pennsylvania, 200 S. 33rd Street, Philadelphia, PA 19104; emails: ekadric@seas.upenn.edu, dlakata@seas.upenn.edu, andre@acm.org.

ACM 1936-7406/2016/08-ART30 $15.00
DOI: http://dx.doi.org/10.1145/2857057

moving bits over interconnect segments between processing elements (PEs). In this article, we explore two interrelated issues that significantly impact data movement energy in FPGA applications: parallelism and memory organization.

How does parallelism impact the energy required for computation? In sequential designs, the energy reading and writing data in large memories is typically the dominant component of energy. Since the energy to read from a memory grows with memory size, parallel designs that use many smaller memories local to each PE rather than one large memory can reduce memory energy. As long as the parallel design does not incur too much energy communicating among its PEs, the decomposition can provide a net energy reduction. We formulate the trade-offs involved, show analytically how they lead to an optimum level of parallelism that minimizes communication energy (Section 2), and illustrate how parallelism tuning can be used to minimize energy both for an ideal limit-study architecture and for designs mapped to a specific, energy-optimized FPGA architecture (Section 6).

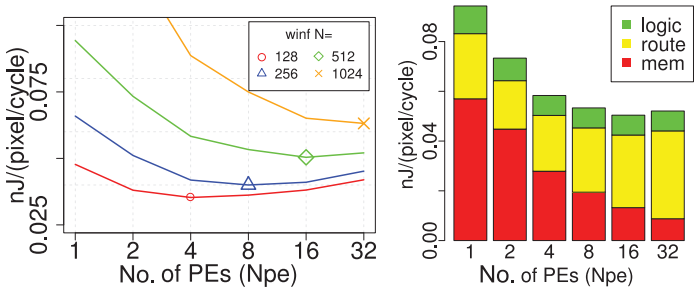
This result underscores the importance of the architectural question, *How do we organize memories in FPGAs to minimize the energy required for a computation?* We have several choices. What are the sizes of memory blocks? Where (how frequently) are memory blocks placed in the FPGA? How are the memories activated? How are they decomposed into sub-block banks? What read and write widths should the memory blocks use? Then there are choices available to the RTL mapping flow. When mapping a logical memory to multiple blocks, should they each get a subset of the data width and be activated simultaneously, or should they each get a subset of the address range and be activated exclusively? We develop simple analytic relations to reason about these choices (Section 3). After setting up the background (Section 4) and methodology (Section 5), we perform an empirical, benchmark-based exploration to identify the most energy-efficient organization for memories in FPGAs and quantify the trade-offs between area- and energy-optimized mappings (Section 7). We show how to choose a one-size-fits-all FPGA memory architecture that contains the worst-case architectural mismatch overhead from memory block size and placement below 60%.

Our contributions are as follows:

—Analytic characterization of the energy-optimal level of parallelism for applications (Section 2)
—Analytic characterization of the energy overhead that results from mismatches between the logical memory organization needed by a task and the physical memory organization provided by an FPGA (Section 3)
—Introduction of the continuous hierarchy memory (Section 3) and characterization of its benefits (Section 7)
—Characterization of how parallelism impacts energy consumption, including demonstration of how parallelism tuning can reduce energy (Section 6)
—First empirical exploration of memory architecture space for energy minimization (Section 7)
—VTR-compatible memory mapping for energy minimization (Section 5.2)

This article is a unification and expansion of Kadric et al. [2014, 2015]. The analysis for parallel scaling (Section 2.3) is new. The analytic bounds on memory mismatch overhead have been tightened using the golden ratio layout (Section 3). This article includes the effects of width mismatch, which were omitted in Kadric et al. [2015], and the parallelism benchmark set has been expanded beyond [Kadric et al. 2014]. We omit parallel mappings to commercial architectures (included in Kadric et al. [2014]) and the treatment of multiple memory sizes (included in Kadric et al. [2015]).

We start by describing the key phenomena involved and developing a simplified analytical framework to illustrate and build intuition on the major effects and trends



(a) Size and PE sweep (b) Detailed breakdown for $N = 512$

Fig. 1. Energy versus PE count (N_{pe}) for the window filter benchmark (WinF).

(Sections 2 and 3). Then we introduce the specific FPGA and energy background (Section 4) and experimental methodology (Section 5) to set up the empirical experiments in Sections 6 and 7.

2. PARALLELISM AND DATA MOVEMENT ENERGY

To perform any computation, we must communicate data between the point in time and space where each intermediate data item is computed and where it is consumed. This communication can occur either through interconnect wires, if the operators are spatially located at different places, or through memories, if the operators are sequentialized on a common physical operator. At the extremes, the design could be fully spatial (e.g., a spatial fast Fourier transform (FFT)) or fully sequential (e.g., a single processor that computes the same FFT, storing intermediate data in a single, large memory). Either way, we spend energy for communication—either toggling long wires or reading and writing from large memories. We show that, in the general case, neither extreme is the most energy efficient. In other words, we can optimize the energy required for communication by tuning the parallelism in the task, and there is typically a level of parallelism that minimizes total communication energy (see Figure 1). The phenomenon here is closely related to the ones explored in DeHon [2015], and we similarly show that it is often better to distribute the data and computation than to centralize it in a single memory. GraphStep [Delorimier et al. 2011] provides one concrete model for how applications might be defined to allow this form of parallelism tuning. In the remainder of this section, we explain and model the opposing communication energy effects and show how they give rise to this optimum energy point.

2.1. Memory Energy

The energy required to access a memory depends on its capacity and the number of output bits. This is driven almost directly by the length of the wires that the bits must traverse to move between the input/output (I/O) port for the memory and the memory cell location within the array. Roughly, the energy (and delay) minimizing organization for an M-bit memory is a $\sqrt{M} \times \sqrt{M}$ array, meaning that all main wires in the memory are of length $\sqrt{M}$ so that their capacitance scales as $\sqrt{M}$ and, consequently, their switching energy scales as $\sqrt{M}$. Everything else being equal, a memory of four times the capacity will cost twice the energy.

Since memory energy is driven by wire lengths, we cannot "cheat" the $\sqrt{M}$ energy growth by decomposing the memory into smaller memories and wiring to them. We would still end up with address wires of length $\sqrt{M}$ and I/O wires of length $\sqrt{M}$. If we were dominated by memory cell capacitance, breaking the large memory into smaller

memory banks and activating only one bank at a time could reduce the memory cell energy, but we are still left with wiring energy that also has a $\sqrt{M}$ dependence. We will see later (Section 3) that banking can help us reduce mismatch energy.

2.2. Between Computations

For many computations, we can reduce the size of the memory by performing the computation in parallel. Rather than having a single PE or computational datapath with a large memory to hold all of the data, we can have multiple PEs, each with its own, smaller memory. Ideally, for a problem with data size N, the size of the memories scales as the serialization factor $S = N/N_{pe}$, where N_{pe} is the number of PEs. Smaller memories reduce the energy for each memory operation as noted earlier. In the extreme, we may be able to eliminate the memories all together and simply connect datapath elements. For example, we could build a completely spatial FFT network with no internal memories. However, increasing the number of PEs also increases the physical size of the computation, potentially increasing the length of the wires in the system and hence increasing energy. In tasks like the FFT, data must now be moved from the PE where it is produced to the PE where it is consumed, and this data movement costs energy.

2.3. Analysis

To understand how these effects interact, we develop simple energy models for the computation. Since communication energy depends on interconnect lengths, which in turn are driven by the design size, we first need a rough understanding of how the design area grows with the number of PEs, N_{pe}, and the problem size, N. We pay area for memory to hold the state of the computation, A_{mem}, area for the logic for each PE, A_{logic}, and area for the interconnect.

$$A_{mem}(M) \propto M \tag{1}$$

Memory area scales linearly with the number of bits stored.

$$S = \frac{N}{N_{pe}} \tag{2}$$

$$A_{pe}(S) = A_{mem}(S) + A_{logic} \tag{3}$$

The memory used by the PE will scale as the serialization factor, S, and that will drive the PE area to scale by the serialization factor as well.

$$p' = \max(0.5, p) \tag{4}$$

We use a Rent's rule [Landman and Russo 1971] wiring model, in the spirit of Thompson [1979] and Bhatt and Leighton [1984], to estimate the growth rate contribution from wiring based on the Rent exponent, p. Rent's rule says that the I/O wiring, IO, out of a region with N computational elements, grows as $IO = cN^p$, where c is a constant dependent on the size of the primitive computational elements and p characterizes the locality in the design. Rent's rule wire area models have a breakpoint at $p = 0.5$, where area grows linearly in N below $p = 0.5$ and as $O(N^{2p})$ above $p = 0.5$. Using p' (Equation (4)), we can describe the area growth in both cases as $O(N^{2p'})$, allowing us to state upcoming results more compactly.

$$A = N_{pe} \cdot A_{pe}(S) + \max\left(C_1\left(\frac{N^{p'}}{S}\right)^2, C_2(N_{pe})^{2p_{net}}\right) \tag{5}$$

Equation (5) captures both PE and interconnect contributions to estimate the area growth of the implementation. The first term is the total area of all PEs, whereas the second captures interconnect. The interconnect area in this max term can be driven by either the wiring requirement for the physical substrate (p_{net}) or the communication requirement of the application (p) and the extent to which the design is serialized, S. C_1 and C_2 are constant factors for each of the cases, which may be different. $p_{net} > p$ means that the substrate (FPGA) is overdesigned for the application and will ultimately determine the interconnect area instead of the design p; $p < p_{net}$ means that the design-required interconnect grows faster than the FPGA provides, meaning that design LUTs must be spread out sparsely over physical FPGA LUTs to get adequate interconnect wiring [DeHon 1999]. $N^{p'}$ captures the volume of traffic that must cross the bisection width of the chip and the denominator S accounts for the fact that it can be serialized over S cycles, reducing the volume of physical wires required. $N^{p'}/S$ or $(N_{pe})^{p_{net}}$ defines one side length of the chip; the other will be proportional to this such that the area is proportional to the square of this length term. Equation (5)'s max term deals with the fact that wiring may drive total area, forcing the design to be spread out just to get the required bandwidth for highly parallel computations when $p > 0.5$. As we will see, this can have an effect limiting the amount of parallelism that should be exploited to minimize energy.

With a model for area, we can now estimate energy requirements.

$$E_{mem}(M) \propto \sqrt{M} \tag{6}$$

$$E_{pe}(S) = S \cdot (E_{mem}(S) + E_{logic}) \tag{7}$$

The energy at the PE, E_{pe}, is driven by the size of the memories and represents the total energy over the PE processing all S computations assigned to the PE.

$$E_{comm} \leq \sum_{i=0}^{\log_2(N)} \left(\frac{N}{2^i} \times c(2^i)^p \times 2^{\lceil i/2 \rceil} \sqrt{A/N} \right) \tag{8}$$

$$\propto N^{p'} \sqrt{A} \tag{9}$$

The communication energy, E_{comm}, deals with both the volume of traffic and the length of the wires that the traffic must traverse. Rent's rule captures the number of wires at each level of a hierarchical decomposition ($(2^i)^p$), and the areas computed above allow us to associate a wire length with each level of the hierarchy ($2^{\lceil i \rceil}\sqrt{A/N}$). The $\frac{N}{2^i}$ term captures the number of subunits at a given level of the hierarchical decomposition.

$$E = N_{pe} \cdot E_{pe} + E_{comm} \tag{10}$$

Equation (10) captures the total energy to perform the task over the entire problem of size N. Substituting Equations (5), (7), and (8) into Equation (10), we get a total energy that scales asymptotically as

$$E = O\left(\frac{N^{1.5}}{\sqrt{N_{pe}}} + N^{p'} \sqrt{N + \max\left((N_{pe})^2 N^{2p'-2}, (N_{pe})^{2p_{net}}\right)} \right). \tag{11}$$

The first term represents the memory operations, and it decreases with N_{pe} as we make the memories in each PE smaller. The second term is the interconnect term, and it increases with N_{pe}. The first component of the area (N in the square root) is the space to hold all of the memory, which is independent of N_{pe} and sets a potential lower

bound on the asymptotic energy of $O(N^{p'+0.5})$. This gives us a goal of selecting an N_{pe} that does not force any of the components to exceed $O(N^{p'+0.5})$. To achieve this,

$$N^{2-2p'} \leq N_{pe} \leq \min\left(N^{1.5-p'}, N^{\frac{1}{2p_{net}}}\right). \tag{12}$$

For p_{net} and $p \leq 0.5$, Equation (12) says that $N_{pe} = O(N)$—we should accommodate larger designs by increasing the number of PEs proportionally with the problem size. This arises because the wire lengths are not growing with N for $p < 0.5$ [Donath 1979]. The example shown in Figure 1(a) illustrates this with the energy-minimizing number of PEs growing linearly with the problem size. For $p > 0.5$, the wire lengths do grow and the number of PEs should grow more slowly as dictated by the terms on the right. Nonetheless, the memory term on the left dictates that the number of PEs must be growing for any $p < 1.0$. The terms on the right allow growth of $O(\sqrt{N})$ even when p_{net} or p becomes 1.0; the allowed growth increases as p and p_{net} reduce from 1.0. When p_{net} is large, we can get cases where the bounds in Equation (12) cross; this reflects cases where $O(N^{p'+0.5})$ is not achievable, and the actual minimum must be formulated differently.

When we select N_{pe} within Equation (12), the total area remains $O(N)$ and is, asymptotically at least, not wire dominated. For the cases where $N_{pe} = O(N)$, the serialization factor becomes a constant $S = \frac{N}{N_{pe}}$ that is independent of N. If we select S such that $A_{mem}(S) = A_{logic}$, the design is only twice as large as a fully serialized design, meaning that the chip crossing wires are at most $\sqrt{2}$ longer than they might be in a fully serialized design. Making S smaller than this increases the total area, and hence the length of chip-crossing communications, whereas making S larger increases the length of the nearest neighbor connections. Therefore, this distribution is, at least, within a small constant factor of the energy-optimal S ratio. Unfortunately, the logic for a PE, A_{logic}, is application specific, whereas an FPGA must typically pick a single, one-size-fits-all organization for memory and logic that must satisfy all applications. As we will see in the next section, we can design the FPGA memory organization so that the overhead energy due to this mismatch is bounded by a small constant.

3. ARCHITECTURE MISMATCH ENERGY

FPGA embedded memories generally improve area- and energy efficiency [Kuon and Rose 2007]. The previous section showed how this works for one class of applications. When the embedded memory perfectly matches the size and organization needed by the application, an FPGA embedded memory can be as energy efficient as the same memory in a custom ASIC. Nonetheless, the FPGA has a fixed-size memory that is often mismatched with the task, and this mismatch can be a source of energy overhead.

As noted (Equation (6)), memory energy scales as the square root of the capacity. When the FPGA memory block (M_{arch}) is larger than the application memory (M_{app}), there is an energy overhead that arises directly from reading from a memory bank that is too large ($E(M_{arch})/E(M_{app})$).

There is also a mismatch overhead when the memory block is smaller than the application memory. To understand this, we must also consider the routing segments needed to link up the smaller memory blocks into a larger memory block. To build a larger block, we take a number of memory blocks ($\lceil M_{app}/M_{arch} \rceil$) and wire them together, with some additional logic, to behave as the desired application memory block. In modern FPGAs, it is common to arrange the memory blocks into periodic columns within the FPGA logic fabric (Figure 2). Assuming square memory and logic blocks, the set of smaller memory blocks used to realize the large memory block might roughly be organized into a square of side length $\lceil \sqrt{M_{app}/M_{arch}} \rceil$, demanding that each

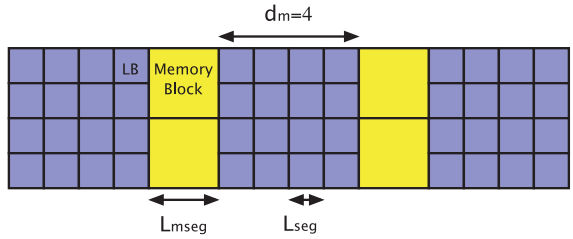


Fig. 2. Column-oriented embedded memories.

address bit and data line connected to the memory cross roughly $d_m \times \lceil \sqrt{M_{app}/M_{arch}} \rceil$ horizontal interconnect segments to address the memory, where d_m is the distance between memory columns in the FPGA architecture. Since there is an asymmetry that we cross logic blocks in the horizontal direction and not in the vertical direction, we can reduce the overhead by a constant factor by composing the large memory from an $h \times v$ rectangle with $v \geq h$, where h and v are the respective horizontal and vertical dimensions ($h \cdot v \cdot M_{arch} = M_{app}$). If E_{seg} is the energy to cross a length-1 segment over a logic island in the FPGA, and $E_{mseg}(M_{arch})$ is the energy to cross a length-1 segment over a memory block of capacity M_{arch}, the horizontal and vertical routing energy to reach across the memory is

$$E_h = (d_m E_{seg} + E_{mseg}(M_{arch})) \times h, \tag{13}$$

$$E_v = E_{mseg}(M_{arch}) \times v. \tag{14}$$

We define $\phi = \frac{d_m E_{seg}}{E_{mseg}(M_{arch})}$, allowing us to restate:

$$E_h = (\phi + 1)\, E_{mseg}(M_{arch}) \times h. \tag{15}$$

Since the routing energy of wires comprises most of the energy in a memory read, and since each bit must travel the height of the memory block (bit lines) and the width (output select), per bit, the energy of a memory read is roughly the energy of the wires crossing it. For a native memory block,

$$E_{bit}(M) \approx 2E_{mseg}(M). \tag{16}$$

Therefore,

$$E_{mseg}(M_{arch}) \left\lceil \sqrt{\frac{M_{app}}{M_{arch}}} \right\rceil \approx E_{mseg}(M_{app}) \approx 0.5 E_{bit}(M_{app}). \tag{17}$$

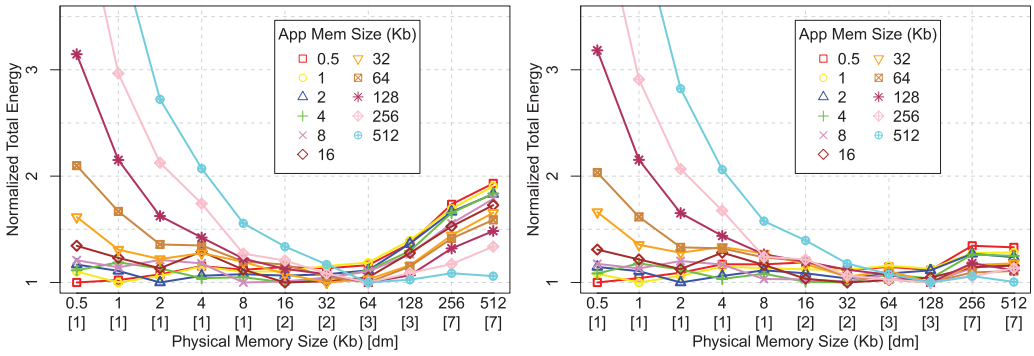
For this composed case, the per bit energy becomes

$$E_{bit}(M_{app} > M_{arch}) = E_h + E_v = (v + h\,(\phi + 1))\, E_{mseg}(M_{arch}). \tag{18}$$

We set the derivative of Equation (18) with respect to v equal to 0 and solve for the v that minimizes $E_{bit}(M_{app} > M_{arch})$, recalling that $h = (1/v) \cdot M_{app}/M_{arch}$, and get $v = \sqrt{(\phi + 1)\, \frac{M_{app}}{M_{arch}}}$, which results in

$$E_{bit}(M_{app} > M_{arch}) = 2\sqrt{(\phi + 1)\, \frac{M_{app}}{M_{arch}}}\, E_{mseg}(M_{arch}). \tag{19}$$

Equations (17) and (19) give the following memory mismatch ratio, driven by the ratio of the energy for routing between memory banks to the energy for routing over



(a) Normal memories (b) With internal banking (1/4, 1/16)

Fig. 3. Energy overhead due to architectural mismatch for matrix-multiply.

memory banks:

$$\frac{E_{bit}(M_{app} > M_{arch})}{E_{bit}(M_{app})} \approx \sqrt{\phi + 1}. \tag{20}$$

To illustrate the mismatch effects when memories are both too large and too small, Figure 3(a) shows the result of an experiment where we quantify how the energy compares between various matched and mismatched designs. Each of the curves represents a single-PE matrix-multiply design that uses a single memory size; the size of the memory varies with the size of the matrices being multiplied. Each curve shows the energy mismatch ratio (y-axis) between the energy required on a particular memory block size (x-axis) and the energy required at the energy-minimizing block size (typically the matched size); hence, all curves go to 1.0 at one memory block size and increase away from that point. In contrast to the previous paragraph, where we used deliberately simplified approximations to provide intuition, Figure 3(a) is based on energy from placed-and-routed designs using tools and models detailed in the following sections. Figure 3 also makes no a priori assumption about large memory mapping, allowing VTR [Luu et al. 2014] to place memories to minimize wiring. The figure shows how the energy mismatch ratio grows when the memory block size is larger or smaller than the matched memory block size. In practice, designs typically demand a mix of memory sizes, making it even harder to pick a single size that is good for all memory needs of an application. Nonetheless, this single-memory size experiment is useful in understanding how each of the mismatched memories will contribute to the total memory energy overhead in a heterogeneous memory application.

There is also a potential energy overhead due to a mismatch in memory placement. Assuming that we accept a column-oriented memory model, this can be stated as a mismatch between the appropriate spacing of memories for the application ($d_{m_{app}}$) and the spacing provided by the architecture ($d_{m_{arch}}$). If the memories are too frequent, nonmemory routes may become longer due to the need to route over unused memories. This gives rise to a worst-case mismatch ratio:

$$E_{route\_overhead\_unused\_memories} \approx \frac{d_{m_{arch}} \cdot E_{seg} + E_{mseg}(M_{arch})}{d_{m_{arch}} \cdot E_{seg}} = \frac{\phi + 1}{\phi} = 1 + \frac{1}{\phi}. \tag{21}$$

If the memories are not placed frequently enough, the logic may need to be spread out, by at most a factor of $\phi + 1$. This effectively forces routes to be longer by a factor of at

most $\sqrt{\phi + 1}$ as they run over unused logic clusters.

$$E_{route\_overhead\_unused\_logic} \leq \sqrt{\phi + 1} \tag{22}$$

If we make $\sqrt{\phi + 1} = 1 + \frac{1}{\phi}$ and solve for ϕ, we get $\phi = \frac{\sqrt{5}+1}{2} \approx 1.6$, which is the golden ratio [Heath and Euclid 1956]. The mismatch ratio due to route mismatch (Equation (21)) is never greater than $1 + \frac{1}{\phi} = \phi \approx 1.6$. Similarly, the mismatch ratio due to memories being too small (Equation (20)) or memories being placed too infrequently (Equation (22)) is never greater than $\sqrt{\phi + 1} = \phi \approx 1.6$. We can observe this phenomenon in Figure 3(a) by looking at the 32Kb memory size that never has an overhead greater than 1.2×. In Figure 3, we also identify the $d_{m_{arch}}$ that minimizes max overhead (shown between square brackets for each memory size in Figure 3). This approximately corresponds to the earlier intuitive explanation, where the energy for routing across memories is balanced with the energy for routing across logic. The 32Kb case has $E_{mseg}(M_{arch})/E_{seg} = 2.53$, suggesting a $d_{m_{arch}}$ around 4. For this 32Kb case, we found $d_{m_{arch}} = 2$ experimentally. Since segment energy is driven by wire length, $d_{m_{arch}} E_{seg} = \phi \cdot E_{mseg}(M_{arch})$ roughly means that $d_{m_{arch}} L_{seg} = \phi \cdot L_{mseg}(M_{arch})$; when we populate memories this way, $\frac{1}{1+\phi} \approx 40\%$ of the FPGA area is in memory blocks. This design point is robust in that it guarantees that the worst-case mismatch overhead is small for any design. This design point gives us an energy-balanced FPGA that makes no a priori assumptions about the mix of logic and memory in the design. In contrast, today's typical commercial FPGAs could be considered logic rich, making sure that the energy (and area) impact of added memories is small on designs that do not use memories heavily.

Although the $d_{m_{arch}} E_{seg} = \phi \cdot E_{mseg}(M_{arch})$ balance can limit the overhead when the memories are too small, we can still have large overhead when the memory blocks are too large ($E(M_{arch})/E(M_{app})$). One way to combat this problem is to use internal banking, or continuous hierarchy memories (CHMs): we can bank the memory blocks internally so that we do not pay for the full cost of a large memory block when we only need a small one. For example, if we cut the memory block into four, quarter-sized memory banks, and only use the memory bank closest to the routing fabric when the application only uses 1/4 (or less) of the memory capacity, we only pay the memory energy of the smaller memory bank (Figure 4). This banking scheme is biased such that the cost of accessing the bank closer to the I/O is lower than the cost of accessing the other ones. This is useful on FPGAs since it reduces the mismatch, whereas on an ASIC, where the sizes can be matched, there is often little reason to prefer one bank over another. In the extreme, we might recursively decompose the memory by powers of two so that we are never required to use a memory more than twice the size of the memory demanded by the application, keeping the mismatch energy ratio down to $\sqrt{2}$. There are some overheads for this banking that may suggest stopping short of this extreme. Figure 3(b) performs the same experiment as Figure 3(a), except with memory blocks that can be decomposed into 1/4 and 1/16 capacity sub-banks. With this optimization, the curves flatten out for larger memory sizes. The physical size with smallest max overhead is now shifted to 128Kb, still at 1.2×.

Another way to reduce the impact of memory block size mismatch is to include memory blocks of multiple sizes in the architecture. This way, the design can use the smallest memory block that will support the application memory. For example, if we had *both* 1Kb and 64Kb memories, we could map the 2Kb and smaller application memories to the 1Kb memory block and the 4Kb and larger application memories to the 64Kb block and reduce the worst-case overhead to 1.1× (Figure 3(a)). The impact of multiple memory sizes is explored experimentally in Kadric et al. [2015].

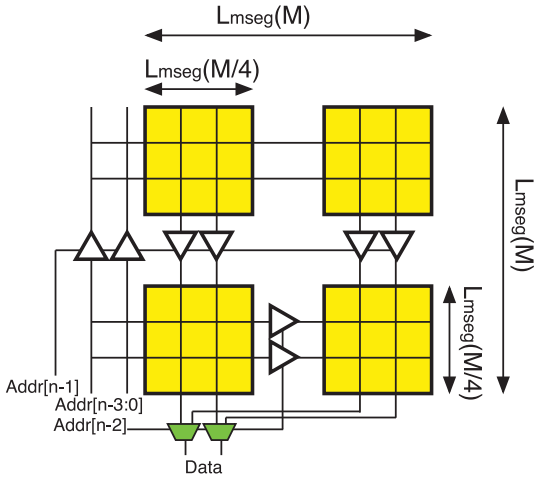


Fig. 4. Internal banking of memory block.

Another point of mismatch between architecture and application is the width of the data written or read from the memory block. Memory energy also scales with the data width. In particular, energizing twice as many bit lines costs roughly twice the energy. Although FPGA memory blocks can be configured to supply less data than the maximum width, this is typically implemented by multiplexing the wider data down to smaller data *after* reading the full width—the same number of bit lines are energized as the maximum width case, so these smaller data reads are just as expensive as the maximum width read and hence are more expensive than they could have been with a matched-width memory. For example, Altera's tools report the same energy for Stratix III M9K memory reads regardless of the width configured [Altera Corporation 2013]. Asymptotically, we expect memory read energy to scale as $W\sqrt{M}$. This means an additional mismatch factor that could be as large as $\frac{W_{arch}}{W_{app}}$. However, the small memory blocks that are appropriate for FPGA embedded memories are dominated by peripheral effects (e.g., address decode, sense amplifiers) such that the width mismatch effect when $W_{arch} > W_{app}$ is less severe in practice. To minimize energy, the memory layout should be square, making cases where $W_{arch} > \sqrt{M_{arch}}$ relatively more expensive.

Another potential point of mismatch is the simultaneous ports provided by the memories. We assume dual-ported memories (two read/write ports) throughout this article.

4. CONCRETE ARCHITECTURE AND ENERGY BACKGROUND

The previous sections dealt abstractly with energy effects. In this section, we describe the architecture and energy modeling we build on to make concrete comparisons.

4.1. FPGA Memory Architecture

We build on the standard island-style FPGA model [Betz et al. 1999]. The basic logic tile is a cluster of K-LUTs with a local crossbar providing connectivity within the cluster (cf. Xilinx CLB, Altera LAB). These clusters are arranged in a regular mesh and connected by segmented routing channels.

To incorporate memories into this mesh, we follow the model used by VTR [Luu et al. 2014], Xilinx, and Altera, where select columns are designated as memory columns rather than logic columns (see Figure 2). Organizing the memory tiles into a homogeneous column rather than placing them more freely in the mesh allows them the

freedom to have a different size than the logic tiles. For example, if the memory block requires more area than the logic cluster, we can make the memory column wider without creating irregularity within rows or columns. Altera uses this column memory model in their Cyclone and Stratix architectures, and the M9K blocks in the Stratix III [Lewis et al. 2009] are roughly 3× the area of the logic clusters (LABs) [Wong et al. 2011] while being logically organized in the mesh as a single tile. Large memories can span multiple rows, such as the M144K blocks in the Stratix III, which are eight rows tall while remaining one logical row wide, accommodated by making the column wider as detailed earlier.

Within this architectural framework, we can vary the proportion of memory tiles to logic tiles by selecting the fraction of columns that are assigned to memory tiles rather than logic tiles. We control this by setting the number of logic columns between memory columns, d_m. VTR identifies this as a `repeat` parameter (`repeat`=$d_m + 1$).

4.2. Energy Modeling and Optimization

Poon et al. [2005] developed energy modeling for FPGAs and identified how to size LUTs (4-LUTs), clusters (8 to 10), and segments (length 1) to minimize energy. However, Poon et al. did not identify an energy-minimizing memory organization. FPGA energy modeling has since been expanded to modern direct-drive architectures and integrated into VTR [Goeders and Wilton 2012].

Recent work on memory architecture has focused on area optimization rather than energy. Luu et al. examined the area efficiency of memory packing and concluded that it was valuable to support two different memory block sizes in FPGAs [Luu et al. 2011]. Lewis et al. showed how to size memories for area optimization in the Stratix V and concluded that a single 20Kb memory was superior to the combination of 9Kb and 144Kb memories in previous Stratix architectures [Lewis et al. 2013] but did not address energy consumption, leaving open the question of whether energy-optimized memory architectures would be different from area-optimized ones. Chin et al. explored the energy impact of embedded memory sizes when they are used to map logic, but not when they are used as read-write memories for application data [Chin et al. 2006].

4.3. Memory Energy Modeling

We use CACTI 6.5 [Muralimanohar et al. 2009] to model the physical parameters (area, energy, delay) of memories as a function of capacity, organization, and technology. In addition to modeling capacity and datapath width, CACTI explores internal implementation parameters to perform trade-offs among area, delay, throughput, and power. We use it to supply the memory block characteristics for VTR architecture files at 22nm. We set it to optimize for the energy-delay-squared product.

For internal banking (see Figure 4), CACTI gives us the area and energy (E_{mem}) of the memory banks, and we compute wire signaling energy (E_{wires}) to communicate data and addresses between the referenced memory bank and the memory block I/O. For example, consider the data in Figure 5 for a 1024×32b (32Kb) internally banked memory. A monolithic 32Kb memory block is 113μm×67μm, which is high enough to contain the 31μm×2 = 62μm required for the height of the two 256 × 32 memories of size 66μm×31μm (plus room for extra logic), as shown in Figure 5. The total width in Figure 5 is 38 × 2 + 66 = 142μm, or 142/113 = 1.26× that of the monolithic 32Kb memory block. We therefore adjust E_{mseg}(32K banked) = 1.26 × E_{mseg}(32K). CACTI directly provides E_{mem}. The I/O is placed close to the first bank (lower left in Figure 5) so that reading the first 64 × 32b comes at no additional wiring cost. Accessing the next small bank has the same memory cost but also a wiring cost of $E_{wires} = (15\mu\text{m})(1 + 30 + 64)C_{wire}(V_{dd})^2$. C_{wire} = 180pF/m, V_{dd} = 0.95V. (1 + 30 + 64)

Addresses	0–63	64–127	128–255	256–511	512–1023
Shape	64×32	64×32	64×32 (×2)	256×32	256×32 (×2)
Size (μm^2)	38×15	38×15	38×15 (×2)	66×31	66×31 (×2)
E_{mem} (pJ)	0.24	0.24	2×0.24	0.51	2×0.51
E_{wires} (pJ)	0.00	0.23	0.82	0.46	1.5

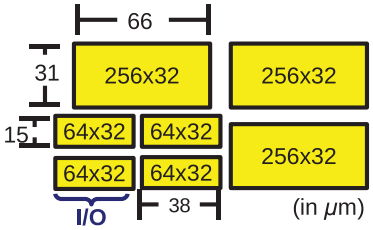


Fig. 5. Internal banking for 1024×32b memory.

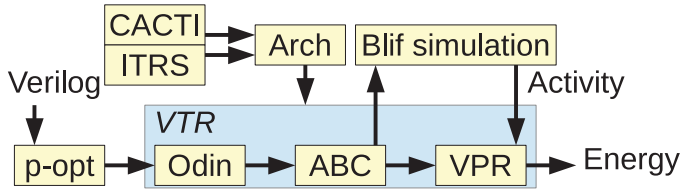


Fig. 6. Energy estimation tool flow.

corresponds to one signal for the enable, $30 = 2 \times 15$ for the address bits (15b covers the $32K \times 1$ operating mode), and 64 for the 32b input and 32b output. Additionally, 15μm is the distance to reach the bank. Similarly, we can compute the other E_{wires} costs. For example, to reach the upper right bank, we need to pay a distance of $(66 + 31)\,\mu$m. Then, the energy of an internally banked memory is given by $E_{banked} = E_{mem} + \alpha E_{wires}$, where α is the average activity factor over all signaling wires.

5. METHODOLOGY

Figure 6 shows our tool flow. We developed and added several components on top of a stock VTR 7 release.

5.1. Activity Factor Simulation

Activity factors and static probabilities assigned to the nets of a design have a major impact on the estimated energy. Common ways to estimate activity include assigning a uniform activity to all nets (e.g., 15%), or performing vectorless estimation with tools such as ACE [Lamoureux and Wilton 2006], as done by VTR. For better accuracy, our flow obtains activity factors by simulating the designs. We run a logic simulation on the BLIF output of ABC (`pre-vpr.blif` file) on a uniformly random input dataset. For example, for the matched 32Kb-memory matrix-multiply design in Figure 3(a), the average simulated activity factor is 11%, whereas ACE estimates it to be 3%, resulting in an energy estimation that is off by approximately 3.7x. The tunable benchmarks (Section 5.6) are designed in a streaming way that activates all memories all the time (independent of the random data), except matrix-multiply (`MMul`), for which the clock-enable signal is on approximately 1/3 of the time. The VTR benchmarks do not come with clock-enable for the memories, so we set them to be always on.

5.2. Power-Optimized Memory Mapping

When mapping logical memories onto physical memories, FPGA tools can often choose to optimize for either delay or energy using power-aware memory balancing [Tessier et al. 2007]. For example, when implementing a 2K×32b logical memory using eight 256×32b physical memories, we could choose to read $W = 4$b from each memory (delay-optimized, Figure 7(b)). Since each memory internally reads at the full, native

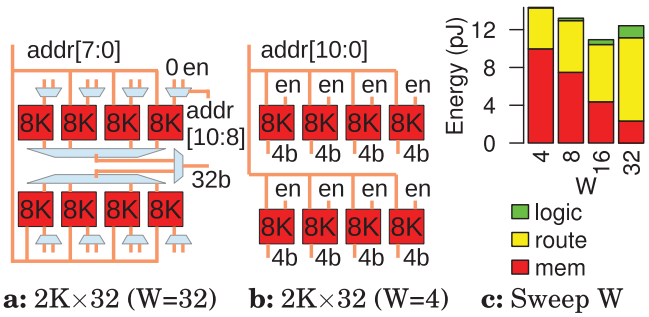


Fig. 7. Effect of memory block activation and output width selection on energy consumption.

width, the cost of the memory operation is multiplied by the number of memory blocks used. Alternatively, we could read $W = 32$b from only one of the memories (Figure 7(a)), in which case only one memory is activated at a time (reducing memory energy), but extra logic and routing overhead is added to select the appropriate memory and data. The power-optimized case often lies between these extremes. For example, in the experiment in Figure 7(c), the optimum is to activate two memories at once and read $W = 16$b from each.

Unfortunately, the VTR flow does not perform this kind of trade-off: it always optimizes for delay. Odin decomposes the memories into individual output bits [Rose et al. 2012], and the packer packs together these 1-bit slices as much as possible within the memory blocks to achieve the intended width [Luu et al. 2011]. In fact, VTR memories do not have a clock-enable, so they must be activated all the time. Instead, we use VTR architectures with special memory block instantiations that contain a clock-enable, modify VTR's architecture-generation script (`arch_gen.py`) to support these blocks, and add a `p-opt` stage before Odin to perform power-optimized memory mapping based on the memories available in the architecture. This includes performing memory sweeps as illustrated in Figure 7(c) to select the appropriate mapping for each application memory. For the robust architecture identified in Section 7, we find that mapping without `p-opt` adds 10% geomean energy overhead, comparable to the 6% benefit reported in Tessier et al. [2007]. Not using `p-opt` adds 41% worst-case energy overhead, suggesting that this optimization is more important for designs with high memory overhead. Our `p-opt` code and associated VTR architecture generation script are available as online supplemental material.

5.3. Logic and Routing Architecture

The logic architecture uses k4n10 logic blocks (clusters of 10 4-LUTs) and 36×36 embedded multipliers (which can be decomposed into two 18×18 multiplies, or four 9×9 multiplies) with $d_{mpy} = 20$ and the same shape and energy as in VTR's default 22nm architectures (a height of four logic tiles, plus we use $L_{mpyseg} = 4L_{seg}$). The routing architecture uses direct-drive segments of length 1 with Wilton switch boxes.

5.4. Technology

We use low-power (LP) 22nm technology [ITRS 2012] for logic evaluation and low stand-by power (LSTP) for memories. We use ITRS parameters for constants such as the unit capacitance of a wire at 22nm ($C_{wire} = 180$ pF/m). Then,

$$C_{metal} = C_{wire} \times \text{tile length}. \tag{23}$$

We evaluate interconnect energy based on this C_{metal} instead of the constant one that is provided in the architecture file. This way, the actual size of the low-level components

of the given architecture and technology, as well as the computed channel width, are taken into account when evaluating energy. It is important to model this accurately, as routing energy can dominate total FPGA energy (see Figure 7(c)).

5.5. Energy and Area of Memory Blocks

VTR assigns one type of block to each column on the FPGA (logic cluster, multiplier, or memory), and can give them different heights, but assumes the same horizontal segment length crossing each column. However, some memories can occupy a much larger area than a logic tile, and laying them out vertically to fit in one logic tile width would be inefficient. For energy efficiency, the memories should be closer to a square shape, and to that end, we allow the horizontal segment length crossing memories to be longer (which costs more routing energy, and hence $E_{mseg}(M) \neq E_{seg}$ in Section 3). We fix the height of the memory (h) ahead of time but keep the horizontal memory segment length (L_{mseg}) floating:

$$h = \left\lceil \frac{\sqrt{A_{sw}(W_0) + A_{mem}}}{\sqrt{A_{logic}(W_0)}} \right\rceil. \tag{24}$$

Here, W_0 is a typical channel width for the architecture and benchmark set. We use $W_0 = 80$. Then, when VPR finds the exact channel width, W_{act}, and hence the tile length and area (A_{logic}), we can adjust L_{mseg} accordingly:

$$L_{mseg} = \frac{A_{sw}(W_{act}) + A_{mem}}{h\sqrt{A_{logic}(W_{act})}}. \tag{25}$$

A_{mem} is the area for the memory obtained from CACTI, and A_{sw} is the switch area required to connect the memory to the FPGA interconnect. We obtain A_{sw} from VPR's low-level models, similar to the way it computes $A_{logic} = A_{luts} + A_{sw}$.

5.6. Benchmarks

To explore the impact of memory architecture, we use the VTR 7 Verilog benchmarks[1] [Luu et al. 2014] and a set of tunable benchmarks that allow us to change the parallelism level, P, in Section 6. Table I summarizes the benchmarks that have memories. We expect future FPGA applications to use more memory than the VTR 7 benchmarks. Some of them, such as stereovision, only model the compute part of the application and assume off-chip memory. We expect this memory to move on chip in future FPGAs. The tunable benchmarks allow us to explore parallelism tuning and provide better coverage of the large memory applications that we think will be more typical of future FPGA applications. For this reason, we do not expect a simple average of the benchmarks, such as the geometric mean, to be the most meaningful metric for the design of future FPGAs—it is weighted too heavily by memory-free and memory-poor applications.

We implemented the tunable benchmarks in Bluespec SystemVerilog [Bluespec 2012]; they are the following:

GMM: Gaussian mixture modeling (GMM) [Genovese and Napoli 2014] for an $N \times N$ pixel image, with 8b per pixel and $M = 5$ models. P pixels are computed every cycle ($N_{pe} = P$). This operation is embarrassingly parallel, as each PE is independent of the other ones. This benchmark has very high locality ($p = 0$).

WinF: 5×5 Gaussian window filter for an $N \times N$ pixel image, with 16b per pixel and power-of-two coefficients. $P = 1$ uses line buffers so that one main memory read

[1]Except `LU32PEEng` and `LU64PEEng` (similar to `LU8PEEng`), on which VPR 7.0 routing does not complete after 10 days, and `spree`, which segfaults through Odin.

Table I. Memory Requirements for the Benchmarks

Benchmark	Mem Bits	Memories (#)	Largest Mem
VTR			
boundtop	32K	1	1K×32
ch_intrinsics	256	1	32×8
LU8PEEng	45.5K	9	256×32
mcml	5088K	10	64K×36
mkDelayWorker32B	520K	9	1K×256
mkPktMerge	7.2K	3	16×153
mkSMAdapter4B	4.35K	3	64×60
or1200	2K	2	32×32
raygentop	5.25K	1	256×21
Tunable			
MMul	$(N^2+N)*32$	2P	$(N^2/P)\times 32$
GMM	$160\ N^2$	P	$(N^2/P)\times 160$
Sort	≈(128N+4NlogN)	log(N/P)-1+2P	(N/P)×[32+logN]
FFT (-twiddle)	≈224N	4P	(N/2P)×56
WinF (-line buffer)	$16\ N^2$	P	$(N^2/P)\times 16$

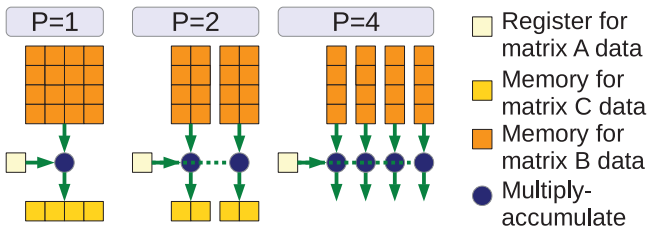


Fig. 8. Parallelism impact on memory and interconnect requirements for a $(4 \times 4)^2$ matrix-multiply.

and four line buffer reads and writes allow a throughput of 1 pixel per cycle. $P = 2$ and $P = 4$ extend the filter's window, share line buffers, and compute 2 and 4 pixels per cycle, respectively. For $P > 4$, every time P is doubled, the image is divided into two subimages, similar to the GMM benchmark ($N_{pe} = P$). This benchmark has medium locality, as each pixel needs to share information with its neighbors, but not with pixels that are farther away ($p = 0.5$).

MMul: $N \times N$ matrix-multiply ($A \times B = C$), with 32b integer values and datapaths (see Figure 8, Section 6.1, $N_{pe} = P$). This benchmark has low locality ($p = 2/3$).

FFT: N-point 28b fixed-point complex streaming Radix-2 FFT, with $P \times \log(N/P)$-stage FFTs followed by $\log(P)$ recombining stages ($N_{pe} = P \log(N)$). The streaming portion has high locality ($p = 0$), whereas the spatial combining section has high communication requirements ($p = 1$).

Sort: N-point 32b streaming mergesort [Koch and Torresen 2011], where each data point also has a $\log(N)$-bit index. One value is processed per cycle, and the parallelism comes from implementing the last $\log(P)$ stages spatially ($N_{pe} = \log(N/P) + (P - 1)$). We build a binary reduce tree to select the final output so that this streaming implementation has $p = 0$.

5.7. Limit Study and Mismatch Lower Bound

Sections 6 and 7 show the energy consumption for different applications and memory architectures. To identify bounds on the mismatch ratio, we also set up limit-study experiments. Our limit study assumes that each benchmark gets exactly the physical memory depth it needs, as if the FPGA were an ASIC. Therefore, there is no overhead

for using memories that are too small (no need for internal banking as in Section 3) or too large (no need to combine multiple memory blocks as in Section 5.2). We further assume that the limit-study memories have the same height as that of a logic tile, making them widely available and keeping the interconnect energy low for vertical memory crossings. Finally, we place memory blocks every two columns ($d_m = 1$) so that place-and-route tools can always find a memory right where they need one. To avoid overcharging for unnecessary memory columns, we modify routing energy calculations and ignore horizontal memory-column crossings for the limit study ($E_{mseg} = 0$). We also want to minimize the effect of width mismatch, for which we could use a similar trick as for the depth: use a large width (e.g., 1024b), but only pay for what the memory actually uses (e.g., 32b). However, this artificially increases the channel width since each input and output data bit needs to access the routing network (this is also the case with the depth trick, but to a lesser extent, as each doubling in depth only adds one bit to the address). Therefore, to minimize the effect of width mismatch, we run multiple limit studies, with memory widths of 8b, 16b, 32b, 64b, and 128b, and choose the one that achieve the lowest energy. CACTI energy does not decrease for width below 8b.

6. PARALLELISM TUNING

In Section 2, we saw that the amount of parallelism used to perform a task affects its energy consumption and that the optimum level of parallelism grows with problem size (Equation (12) and its implications). In this section, we show experimental results from optimizing the parallelism for the tunable benchmarks at different dataset sizes.

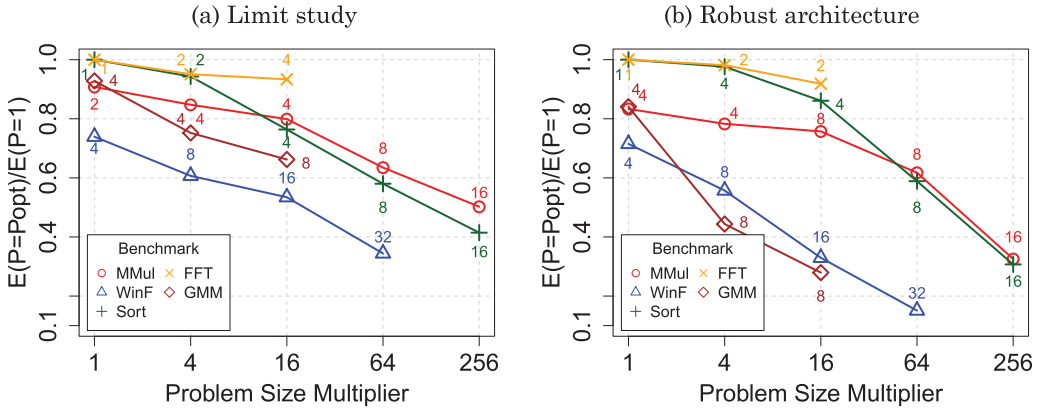
6.1. Example: MMul

Let us first review a specific example of a task where parallelism can be tuned to illustrate how memory is decomposed and how communication requirements change. Figure 8 shows the shape of an $N \times N$ by $N \times N$ matrix-multiply $A \times B = C$ for different parallelism levels P ($N = 4$ is shown). The computation is decomposed by columns, with each PE performing the computation for N/P columns of the matrix. The B data is streamed in first and stored in P memories of size N^2/P, then A is streamed in row major order. Each A data point ($A[i, k]$) is stored in a register, data for each column (j) is read from each B memory, a multiply-accumulate is computed ($C[i, j] = C[i, j] + A[i, k] \cdot B[k, j]$), and the result is stored in a C memory of size N/P.[2] Once all of the A data points of a row have been processed, the results of the multiply-accumulates can be streamed out, and the C memories can be used for the next row. When $P = N$, C does not need memories. Either way, increasing P keeps the total number of multiply-accumulates and memory operations constant. However, since the memories are organized in smaller banks, each memory access now costs less, and energy is reduced, as long as the interconnect per PE does not increase too much.

6.2. Parallelism Tuning with Limit-Study Architecture

To explore the effect of parallelism on energy without the bias introduced by a fixed memory architecture, we first sweep the size and parallelism of our five tunable benchmarks using the limit-study architecture described in Section 5.7. This is how we obtain Figure 1 for WinF. We can see that as the number of PEs increases, the total energy

[2]This is different from the matrix-multiply in Section 3, where C was stored in an output memory of size N^2 ($P = 1$), keeping only one size of memory for the application.



[The optimum PE count is shown next to each point]

	MMul	FFT	WinF	GMM	Sort
Base problem sizes: N =	16×16	1024	128×128	128×128	512

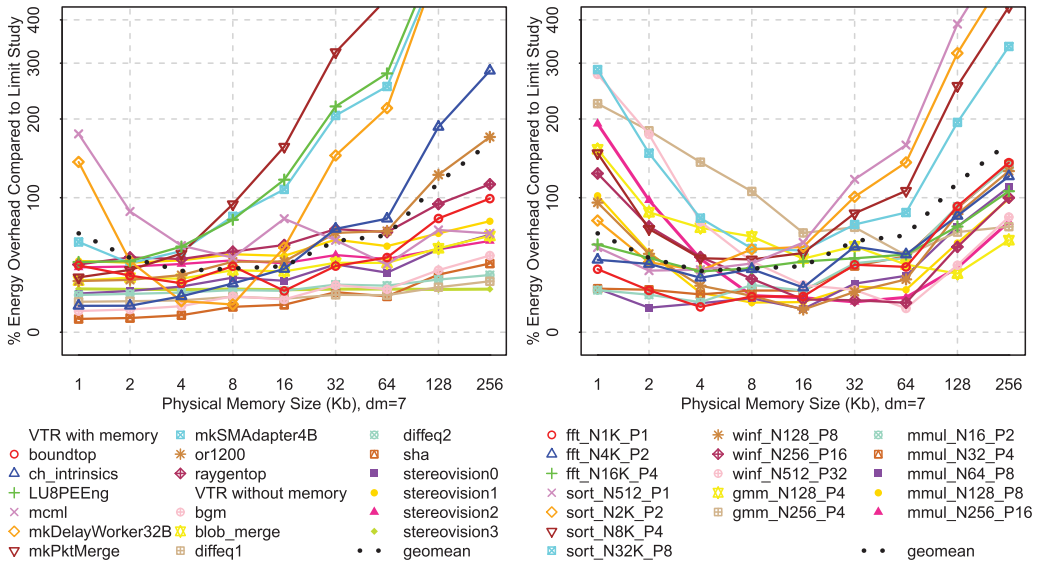
Fig. 9. Optimum parallelism level versus problem size.

decreases, reaches a minimum, then starts increasing. The decrease is due to decreasing memory energy (see Figure 1(b)). The increase is due to increasing interconnect energy. We also notice that the optimum number of PEs (P_{opt}) increases with increasing problem size. We run the same experiment for the other benchmarks, normalize the energy at P_{opt} to the energy at $P = 1$, and we get Figure 9(a). We see that the other benchmarks follow the same trend. The benefit of tuning parallelism to the optimum number of PEs grows with the problem size. For the largest sizes shown, tuning parallelism provides energy savings of 2.1× for WinF, 1.7× for MMul, 1.1× for FFT, 2.4× for Sort, and 1.4× for GMM. It should be clear from both Figure 9(a) and Equation (11) that the parallelism benefit will continue to grow with problem size.

6.3. Parallelism Tuning with Concrete FPGA Architecture

We now explore the benefits of parallelism with a concrete FPGA architecture using internally banked 16Kb memory blocks with $width = 32$ and $d_m = 7$ (the robust architecture identified in Section 7). The optimum level of parallelism may change from the limit study to adapt to the given physical architecture, but the major trend still holds. Figure 9(b) shows the same experiments as before on this concrete FPGA memory organization. This time, the energy savings at P_{opt} compared to $P = 1$ are larger than in the limit-study case: 4.7× for WinF, 2.6× for MMul, 1.1× for FFT, 3.3× for Sort, and 3.0× for GMM. In addition to finding the right level of parallelism, these concrete designs benefit from selecting a logic-memory balance that reduces the mismatch overhead.

The largest design in Figure 9(b) is the 512×512, eight PEs GMM, using 250,000 LUTs, 90,000 registers, 40Mb of memory, a 206×206 array, and a channel width of 128. This fits comfortably on modern FPGAs. In fact, Kadric et al. [2014] showed the same energy reduction through optimal parallelism trends for a subset of the benchmarks (and fixed problem sizes) on the Stratix IV. In cases where the ideal parallelism level is too large, we may be limited by the FPGA size. This is, for example, the case for larger GMM sizes, not shown in Figure 9(b), which require more PEs, and hence larger arrays, than our tools can currently support.



(a) VTR benchmarks (b) Tunable benchmarks

Fig. 10. Sweep of physical memory block size at fixed [dm = 7, width = 32b].

7. MEMORY EXPLORATION

Section 3 suggested that we could build a robust FPGA that bounds the worst-case energy mismatch. In this section, we explore the different memory architecture parameters (*depth*, *width*, d_m, internal banking) and identify optimum regions of operation.

7.1. Memory Block Size Sweep

We start with the simplest memory organization that uses a single memory block size and no internal banking (Figure 10) at a fixed $d_m = 7$ and $width = 32$b. For comparison, energy is normalized to the lower bound obtained using the limit study. We include all of the benchmarks from Section 5.6, including multiple sizes for the tunable benchmarks, where we set the number of PEs to the optimum values found in Figure 9(a). Most of the curves have an energy-minimizing memory size between the two extreme ends (1Kb and 256Kb), including the geomean curve. Benchmarks with little memory have an energy-minimizing point at the smallest memory size (1Kb). Benchmarks with no memory have a close-to-flat curve, paying only to route over memories but not for reads from large memories. The 4Kb memory architecture minimizes the geometric mean energy overhead of all benchmarks at 37%. As noted (Section 5.6), the geometric mean is weighted heavily by the many benchmarks with little or no memory and thus may not be the ideal optimization target for future FPGA applications. `gmm_N256_P4` and `mkPktMerge` define the maximum energy overhead curve and suggest that an 8Kb memory minimizes worst-case energy overhead at 110% of the lower bound.

Figure 11 shows the detailed breakdown of energy components for three benchmarks, where the memory capacity is varied, both for the normal case (top row) and for the internally banked memories (bottom row). We also show a blue line highlighting the lower bound obtained from the limit study. Most benchmarks have the shape of `Sort_N2K_P2`, with an energy-minimizing memory size between 4Kb and 32Kb. Small benchmarks with small memories have the shape of `mkSMAdapter4B`, with large increases in memory energy with increasing memory block size. The bottom row shows how internal banking

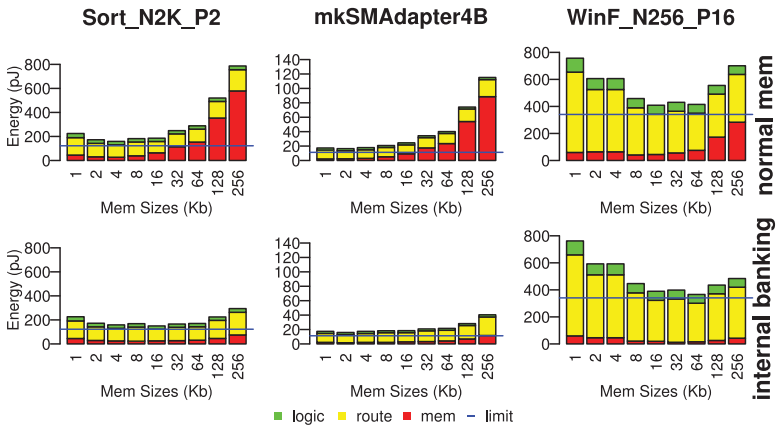


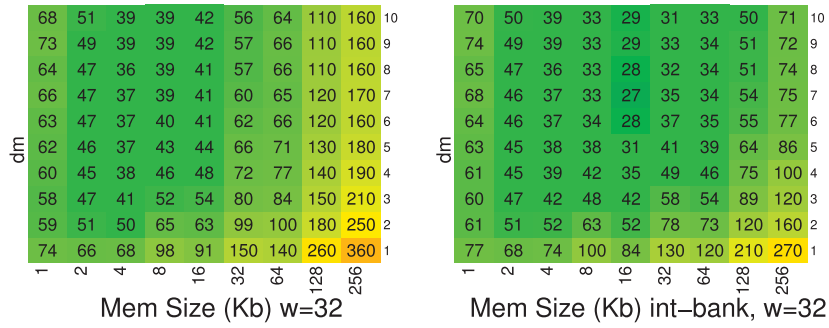
Fig. 11. Detailed breakdown of energy versus memory block size [dm = 7, width = 32b].

reduces this effect. It may even allow the minimum energy point to shift. For example, in `WinF_N256_P16`, the minimum shifts from 16Kb to 64Kb, reducing total energy at the energy-minimizing block size from 410pJ to 370pJ, or by 11%.

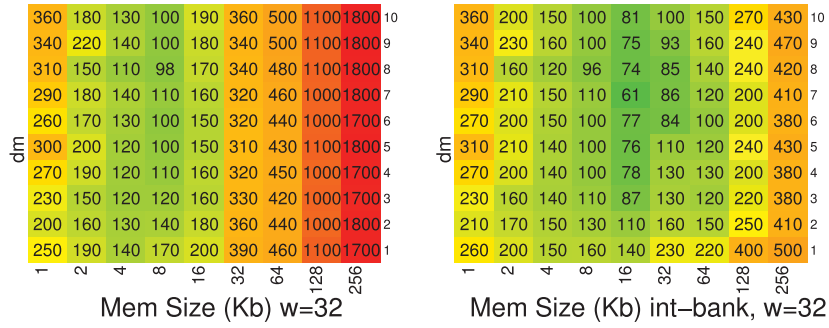
7.2. Impact of d_m

In Section 3, we showed analytically why the spacing between memory columns, d_m, should be chosen to balance logic and memory to minimize worst-case energy consumption. For simplicity, we limited Figure 10 to only use $d_m = 7$. Since the optimal values of d_m may vary among benchmarks, Figure 12 shows geomean (a) and worst-case (b) energy overheads when varying both memory block size and d_m, still at a fixed $width = 32$b. Without adding internal banking, sweeping d_m allows us to identify a lower energy point with 98% worst-case overhead ($d_m = 8$) versus the 110% we found previously when only looking at $d_m = 7$. We see broad ranges of values that achieve near the lowest geometric mean point, with narrower regions that minimize the worst-case overhead. The heatmap shows that overhead has a stronger dependence on memory block size than memory spacing. If we approximate the Cyclone V as 8Kb and $d_m = 9$, and the Stratix V as 16Kb and $d_m = 9$,[3] we can see that these two commercial architectures are around the energy-minimizing valley for both geometric mean and worst case. However, we can improve energy further by using internal banking, which broadens the energy-minimizing valleys, shifts them toward larger memory sizes, and overall reduces the overhead. Compared to the 16Kb, non–internally banked commercial architectures ($\approx$ Stratix V), we can reduce the worst case by 46% ((180-98)/180) by selecting an energy robust design point that bounds mismatch effects, which means using an architecture closer to that of the Cyclone V. Then we can reduce the energy by another 38% ((98-61)/98) by using internal banking and retuning the memory organization. This also reduces the geomean by 36% ((42-27)/42). We achieve these benefits with the 16Kb, internally banked, $d_m = 7$ architecture. Since our logic block is smaller, our energy minimizing cases tend to place the memories more frequently than the commercial architectures, closer to the robust balance point identified analytically in Section 3. In Kadric et al. [2015], we also explored architectures with two memory sizes and found that without internal banking, they achieved a similar reduction in

[3]Modeled points have square logic clusters and memories, whereas real Stratix and Cyclone devices are rectangular. Cyclone V and Stratix V logic blocks have 10 6-LUTs—larger logic blocks than the 10 4-LUT clusters that we use here.



(a) Geomean (% overhead)



(b) Worst case (% overhead)

Fig. 12. Energy overhead versus memory block size and d_m.

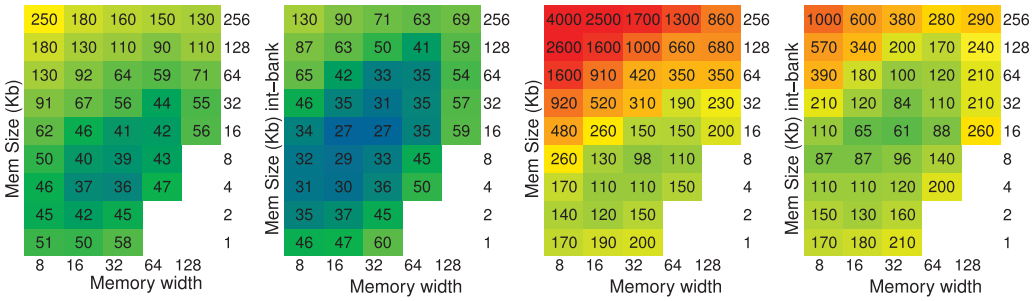
Table II. Area Comparison of Select Memory Organizations

Architecture	Internal Bank?	Size	Width	d_m	Maximum Overhead	Relative Area
robust, energy-min.	Yes	16Kb	32	7	61%	1.09×
≈ "Cyclone V"	No	8Kb	32	9	98%	1.00×
≈ "Stratix V"	No	16Kb	32	9	180%	1.07×

worst-case overhead to one memory with internal banking. Combining internal banking and two memory sizes can further reduce energy overheads at the expense of higher area. Table II shows that our robust, internally banked memory architecture has modest area impact compared to the alternatives.

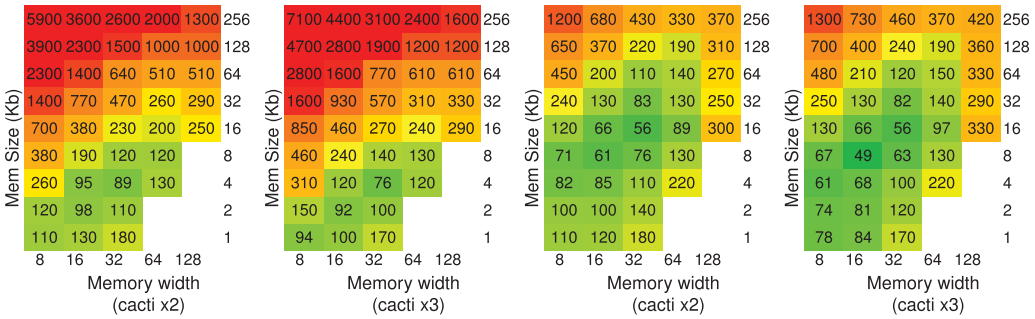
7.3. Impact of Memory Width

Figure 13 shows geomean (a) and worst-case (b) energy overheads when varying memory capacity and data width, with and without internal banking. Each point on the heatmap shows the energy overhead at the d_m that minimizes it. The lower-right corner is missing because we do not explore the cases where $width > depth$. Each heatmap shows an energy-minimizing valley running along the bottom-left to top-right axis. Once again, we observe that using internal banking broadens the valley, highlighting the most robust architectural point at 16Kb and $width = 32$b, with internal banking. Not shown on the heatmap, this point has $d_m = 7$. This architecture keeps the worst-case overhead below 61% and the geomean overhead below 27% across mismatches in memory block size, memory column spacing, and memory width.



(a) Geomean (% overhead) (b) Worst case (% overhead)

Fig. 13. Energy overhead versus memory block size and data width.



(a) Normal memories (% overhead) (b) Internal banking (% overhead)

Fig. 14. Sensitivity of Figure 13(b) (worst-case overheads) to CACTI estimates.

7.4. Sensitivity

The best memory sizes and the magnitude of benefits achievable are sensitive to the relative cost of memory energy compared to interconnect energy. Since PowerPlay [Altera Corporation 2013] estimates that the Altera memories are more expensive (about 3× the energy—perhaps because the Altera memories are optimized for delay and robustness rather than energy) than the energy-delay-squared-optimized memories that CACTI predicts are possible, it is useful to understand how this effect might change the selection of architecture. Therefore, we perform a sensitivity analysis where we multiply the energy numbers reported by CACTI by factors of 2× and 3× (Figure 14). Without internal banking, the relative overhead cost of using an oversized memory is increased, shifting the energy-minimizing bank size down to 4Kb (instead of 8Kb previously). The benefit of internal banking remains around 60% throughout all cases.

Even though CACTI can be configured to tune memories for energy, its basic architecture is delay oriented and does not include many energy-oriented optimizations that could be relevant to energy-minimizing FPGA embedded memories. There is considerable room for future work to optimize these memories, and impact both the energy of the ideally matched, limit-study memories, and the fixed-size FPGA memory blocks.

8. CONCLUSIONS

Communication energy dominates computations, whether the communication is moving data in and out of memories or moving data over wires to different processing points on the chip. Tuning the level of parallelism exploited for an application can shift this communication energy between memories and interconnect, often changing the energy

required for the computation. As a result, for each application and dataset size, there is an optimal level of parallelism that minimizes energy. This minimum energy point balances the energy spent communicating between PEs with the energy spent reading from local data memories. The optimal level of parallelism grows with problem size, as does the energy benefit compared to a nonparallel design. We show 4.7× energy reduction compared to the nonparallel design.

These communication results underscore the need for an energy-optimized on-chip memory system, and the need to support flexible memory and processing systems that can be tuned to the application and dataset size. We have shown how to size and place embedded memory blocks to guarantee that energy is within a factor of 1.6 of the optimal organization for the application. On the benchmark set, we have seen that a 32b wide, 16Kb, internally banked memory block keeps the worst-case mismatch energy overhead below 61% compared to an optimistic limit-study lower bound. Without internal banking, the Cyclone V memory organization of 32b wide, 8Kb memories with $d_m = 9$ achieves close to the smallest worst-case overhead of 98%. Although the memory organization is similar, the Cyclone V logic block is closer to 20 6-LUTs, thus it represents a more logic rich design. Commercial architecture with 32b wide, 16Kb, non–internally banked memory blocks have a worst-case overhead of 180%. Tuning for robust energy cuts the worst-case mismatch overhead by 46%, and internal banking provides 38% savings on top of that.

REFERENCES

Altera Corporation. 2013. *PowerPlay Early Power Estimator*. Altera Corporation, San Jose, CA. http://www.altera.com/support/devices/estimator/pow-powerplay.jsp.

Vaughn Betz, Jonathan Rose, and Alexander Marquardt. 1999. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer, Norwell, MA.

Sandeep Bhatt and Frank Thomson Leighton. 1984. A framework for solving VLSI graph layout problems. *Journal of Computer System Sciences* 28, 300–343.

Bluespec. 2012. Bluespec SystemVerilog 2012.01.A. Available at http://www.bluespec.com.

S. Y. I. Chin, C. S. P. Lee, and Steven J. E. Wilton. 2006. Power implications of implementing logic using FPGA embedded memory arrays. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*. 1–8. DOI:http://dx.doi.org/10.1109/FPL.2006.311200

André DeHon. 1999. Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization). In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 69–78.

André DeHon. 2015. Fundamental underpinnings of reconfigurable computing architectures. *Proceedings of the IEEE* 103, 3, 355–378. DOI:http://dx.doi.org/10.1109/JPROC.2014.2387696

Michael Delorimier, Nachiket Kapre, Nikil Mehta, and André DeHon. 2011. Spatial hardware implementation for sparse graph algorithms in GraphStep. *ACM Transactions on Autonomous and Adaptive Systems* 6, 3, Article No. 17. DOI:http://dx.doi.org/10.1145/2019583.2019584

Wilm E. Donath. 1979. Placement and average interconnection lengths of computer logic. *IEEE Transactions on Circuits and Systems* 26, 4, 272–277.

M. Genovese and E. Napoli. 2014. ASIC and FPGA implementation of the Gaussian mixture model algorithm for real-time segmentation of high definition video. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 3, 537–547. DOI:http://dx.doi.org/10.1109/TVLSI.2013.2249295

J. B. Goeders and Steven J. E. Wilton. 2012. VersaPower: Power estimation for diverse FPGA architectures. In *Proceedings of the International Conference on Field-Programmable Technology*. 229–234. DOI:http://dx.doi.org/10.1109/FPT.2012.6412139

Thomas L. Heath and Euclid. 1956. *The Thirteen Books of Euclid's Elements, Books I and II* (2nd ed.). Dover Publications.

ITRS. 2012. International Technology Roadmap for Semiconductors. Available at http://www.itrs2.net/itrs-reports.html.

Edin Kadric, David Lakata, and André DeHon. 2015. Impact of memory architecture on FPGA energy consumption. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 146–155.

Edin Kadric, Kunal Mahajan, and André DeHon. 2014. Kung Fu data energy-minimizing communication energy in FPGA computations. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*.

Dirk Koch and Jim Torresen. 2011. FPGASort: A high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 45–54.

Ian Kuon and Jonathan Rose. 2007. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 2, 203–215.

J. Lamoureux and Steven J. E. Wilton. 2006. Activity estimation for field-programmable gate arrays. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*. 1–8. DOI:http://dx.doi.org/10.1109/FPL.2006.311199

B. S. Landman and R. L. Russo. 1971. On pin versus block relationship for partitions of logic circuits. *IEEE Transactions on Computers* 20, 1469–1479.

David Lewis, Elias Ahmed, David Cashman, Tim Vanderhoek, Chris Lane, Andy Lee, and Philip Pan. 2009. Architectural enhancements in Stratix-III and Stratix-IV. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 33–42. DOI:http://dx.doi.org/10.1145/1508128.1508135

David Lewis, David Cashman, Mark Chan, Jeffery Chromczak, Gary Lai, Andy Lee, Tim Vanderhoek, and Haiming Yu. 2013. Architectural enhancements in Stratix V. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 147–156. DOI:http://dx.doi.org/10.1145/2435264.2435292

Jason Luu, Jason Helge Anderson, and Jonathan Scott Rose. 2011. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 227–236. DOI:http://dx.doi.org/10.1145/1950413.1950457

Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, et al. 2014. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems* 7, 2, 6:1–6:30. DOI:http://dx.doi.org/10.1145/2617593

Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. 2009. *CACTI 6.0: A Tool to Model Large Caches*. HPL 2009-85. HP Labs, Palo Alto, CA. http://www.hpl.hp.com/techreports/2009/HPL-2009-85.html.

Kara K. W. Poon, Steven J. E. Wilton, and Andy Yan. 2005. A detailed power model for field-programmable gate arrays. *ACM Transactions on Design Automation of Electronic Systems* 10, 2, 279–302.

Jonathan Rose, Jason Luu, Chi Wai Yu, Opal Densmore, Jeffrey Goeders, Andrew Somerville, Kenneth B. Kent, Peter Jamieson, and Jason Anderson. 2012. The VTR Project: Architecture and CAD for FPGAs from Verilog to routing. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. ACM, New York, NY, 77–86.

R. Tessier, V. Betz, D. Neto, A. Egier, and T. Gopalsamy. 2007. Power-efficient RAM mapping algorithms for FPGA embedded memory blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 2, 278–290. DOI:http://dx.doi.org/10.1109/TCAD.2006.887924

C. Thompson. 1979. Area-time complexity for VLSI. In *Proceedings of the ACM Symposium on Theory of Computing*. 81–88.

Henry Wong, Vaughn Betz, and Jonathan Rose. 2011. Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 5–14.

Received July 2015; accepted December 2015