AnyCore: A Synthesizable RTL Model for Exploring and Fabricating Adaptive Superscalar Cores

Rangeen Basu Roy Chowdhury, Anil K. Kannepalli, Sungkwan Ku, and Eric Rotenberg

Department of Electrical and Computer Engineering
North Carolina State University
{rbasuro, akannep, sku2, ericro}@ncsu.edu

Abstract

Adaptive superscalar cores have the ability to dynamically adjust their execution resources to match the instruction-level parallelism (ILP) of different program phases. The goal of adaptivity is to maximize performance in as energy-efficient a manner as possible. This is achieved by disabling execution resources that contribute only marginally to performance for the code at hand. Researchers have proposed many adaptive features, including structures, superscalar width, and pipeline depth. The benefits of adaptivity are eroded by its circuit-level overheads. Unfortunately, circuit-level overheads cannot be effectively estimated or appreciated without a hardware design. To this end, we developed a register-transfer-level (RTL) design of a highly adaptive superscalar core, called AnyCore. AnyCore can be used to quantify logic overheads of an adaptive core with respect to fixed cores, synthesize and compare different adaptive cores, synthesize and compare an adaptive core to a multi-core comprised of multiple fixed core types, and fabricate adaptive superscalar cores. We provide examples of these use-cases.

1. Introduction

As speed improvements from technology scaling slow with the end of Dennard scaling, processor architectures are becoming increasingly heterogeneous to eke out the most performance and energy efficiency from silicon. Heterogeneity ranges from CPU-GPU hybrids, to accelerators, to single-ISA heterogeneous multi-core processors, to adaptive/reconfigurable processors.

Adaptive superscalar cores have the ability to dynamically adjust their execution resources to match the instruction-level parallelism (ILP) of different program phases. The goal of adaptivity is to maximize performance in as energy-efficient a manner as possible. This is achieved by disabling execution resources that contribute only marginally to performance for the code at hand. Researchers have proposed various adaptive features, including size-adjustable structures (reorder buffer, physical register file, issue queue, load and store queues, caches, etc.) [28, 5, 7, 21], superscalar width (number of pipeline ways) [16], and pipeline depth [22, 30].

The benefits of adaptivity are eroded by its circuit-level overheads. Unfortunately, circuit-level overheads cannot be effectively estimated or appreciated without a hardware design. To this end, we developed a register-transfer-level (RTL) design of a highly adaptive superscalar core, called *AnyCore*. AnyCore RTL is the centerpiece of the overall *AnyCore Toolset* which enables computer architects to explore, and even fabricate, adaptive superscalar cores. The AnyCore Toolset, which will be open-sourced, is comprised of the following:

- *AnyCore RTL Model.* This is the synthesizable RTL model of AnyCore (written in Verilog). The RTL description of the core is heavily parameterized, allowing AnyCore processors of different sizes to be synthesized. A given AnyCore processor is adaptive, *i.e.*, its widths and structures can be dynamically adjusted within its maximum dimensions.
- *Clock gating and power gating tools.* Adaptivity in the AnyCore RTL is achieved nominally through combinational logic that configures the formation of fetch bundles (fetch width), the execution lanes used (issue width), and the structure partitions used (structure sizes). To maximize power savings, unused lanes and partitions should also be clock-gated or power-gated. For the former, the AnyCore RTL instantiates a clock gate cell for each lane and partition. A Verilog wrapper makes it easy for the user to specify the clock gate cell for a given standard cell library, and clock tree synthesis automatically sizes clock gate cells. For the latter, the RTL description is paired with a Unified Power Format (UPF) [2, 3] description of power domains, which today's EDA tools can handle for automatically synthesizing, simulating, measuring, and generating layouts of, power-gated designs. The provided UPF serves as an example of placing lanes and partitions into power domains, upon which the user can improvise to explore different configurations of power domains.
- *Synthesis and Analysis Flow.* Low-level power estimation flows, based on post-synthesis/post-layout netlist simulation, are beyond the experiences of many computer architecture researchers. The AnyCore Toolset includes such a flow out-of-the-box. The utility of low-level power estimation is demonstrated in this paper's first use-case: measuring logic overheads of an adaptive core with respect to several fixed cores.
- *AnyCore PAT Tool.* The Power, Area, and Timing (PAT) tool enables computer architecture researchers to automatically generate per-stage/per-lane power, cycle time, and area estimates, for:
  - both fixed cores (AnyCore RTL can excise its logic for adaptivity) and adaptive cores,

978-1-5090-1953-3/16/$31.00 ©2016 IEEE

– different maximum fetch widths, issue widths, and structure sizes,
– different dynamic configurations (for adaptive cores),
– either synthesized (standard-cell based) or custom memories (*e.g.*, from a memory compiler or memory estimation tool).

The per-unit power estimates can be combined with respective per-unit activity counts produced by a C++ cycle-level microarchitecture simulator, thereby enabling architecture level studies with synthesis-based power models and comprehensive logic coverage. AnyCore RTL can be synthesized to different technology nodes and standard cell libraries, accommodating researchers with access to commercial IP and/or open-source IP such as FreePDK 45nm [33], FreePDK 15nm [12], and Synopsys 32/28nm [1].

AnyCore can be used to quantify logic overheads of an adaptive core with respect to fixed cores, synthesize and compare different adaptive cores, synthesize and compare an adaptive core to a multi-core comprised of multiple fixed core types, and fabricate adaptive superscalar cores. We provide several such use-cases in this paper.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes the AnyCore Toolset. Sections 4, 5, and 6 provide three use-cases. Section 4 provides an example of quantifying the logic overheads of an adaptive core with respect to fixed cores. Section 5 provides an example of using the AnyCore PAT tool and a C++ cycle-level simulator, to compare an adaptive core and a heterogeneous multi-core. In Section 6, we describe the AnyCore prototype that we fabricated. This section of the paper describes the chip bring-up results and early performance and power measurements for different configurations.

2. Related Work

Past works on adaptive/reconfigurable processors lay the foundation for AnyCore. Researchers have separately proposed structure resizing [28, 5, 7, 21, 6, 10, 17, 26], pipeline depth adaptation [22, 30], and pipeline width adaptation [16, 9, 14]. Each of these works explore a single superscalar dimension. The Flicker microarchitecture [27] can scale both width and structures, but structure partitions are tied to lanes, hence, not independently adjustable. The chief distinction, however, is that we develop an RTL model of a width and size adaptive core and a toolset for using it. Thus, our contribution is an RTL-based tool for exploring and fabricating adaptive cores. In particular, the ability to measure and understand logic overheads of a reconfigurable core with respect to fixed cores is critical as the community considers adaptive cores in the era of specialization.

AnyCore and the above approaches carve out smaller monolithic cores from a superset monolithic core by deconfiguring lanes and/or structure partitions, with the goal of mimicking the optimal fixed core for the instruction-level behavior at hand. A different form of microarchitectural adaptivity is aggregating multiple small cores to emulate a single larger and wider superscalar core [18, 20, 34, 29]. The two modes offer high thread-level parallelism (TLP) (disaggregated cores) and high instruction-level parallelism (ILP) (aggregated cores).

Like the core-aggregation techniques, MorphCore [19] attempts to modulate the balance between single-thread performance and multi-threaded performance. Its approach is different, however, and advocates the same philosophy as AnyCore which is to begin with a large monolithic core and adapt it.

Single-ISA heterogeneous multi-core processors [23, 24, 25] achieve adaptation by migrating a thread among multiple, diverse, fixed core designs. Using fixed core designs eliminates the logic overheads of an adaptive core, but the number of fixed core designs is limited and the overhead manifests in thread migrations.

3. AnyCore Toolset

3.1. AnyCore RTL Model

The Verilog description of AnyCore is *parameterized*, in that we have the ability to synthesize AnyCore processors of different maximum widths (up to 8) and sizes (structures can be as large as desired), and a specific AnyCore processor is *adaptive* (*i.e.*, dynamically configurable) within its maximum width and size. Moreover, the logic for adaptivity can be excised, *i.e.*, its inclusion or exclusion is also parameterized, such that the same Verilog description can be used to synthesize both fixed cores and adaptive cores.

The starting point for the AnyCore RTL model was a 8-wide superscalar core generated by FabScalar [15]. This initial Verilog description was modified extensively to arrive at the AnyCore RTL model.

1. *Restructuring for lanes*: With FabScalar, each pipeline stage is a top-level Verilog module and each pipeline register between stages is also a top-level Verilog module. This top-level structure was preserved. Rather, within each stage, as much logic as practical was partitioned into lanes. That is, all of the logic that can be solely-affiliated with an instruction slot was grouped together in a submodule within the pipeline stage module. The same was done within pipeline register modules, *i.e.*, submodules were created for individual instruction packets.
2. *Partitioning of structures*: Each memory array that implements a structure (queue, list, table, register file, etc.) is already encapsulated as a dedicated Verilog module. Within this module, the memory array was partitioned into multiple submodules and MUXes aggregate them into the overall memory array.
3. *Static configurability*: While FabScalar can generate fixed cores of different fetch and issue widths, it achieves this via composition from a library of fixed-width stage designs (dedicated Verilog descriptions for 1-wide, 2-wide, *etc.*) [15]; only structure sizes are parameterized. In contrast, AnyCore is a single Verilog description in which both widths and structure sizes are parameterized. Moreover, a "fixed-versus-adaptive" parameter controls whether or not the logic for dynamic configurability (next item) is synthesized.

4. *Dynamic configurability (adaptivity)*: Logic was added to dynamically configure the design for arbitrary core configurations. First, previously fixed width parameters were made adjustable. The dynamic FETCH_WIDTH controls the number of fetched instructions that flow through the fetch through decode stages and into the Instruction Buffer; the dynamic DISPATCH_WIDTH controls the number of instructions that are withdrawn from the Instruction Buffer, moved through the rename and dispatch stages, and dispatched into the Issue Queue; and the dynamic ISSUE_WIDTH controls the number of issued instructions that flow through configured execution lanes. Second, previously fixed size parameters were made adjustable for all structures. Ultimately, the dynamic width and size parameters synthesize to two types of control signals: signals that ensure instructions flow through only the active lanes and structure partitions, and signals that control clock-gating or power-gating cells (to turn on or off power domains) and isolation cells (to clamp floating inputs from off-domains).

There are two different versions of the AnyCore RTL model, one that implements the PISA ISA [13] and another that implements the RISC-V ISA [8]. These are referred to as AnyCore-PISA and AnyCore-RISCV, respectively. AnyCore-PISA was developed first, owing to its derivation from FabScalar. AnyCore-RISCV is a port of AnyCore-PISA [11].

3.2. Clock Gating and Power Gating Tools

The AnyCore Toolset provides the computer architect with tools to implement clock gating and power gating with low effort. The key lies in the Verilog hierarchy within pipeline stages, which are subdivided into lanes, and pipeline structures, which are subdivided into partitions. This Verilog hierarchy enables efficient expression of clock gating or power gating of lanes and partitions.

3.2.1. Clock Gating. Each lane and partition receives a clock input. This clock input is sourced by a generic clock gate module. The generic clock gate module has the same interface as a clock gate cell in a standard cell library, but it is just a Verilog wrapper. Inside the wrapper, the user either instantiates (if doing clock gating) or does not instantiate (if not doing clock gating) a clock gate cell from his/her standard cell library. Clock tree synthesis automatically takes care of sizing of clock gate cells.

In addition to RTL-based clock gating or power gating of entire lanes and partitions, one may judiciously apply automatic clock gating during synthesis. We did not apply synthesis-based clock gating in the three use-cases that follow. For the chip (third use-case), we used partition-level and lane-level clock gating to turn off clocks to entire structure partitions and lanes, respectively. This is orthogonal to fine-grained clock gating and the two can be applied simultaneously on a design to save more power. This, however, would not change the trend of power consumption by the different configurations of AnyCore.

3.2.2. Power Gating. With power gating becoming increasingly important in future technology nodes, today's EDA tools and commercial standard cell libraries are increasingly sophisticated in their end-to-end support, from specification to layout, of multiple switchable power domains. Designers can specify power domains and assign Verilog module instances to power domains using Unified Power Format (UPF). The UPF power domain description accompanies the Verilog description throughout the tool flow. Different tools use the UPF according to their roles.

The AnyCore Toolset currently has several example UPF descriptions, which the user can adapt for his/her adaptive core designs.

3.3. Synthesis and Analysis Flow

Low-level power estimation, based on post-synthesis/post-layout netlist simulation, has utility in measuring and understanding logic overheads of adaptivity (as done in Section 4). Many computer architects do not routinely perform such experiments. Moreover, it is challenging to get such flows up-and-running from scratch. The AnyCore Toolset comes with a synthesis and analysis flow that can be used out-of-the-box.

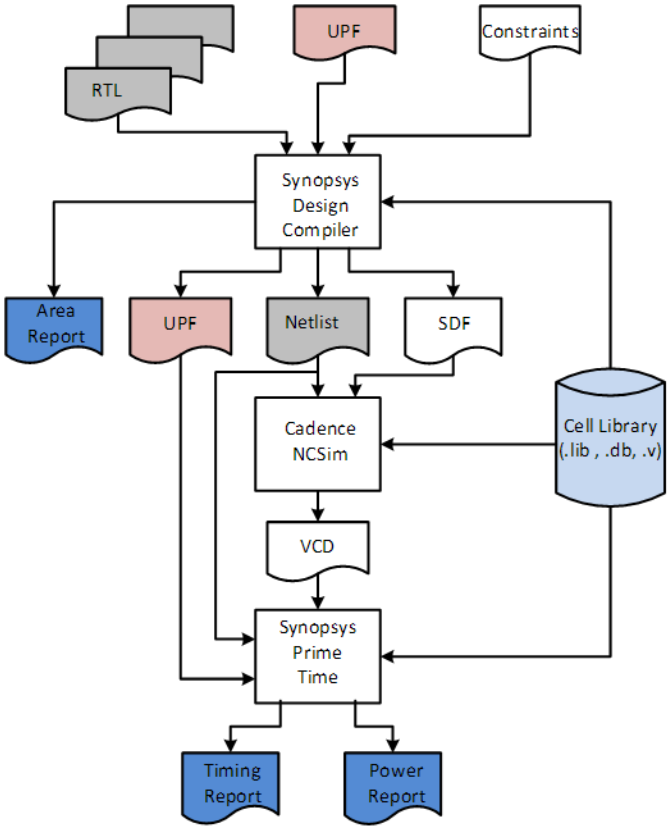


Figure 1: Synthesis & Analysis Flow

3.3.1. Synthesis. As shown in Figure 1, the inputs to Synopsys Design Compiler™ are the RTL description, an optional UPF description, and constraints. A UPF description is supplied for an adaptive core that employs power gating. After performing synthesis, Design Compiler outputs three key files: (1) the gate-level netlist in Verilog format, (2) the delays of gates and nets in Standard Delay Format (SDF), and (3) a UPF description

corresponding to the gate-level netlist. Synthesis also outputs area and frequency reports.

The next section explains how the post-synthesis tool flow uses these three key files to analyze timing and power consumption.

3.3.2. Analysis: Obtaining Cycle Time and Power. Synopsys Prime Time™ is used for analysis. Accurate cycle times are obtained using *static timing analysis* in Prime Time™. Power profiles of fixed cores and adaptive cores under chosen configurations are characterized via the two-step process of (1) netlist simulation (*i.e.*, gate-level simulation) and (2) time-based power estimation in Prime Time PX™.

1. *Netlist simulation*: After synthesizing a design, benchmarks are simulated on the synthesized netlist. As shown in Figure 1, Cadence NCSim takes in the netlist and SDF, and uses the SDF to annotate the netlist for detailed delay modeling. The output of a netlist simulation is switching activity information in the form of a Value Change Dump (VCD) file. Not shown in the diagram, is a top-level Verilog testbench that instantiates the netlist of the core and loads a benchmark for simulation on the core.
2. *Time-based power estimation*: As shown in Figure 1, the gate-level netlist, gate-level UPF, and activity file (VCD) are read into Prime Time PX™. Prime Time PX™ builds a virtual power delivery network based on the UPF. The time-based power estimation observes power domains being on or off, according to the states of power-gating control signals obtained from the VCD.

3.4. AnyCore PAT Tool

The AnyCore PAT tool automatically generates a database of power, area, and timing estimates for all variations of each pipeline stage (front-end) and each execution lane (back-end). The following description is geared towards "pipeline stage", but similar principles apply to "execution lane" (moreover, a given lane is impacted by the number of other lanes). For each pipeline stage, the following dimensions are varied: (1) Whether the pipeline stage is for a fixed core or an adaptive core. (2) The width of the pipeline stage. (3) The sizes of structures referenced by the pipeline stage. (4) The dynamically-configured width of the pipeline stage (only applies to an adaptive pipeline stage). (5) The dynamically-configured sizes of referenced structures (only applies to an adaptive pipeline stage). The first three dimensions affect the number of unique synthesis runs. The last two dimensions are handled by UPF-driven analysis of the synthesized netlist, for both clock gating (dynamic power) and power gating (static and dynamic power).

The user can specify the implementation strategy for each structure: synthesized memory (standard-cell based) versus custom memory (IP block). The latter is facilitated by a library of custom memories. Just like standard cells, each custom memory is abstracted by a .lib file which specifies its power, area, and timing parameters. A lower level tool within the AnyCore PAT tool generates .lib files from power, area, and timing data obtained by third-party memory estimation tools, *e.g.*, FabMem [32], CACTI [31], commercial or academic-use memory compilers [1], *etc.* The generated .lib files have the same names as corresponding leaf Verilog modules representing the structures or structure partitions. If the leaf Verilog module is black-boxed (empty module), synthesis uses the corresponding .lib file, otherwise it synthesizes the module.

4. Use-Case 1: Logic Overheads of an Adaptive Core

The goal of this use-case is to measure the logic overheads of an adaptive core with respect to fixed cores, and understand the sources of these overheads. The logic overheads depend on (1) the specific adaptive core design and (2) the configurations that will be compared against corresponding fixed cores. The adaptive core design and its configurations chosen for overhead analysis are described in Sections 4.1 and 4.2, respectively. We use the Synthesis and Analysis Flow from the AnyCore Toolset (from Section 3.3) to measure area, frequency, and energy overheads of adaptivity. Area and frequency overheads are studied in Section 4.3 and energy overhead is studied in Section 4.4.

Please note the following additional methodological points, *which are unique to this section*. First, the AnyCore-PISA version of the RTL was used. Second, the flow was configured to use a commercial 45nm standard cell library. Third, L1 instruction and data caches are not included in the analysis for both adaptive and fixed cores. Inclusion of identical L1 caches in the adaptive and fixed cores will reduce the reported *percentage* energy overheads as the caches will contribute a large in-common energy component. Fourth, all memories are synthesized. In later sections: Use-Case 2 uses the AnyCore-RISCV version of the RTL, an open-source 45nm standard cell library, custom memories, and L1 caches; Use-Case 3 is a AnyCore-PISA based chip fabricated in IBM 130nm with all memories synthesized, including L1 caches. Table 1 summarizes this information for the three use-cases.

4.1. Specific Adaptive Core Design

Figure 2 depicts the adaptive core that we synthesized from the AnyCore RTL model and the power domains specified by its UPF description. In its widest dynamic configuration, the pipeline has a fetch width of four instructions and an issue width of five instructions. The first three execution lanes – memory, control, and simple/complex ALU – suffice to support all integer instructions. Two additional simple ALU lanes round out the back-end. Key structures for exposing and extracting instruction-level parallelism (ILP) are also shown. They include the Issue Queue (IQ), Load and Store Queues (LQ, SQ), Physical Register File (PRF), Active List (AL), and Free List.

The UPF description specifies 9 power domains, one of which is the TOP domain that is always on. The power domains can be seen in Figure 2. Unshaded lanes and structure partitions (white) are in the TOP domain, hence, always on. Conversely, each shaded lane or structure partition (gray) is in its own power domain, and is labeled with a power domain number. Several structures are configured in unison and these exceptions will be pointed out.

Table 1: Methodology choices for the three use-cases in this paper.

Use-Case	Version of RTL Model	Technology	Memories	L1 Caches
1 (§4)	AnyCore-PISA	commercial 45nm	synthesized	excluded
2 (§5)	AnyCore-RISCV	open-source 45nm (FreePDK)	custom (FabMem)	included (cacti)
3 (§6)	AnyCore-PISA	IBM 130nm	synthesized	included (synthesized)

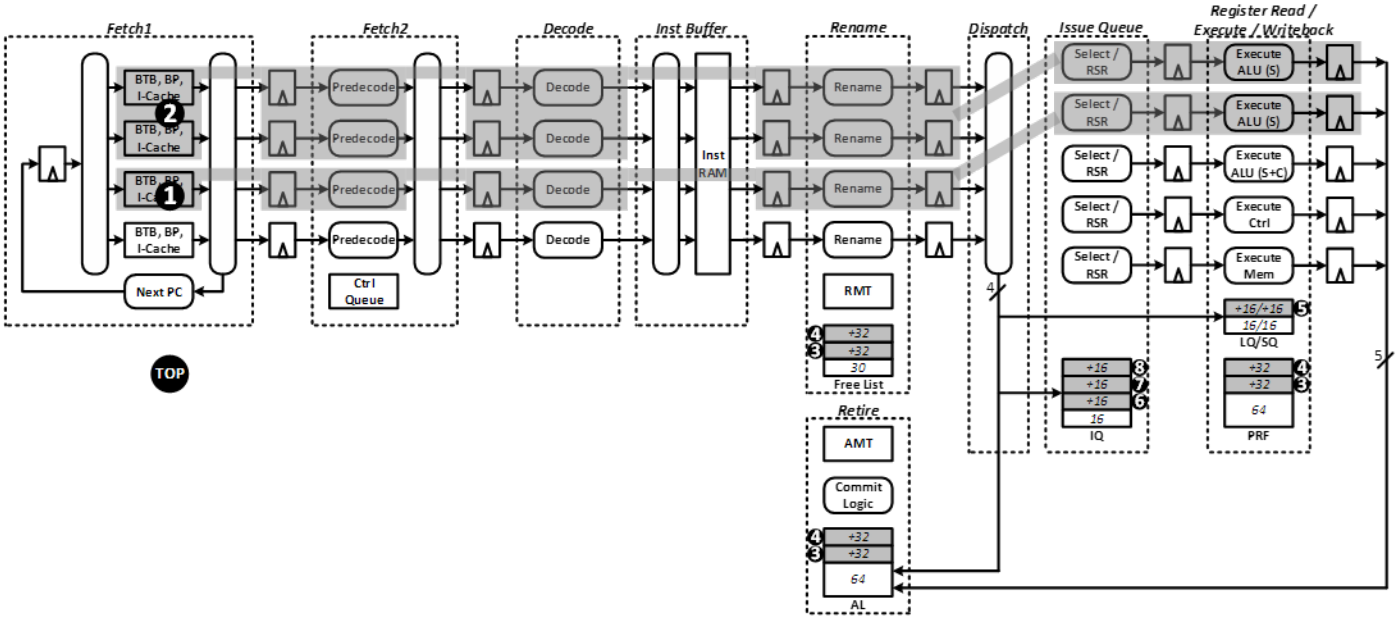


Figure 2: The adaptive core used in this section.

The TOP domain has one lane in the front-end and the first three execution lanes in the back-end, so that the minimim-width configuration is fully functional. All structures have a base partition in the TOP domain. The sizes of the base partitions, as well as the power-gated expansion partitions, are annotated in the figure. Some logic blocks within the front-end pipeline stages could not be efficiently or correctly partitioned by lane, typically due to operating on the bundle as a whole. These logic blocks are shown as spanning the whole width of the pipeline stage and are included in the TOP domain.

Superscalar width is increased via the power domains labeled "1" and "2" in the figure. Power domain "1" adds one more lane in the front-end plus one simple ALU lane in the back-end, for a 2-way fetch, 4-way issue superscalar processor. Power domain "2" adds another two lanes in the front-end and another simple ALU lane in the back-end, for a 4-way fetch, 5-way issue superscalar processor.

The remaining six power domains are for adapting the sizes of structures. The Physical Register File (PRF), Free List, and Active List (AL) scale together: each has two power-gated expansion partitions, in power domains "3" and "4". The Load and Store Queues (LQ/SQ) have one power-gated expansion partition, in power domain "5". Finally, the Issue Queue (IQ) has three power-gated expansion partitions, in power domains "6", "7", and "8".

4.2. Configurations Chosen for Overhead Analysis

The six configurations shown in Table 2 were chosen for overhead analysis. These configurations represent a range of widths and sizes. Three widths are represented (1, 2, and 4) and two different sizings for each width. Each configuration is named according to its width and relative size. For example, 4WL is a 4-wide core with large structures, 1WS is a 1-wide core with small structures, and so forth. The table uses abbreviations for the function unit mix: M (memory lane), B (control lane), S/C (both simple and complex ALUs), S (simple ALU only).

4.3. Area and Frequency

Figure 3 shows the synthesized areas of the six fixed cores and AnyCore. AnyCore's area includes isolation cells necessary for power gating. AnyCore's area overhead, with respect to the biggest fixed core (4WL), is 0.15 sq mm or about 19%.

Figure 4 shows the frequencies of the six fixed cores and AnyCore. An interesting result is that the frequencies across the different fixed cores are fairly flat and changes by only 6.7%. This can be attributed to the use of fully synthesized structures and our constraint strategy for synthesis (discussed further in Section 4.4). As logic depth for such a structure only increases logarithmically, doubling its size does not double the propagation delay through it. The frequency of AnyCore is 1.67 GHz, 8.3% lower than the smallest fixed core and only 1.67% lower than the

Table 2: Core Configurations.

Parameter	1WS	1WL	2WS	2WL	4WS	4WL
Fetch Width	1	1	2	2	4	4
Issue Width	3	3	4	4	5	5
Issue Queue	16	32	32	48	48	64
Load/Store Queues	16	16	16	32	16	32
Active List	64	96	64	96	96	128
Register File	64	96	64	96	96	128
BTB / BPU / RAS	256/1024/8	256/1024/8	512/2048/16	512/2048/16	1024/4096/16	1024/4096/16
Func Unit Mix	M,B,S/C	M,B,S/C	M,B,S/C,S	M,B,S/C,S	M,B,S/C,S,S	M,B,S/C,S,S

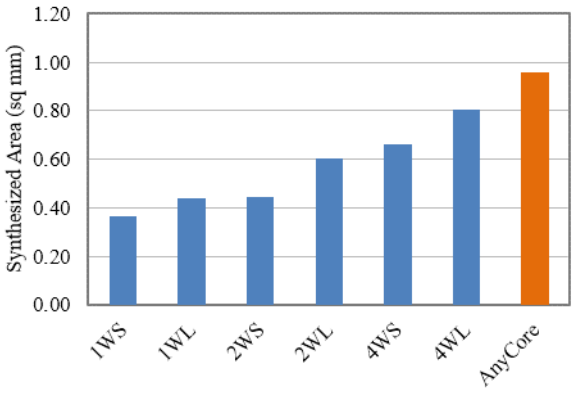


Figure 3: Synthesized areas of the six fixed cores and AnyCore.

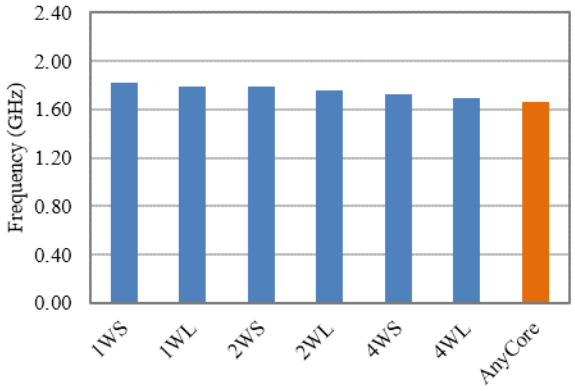


Figure 4: Frequencies of the six fixed cores and AnyCore.

largest one.

4.4. Energy

As explained in Section 3.3.2, the AnyCore Toolset's power analysis flow uses netlist simulation of benchmarks followed by time-based power estimation in Prime Time PX^{TM}. While we can use (and have used) SPEC benchmarks for netlist simulations, they do not utilize the entire core well (*e.g.*, frequent branch mispredictions cause underutilization of structures). Yet, we need high utilization in order to measure the intrinsic energy of AnyCore and the fixed cores. Therefore, we use a high-ILP microbenchmark for the netlist simulations that drive time-based power estimation.

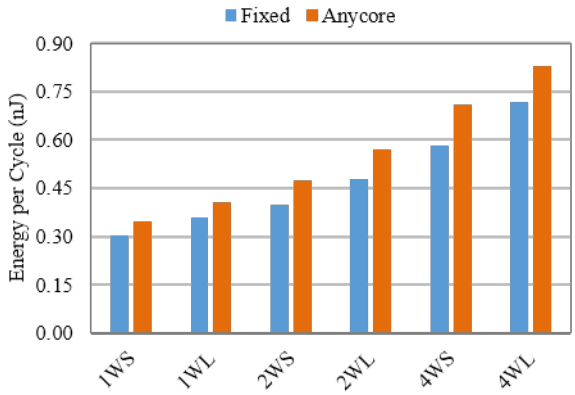


Figure 5: Energy per cycle of fixed cores and corresponding configurations of AnyCore.

The total energy/cycle (static and dynamic) of the different configurations are plotted in Figure 5. Each configuration has two data points, for the fixed core and AnyCore. Figure 6 shows the percent energy overhead of each AnyCore configuration relative to the corresponding fixed core. The energy overheads are significant, ranging from 17% (1WL) to 27% (4WS). Each bar in the graph is subdivided into static and dynamic energy contributions to the total overhead.

One trend is that the static energy contribution to overhead is larger for the smaller cores. This is due to two reasons. First, the cycle time difference is greatest between AnyCore and smaller fixed cores. Static energy increases with cycle time (whereas dynamic energy only increases with activity, hence, with number of cycles). Second, AnyCore provisions read and write ports to all structures for the widest superscalar configuration. The number of extra ports is greatest with respect to the narrowest cores. Conversely, there are no extra ports with respect to the widest cores. In narrow configurations of AnyCore, lanes (pipeline ways) are turned-off but their dedicated ports to structures – while not consuming dynamic power – still consume static power.

We were expecting energy overhead to be less with respect to fixed 4WL. Primarily, fixed 4WL is of the same superscalar complexity as AnyCore's maximum configuration. Their structures are the same size and have the same number of ports. Moreover, the cycle time difference is modest. For both reasons, we expect static energy overhead to not be a major factor for this comparison, and indeed it is not. So why is dynamic energy

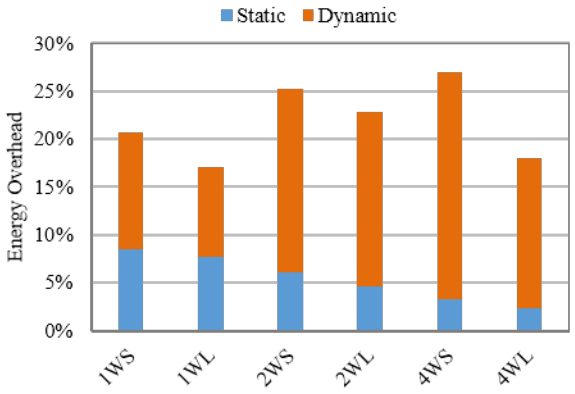


Figure 6: Percent energy overhead of AnyCore configurations w.r.t. fixed counterparts.

overhead so high for equal superscalar complexity? Partitioned structures are the root cause. Synthesis is able to optimize better with a monolithic structure than a structure with artificial internal boundaries. This factor manifested more as dynamic energy overhead than cycle time overhead owing to our strategy for constraining synthesis.

Due to the richness of modern standard cell libraries, synthesis timing constraints have a profound impact on tradeoffs among cycle time, dynamic power, and static power. Our strategy for synthesizing a core (whether it be AnyCore or a fixed core) is to make multiple synthesis runs, successively tightening the cycle time constraint by 10ps until it cannot be met. As the constraint is tightened, synthesis chooses cells that are faster but that consume greater area, static power, and dynamic power. Being too aggressive may lead to an incremental frequency gain while paying too much in terms of both dynamic and static energy. Static energy is a duplicitous factor, however: a key justification for minimizing cycle time (aside from performance considerations) is to decrease static energy/cycle, which increases with cycle time (unlike dynamic energy).

There are three key takeaways from this section.

1. The dominant source of static energy overhead is excess ports, which afflicts configurations that are narrower than the maximum width.
2. The dominant source of dynamic energy overhead is partitioning of structures. Artificial internal boundaries increase logic depth. This creates a tension for synthesis, between minimizing cycle time versus choosing slower energy-efficient cells.
3. These results motivate research on reducing logic overheads of adaptivity. Power gating excess ports within memory structures should be looked at. Reliably sensing and tuning frequency to the configuration is an important goal both for reducing static energy/cycle overhead and improving performance. Additionally, AnyCore's design itself may need to be adjusted to enhance the frequency differentials that may be possible.

5. Use-Case 2: Adaptive vs. Heterogeneous Cores

The purpose of this section is to show a use-case of the AnyCore PAT Tool. For per-stage/per-lane power estimates and whole-core cycle time estimates, we used the AnyCore PAT Tool targeting the FreePDK 45nm standard cell library [33], FabMem [32] for highly-ported RAMs/CAMs, and CACTI [31] for caches. The per-stage/per-lane power data are combined with corresponding activity counts from an in-house C++ cycle-level execute-at-execute microarchitecture simulator.

We compare an adaptive core and a heterogeneous multi-core processor customized to the workload. The workload consists of the highest-weighted 100M SimPoint from each of 15 SPEC2006 benchmarks compiled using "gcc -O3". Thus, the hetero palette consists of 15 core types and we will consider scheduling only the corresponding 15 dynamic configurations for the adaptive core. Since we are interested in trading some performance for an even larger energy savings, for each benchmark, we found the narrowest and smallest core that yields an IPC within 15% (or better) of the peak IPC (which is generally obtained on the widest and largest core).

We compare four architectures. "cg" and "fg" denote coarse-grain and fine-grain scheduling.

- Homogeneous: All benchmarks are run on the one core type that yields the highest harmonic mean BIPS.
- Hetero-cg: Each benchmark is run on the core type that yields the highest BIPS for that benchmark. Energy efficiency is built-in to some extent in the way the core palette was derived, as explained above. When maximum BIPS is the objective, 5 of the 15 core types in the hetero palette are utilized by the entire workload.
- AnyCore-fg: Each benchmark is divided into 10K-instruction intervals. Each interval is scheduled on the lowest-energy configuration that yields IPC within X% of the highest-IPC configuration (for that interval). We consider values of X between 0% and 25%, in 5% increments, yielding a pareto frontier of six points in the energy vs. delay graphs that follow.
- Hybrid: This architecture is explained below.

The graph in Figure 7a plots energy versus delay for the *namd* benchmark. For *namd*, the large disparity in execution time between Hetero-cg (its best core type) and Homogeneous (the best single core type for the whole workload) highlights the benefit of heterogeneity. Furthermore, *namd* is an example where AnyCore's pareto frontier (AnyCore-fg) is not entirely eclipsed by either Homogeneous or Hetero-cg. Therefore, the adaptive core provides a useful energy-delay continuum. 8 of the 15 benchmarks fall into the same class as *namd*, wherein the adaptive core is not entirely eclipsed, despite its significant cycle time and energy overhead with respect to fixed cores. The benefits of fine-grain adaptivity compensate for the adaptive core's cycle time and energy tax.

On the other hand, for the other 7 benchmarks, the benefits of fine-grain adaptivity are insufficient to overcome the adaptive core's cycle time and energy tax. The *mcf* benchmark is a representative example. Its energy and delay are plotted in the graph of Figure 7b. For one thing, coarse-grain heterogeneity is of little

value for *mcf*: its chosen core type (Hetero-cg) *is* the homogeneous core type (Homogeneous). This fixed core fully eclipses AnyCore's pareto frontier. This negative result highlights the importance of accounting for the cycle time and energy tax of within-core adaptivity, an aspect that has been largely ignored in past work. It also highlights the need for research on reducing the logic overhead of within-core adaptivity.

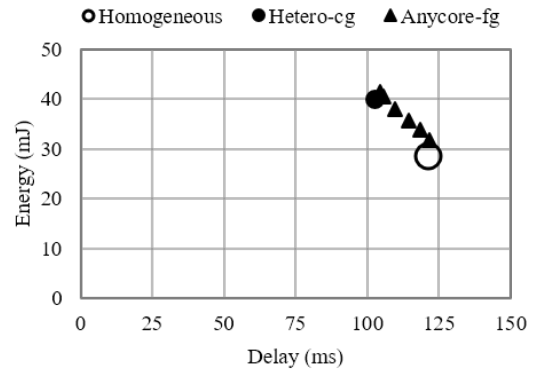
The graph in Figure 7c plots average energy versus average delay for all benchmarks. A zoomed-in version is shown in Figure 7d. The average looks like *mcf*: AnyCore-fg is eclipsed by Homogeneous. Several low-IPC benchmarks, including *mcf*, contribute disproportionately to average delay, benefit little from fine-grain adaptivity, and are sensitive to the cycle time tax of the adaptive core. The graph shows a fourth processor design, Hybrid, comprised of two core types: the homogeneous core and the adaptive core. The seven benchmarks, for which AnyCore-fg is eclipsed, run on the homogeneous core. The other eight benchmarks run on the adaptive core with fine-grain switching. Hybrid is robust in that it provides a single best fixed core and an adaptive core for accelerating benchmarks with exploitable fine-grained phase behavior. Such a Hybrid would not be conceived in the context of prior work that largely ignores logic overheads of reconfigurable cores.

6. Use-Case 3: AnyCore Chip

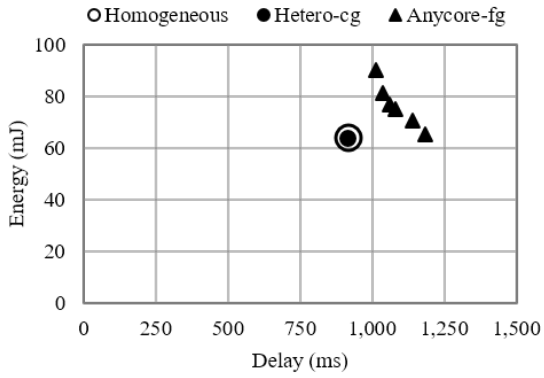
As a third use-case, we fabricated a chip that implements the same adaptive core shown in Figure 2 (maximum fetch and issue widths of 4 and 5, respectively). The core includes L1 instruction and data caches. All memories, including the L1 caches and branch predictor structures, are synthesized (standard-cell based RAMs). Unlike the first use-case, which used power-gating, the chip implements clock-gating of inactive lanes and partitions. Static power is negligible in the older foundry process. Moreover, the older standard cell library does not have the built-in power-gating support that is present in newer standard cell libraries, rendering it incompatible with physical design flows that exploit UPF and its peer CPF (Common Power Format).

The physical design data, final layout, floorplan, and printed circuit board (PCB) are shown in Figure 8. The chip is 25 mm^2 in an IBM 130nm process and has 100 pads, 79 of which are signal pads. The 79 signal pads primarily consist of dedicated instruction and data buses (for cache miss handling), a debug bus for directly interacting with the chip, such as configuring the adaptive core, directly reading/writing the caches, *etc.*, and clock, reset, and assorted control signals. The chip is packaged in a CQFP-100 package (100 pins, 25 on each side) and the package is housed in a compatible 100-pin socket on a custom-designed 4-layer PCB. The package and socket can be seen in the top-side image of the PCB. The bottom-side, not shown, has a standard FMC LPC connector and other passive components (decoupling capacitors, power measurement resistor). The LPC connector mates the PCB to an FPGA board (we currently use the Xilinx ML605). The FPGA is programmed with a testbench that services cache miss requests and manages the debug bus.

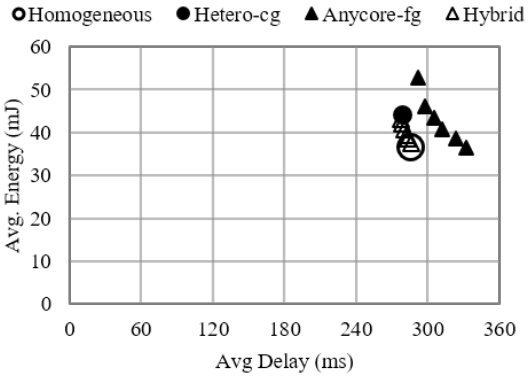
The chip has a built-in "liveness test", the results of which



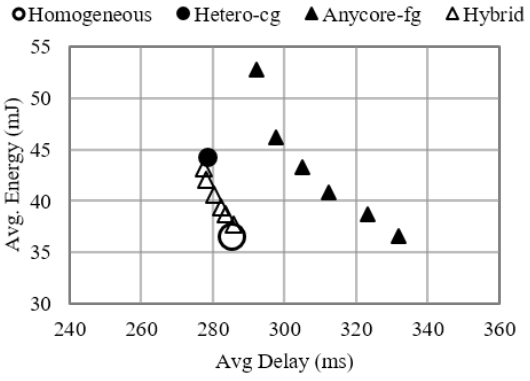
(a) namd



(b) mcf



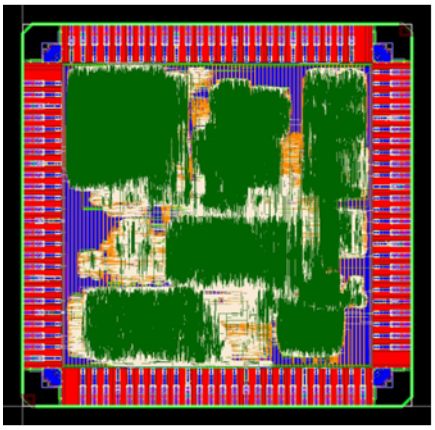
(c) average



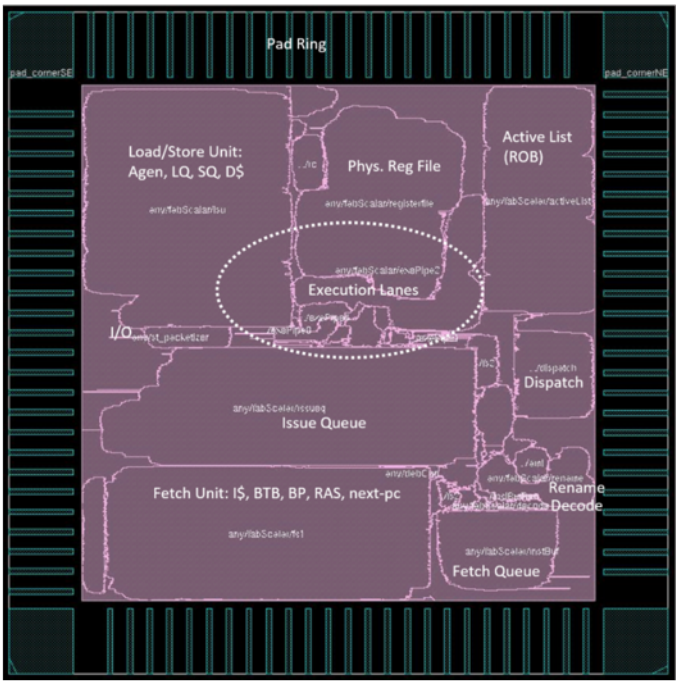
(d) average-zoomed

Figure 7: Adaptive vs. Hetero.

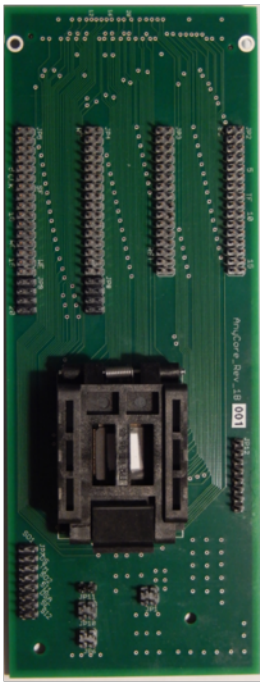
Physical design data	
Technology	IBM 8RF (130nm)
Dimensions	5 mm x 5 mm
Area	25 mm^2
Pads (signal, power)	100 (79, 21)
Transistors	3.4 million
Cells	1.5 million
Nets	7.6 million



Final Layout



Floorplan



PCB

Figure 8: AnyCore chip and printed circuit board.

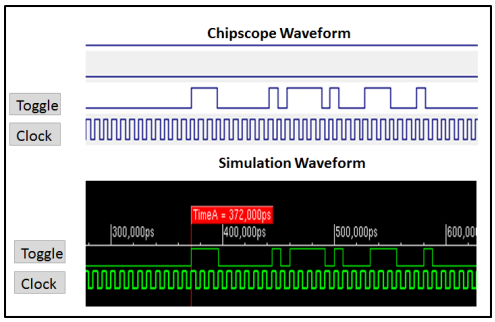


Figure 9: Chip liveness test.

are shown in Figure 9. This test is faciliated by three features. First, the caches can be configured as either cache or scratchpad, controlled either by a configuration register or a dedicated pin. Second, a chip reset initializes the top several rows of the caches (presuming scratchpad mode) to a test program. Third, the test program includes several "toggle" instructions co-mingled with various other instructions of all types (loads, stores, ALU ops, branches). This is a custom instruction added for the sole purpose of toggling a dedicated pin. To initiate the liveness test, the clock is started, the pin for scratchpad mode is asserted, and reset is asserted. We monitor the toggle pin using Xilinx ChipScope ILA™ (Integrated Logic Analyzer in the FPGA). The top waveform is a screen capture of the toggle pin of the chip (from ChipScope). It matches the bottom waveform obtained from simulation. We ran other microbenchmarks, loaded through cache misses, to test functionality. Functional testing is on-going and no bugs or defects have been observed so far.

Figure 10 shows measured IPC and current (mA) for a high-IPC and long-running microbenchmark that repeatedly sums elements of an array. Measurements were taken for the three different core configurations shown in Table 3 with the expectation that IPC and current would both increase monotonically with superscalar width and size. The measurements bore this trend. IPC is very close to 1 and 2 for the 1-wide and 2-wide configurations, and just under 3 for the 4-wide configuration. We hypothesize, but have not yet investigated, that the innermost taken loop-branch is limiting average fetch bundle size below 4 and/or data dependencies limit IPC below 4. The current increases linearly from 22 mA to 31 mA.

Table 3: Three configurations for current measurements.

Parameter	conf. 1	conf. 2	conf. 3
Fetch Width	1	2	4
Issue Width	3	4	5
Issue Queue	32	64	64
Load/Store Queues	32/32	32/32	32/32
Phys. Register File	96	128	128

The first configuration's current consumption (22 mA) seems disproportionate with respect to the current differential between it and the maximally configured core. The reason is that a lot of power is consumed by the clock tree and in particular the many D flip-flop sinks in the synthesized caches. Table 4 shows results of our "idle power tests". In all four tests, the dedicated *stall_fetch* input pin is asserted to stall the fetch stage so that no

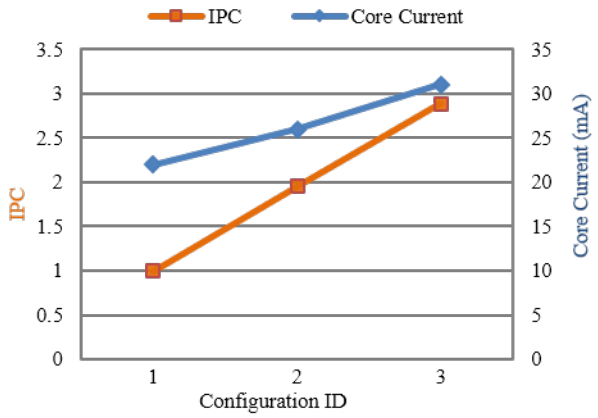


Figure 10: Chip measurements: IPC and current for three configurations.

new fetch bundles are brought into the pipeline. In general use, the FPGA uses this control signal and the debug bus in the protocol for reconfiguring the core, but for the idle power tests it is used simply for fetch gating indefinitely. The four tests cover the minimum and maximum configurations with the clock disabled and enabled. With the clock disabled, power is near-zero for both the minimum and maximum configurations. With the clock enabled, power is 17 mA and 20 mA for the minimum and maximum configurations. This is solely clock tree power (owing to fetch gating the core), including the clock routing, clock buffers, and limited internal switching in the D flip-flop sinks. The 3 mA delta in clock tree power gives a sense of the number of clock-gated flip-flops between the minimum and maximum configurations, and its contrast with the 17 mA baseline also gives a sense of the large contribution of the always-clocked instruction and data caches. Although we cannot precisely quantify their contribution (because we cannot clock-gate the minimal pipeline configuration), a practical estimate is about 16 mA (since 1-wide to 4-wide exhibits a clock power delta of 3 mA, let's assume 1 mA if the 1-wide configuration were clock-gated). Therefore, to implicitly factor-out the synthesized caches' contribution to clock tree power, one can view the current measurement graph as starting at about 16 mA.

Table 4: Idle power tests.

stall_fetch	clock	config.	current (mA)
true	disabled	minimum	0.01
true	disabled	maximum	0.01
true	enabled	minimum	17
true	enabled	maximum	20

7. Summary

This paper presented AnyCore, a synthesizable RTL model of an adaptive superscalar core, and the AnyCore Toolset for effectively using it. A key advantage of an RTL-based tool, over conventional analytical frameworks, is the ability to account for logic overheads of a reconfigurable core with respect to fixed cores in its configuration space. Moreover, it enables doing this accounting in the context of a whole core and commercial/open standard cell libraries. In the era of specialization, where the objective is to eke out the most performance and energy efficiency from silicon, measuring, understanding, accounting for, and ultimately reducing logic overheads is critically important.

The AnyCore RTL model is a single Verilog description with rich static configuration options. The user can synthesize (1) fixed and adaptive cores, (2) fixed/adaptive cores of different maximum widths and sizes, and (3) adaptive cores with clock-gating or power-gating. And, of course, a given adaptive core is dynamically configurable within its maximum dimensions. The AnyCore Toolset comes with a Synthesis and Analysis Flow for low-level power estimation (crucial for measuring power overheads of reconfigurable logic) and the AnyCore PAT tool for automatically generating per-stage/per-lane power, area, and timing databases for use with C++ cycle-level simulators.

In this paper, we demonstrated three use-cases: measuring logic overheads of an adaptive core with respect to fixed cores, comparing an adaptive core to a heterogeneous multi-core for different configuration/migration granularities, and fabrication of a prototype adaptive core.

The AnyCore Toolset is a collection of open-source software, gateware, and scripts, and is available for download from North Carolina State University [4].

8. Acknowledgments

This research was supported by NSF grant CCF-1018517 and a gift from Qualcomm. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] "Synopsys worldwide university program," available at www.synopsys.com/community/universityprogram/Pages/default.aspx. [Online]. Available: www.synopsys.com/community/universityprogram/Pages/default.aspx

[2] "Ieee standard for design and verification of low power integrated circuits," pp. 1–218, 2009, iEEE Std 1801-2009.

[3] "Ieee standard for design and verification of low-power integrated circuits," pp. 1–348, 2013, iEEE Std 1801-2013 (Revision of IEEE Std 1801-2009).

[4] "Anycore toolset," 2016. [Online]. Available: http://people.engr.ncsu.edu/ericro/research/anycore.htm

[5] D. H. Albonesi, "Dynamic ipc/clock rate optimization," in *Computer Architecture, 1998. Proceedings. The 25th Annual International Symposium on*, 1998, pp. 282–292.

[6] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," in *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on*, 1999, pp. 248–259.

[7] D. H. Albonesi, R. Balasubramonian, S. G. Dropsbo, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster, "Dynamically tuning processor resources with adaptive processing," *Computer*, vol. 36, no. 12, pp. 49–58, 2003.

[8] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for risc-v," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, Aug 2014. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html

[9] R. I. Bahar and S. Manne, "Power and energy reduction via pipeline balancing," in *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, 2001, pp. 218–229, iD: 1.

[10] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, "Reducing the complexity of the register file in dynamic superscalar processors," in *Microarchitecture, 2001. MICRO-34. Proceedings. 34th ACM/IEEE International Symposium on*, 2001, pp. 237–248.

[11] R. Basu Roy Chowdhury, A. K. Kannepalli, and E. Rotenberg, "Fabscalar-risc-v," 2nd RISC-V Workshop, 2015.

[12] K. Bhanushali and W. R. Davis, "Freepdk15: An open-source predictive process design kit for 15nm finfet technology," in *Proceedings of the 2015 International Symposium on Physical Design*. ACM, 2015, pp. 165–170.

[13] D. Burger, T. Austin, and S. Bennett, "Evaluating future microprocessors: The simplescalar toolset," University of Wisconsin-Madison, Tech. Rep. CS-TR-1308, 1996.

[14] A. Buyuktosunoglu, T. Karkhanis, D. H. Albonesi, and P. Bose, "Energy efficient co-adaptive instruction fetch and issue," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, 2003, pp. 147–156.

[15] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "FabScalar: Composing Synthesizable RTL Designs of Arbitrary Cores Within a Canonical Superscalar Template," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA-38, June 2011, pp. 11–22.

[16] E. Chun, Z. Chishti, and T. N. Vijaykumar, "Shapeshifter: Dynamically changing pipeline width and speed to address process variations," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 41. Washington, DC, USA: IEEE Computer Society, 2008, pp. 411–422. Available: http://dx.doi.org/10.1109/MICRO.2008.4771809

[17] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott, "Integrating adaptive on-chip storage structures for reduced dynamic power," in *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '02. Washington, DC, USA: IEEE Computer Society, 2002, p. 141. Available: http://dl.acm.org/citation.cfm?id=645989.674326

[18] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, "A reconfigurable chip multiprocessor architecture to accommodate software diversity," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–6, iD: 1.

[19] K. Khubaib, M. A. Suleman, M. Hashemi, C. Wilkerson, and Y. N. Patt, "Morphcore: An energy-efficient microarchitecture for high performance ilp and high throughput tlp," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, 2012, pp. 305–316.

[20] C. Kim, S. Sethumadhavan, M. S. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. W. Keckler, "Composable lightweight processors," in *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, 2007, pp. 381–394, iD: 1.

[21] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar, "Reducing peak power with a table-driven adaptive processor core," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009, pp. 189–200, iD: 1.

[22] J. Koppanalil, P. Ramrakhyani, S. Desai, A. Vaidyanathan, and E. Rotenberg, "A case for dynamic pipeline scaling," in *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '02. New York, NY, USA: ACM, 2002, pp. 1–8. Available: http://doi.acm.org/10.1145/581630.581632

[23] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: the potential for processor power reduction," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, 2003, pp. 81–92, iD: 1.

[24] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, 2004, pp. 64–75, iD: 1.

[25] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '06. New York, NY, USA: ACM, 2006, pp. 23–32. Available: http://doi.acm.org/10.1145/1152154.1152162

[26] S. Lopez, O. Garnica, D. H. Albonesi, S. Dropsho, J. Lanchares, and J. I. Hidalgo, "Adaptive cache memories for smt processors," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, 2010, pp. 331–338.

[27] P. Petrica, A. M. Izraelevitz, D. H. Albonesi, and C. A. Shoemaker, "Flicker: a dynamically adaptive architecture for power limited multicore systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 13–23. Available: http://doi.acm.org/10.1145/2485922.2485924

[28] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," in *Microarchitecture, 2001. MICRO-34. Proceedings. 34th ACM/IEEE International Symposium on*, 2001, pp. 90–101.

[29] M. Pricopi and T. Mitra, "Bahurupi: A polymorphic heterogeneous multi-core architecture," *ACM Trans.Archit.Code Optim.*, vol. 8, no. 4, pp. 22:1–22:21, jan 2012. Available: http://doi.acm.org/10.1145/2086696.2086701

[30] P. S. Ramrakhyani, "Dynamic pipeline scaling," Master's thesis, 2003, iD: NCSU1627499; Formats: Theses and Dissertations; x, 57 p. : ill.; M2: OCLC Number: 52407662; Includes bibliographical references (p. 55-57).; Includes vita.; Thesis (M.S.)–North Carolina State University. Available: http://www2.lib.ncsu.edu/catalog/record/NCSU1627499;Getanonlineversion(NCSUonly)(http://www.lib.ncsu.edu/resolver/1840.16/2106)

[31] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," *Tech. Report HPL-2008-20, HP Labs*, 2008.

[32] T. A. Shah, "Fabmem: A multiported ram and cam compiler for superscalar design space exploration," Master's thesis, 2010. Available: http://repository.lib.ncsu.edu/ir/handle/1840.16/5999

[33] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "Freepdk: An open-source variation-aware design kit," in *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*. IEEE, 2007, pp. 173–174.

[34] D. Tarjan, M. Boyer, and K. Skadron, "Federation: Repurposing scalar cores for out-of-order instruction issue," in *Proceedings of the 45th Annual Design Automation Conference*, ser. DAC '08. New York, NY, USA: ACM, 2008, pp. 772–775. Available: http://doi.acm.org/10.1145/1391469.1391666