IEEE Open Journal of
Circuits and Systems

Received 6 September 2024; revised 7 November 2024; accepted 7 December 2024. Date of publication 16 December 2024;
date of current version 6 January 2025.

Digital Object Identifier 10.1109/OJCAS.2024.3518110

Pasithea-1: An Energy-Efficient Sequential Reconfigurable Array With CPU-Like Programmability

TOBIAS KAISER[1] (Graduate Student Member, IEEE), ESTHER GOTTSCHALK[2], KAI BIETHAHN[1], AND FRIEDEL GERFERS[1] (Senior Member, IEEE)

[1]Chair of Mixed Signal Circuit Design, Technische Universität Berlin, 10623 Berlin, Germany
[2]Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut, 10587 Berlin, Germany

This article was recommended by Associate Editor J. Viraraghavan.

Corresponding author: T. KAISER (e-mail: kaiser@tu-berlin.de)

ABSTRACT This work presents Pasithea-1, a coarse-grained reconfigurable array (CGRA) that combines energy efficiency with CPU-like programmability. Its extensible instruction set uses sequential control flow in code fragments of up to 64 RISC-like instructions, which encode control and dataflow graphs in adjacency lists. Combined with dedicated, uniform processing elements, this enables fast compilation from C source code (1.4 s mean compile time). Demonstrator measurements reveal energy efficiency of up to 601 int32 MIPS/mW at 0.59 V and performance of up to 148 MIPS at 0.90 V. Compared to a RISC reference system, mean energy efficiency is improved by 2.24× with 1.71× higher execution times across 12 of 14 benchmarks. Program-dependent factors underlying variations in energy efficiency are identified using dynamic program analysis. To reduce operand transfer energy, seven interconnect topologies are evaluated: a flat bus, five crossbar variants and a logarithmic network. Best results are obtained for a crossbar topology, reducing mean dynamic tile energy by 19 %. Furthermore, floating-point (FP) support is added to the instruction set and evaluated using three binary-compatible microarchitectures, presenting distinct area-performance-energy tradeoffs. The interconnect and FP microarchitecture explorations demonstrate that, unlike CGRAs utilizing low-level bitstreams, Pasithea's instruction set hides microarchitectural details, which makes it possible to optimize hardware without severing binary compatibility.

INDEX TERMS Computer architecture, reconfigurable architectures, microprocessor chips, energy efficiency, code generation, on-chip interconnection networks, floating-point arithmetic.

I. INTRODUCTION

THERMAL limits, battery size and energy costs restrict computing systems in all application domains, making energy efficiency a paramount design criterion. Von Neumann CPUs, while convenient to program, are limited in energy efficiency. Under today's constraints of Post-Dennard scaling, technological advancements in energy efficiency no longer keep pace with increasing logic densities [1]. Thus, new architectures are needed.

One such approach for future general-purpose computing is coarse-grained reconfigurable arrays (CGRAs) [2], which are "interconnected network[s] of configurable logic and storage elements" [3] with word-level logic, storage and interconnect primitives. Through massive spatial parallelism, CGRAs can achieve high performance in applications that can be suitably parallelized. Aside from this, their construction has unique advantages in energy efficiency over Von Neumann CPUs: Instead of relying on continuous instruction fetching and decoding, configuration data can be locally retained for multiple executions, which saves energy *(instruction reuse)*. Additionally, array units exchange data locally (*spatial dataflow*), which bypasses the indirection of a global CPU register file and reduces energy use further.

CGRAs have been successfully employed as accelerators for regularly structured compute-intensive kernels, such as in digital signal processing [4], [5], [6], image processing [6], [7], [8] and machine learning applications [7]. They have also been proposed as general-purpose CPU replacement [3], [9], [10], but broad adoption has failed to emerge. This can be attributed to challenges in programmability [10], [11], [12]. We highlight the following **programmability and extensibility** characteristics, which supported the proliferation of CPUs, but are often lacking in CGRAs:

1) support for the *C programming model* and source code, enabling the reuse of existing source code and programming methods,
2) *fast code generation*, enabling quick development cycles for large and complex programs,
3) *hiding microarchitectural features from the instruction set (ISA/μarch decoupling)*, which makes it possible to improve microarchitectures for performance, power or area while preserving machine code compatibility, and
4) *instruction-set extensibility*, which makes it possible to adapt the architecture to specific task domains without breaking machine code compatibility.

This paper demonstrates how these **programmability advantages of CPUs** can be reconciled with **energy efficiency advantages of CGRAs**. For this purpose, the architecture **Pasithea-1** is introduced, evaluated and extended. Unlike CGRAs in which instructions run in parallel, Pasithea-1 follows a **sequential execution model**. This limits performance, but facilitates CPU-like programmability and extensibility. At the same time, it advances beyond CPU energy efficiency through instruction reuse and spatial dataflow.

From a software perspective, Pasithea-1 is designed to **seamlessly replace a CPU-based microcontroller**. It is programmed in C without architecture-specific annotations, which makes it easy and affordable to adopt in existing and new software projects.

The base architecture of Pasithea-1, its code generation and first silicon measurement results were presented at prior conferences [13], [14]. This paper extends this research with following **novel contributions**:

1) To reduce energy of operand transfers, **seven interconnect networks** are evaluated. For ISA/μarch decoupling, machine code compatibility is preserved.
2) A **floating-point** instruction-set extension is proposed and implemented in three binary-compatible microarchitectural variants, which provide different area-energy-performance tradeoffs.
3) Using an **extended set of benchmarks** and dynamic program analysis, **task-dependent factors** determining the energy efficiency of Pasithea-1 are investigated.
4) By **correlating silicon and simulation results** of the base architecture, the validity of energy efficiency predictions for new design variants is substantiated.

The paper is structured as follows: Section II introduces related work. The Pasithea-1 instruction set and microarchitecture including new floating-point variants and interconnect topologies are described in Section III. Section IV covers physical design and the silicon prototype. The C compiler is described in Section V, followed by benchmarking, measurement and simulation methodology in Section VI. Results are shown in Section VII. Section VIII concludes the paper.

II. RELATED WORK

A key distinction in CGRA design is whether individual PEs time-multiplex instructions on a cycle-to-cycle basis (shared PEs) or are only assigned a single instruction (dedicated PEs) [15]. With **dedicated PEs**, instructions remain stationary throughout the execution of a mapped subtask. This increases energy efficiency and simplifies code generation [12]. Pasithea-1 is a dedicated-PE architecture. Other examples are Tartan [9], BilRC [5], HyCUBE [6], [16], Plasticine [17], RipTide [10] and Amber [7]. In contrast to this, **shared-PE** architectures [8], [18], [19], [20] facilitate greater utilization of logic resources. Statically scheduled shared-PE architectures share many properties of VLIW processors [21].

The spatial design principles underlying CGRAs have also been applied to CPU design. **In-place processors** execute CPU machine code, which is based on **sequential control flow**, distributing instructions spatially. Ultrascalar [22] and CRIB [23] pioneered this approach to reduce multi-issue CPU control overhead and to maximize instruction-level parallelism. DiAG [24] extended the approach by thread pipelining and instruction reuse: Instructions are kept stationary for multiple executions, which avoids repetitive fetching and saves energy. Like these architectures, Pasithea-1 follows a sequential control flow model. In contrast to them, it uses a custom instruction set that exposes spatial characteristics of the computing fabric. This avoids the need to map general-purpose register references to spatial resources on the fly and thus enables a significantly simpler interconnect network.

Many CGRAs [8], [10], [15], [17], [19] as well as in-place processors [22], [23], [24] are furthermore categorized as dataflow architectures: Analogous to classical dataflow architectures [25], they execute instructions based on operand availability. In this narrow sense, Pasithea-1 is not a dataflow architecture. Nevertheless, like classical dataflow architectures, its instructions directly encode edges of the dataflow graph, and operand values are linked to instructions, omitting the logical indirection of a central register file.

A. INTERCONNECT NETWORKS

CGRAs vary in the organization of their interconnect networks. Most notable are mesh [5], [6], [7], [9], [10] and linear topologies [23], [24], [26], [27]. Less commonly,

bus [3] and crossbar [4] networks are found. To reduce the delay of a linear network, Ultrascalar proposes a logarithmic topology [22]. Using ADRES [18] as template, [28] evaluated the energy and performance impact of various mesh and bus interconnects. In contrast to the interconnect exploration in this work, ADRES machine code must be recompiled for each interconnect network modification.

On top of this, architectures vary in whether interconnect resources are statically assigned to specific data sources [7] or dynamically shared [5], [10]. Furthermore, some architectures provide pipeline registers as part of the interconnect [5], [7], while others omit them to optimize for energy efficiency rather than performance [10]. In Pasithea-1, interconnect resources are dynamically shared and not pipelined.

B. COMPILING FOR CGRAS

Most CGRAs use custom instruction sets (bitstream formats) and require machine code to be generated ahead of time. While some support the C programming model [10], [18], others require the use of custom domain-specific programming languages [5], [7]. Spatial and temporal aspects impose unique challenges and opportunities on CGRA compiler design, including placement/binding, routing and scheduling problems. This typically requires compilers to use computationally demanding algorithms [12] such as simulated annealing [5], [7]; integer linear programming, SAT solving [10]; SMT solving, the conjugate gradient method and A* search [7]. The associated complexity and time overhead contribute to the programmability barriers seen in CGRAs.

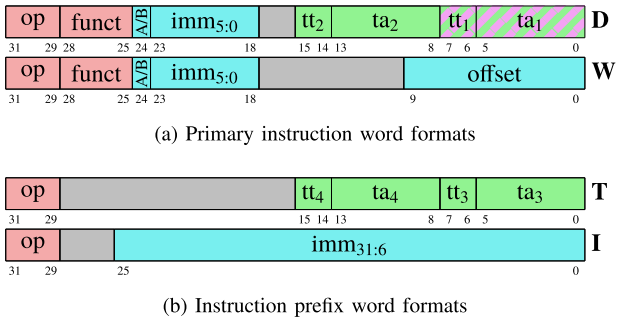
To simplify programming, some CGRAs run CPU machine code using dynamic binary translation (DBT). Reference [26] and MuTARe [27] perform DBT in hardware to offload instruction sequences to a CGRA that is integrated into a CPU datapath. Using configuration caches, time and energy is saved on repeated executions. An alternative approach is to perform DBT in software [29], which enables more complex optimizations and decouples the CGRA from the CPU datapath.

III. ARCHITECTURE

A. INSTRUCTION SET

Pasithea's instructions operate on 32-bit integers. Groups of up to 64 instructions form *code fragments*, which are loaded contiguously into CGRA fabric and act as machine-level functions. Unlike in typical CGRAs, instructions within a fragment are executed using sequential control flow.

Each instruction encodes fragment-local dataflow and control flow using up to four *target instruction pointers* (TIPs). By directly referencing fragment-local instructions, TIPs facilitate spatially distributed execution. In contrast to RISC CPUs, where instructions obtain inputs from a global set of general-purpose registers, Pasithea instructions obtain input operands from two fixed local operand registers *opA* and *opB*. Through dataflow TIPs, instructions write



(a) Primary instruction word formats

(b) Instruction prefix word formats

FIGURE 1. Pasithea-1 instruction encoding. Color legend: op/funct codes, dataflow TIP, control flow TIP, immediates. Adapted from Kaiser [13, Fig. 1].

their results to operand registers of other instructions. This way, dataflow TIPs encode the dataflow graph (DFG) of a fragment in the form of adjacency lists. Additionally, TIPs encode local control flow in the form of conditional branches.

Fig. 1 summarizes instruction encoding. Instructions consist of a primary instruction word (Fig. 1(a)) with optional prefix words (Fig. 1(b)). D type instructions produce results that can be written to other fragment-local operand registers or used as branch condition. W type instructions, e.g., *sw* (store word), do not produce such results. In the primary D type instruction word, two TIPs can be encoded; two more TIPs are available through the optional T prefix word.

Each TIP consists of a target address ta, referencing any instruction within the same fragment, and a target type (tt). The first TIP (tt_1, ta_1) is mandatory and can encode either a conditional branch (control flow TIP) or a write to an operand register (dataflow TIP). The remaining three TIPs are optional and can only encode operand register writes.

Instructions also comprise an immediate value for initialization of either opA or opB. Arbitrary 32-bit immediates are enabled through the I prefix word. When it is absent, the 6-bit immediate of the primary instruction is sign-extended.

Like subroutines, fragments can be invoked multiple times. Each invocation creates a new *fragment instance* (FI). FIs of a particular fragment share the same machine code but can differ in runtime data (operand register values, control flow state), enabling reentrancy.

Each FI possesses eight logical message registers through which it can receive data (such as function arguments or return values) from other FIs. Every FI is identified by a unique FI address. By passing messages between FIs, inter-fragment dataflow and control flow is established.

Table 1 lists all instructions of Pasithea-1, including new floating-point (FP) instructions. ALU and load/store instructions borrow basic semantics from RISC-V [30]. FIs can exchange data words, such as function arguments and return values, using message passing instructions. To invoke and terminate FIs, FI management instructions are provided.

B. CONCURRENCY AND EVENT ORDERING

While execution within FIs is sequential, coexisting FIs are concurrent by default. Microarchitectures are thus free to

TABLE 1. Pasithea-1 instruction set overview.

ALU	FP extension	Load/store	Msg. passing	Manage FIs
or, and, xor, add, sub, slt, sltu, sll, srl, sra	fadd, fsub, fmul, fcmp, fmin, fmax	lw, lh, lb, lhu, lbu, sw, sh, sb	send, recv	inv, term
Results depend only on opA and opB; no side effects.		*Access global memory state.*	*Blocking FI-to-FI communication.*	*Create and terminate FIs.*

execute coexisting FIs in parallel. Event ordering can be enforced using send and receive instructions: If the requested message register is empty, the receiving FI is paused until a message with the requested identifier is received.

When an FI is used as subroutine, the newly created callee FI initially waits for argument values. Later, the caller FI waits for a return message from the callee FI before continuing. This message serves as completion indicator and also transmits the return value, where applicable.

In this work, all fragments are subroutines, and thus all FIs are subroutine calls. We have previously demonstrated the possibility of using FIs as coroutines or lightweight threads [13]. To reconcile these inherent multithreading capabilities with high-level language constructs and code generation, further research is needed.

C. BASE MICROARCHITECTURE

Fig. 2 shows the microarchitecture of Pasithea-1. A 2 × 4 array of tiles makes up the CGRA fabric. Each tile comprises 16 processing elements (PEs), each serving one instruction. A fragment of 16 or fewer instructions can be loaded into a single tile. For fragments of up to 32/64 instructions, 2/4-tile groups are formed from adjacent tiles as shown in Fig. 2(a).

The **S bus** connects PEs among each other and to the *tile interface node* (TIN). T links connect S buses of adjacent tiles if they are part of the same large fragment (> 16 instructions). At its core, the S bus is a 32-bit data bus with additional control signals that is used to (1) load instructions into PEs, (2) send results from PEs to operand registers of other PEs, (3) invoke/terminate FIs or send/receive messages through the TIN, (4) perform load/store operations through the TIN, (5) evict FI runtime state to memory or restore FI runtime state from memory and (6) locally coordinate control flow using a network of execute-enable signals. Using these execute-enable signals, a control flow token is passed around among PEs and the TIN. This mechanism fulfills the task of the CPU program counter in a spatially distributed manner.

Each **PE**, shown in Fig. 2(c), contains two operand registers, opA and opB, and an instruction register (IR) for a primary instruction word and optional I and T prefixes. IR, opA and opB are written through the S bus. Execution is triggered by an incoming execute-enable signal of the S bus. The control unit, implemented as finite state machine (FSM), coordinates the process of execution. For load/store, message passing and FI management instructions, this includes communication with the TIN. Upon completion, the PE uses the S bus to write the result word to operand

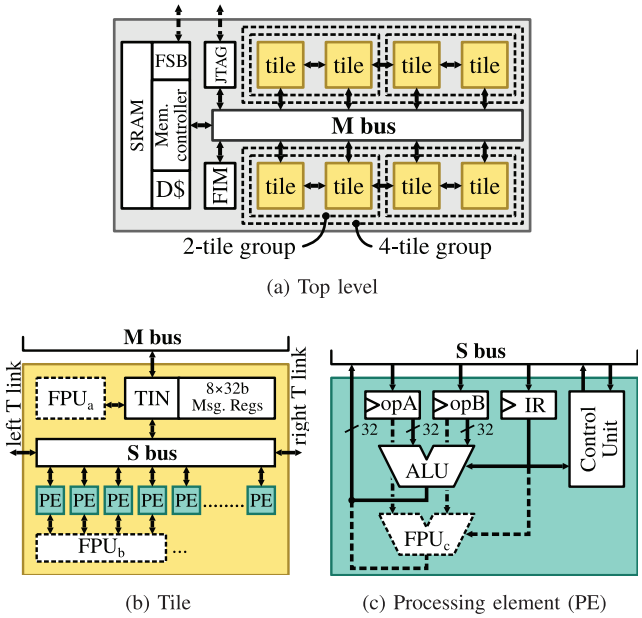


FIGURE 2. Pasithea-1 microarchitecture. No FPU is present in the base microarchitecture. FPU$_{a/b/c}$ are mutually exclusive. Adapted from Kaiser [14, Fig. 1].

registers of other PEs, as requested by its TIPs. Concurrently, the control unit hands control flow over to either the next instruction in sequence or to a branch target, in case a control flow TIP is present and the encoded branch condition is satisfied.

The **M bus** connects all tiles, the memory controller and the *fragment instance manager* (FIM). In addition to FI-to-FI communication, the M bus fulfills load/store operations, transfers machine code from memory to fabric and is used for evict/restore operations. The **TIN** is situated between M bus and S bus and manages execution of the FI contained in a tile or linked tile group through an FSM. The TIN also contains the eight message registers for inter-FI communication.

When an FI terminates, its machine code remains loaded as a **residual fragment**. Subsequent fragment invocations can reuse such residual fragments, enabling instruction reuse across subroutine calls. A least-recently-used (LRU) queue of residual fragments is maintained in hardware.

FIs in memory: Only a limited number of FIs can reside concurrently in fabric (e.g., eight 16-instruction FIs, two 64-instruction FIs or combinations thereof). To hide this hardware limitation and allow greater numbers of FIs to coexist logically (e.g., for call stacks whose depth exceeds fabric capacity), FIs can be moved from fabric to SRAM *(evict)* and back from SRAM to fabric *(restore)*. Up to 254 FIs can be evicted to memory. Throughout the evict/restore process, FIs retain their unique addresses and the ability to receive messages for later processing.

The **FIM** supervises FI invocation, instruction fetching and FI termination. Furthermore, it initiates and coordinates the FI evict and restore processes: When all FIs in fabric are waiting for messages, the FIM detects a stall condition

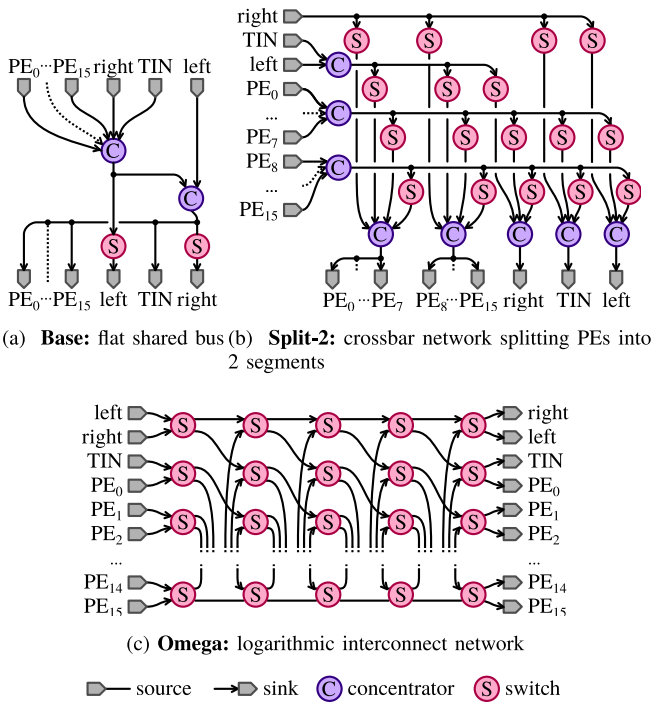
and initiates a multi-step *unstall* procedure, at the end of which the next pending FI from the ready queue is restored to fabric. In many instances, some fabric space must be freed before restoring. For this purpose, residual fragments are firstly cleared in LRU order. Secondly, when no residual fragments are present, waiting FIs are evicted in LRU order.

FIs are only evicted while waiting for a message. If a message is sent to an evicted FI, the FIM delivers it to its memory representation. If the message index is equal to the index that the FI is waiting for, the FI is added to the ready queue. If no fabric space is available for a new FI at time of invocation, the new FI is created in memory and immediately added to the ready queue.

The ready queue and a pool of unused FI addresses for off-fabric FI creation are maintained as linked lists in memory.

Notably, the **evict/restore processes are logically hidden from the executed program** (transparency). This makes it possible to add or remove hardware tiles while maintaining full machine code compatibility *(ISA/µarch decoupling)*.

The **memory controller** connects the SRAM to the M bus for loads, stores and instruction fetching. It includes a 32 B instruction buffer and a 4×32 B fully-associative data cache with LRU replacement policy. Through the front-side bus (FSB), the memory controller provides character I/O and an external memory interface.



(a) **Base:** flat shared bus (b) **Split-2:** crossbar network splitting PEs into 2 segments

(c) **Omega:** logarithmic interconnect network

FIGURE 3. Interconnect topologies used in S bus design exploration. Switch units propagate inputs conditionally (based on destination addresses); concentrator units propagate inputs unconditionally.

D. INTERCONNECT EXPLORATION

The S bus enables PE-PE and PE-TIN communication within a tile or a linked tile group. It is logically flat and fully combinational. To optimize its dynamic energy, the average switched capacitance per data transfer must be minimized. We approach this by inserting conditional switches to reduce propagation of signals to unaffected S bus endpoints. Three types of topologies are evaluated, shown in Fig. 3:

1) The **Base** microarchitecture uses the flat shared bus topology depicted in Fig. 3(a). Within 2/4-tile groups, the switches located before the *right* and *left* T link outputs confine data propagation to tiles affected by the current bus operation. Furthermore, they prevent propagation of signals between tiles that are not part of the same FI.
2) The **Split-***N* topologies reduce signal propagation by splitting PEs of a tile into *N* groups. These PE groups, TIN, and T link interfaces *left* and *right* are interconnected using a single-stage crossbar switch. Fig. 3(b) shows the Split-2 topology as an example of this. Switches for unused paths such as TIN→TIN or left→left are omitted. Design variants Split-1 (most similar to Base), Split-2, Split-4, Split-8 and Split-16 (one switch per PE) have been implemented.
3) The **Omega** logarithmic interconnect is shown in Fig. 3(c). Its topology was pioneered in the context of array processors [31]. A multi-stage switch network restricts signal propagation to paths needed for the present bus operation. Even though each stage adds switching and control overhead, an overall activity reduction is possible due to the low fan-out per switch.

In each of these networks, PE-to-PE paths within a tile have uniform lengths (number of traversed switches), which balances combinational paths to meet timing requirements. Networks with unbalanced path lengths, e.g., mesh, torus or ring topologies, were not explored due to the timing drawbacks of their longest combinational paths and their need for more complex control logic.

All design variants share the same underlying instruction set and require no machine code modification *(ISA/µarch decoupling)*.

E. FLOATING-POINT EXTENSION

Many applications, e.g., in signal processing, machine learning or computer graphics, rely on floating-point (FP) math. To enable fast FP operations, Pasithea's instruction set is extended by a set of FP instructions, shown in Table 1. The FP instructions use the IEEE 754 single-precision format and share the registers opA and opB with integer instructions.

In accord with the base instruction set, all PEs must support the FP instructions. Hardware floating-point units (FPUs) are significantly larger and more complex than integer ALUs. Hence, how they are integrated matters for power, performance and area. Three FPU integration variants are proposed, named by their number of FPUs per tile:

1) **FPU-1** adds a single FPU per tile by connecting it to the TIN, shown as FPU_a in Fig. 2(b). Multiple S bus cycles are needed to complete FPU operations.

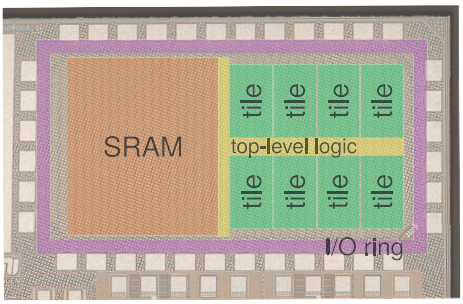


FIGURE 4. Die micrograph of the 8-tile CGRA demonstrator with layout view overlay. Area is 1228 μm × 608 μm, excluding I/O ring and pads. Reprinted from Kaiser [14, Fig. 2].

2) In **FPU-4**, groups of four PEs each share one FPU, leading to four PEs per tile. This is depicted as FPU_b in Fig. 2(b). Bypassing the S bus, a single clock cycle suffices to apply FP operands and obtain the result.
3) **FPU-16** equips each PE with one FPU, leading to 16 PEs per tile. This is shown as FPU_c in Fig. 2(c).

All three FPU variants utilize a fully combinational FPU design, which comprises an adder, multiplier, special number handler and a rounding unit.

Despite their microarchitectural differences, all three FPU variants implement the identical underlying instruction set *(ISA/μarch decoupling)*.

IV. PHYSICAL DESIGN & SILICON PROTOTYPE

A. SILICON PROTOTYPE

A silicon prototype of the base microarchitecture without FPU, with flat shared S bus and 256 kB SRAM was successfully fabricated [14] in GlobalFoundries 22 nm FD-SOI CMOS [32]. The die is shown in Fig. 4.

Its core area, including SRAM but excluding I/O, amounts to 0.75 mm^2. Considering only tile and top-level logic area, the prototype attains a fabric density of 437 PEs/mm^2.

Compared to a CPU core of equal capabilities, Pasithea utilizes more logic area but exhibits significantly less switching activity per logic area. Leakage power is thus of great concern. To ensure that leakage contributions do not nullify savings in dynamic energy, core logic was implemented with ultra-low-leakage (ULL) standard cells, which use ultra-high threshold voltage transistors and 28 nm gate-length biasing.

Synopsys Design Compiler and IC Compiler II were used for synthesis and place-and-route. To reduce tool runtime, a bottom-up hierarchical design flow was used, in which the tile was implemented as a separate unit in synthesis and layout. On top level, eight identical tiles were instantiated.

Sign-off was completed for a nominal V_{DD} of 0.8 V using the RC-extracted design with metal fill and parametric on-chip variation (POCV) cell models. Corners of 0.76 V to 0.88 V, −40 °C to 125 °C, fast and slow process and RC corners were applied. Additional hold margins and transition times constraints were specified to facilitate operation at V_{DD} below 0.76 V.

The design comprises a total of 34.2k flip-flops, of which only a small fraction is toggled in any given clock cycle. Clock gates were manually inserted at various points in the design hierarchy to limit switching to active parts of the design and restrict clock propagation to relevant clock subtrees. In total, the design uses 500 clock gating cells.

B. EXPLORATORY DESIGN VARIANTS

An exploratory design flow was used to implement and evaluate the proposed floating-point and interconnect variants. It differs from the silicon prototype's design flow in following aspects: I/Os, pad cells and JTAG TAP were removed to simplify simulation. To accommodate additional FPU and S bus logic, the floorplan was enlarged. Relaxed timing margins and transition time constraints were applied to reduce tool runtime.

Using this design flow, the following ten separate design variants were implemented and simulated: Split-1, Split-2, Split-4, Split-8, Split-16, Omega, FPU-1, FPU-4, FPU-16, Base (reimplemented as reference for comparisons).

C. RISC REFERENCE SYSTEM

To compare the architecture against energy-efficient embedded CPUs, a single-core reference system based on Ibex [33], an open-source 32-bit RISC-V CPU, was implemented. The selected configuration of Ibex supports the RV32IMC instruction set with 3-cycle multiplication.

This reference system was implemented using the same design flow as the Pasithea-1 exploratory design variants, identical technology, standard cell library and SRAM macro. At the SRAM interface, the reference system uses the same 32 B instruction buffer and 4 × 32 B data cache as Pasithea-1.

V. CODE GENERATION

While Pasithea's instruction set exposes spatial dataflow and control flow, it is also designed to minimize the effort of code generation [14]: Similarly to a CPU, it uses **sequential control flow** and provides conditional branches. This enables **arbitrary control flow structures**, such as nested loops and conditionals. In contrast to architectures that differentiate PEs into different types for specific purposes (e.g., memory access PE, ALU PE), **all PEs are functionally equivalent**. Based on this, instructions in a fragment can be mapped to **dedicated PEs** (one instruction per PE) in simple control flow order. Using dataflow TIPs and the logically flat S bus, all PEs/instructions within a fragment can freely communicate, without the need for complex routing at compile time.

Exploiting these characteristics, a C compiler backend for Pasithea based on LLVM [34] (version 15.0.6) was implemented [14] and is used in this work for compiling benchmark code. It leverages LLVM's existing frontend, middle-end and RISC-V instruction selection passes and compiles LLVM's machine intermediate representation (MIR) to Pasithea's intermediate assembly-like

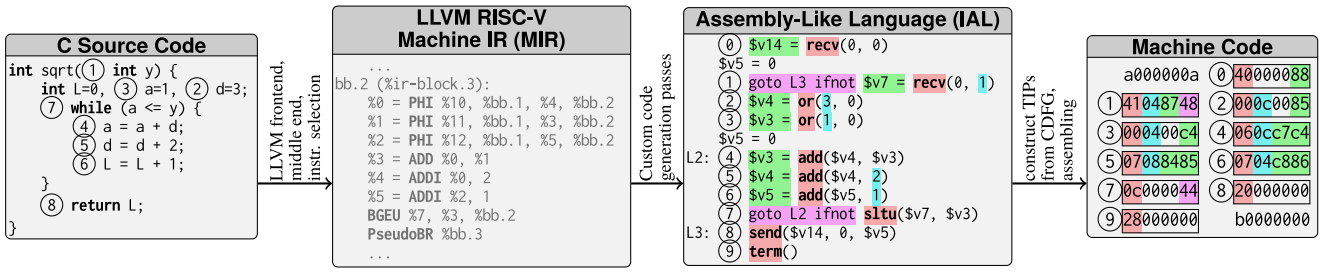


FIGURE 5. Code generation example. Color legend: op/funct codes, dataflow, control flow, immediates. Adapted from Kaiser [14, Fig. 3].

language (IAL). IAL exposes only available CGRA primitives (instructions) and represents dataflow using virtual registers. In a final step, IAL is translated to machine code.

The custom MIR-to-IAL code generation is similar to CPU code generation. It leverages conventional dataflow analysis and includes: opcode translation; redistribution and simplification of immediate operands; lowering of function calls, argument reads and return value writes; SSA elimination; constant and copy propagation; fusion of add-loads, add-stores and assignment-branches where possible; dead and redundant code elimination. For the subsequent translation of IAL to machine code, dataflow is analyzed once more to derive dataflow TIPs, as described in [13]. For IAL instructions that require more than four TIPs, supplemental copy machine instructions are inserted.

Fig. 5 shows code generation for a small example function in C. The IAL code uses virtual registers (`$v...`), which are subsequently resolved to dataflow TIPs. The caller FI address is received as `$v14` (instr. 0) and is used to send the return value `L` (`$v5`) back to the caller (instr. 8). Instruction 1 assigns the function argument `y` to `$v7` and furthermore branches to `L3` if `y` equals zero. Instructions 2 and 3 initialize C variables `a` and `d`. Instructions 4–7 implement the while loop. In the shown example, no instruction has more than two TIPs or immediate operands exceeding six bits. Therefore, no instruction prefix words are used.

This TIP-based dataflow can be characterized as *static single use* (SSU), as operand registers are used (read) exclusively by their individual associated instruction but can be assigned (written) by any fragment-local instruction. In earlier literature, SSU form is used for code optimization [35].

Present usability limitations: Currently, each C function is translated to a single code fragment, which means that functions cannot exceed 64 instructions. Large functions must therefore be manually divided into smaller functions in C code. A more comprehensive compiler should perform this step automatically. Furthermore, as Pasithea's call stack differs from a typical CPU call stack, non-static local variables can currently only be allocated in operand registers, not in stack memory.

VI. EXPERIMENTAL METHODS

A. BENCHMARKS

To evaluate performance and energy efficiency, the following 14 general-purpose benchmarks, implemented in C, are run on both Pasithea and the RISC reference system: Euclid's algorithm, Stein's algorithm, square root by linear search (shown in Fig. 5), CRC32 checksum, MD5 cryptographic hash, treesort, quicksort, heapsort, the Wagner-Fisher algorithm for Levenshtein distance [36], Huffman compression [36], Hume-Sunday string search [36], a finite impulse response filter (FIR) [36], Dijkstra's algorithm [36], and a conflict driven boolean satisfiability (SAT) solver [37]. C sources were manually adapted to work around mentioned limitations of the C compiler.

These benchmarks were selected to vary in code complexity, locality of control and dataflow, call stack usage and load/store behavior. They exclude FP operations and focus on algorithms without multiplication or division, which Pasithea does not natively support. Exceptions are FIR and Dijkstra, for which mul/div are emulated using shift-and-add subroutines on Pasithea.

Further programs were included to evaluate specific system functions in isolation. Nop and xor loops across one or two tiles were used to evoke minimal and maximal S bus activity. The memory subsystem was assessed using load/store hit/miss loops.

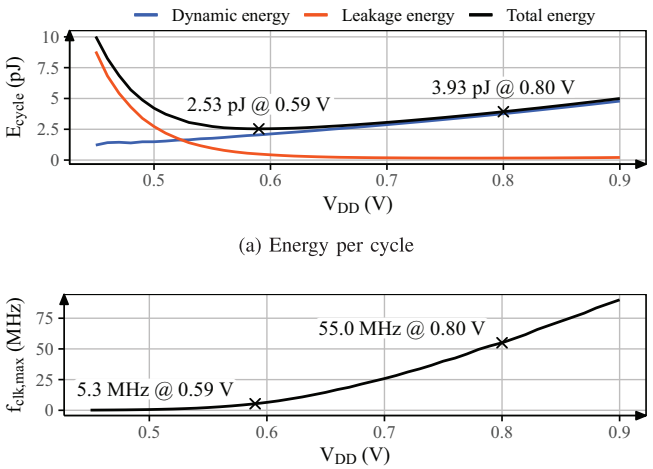
To evaluate the floating-point extension and design variants, CORDIC cosine and matrix multiplication programs were implemented in IAL.

All programs are run in loops, making full use of filled caches and reusing residual code fragments in fabric.

B. SIMULATION

Using Siemens Questa, timing-annotated post-layout netlist simulations were performed for the Pasithea-1 silicon design, the exploratory design variants, and the RISC reference system. Generated VCD activity vectors were subsequently used in time-based power analysis with Synopsys PrimeTime at nominal operating conditions ($V_{DD} = 0.8\,V$), including RC parasitics from layout with metal fill.

To investigate factors underlying switching activity in both Pasithea and the RISC reference system, execution traces and event counts were recorded for both platforms.



(a) Energy per cycle

(b) Maximum operating frequency

FIGURE 6. V_{DD} sweep measurement results for execution of *heapsort*.

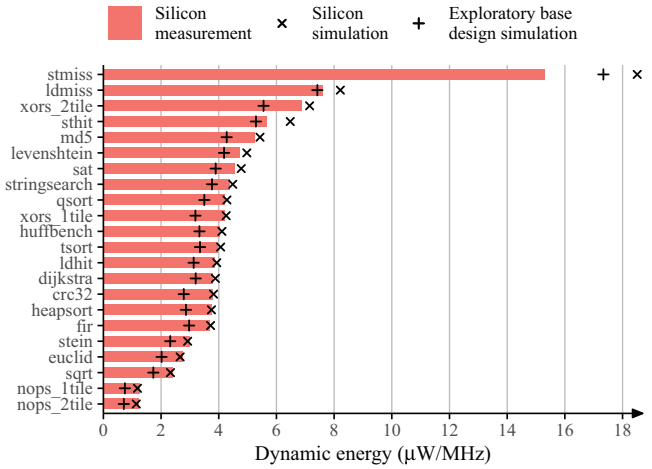


FIGURE 7. Measured and simulated dynamic energy during benchmark execution, at V_{DD} = 0.8 V.

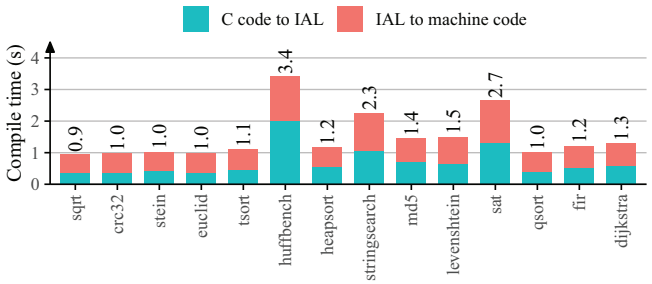


FIGURE 8. Benchmark compile times (C to Pasithea machine code) on an Intel Core i5-1135G7 machine.

C. MEASUREMENT

The silicon prototype was connected to a Xilinx Artix 7 FPGA for dynamic clock generation, programming and verification of execution results. A Keithley 236 source measure unit was used to provide the core supply voltage V_{DD} and measure the supply current. For each benchmark and V_{DD}, the maximum error-free clock frequency $f_{clk,max}$ and the supply current were recorded.

VII. RESULTS

A. SILICON PROTOTYPE

Measurements of energy and $f_{clk,max}$ for execution of *heapsort* are shown in Fig. 6. Its minimum energy point (MEP), $V_{DD} = 0.59$ V, and nominal operating point, $V_{DD} = 0.8$ V, are highlighted. $f_{clk,max}$ was found to vary significantly between benchmarks. As a result of program-dependent switching activity and $f_{clk,max}$, the MEP varies between $V_{DD} = 0.57$ V and 0.61 V across benchmarks.

Reliable operation for all benchmarks was observed between 0.51 V to 0.9 V at room temperature. For some benchmarks, correct operation was maintained down to 0.42 V.

Fig. 7 shows dynamic energy measurements for each benchmark. The measured values are compared to simulation results. Results from simulation of the fabricated design match the measurement results closely (simulation/measurement ratios yielding geom. mean = 1.02 and geom. std. dev. = 1.06).

Energy figures from base microarchitecture simulation using the exploratory design flow yielded systematically lower power figures than the silicon measurements (geom. mean = 0.83, geom. std. dev. = 1.13), due to the relaxed timing margins and transition time constraints. To account for this, exploratory design variants are subsequently only compared to other exploratory design variants.

B. BENCHMARK RESULTS

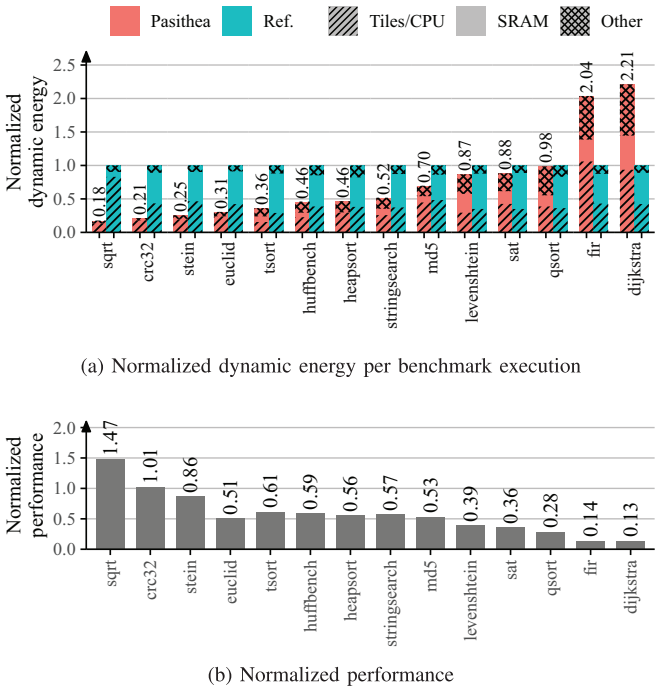
In the following, for mean values across benchmarks, the geometric mean is used.

Fig. 8 shows compile times of the C benchmarks. The mean compile time over benchmarks is 1.4 s.

At MEP, a mean energy efficiency of 171 MIPS/mW and a peak energy efficiency (*sqrt*) of 601 MIPS/mW were measured. At 0.9 V, a mean performance of 44 MIPS and a peak performance (*sqrt*) of 148 MIPS were measured.

Fig. 9 shows relative dynamic energy and performance of Pasithea-1 in comparison to the RISC reference system. Benchmarks are sorted by energy efficiency. In 12 benchmarks, Pasithea-1 surpasses the reference system in energy efficiency, while exhibiting lower energy efficiency for *fir* and *dijkstra*. Lower performances in Fig. 9(b), equalling greater execution times, coincide with greater energies in Fig. 9(a). Considerable reductions in SRAM energy due to instruction reuse are seen in all benchmarks except for *dijkstra*. The contributions marked as "other" reflect activity of the M bus, FIM and memory controller. For the reference system, "other" comprises only the memory controller.

Excluding benchmarks *fir* and *dijkstra*, Pasithea-1 provides a 2.24× higher energy efficiency at 1.71× higher execution time relative to the RISC reference system.



(a) Normalized dynamic energy per benchmark execution

(b) Normalized performance

FIGURE 9. Performance and energy of Pasithea-1, normalized to RISC reference system, simulation results.

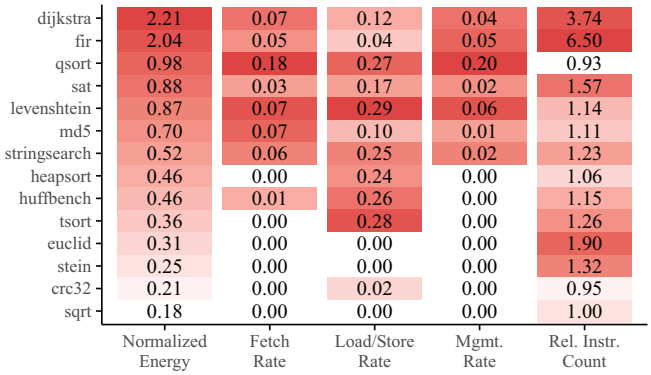


FIGURE 10. Benchmarks sorted by normalized energy with quantities from dynamic program analysis. Values are colored by rank in column.

Including all benchmarks, energy efficiency is on average 1.80× higher with a 2.11× higher execution time.

C. FACTORS DETERMINING ENERGY EFFICIENCY

Fig. 9 shows that programs benefit from CGRA execution to varying degrees: While for some programs energy is reduced by up to 5.6× (*sqrt*), others run less efficiently than on the reference CPU (*fir*, *dijkstra*). To understand what makes programs particularly energy efficient or inefficient on Pasithea, event counts from dynamic program analysis were analyzed. It was found that the major differences in energy efficiency can be explained by the quantities *fetch rate*, *load/store rate*, *management rate* and *relative instruction count*, which are visualized in Fig. 10 alongside normalized energy. In the following, we define these four quantities and trace them back to program characteristics.

The **fetch rate** is defined as instruction words fetched per executed instruction. Low fetch rates indicate frequent instruction reuse, which can be attributed to high instruction locality. The benchmarks *sqrt*, *crc32*, *stein*, *euclid*, *tsort* and *heapsort* exhibit fetch rates of zero. This means that their code fits into fabric completely, allowing subsequent loop iterations to fully reuse residual fragments and forgo all instruction fetching. As instruction fetching uses energy, higher fetch rates correspond to higher energy use.

Another energy overhead comes from evict/restore operations, which are initiated by the FIM. The **management rate** reports the number of FIM-initiated memory accesses per executed instruction. The benchmarks *sqrt*, *crc32*, *stein*, *euclid*, *tsort*, *huffbench* and *heapsort* exhibit management rates of zero, indicating that no evict/restore operations are performed. Fig. 10 shows that greater management rates are associated with greater normalized energies. Such increased management rates are caused by frequent deeply nested function calls. For example, the recursive *qsort* incurs a particularly high management rate (0.20).

The **load/store rate** quantifies the number of load/store instructions per executed instruction. High load/store rates are found to increase normalized energy, as memory requests and responses must travel long paths through S bus and M bus. The benchmarks *sqrt*, *crc32*, *stein*, *euclid* and *fir* perform comparatively few load/store operations (load/store rates below 0.04), which positively affects their energy use. The sorting algorithms *tsort*, *heapsort* and *qsort* exhibit among the highest load/store rates (0.24–0.27), due to frequent comparison and swapping of array elements.

Lastly, the **relative instruction count** quantifies the number of executed instructions on Pasithea relative to the reference system. Greater relative instruction counts are found to increase energy use. The benchmarks *fir* and *dijkstra* are found to have the highest relative instruction counts (6.50 and 3.74). This is due to their use of multiplication and division, which are emulated with shift-and-add subroutines due to Pasithea's lack of native mul/div instructions.

The preceding observations make it possible to propose **measures for further energy reduction**: To increase reuse probabilities of residual fragments and enable deeper nested calls without evict/restore operations, the number of tiles could be scaled up. We expect this to reduce fetch and management rates. Relative instruction counts could be improved by adding support for missing integer mul/div operations on PE or tile level. The observed load/store overhead could be reduced by optimizing the TIN or M bus or by introducing distributed data caches.

D. INTERCONNECT VARIANTS

The impact of S bus topologies on energy efficiency and area was explored using the base, Split-1/2/4/8/16 and Omega design variants. The variants do not differ in performance, as

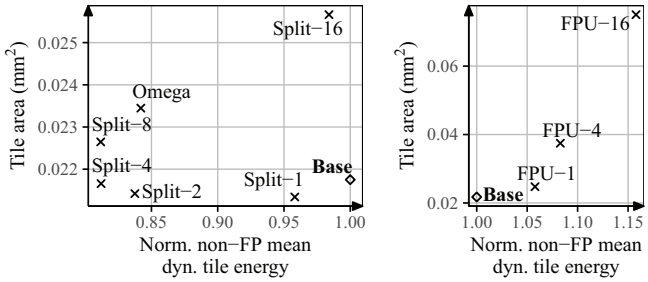


FIGURE 11. Impact of interconnect design variants (left) and FP design variants (right) on tile area and dynamic tile energy (mean value across all non-FP benchmarks, normalized to base variant), simulation results.

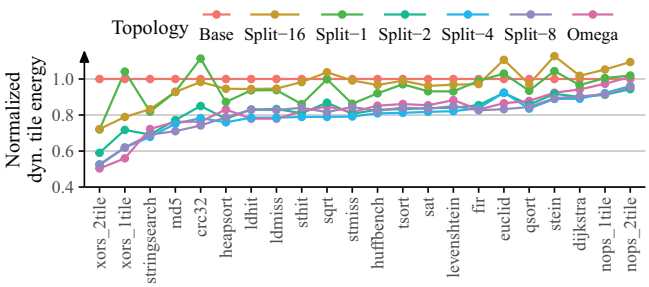


FIGURE 12. Interconnect topology exploration: per-benchmark impact on energy, normalized to base variant, simulation results.

they exhibit identical cycle timing and used identical clock constraints for timing closure.

Fig. 11 shows energy and area characteristics of all variants. The greatest energy reductions are achieved by Split-4 and Split-8: Both reduce dynamic tile energy by 19 % below base. Of these two, Split-4 has a 4 % lower tile area and can therefore be considered the best overall network.

Compared to Split-4/8, Split-1 and Split-2 have fewer switches, require less area and use more energy. Their higher energy use is attributed to broader data propagation to uninvolved endpoints. Compared to Split-4/8, Omega and Split-16 use more energy despite having more switches and requiring more area. This is attributed to the energy overhead of added switches within the interconnect network.

Fig. 12 shows energy use for each combination of benchmark and topology. The highest efficiency gains are realized for *xors_1tile* and *xors_2tile*, which synthetically evoke highest S bus toggling; lowest gains are found for *nops_1tile* and *nops_2tile*, which minimize S bus toggling.

E. FLOATING-POINT VARIANTS

Fig. 13 shows energy and execution time of the FP benchmarks on the evaluated three design variants FPU-1/4/16. For the FP benchmarks, FPU-16 exhibits the lowest energy use. FPU-4 and FPU-1 use more energy, as they share FPUs among multiple PEs. FPU-4 and FPU-16 exhibit identical execution times; FPU-1 uses additional cycles for S bus communication. The execution time and energy differences between FPU variants are greater for *cordic* than for *matmul*, as it performs FP operations more frequently.

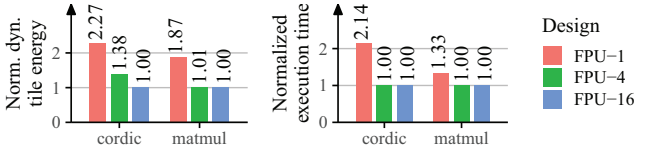


FIGURE 13. Energy use (left) and execution time (right) of FP benchmarks *cordic* and *matmul*, simulation results.

TABLE 2. Comparison of related CGRAs and MCU.

	SleepRunner [38]	HyCube [6, 16]	Amber [7]	RipTide [10]	Pasithea-1 (this work)
Node	28 nm FD-SOI	40 nm LP	16 nm FinFET	22 nm FinFET	22 nm FD-SOI
Results	silicon	silicon	silicon	post-syn.	silicon
Type	RISC MCU	CGRA			
		CPU-supervised			self-managed
ISA/μarch Decoupling	yes	no			yes
Compile Time	< 1 s	$20-3000$ s	~ 73 s	$5-1000$ s	$0.9-3.4$ s
$f_{clk,max}$	80 MHz	753 MHz	955 MHz	50 MHz	150 MHz
V_{DD} (V)	0.8/0.5/0.4	0.8 – 1.1	0.84 – 1.29	unknown	0.42 – 0.9
Fabric Size	N/A	4×4 PEs	384 PEs	6×6 PEs	8×16 PEs
PEs/mm²		5.6	19.1	144	437
μW/MHz	3.3	84.4	unknown	14.8	1.64
Total mm²	0.574	2.87	20.1	0.25	0.75
Memory	64 kB	4 kB	4608 kB	256 kB	256 kB
Benchmark	Dhrystone	FFT	dense lin. algebra		sqrt
Peak Performance	100 DMIPS	5380 MOPS	367 int16 GOPS	164 MOPS	148 int32 MIPS
Peak Efficieny	385 DMIPS/mW	26.4 int32 MOPS/mW	538 int16 MOPS/mW	180 MOPS/mW	601 int32 MIPS/mW
Static Power	8.4 μW	unknown	unknown	< 9.6 μW	2.06 μW

Adapted from Kaiser [14, Tab. II].

The impact of FPU variants on tile area and energy use of non-FP benchmarks is depicted in Fig. 11. FPU-1 increases tile area by only 14 % over the base architecture, FPU-4 increases tile area by 72 % and FPU-16 by 245 %. Fig. 11 shows that this increase of tile area degrades non-FP energy efficiency, due to increased wire lengths inside the tile.

Overall, the optimal choice between FPU-1/4/16 depends on the application. FPU-16 achieves the highest FP energy efficiency at the cost of area and non-FP energy efficiency. FPU-4 offers a more balanced trade-off, requires much less FPU area and degrades non-FP energy efficiency to a lesser degree. FPU-1 requires additional cycles and energy for FP operations but comes with only a minimal area overhead. It is thus a sensible choice for applications with only infrequent FP operations.

F. COMPARISON TO RELATED ARCHITECTURES

Table 2 compares Pasithea-1 with recent CGRA architectures and an ultra-low-power MCU.

Compared to the CGRAs, Pasithea-1 offers significantly faster code generation (1.4 s mean compile time) and requires no CPU supervision. With the ability to change microarchitectural features such as FPU integration variant,

interconnect network and number of hardware tiles without breaking machine code compatibility, it demonstrates decoupling between instruction set and microarchitecture. With its high energy efficiency and low static power, Pasithea-1 enables energy reduction in embedded low-power applications without the software overhead of traditional CGRAs. Furthermore, its high fabric density enables comprehensive instruction reuse even when area is tightly limited.

Recognizing the high SRAM energy overhead of CPU instruction fetching, SleepRunner [38] uses a custom ultra-low-power SRAM as program memory. Further circuit-level measures including split supply voltages are taken in SleepRunner to minimize its energy use. Due to its extensive instruction reuse, Pasithea-1 can surpass such ultra-low-power MCUs in energy efficiency despite its larger memory and lack of circuit-level energy optimization.

VIII. CONCLUSION

The presented CGRA architecture Pasithea-1 demonstrates a novel approach for reconciling energy efficiency advantages of CGRAs with CPU programmability advantages. It supports the C programming model and, due to its instruction set design, allows fast code generation. Using a wide range of benchmarks, this work showed that Pasithea-1 supports complex application logic without requiring a CPU for supervision. By hiding microarchitectural features from the instruction set, hardware optimizations can proceed without breaking the existing compiler or machine code. When instruction set features such as FP operations are added, backwards compatibility can be maintained.

For 12 of 14 benchmarks, Pasithea-1 surpassed the RISC reference system in energy efficiency, reducing energy by up to 5.6×. This is attained through *spatial dataflow*, with distributed operand registers supplanting a central register file, and comprehensive *instruction reuse*, both within and across function calls. Its combination of CPU-like programmability, high energy efficiency, area efficiency (437 PEs/mm^2) and low static power (2.06 µW) makes Pasithea-1 highly suitable as CPU replacement in small embedded applications that prioritize energy efficiency, which are particularly common in the Internet of Things (IoT).

The presented exploration of design variants illustrates optimization opportunities and tradeoffs in CGRA design space. Optimized interconnects were able to reduce signal propagation and increase energy efficiency. The best energy-area results were achieved with the Split-4 crossbar topology, demonstrating that a balance must be found between energy saved by improved signal propagation on the one hand and the added energy of control structures on the other hand.

The FPU integration variants show that the level of hardware resource replication should match the expected frequency of use to maximize energy efficiency: Highly replicated hardware units, such as in the FPU-16 variant, reduce energy when used frequently but can otherwise increase energy. When FP operations are rarely used, low replication, such as in the FPU-1 variant, is energy-optimal.

ACKNOWLEDGMENT

The authors thank GlobalFoundries for technology access and EUROPRACTICE for providing design tools.

REFERENCES

[1] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, Sep. 2013, doi: 10.1109/MM.2013.90.

[2] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *Proc. Int. Conf. Embed. Comput. Syst., Archit., Model. Simul. (SAMOS)*, 2016, pp. 235–244, doi: 10.1109/samos.2016.7818353.

[3] S. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, Apr. 2000, doi: 10.1109/2.839324.

[4] A. Yeung and J. Rabaey, "A reconfigurable data-driven multiprocessor architecture for rapid prototyping of high throughput DSP algorithms," in *Proc. 26th Hawaii Int. Conf. Syst. Sci.*, 1993, pp. 169–178, doi: 10.1109/hicss.1993.270747.

[5] O. Atak and A. Atalar, "BilRC: An execution triggered coarse grained reconfigurable architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 7, pp. 1285–1298, Jul. 2013, doi: 10.1109/tvlsi.2012.2207748.

[6] B. Wang, M. Karunarathne, A. Kulkarni, T. Mitra, and L.-S. Peh, "HyCUBE: A 0.9 V 26.4 MOPS/mW, 290 pJ/op, power efficient accelerator for IoT applications," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2019, pp. 133–136, doi: 10.1109/a-sscc47793.2019.9056954.

[7] K. Feng et al., "Amber: A 16-nm system-on-chip with a coarse-grained reconfigurable array for flexible acceleration of dense linear algebra," *IEEE J. Solid-State Circuits*, vol. 59, no. 3, pp. 947–959, Mar. 2024, doi: 10.1109/jssc.2023.3313116.

[8] D. Voitsechov, O. Port, and Y. Etsion, "Inter-thread communication in multithreaded, reconfigurable coarse-grain arrays," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2018, pp. 42–54, doi: 10.1109/micro.2018.00013.

[9] M. Mishra, T. J. Callahan, T. Chelcea, G. Venkataramani, S. C. Goldstein, and M. Budiu, "Tartan: evaluating spatial computation for whole program execution," *ACM SIGOPS Oper. Syst. Rev.*, vol. 40, no. 5, pp. 163–174, Oct. 2006, doi: 10.1145/1168917.1168878.

[10] G. Gobieski et al., "RipTide: A programmable, energy-minimal dataflow compiler and architecture," in *Proc. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2022, pp. 546–564, doi: 10.1109/micro56248.2022.00046.

[11] L. Liu et al., "A survey of coarse-grained reconfigurable architecture and design," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–39, Nov. 2020, doi: 10.1145/3357375.

[12] K. J. M. Martin, "Twenty years of automated methods for mapping applications on CGRA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2022, pp. 679–686, doi: 10.1109/ipdpsw55747.2022.00118.

[13] T. Kaiser and F. Gerfers, "Pasithea-1: An energy-efficient self-contained CGRA with RISC-like ISA," in *Proc. Int. Conf. Archit. Comput. Syst.*, 2022, pp. 33–47, doi: 10.1007/978-3-031-21867-5_3.

[14] T. Kaiser and F. Gerfers, "A 2.41-μW/MHz, 437-PE/mm^2 CGRA in 22 nm FD-SOI with RISC-like code generation," in *Proc. IEEE Symp. Low-Power High-Speed Chips Syst. (COOL Chips)*, 2023, pp. 1–6, doi: 10.1109/COOLCHIPS57690.2023.10121985.

[15] J. Weng, S. Liu, Z. Wang, V. Dadu, and T. Nowatzki, "A hybrid systolic-dataflow architecture for inductive matrix algorithms," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 703–716, doi: 10.1109/hpca47549.2020.00063.

[16] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *Proc. 54th Annu. Design Autom. Conf.*, 2017, pp. 1–6, doi: 10.1145/3061639.3062262.

[17] R. Prabhakar et al., "Plasticine: A reconfigurable architecture for parallel patterns," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 389–402, doi: 10.1145/3079856.3080256.

[18] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2003, pp. 61–70, doi: 10.1007/978-3-540-45234-8_7.

[19] M. S. S. Govindan, D. Burger, and S. Keckler, "TRIPS: A distributed explicit data graph execution (EDGE) microprocessor," in *Proc. IEEE Hot Chips Symp. (HCS)*, 2007, pp. 1–13, doi: 10.1109/HOTCHIPS.2007.7482519.

[20] A. Parashar et al., "Efficient spatial processing element control via triggered instructions," *IEEE Micro*, vol. 34, no. 3, pp. 120–137, May/Jun. 2014, doi: 10.1109/MM.2014.1.

[21] B. D. Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. D. F. Deprettere, R. Leupers, and J. Takala, Eds., Boston, MA: Springer, Sep. 2010, pp. 449–484, doi: 10.1007/978-1-4419-6345-1.

[22] D. Henry, B. Kuszmaul, and V. Viswanath, "The ultrascalar processor-an asymptotically scalable superscalar microarchitecture," in *Proc. 20th Anniv. Conf. Adv. Res. VLSI*, 1999, pp. 256–273, doi: 10.1109/ARVLSI.1999.756053.

[23] E. Gunadi and M. H. Lipasti, "CRIB: consolidated rename, issue, and bypass," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 23–32, Jun. 2011, doi: 10.1145/2024723.2000068.

[24] D. K. Wang and N. S. Kim, "DiAG: A dataflow-inspired architecture for general-purpose processors," in *Proc. 26th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Apr. 2021, pp. 93–106, doi: 10.1145/3445814.3446703.

[25] Arvind and D. E. Culler, "Dataflow architectures," *Annu. Rev. Comput. Sci.*, vol. 1, no. 1, pp. 225–253, Jun. 1986, doi: 10.1146/annurev.cs.01.060186.001301.

[26] A. A. D. Farahani, H. Beitollahi, and M. Fathi, "A dynamic general accelerator for integer and fixed-point processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 12, pp. 2509–2517, Dec. 2020, doi: 10.1109/tvlsi.2020.3023106.

[27] M. Brandalero, L. Carro, A. C. S. Beck, and M. Shafique, "Multi-target adaptive reconfigurable acceleration for low-power IoT processing," *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 83–98, Jan. 2021, doi: 10.1109/TC.2020.2984736.

[28] A. Lambrechts, P. Raghavan, M. Jayapala, B. Mei, F. Catthoor, and D. Verkest, "Interconnect exploration for energy versus performance tradeoffs for coarse grained reconfigurable architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 1, pp. 151–155, Jan. 2009, doi: 10.1109/tvlsi.2008.2002993.

[29] R. Wirsch and C. Hochberger, "Towards transparent dynamic binary translation from RISC-V to a CGRA," in *Proc. Int. Conf. Archit. Comput. Syst.*, 2021, pp. 118–132, doi: 10.1007/978-3-030-81682-7_8.

[30] A. Waterman and K. Asanović, "The RISC-V instruction set manual, volume I: User-level ISA, version 2.1," Dept. Electr. Eng. Comput. Sci., Univ. California Berkeley, Berkeley, CA, USA, Rep. UCB/EECS-2016-118, 2019.

[31] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, no. 12, pp. 1145–1155, Dec. 1975, doi: 10.1109/t-c.1975.224157.

[32] R. Carter et al., "22nm FDSOI technology for emerging mobile, Internet-of-Things, and RF applications," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, Dec. 2016, pp. 2.2.1–2.2.4, doi: 10.1109/IEDM.2016.7838029.

[33] P. D. Schiavone et al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," in *Proc. 27th Int. Symp. Power Timing Model. Optim. Simul. (PATMOS)*, 2017, pp. 1–8, doi: 10.1109/PATMOS.2017.8106976.

[34] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int. Symp. Code Gener. Optim.*, 2004, pp. 75–86, doi: 10.1109/cgo.2004.1281665.

[35] R. Lo, F. Chow, R. Kennedy, S.-M. Liu, and P. Tu, "Register promotion by sparse partial redundancy elimination of loads and stores," *ACM SIGPLAN Notices*, vol. 33, no. 5, pp. 26–37, May 1998, doi: 10.1145/277652.277659.

[36] J. Pallister, S. Hollis, and J. Bennett, "BEEBS: Open benchmarks for energy measurements on embedded platforms," 2013, *arXiv:1308.5174*.

[37] M. Gammelsæter. "Ersatz: A simple, small, work in progress SAT-solver, written in ANSI C." Feb. 2024. [Online]. Available: https://github.com/martingms/ersatz

[38] D. Bol et al., "SleepRunner: A 28-nm FDSOI ULP cortex-M0 MCU with ULL SRAM and UFBR PVT compensation for 2.6–3.6-uW/DMIPS 40–80-MHz active mode and 131-nW/kB fully retentive deep-sleep mode," *IEEE J. Solid-State Circuits*, vol. 56, no. 7, pp. 2256–2269, Jul. 2021, doi: 10.1109/JSSC.2021.3056219.

TOBIAS KAISER (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering from Technische Universität Berlin, in 2016 and 2017, respectively, where he is currently pursuing the Doctoral degree.

In 2018, he joined the Mixed Signal Circuit Design Group, Technische Universität Berlin. His research interests include energy-efficient processor design, design automation, and mixed-signal applications.

ESTHER GOTTSCHALK received the B.Sc. and M.Sc. degrees in electrical engineering from Technische Universität Berlin in 2021 and 2024, respectively.

In 2023, she joined as a Master Thesis Student with the Mixed Signal Circuit Design Group, Technische Universität Berlin. She is currently a Research Assistant with the Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut. Her research interests include energy-efficient processor design and mixed-signal applications in wireless communication.

KAI BIETHAHN received the B.Sc. and M.Sc. degrees in computer engineering from Technische Universität Berlin in 2020 and 2024, respectively.

From 2020 to 2024, he worked as a Student Researcher with the Mixed Signal Circuit Design Group. His research interests include energy-efficient architectures, life cycle assessment, and design automation.

FRIEDEL GERFERS (Senior Member, IEEE) received the Dr.-Ing. degree from the University of Freiburg, Germany, in 2005.

He has gained his first industrial R&D experience with Philips Semiconductor, Starnberg, Germany. In 2006, he joined as a Postdoctoral Research Fellow working on new types of piezoelectric MEMS sensors and their readout circuits with Intel Research, Santa Clara, CA, USA. His entrepreneurial spirit led him from 2007 to 2011 to the start-up companies Aquantia and Alvand Technologies, CA, USA. In the role of a Technical Director, he led the mixed-signal departments, which were crucial for the successful market positioning of these companies in the field of high-speed data transmission systems that operate close to the Shannon limit. In 2009, he founded the technology startup, NiederRhein Technologies, Mountain View, CA, USA. More recently, he was responsible for the worldwide development of high-precision data-converters for Integrated Device Technology, San José, CA, USA. The team was later in 2014 acquired by Apple Inc., Cupertino, CA, USA. Since 2015, he has been a Full Professor with the Technische Universität Berlin, Germany, where he is currently the Director and the Head of the "Mixed Signal Circuit Design" Chair. In 2018, he co-founded IC4X GmbH, Berlin, which specializes in the development of high-performance analog and mixed-signal circuits and systems. He is currently a member of the Scientific Advisory Board of the Leibniz Institutes – Innovations for High Performance Microelectronics and the Ferdinand Braun Institute, and of the Research Fab Microelectronics Germany. He is co-author of the book *Continuous-Time Sigma-Delta A/D Conversion, Fundamentals, Error Correction and Robust Implementations*. In addition, he has authored or co-authored several book chapters and holds more than 18 patents. His current research interests include energy-efficient mixed-signal integrated circuit design, self-correcting and reconfigurable analogue circuits, and high-speed and high-performance data converters for wireless, wireline and optical communications systems.

Dr. Gerfers was awarded the Einstein-Professorship for Mixed Signal Circuit Design from the Einstein Foundation in 2019. He is currently a member of the Technical Program Committee of IEEE Custom Integrated Circuits Conference, IEEE European Solid-State Devices and Circuits Conference, European Microwave Week, and Optical Fiber Communication Conference. He was a Guest Editor of IEEE JOURNAL OF SOLID-STATE CIRCUITS in 2021.