



Evaluation of Different Processor Architecture Organizations for On-Site Electronics in Harsh Environments

Sven Gesper¹ · Moritz Weißbrich¹ · Tobias Stuckenberg¹ · Pekka Jääskeläinen² · Holger Blume¹ · Guillermo Payá-Vayá¹ 

Received: 27 April 2020 / Accepted: 3 November 2020 / Published online: 26 December 2020
© The Author(s) 2020

Abstract

Microcontrollers to be used in harsh environmental conditions, e.g., at high temperatures or radiation exposition, need to be fabricated in robust technology nodes in order to operate reliably. However, these nodes are considerably larger than cutting-edge semiconductor technologies and provide less speed, drastically reducing system performance. In order to achieve low silicon area costs, low power consumption and reasonable performance, the processor architecture organization itself is a major influential design point. Parameters like data path width, instruction execution paradigm, code density, memory requirements, advanced control flow mechanisms etc., may have large effects on the design constraints. Application characteristics, like exploitable data parallelism and required arithmetic operations, have to be considered in order to use the implemented processor resources efficiently. In this paper, a design space exploration of five different architectures with MIPS- or ARM-compatible instruction set architectures, as well as transport-triggered instruction execution is presented. Using a 0.18 μm SOI CMOS technology for high temperature and an exemplary case study from the fields of communication, i.e., powerline communication encoder, the influence of architectural parameters on performance and hardware efficiency is compared. For this application, a transport-triggered architecture configuration has an $8.5\times$ higher performance and $2.4\times$ higher computational energy efficiency at a $1.6\times$ larger total silicon area than an off-the-shelf ARM Cortex-M0 embedded processor, showing the considerable range of design trade-offs for different architectures.

Keywords ASIC · Design tradeoff analysis · Harsh environment · Processor architecture organization · Transport-triggered architecture

✉ Guillermo Payá-Vayá
guipava@ims.uni-hannover.de

¹ Institute of Microelectronic Systems, Leibniz Universität Hannover, Hannover, Germany

² Customized Parallel Computing Group, Tampere University, Tampere, Finland

1 Introduction

Automotive, deep-drilling and aerospace applications with on-site microcontroller-like systems are still an important field of research. These systems should provide enough flexibility to facilitate continuous maintainability. Moreover, the electronic components themselves have to ensure reliable operation even in harsh environmental situations, such as high temperatures or increased radiation.

Nowadays, integrated circuits for harsh environments are manufactured using relatively large technology nodes and silicon on insulator (SOI) stacks to reduce leakage current and latch-up effect probability [8]. Current node sizes are in the range of 1 μm down to 130 nm provided by, e.g., Fraunhofer H035 (350 nm, 250 °C, [7]), Vorago HARDSIL (130 nm, 200 °C, [6]), Tekmos (0.6 μm , 210 °C, [29]), or X-Fab XT018 (180 nm, 175 °C, [36]). As a drawback, these large technology nodes only provide a moderate operating frequency, reducing the overall system performance. Furthermore, the number of transistors and consequently, the circuit complexity on a die is limited. Considering that the power consumption is usually restricted for embedded applications, the according system's processor architecture organization has a high impact on the overall efficiency.

The instruction set architecture (ISA) and the way how this ISA is implemented in hardware (i.e., processor architecture organization or microarchitecture) determine the performance of a processor architecture, which is constrained by the above mentioned technology node limitations. On the hardware implementation side, two of the most significant design parameters are the data path width and the instruction execution paradigm. Predominant microcontroller data path widths can be classified into 8-, 16-, or 32-bit architectures. The most common instruction execution paradigms are single-cycle, multi-cycle or pipelined execution, i.e., RISC- or CISC-like design concept. A completely different design concept is presented by the transport-triggered architecture [9]. This architecture performs instructions as side effects to move operations. On the other side, different design decisions in the ISA, e.g., register- or memory-based computation, the number and utilization of pipeline stages, the existence of *branch delay* and *stall* cycles, instruction and immediate operand width, the availability of hardware acceleration for multiplication and division operations, as well as the possibilities for *conditional instruction execution*, have a significant impact on operation execution throughput, instruction memory size and code density of a processor [35]. All of these parameters highly influence the processor's performance as well as silicon area and energy requirements.

One exemplary application for harsh environment systems is the powerline communication. This technology offers the benefit of reducing the hardware points of failure in the application systems by reducing the amount of communication wires used for data transfers. Since replacing a broken wire in, e.g., a satellite system is extremely costly or not possible, the powerline communication bus protocol can reduce the amount of wiring significantly when adding more and more communication nodes to the system [11]. In case of a harsh environment system, sensor and control data are the most common types of information that are transferred. The required data rates of such information usually reach few kbit/s, but

modern powerline communication standards can provide several Mbit/s, which enables the connection of over 100 sensors and actors onto a single wire [27]. The main challenge for communication systems in a harsh environment is to provide a stable and reliable link even at low Signal-to-Noise-Ratio (SNR) or over frequency fading channels. Here, the powerline communication uses Orthogonal Frequency Division Multiplex (OFDM) and can disable certain frequency carriers to ensure communication over frequency selective channels. Additionally, the use of a repeated preamble allows the receiver to measure the current SNR precisely and perform an exact synchronization, making the powerline communication system a good case study as a harsh environment application [1, 20].

This paper is an extension of a previously published paper [13], where a Reed–Solomon encoder was used to evaluate different instruction execution paradigms (i.e., single-cycle, multi-cycle, and pipeline) of five processor architectures. Based on this case study, the design space has been explored, regarding processing performance, silicon area, and power consumption. The design exploration is extended in this paper to a more complete application, including more implementation details and other processor architectures. The main contributions of this paper are:

- Description of five different processor architectures, which can be used as a microcontroller in harsh environment systems. This includes a microarchitecture, which can execute two instructions at the same time and implements the MIPS ISA. These architectures are implemented in HDL.
- Use of a real application for communication, i.e., powerline physical layer encoder, which is used in harsh environments.
- Design space exploration, regarding processing performance, silicon area and power consumption, using a high-temperature capable technology, is presented and analyzed to compare the different architecture organizations.

This paper is organized as follows: in Sect. 2, exemplary commercial microcontrollers for harsh environments are compared. Section 3 describes the features of five different processor architectures, making emphasis on their hardware implementation. The evaluation of these architectures, using the aforementioned powerline application is given in Sect. 4. Finally, a conclusion is drawn in Sect. 5.

2 Related Work

For both high-performance computers and low-power embedded systems, the choice of a suitable instruction set architecture is crucial for the overall processor performance as well as the hardware and energy efficiency of the system. In [35], the properties, advantages and drawbacks of several popular RISC-style ISAs, i.e., MIPS, SPARC, Alpha, ARMv7 and ARMv8, and OpenRISC, are discussed. Furthermore, the 80×86 ISA, being most popular in desktop computers, is considered. All of these ISAs are intended to be used on 32-bit or 64-bit data path processors with pipelined instruction execution for a high instruction throughput. However, especially for smaller 8-bit or 16-bit microcontrollers, more simple

single-cycle or multi-cycle processor organizations may still be used, as shown in our previous work [13] for the AVR and MSP430 architectures. These organizations can lead to implementations with lower silicon area requirements when performance is not the major design issue, which is important in area-constrained large technology nodes for high-temperature applications.

Table 1 shows exemplary commercial microcontrollers specialized for harsh environments. They differ regarding their maximum operating temperature, which ranges from 150 °C up to 225 °C, and clock frequency due to their semiconductor technology and the underlying processor architecture organization. For many architectures, the execution is pipelined in 2 to 8 stages or is performed by a multi-cycle structure. Architectures with a deeper pipeline have a higher operating frequency, which is not a direct increase in performance due to data and control hazards in the pipelined execution. Most cores follow the RISC-like design paradigm and include a multiplication unit. The core in [31] also features a floating-point unit. The available space for programs in the instruction memory ranges from 2 kB up to 4MB when using external Flash memory. This exemplary portfolio shows 8-, 16- and 32-bit architectures, allowing a wide spectrum of applications being efficiently implemented.

3 Processor Architecture Organizations

In this paper, five different processor architectures, representing a wide variety of different design concepts and implemented in HDL, are described. These are an ARM Cortex-M0, an ARM Cortex-M3, a MIPS32-ISA-compatible, a VLIW-MIPS processor with two issue slots, and a transport-triggered architecture (TTA) [19]. A

Table 1 Commercial architectures for digital signal processing in high temperature environment

Company	Model	Features	T_{\max} (°C)	f_{\max} (MHz)	Instruction memory
TI	[31]	32-bit, RISC, 8-stage pipeline, MAC, FPU	125/ 210	150/ 100	Flash 512 kB
TI	[30]	32-bit, RISC, 8-stage pipeline, MAC	220	150	128 kB (ext. Flash 4 MB)
TI	[32]	16-/32-bit, RISC, 3-stage pipeline, MUL	220	60	64 kB (ext. Flash 1 MB)
Tekmos	[29]	8-bit, RISC, 2 to 3-stage pipeline, DIV and MUL	210	16	2 kB (ext. 64 kB)
Honeywell	[16]	8-bit, RISC, 2 to 3-stage pipeline	225	16	64 kB
VORAGO	[34]	32-bit, RISC, 3-stage pipeline, MUL	200	50	128 kB
Microchip	[21]	16-bit, CISC, 2-stage pipeline, DIV and MUL	150	80	16 kB / 32 kB
Freescale	[10]	16-bit, RISC, multi-cycle, DIV and MUL	150	25	16 kB

T Temperature, *MUL* multiplier, *DIV* divider, *FPU* floating-point unit

selection of the most important features for this work is given for the aforementioned five cores and the AVR8 and the MSP430-compatible NEO430, implemented in [13], in Table 2. Each processor executes C-Code, compiled and assembled by toolchains based on LLVM (MIPS, VLIW-MIPS and TTA) or GCC (ARM). The optimization flags are set to $-O3$ for best throughput. According to [35], the main distinctive ISA features of architecture organizations are instruction execution paradigm (pipeline/multi-cycle/transport-triggered), number of registers and pipeline stages, *branch delay*, *stall* and *conditional execution* mechanisms, instruction and immediate operand width, as well as the configuration of a dedicated hardware accelerator for integer division and multiplication. These main distinctive ISA features are covered in the following. The implemented architectures directly accesses instructions and data memory without additional cache systems.

For a fixed application, the performance and hardware efficiency of a processor can be increased by applying the *application-specific instruction-set processor* (ASIP) concept. This could be especially interesting for the TTA, which allows complete customization of the data flow and operations required by the application at hand. However, it should be emphasized that this paper is focused on the inherent properties of the different baseline ISAs and their hardware implementation for a given application only. For the TTA, this is achieved by exclusively using non-specialized functional units, which provide general ALU operations comparable to the other evaluated architectures.

The aforementioned single-cycle AVR8 and multi-cycle NEO430 microcontrollers are not further considered due to their low performance results and inefficient data processing, evaluated previously in [13]. This is mainly caused by the 8- and 16-bit data path and the lack of a dedicated multiply unit in the AVR8, which are not appropriate for the performance requirements of the selected application case study used in this work. All five evaluated architectures have a

Table 2 Main distinctive architectural features of the presented cores

CORE	Data word size, design	Regs.	Dataflow type	Instruction execution	Multiplier/divider unit (latency)
ARM CM0	32-bit, RISC	17	R \Leftrightarrow R	Pipelined (3 stages) and multi-cycle	Yes (1 cycle)/no
ARM CM3	32-bit, RISC	17	R \Leftrightarrow R	Pipelined (3 stages) and multi-cycle	Yes (1/2..12 cycle)
MIPS32	32-bit, RISC	32	R \Leftrightarrow R	Pipelined (5 stages)	Yes (1..32 cycles)
VLIW-MIPS	32-bit, RISC	32	R \Leftrightarrow R	Pipelined (4 stages)	Yes (1..32 cycles)
TTA	32-bit (variable)	var.	R/FU \Leftrightarrow R/FU	Transport-triggered	Yes (1..32 cycles)
AVR8	8-bit, RISC	32	R \Leftrightarrow R	Pipelined (2 stages)	No
NEO430	16-bit, CISC	16	R/M \Leftrightarrow R/M	Multi-cycle	Yes (16 cycles)/no

Regs/R Register(s), *M* memory, *FU* functional unit

32-bit data path and therefore support the 32-bit data processing required by the case study of this work.

3.1 ARM Cortex-M Processors

The Cortex-M processor series from ARM Limited is designed to operate as microcontroller-like processors. In this paper, the Cortex-M0 (hereafter named CM0) and Cortex-M3 (hereafter named CM3) processors, which are two of the smallest processors in this series, are used for evaluation. Both processors have three pipeline stages, in-order instruction execution, a 32-bit data path width, 16 registers, and a 32-bit hardware multiplier. Figure 1 shows the microarchitecture of both processors, including the three pipeline stages: instruction fetch, instruction decode and execution. The multiplier of the CM0 can be configured to be either a pipelined, large single-cycle or an iterative, smaller 32-cycle multiplier. Additional to its single-cycle multiplier unit, the CM3 also implements a divider and MAC unit (dotted lines in Fig. 1), which are designed to use multi-cycle execution.

Both processor configurations have a prefetch unit, which prefetches at least two consecutive instructions. The CM0 only has one 32-bit register prefetch, which can hold two 16-bit Thumb instructions. In case of conditional branches, the pipeline of the CM0 has to be *stalled* until the branch target is loaded by the prefetch unit. The CM3 has a FIFO for up to three 32-bit instructions, meaning that a maximum of six 16-bit Thumb or a minimum of three 32-bit Thumb2 instructions can be prefetched. This FIFO allows the CM3 to use *branch forwarding* and *speculative branch target prefetch*. *Branch forwarding* decodes unconditional branches during the ID stage. Therefore, the instruction located in the target address is scheduled immediately, saving one cycle of pipeline *stalling*. *Speculative branch target prefetch* in this case means that both targets of a branch (*taken* and *not taken*) are prefetched and stored

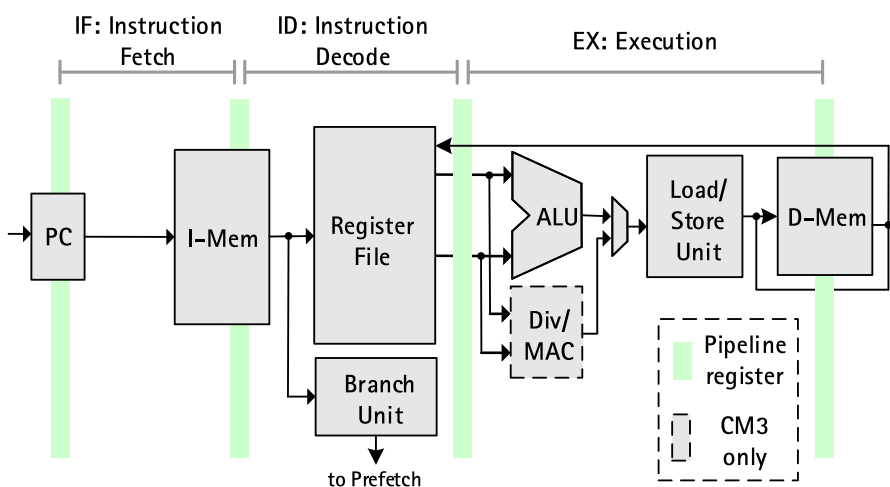


Fig. 1 Pipeline structure of the ARM Cortex-M processors

in the FIFO buffer. After the branch decision is computed in the execution stage, one of the prefetched target is dropped which could in the worst case lead to one cycle of pipeline *stalling* [2].

The instruction set of both the CM0 and CM3 is designed to keep the instruction memory (I-Mem) size at a minimum. They apply compression of instruction code by combining multiple incremental load or store instructions to be executed in a single command. For instance, the code listed in Listing 1 encodes first a load instruction of consecutive values from the stack using the stack pointer (sp) into the registers r4–r6, followed by a store instruction that stores the registers r0–r3 at the address given in register r4, incrementing that address value after each store execution [4].

Listing 1: Load and Store instruction with multiple execution cycles

```
ldmia.w sp!, {r4, r5, r6}
// Execution time: 1 initial cycle + 3 RF loads = 4 cycles
stmia.w r4, {r0, r1, r2, r3}
// Execution time: 1 initial cycle + 4 RF writes = 5 cycles
```

The instruction set of the CM0 is the ARMv6-M [5], which is a reduced set of 56 microcontroller-optimized instructions from the ARMv7-M standard [4]. Only 16-bit Thumb instructions are supported, which are fetched from the 32-bit wide I-Mem. Due to the reduced instruction width, only short immediate values of up to 8-bit can be encoded within the instructions. *Conditional execution* is only available for branch instructions. Compared to the CM3, the execution time of the ARMv6-M instruction is deterministic in the sense that no data value-dependent execution is present, resulting in a fixed amount of cycles for computation [3]. The instruction set architecture of the CM3 is the ARMv7-M, which is a microcontroller-oriented set of the ARMv7 instructions [4]. The 32-bit wide instructions allow the encoding of 8-bit short immediate values with the instruction, and additional immediate modifier bits are provided to rearrange the value to a long 32-bit immediate following 28 frequently used pattern templates to achieve a high instruction memory compaction [4]. To avoid unnecessary control flow, most instructions allow *conditional execution* using condition codes, however, a preceding condition block initialization instruction is required. Moreover, the ARMv7-M instruction set has various multi-cycle instructions, e.g., a signed multiply-and-accumulate long instruction SMLAL <RdLo>, <RdHi>, <Rn>, <Rm>, which multiplies two 16-bit values (Rn, Rm) and accumulates them in a 64-bit accumulation register, consisting of two 32-bit registers (RdLo, RdHi). The execution time of this instruction depends on the type (i.e., width) of the input operands and can be in the range of 2 to 7 cycles. During the execution of these instructions, the instruction fetch is *stalled*. Another multi-cycle instruction is the division which can take between 2 and 12 cycles. Depending on the number of leading zeros (or ones if signed operation) of the input operands, the 12-cycle division is stopped early to speed up computation time. If a data-dependent instruction needs access to the division result, a pipeline *stall* occurs until the result is available.

3.2 MIPS32 Processor

The implemented MIPS32, based on [22], is a 32-bit pipelined architecture with in-order instruction execution and has 5 pipeline stages that support hazard resolution (see Fig. 2). Data and instruction memory use independent 32-bit address spaces with byte-wise alignment. The 32-bit wide instructions of the ISA form a RISC architecture and support register-to-register operations using a register file with 32 entries and the encoding of 16-bit short immediate values within the instruction. Each instruction can have up to three operands, defining two sources and one destination, making the MIPS32 a three-operand machine (e.g., $R1 + R2 \rightarrow R3$: ADD R3, R1, R2).

In case of executing branch instructions (and also jumps), the instruction following in the code is always executed independently of if the branch is taken or not. This is known as *branch delay slot*. If the compiler can not allocate any instruction after a branch/jump instruction, a NOP instruction will be used, which will decrease the performance. However, branch instructions can be avoided by using partial *conditional execution*, which is implemented using conditional move operations.

The MIPS32 processor also includes a Multiply-and-Accumulate (MAC) unit as well as a divider unit in the execution stage. Those hardware units are implemented using a configurable processing latency of 1 to 32 cycles at design time (see Sect. 3.5). In case of accessing the 64-bit wide result register, the execution stage may cause a *stall* for instruction fetch and decode until the multiplication (or division) operation is finished.

3.3 VLIW-MIPS Processor

The VLIW-MIPS processor is derived from the aforementioned MIPS32 [12]. In contrast to the MIPS32, the VLIW-MIPS uses two parallel issue slots. Thus, the 64-bit instruction word contains two separate instructions based on the ISA of the MIPS32. Instructions allocated in the second issue slot are mainly used for

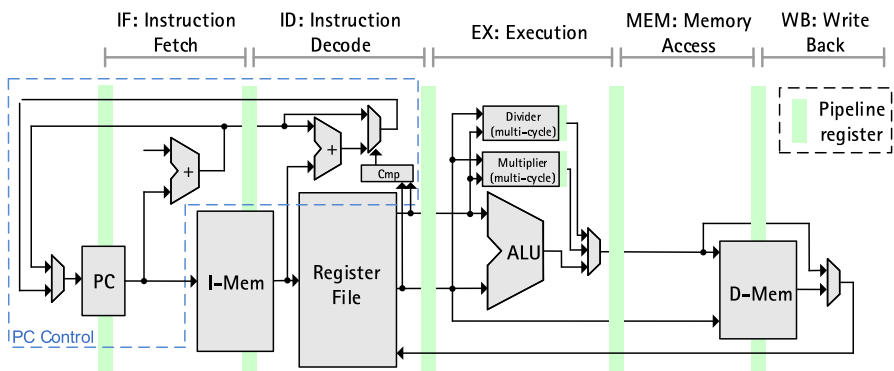


Fig. 2 Five stage pipeline structure of the MIPS processor

performing memory accesses, while the other issue slot is dedicated to arithmetic/logic and control flow operations. However, the second issue slot can still be used to execute `add` or `sub` instructions (without any flag computation) by using the second small adder of the execution stage, which is required for calculation of base-offset memory addresses initially.

The use of a specialized issue slot for memory access allows to reduce the number of pipeline stages to 4. Now, the execution (EX) and the memory access (MEM) stage of the MIPS32 are combined (see Fig. 3). In order to avoid increasing the critical path of the VLIW-MIPS hardware architecture implementation, the complexity of the ALU in the MEM stage is reduced for address computation and therefore supports simple arithmetic only. The parallel execution of memory accesses along with arithmetic/logic instructions allows the use of instruction-level parallelism in applications, due to the high amount of load/store instructions in contrast to pure computational instructions. A typical amount of parallelizable memory instructions is about 35% (based on SPEC CPU2006 benchmark [14]). However, due to data dependencies within an application, the two issue slots cannot always be fully utilized. The number of ports of the 32-entry register file of the VLIW-MIPS is doubled in comparison to the MIPS32, due to the dual-issue data accesses.

Another drawback is the implementation of two *branch delay slots*. In contrast to the MIPS32 microarchitecture, the VLIW-MIPS microarchitecture evaluates the branch condition during the EX stage, fetching always the two address-consecutive very long instruction words before taking or not taking the branch. In case of taking the branch, 4 instructions (i.e., 2 very long instruction words) will be flushed with the corresponding loss of performance. This part of the microarchitecture could be improved by evaluating the branch condition on the ID stage (like in MIPS32

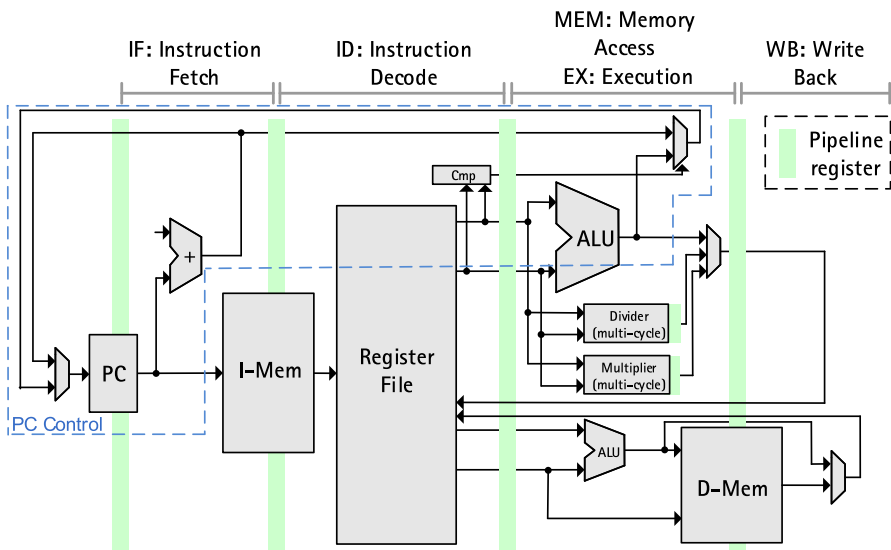


Fig. 3 Four stage pipeline structure of the VLIW-MIPS processor

microarchitecture), which introduces a trade-off between the required silicon area, the performance and the code density [35].

3.4 Transport-Triggered Architecture Processor

The Transport-Triggered Architecture (TTA) can be described as an exposed data path processor. Functional units are connected through a programmable bus interconnect, which creates a programmable data path [9]. In contrast to describing specific computational operations, the instruction words contain configurations for the interconnections between the Functional Units (FUs). Operands are transferred to a functional unit through programmable sockets, which are basically implemented with switches, connecting a FU to a set of buses. A unit's operation is triggered as a side-effect of writing data to it. An exemplary setup of the TTA is shown in Fig. 4. The configurable architectural concept allows to add new FUs with custom functionality as well as more buses to enable parallel data transfer. Furthermore, the selective configuration of bus sockets is possible, taking profit of the processing characteristics of the target application and considering a trade-off between programming flexibility and interconnect complexity. An open-source *TTA HW/SW Co-Design Environment* (TCE) including an LLVM-based C compiler is available for the TTA designer to aid in the configuration, customization and design space exploration process [19].

TTA configurations with resources similar to conventional processor architectures are good starting points for application profiling and hardware optimization. Figure 5a shows a data path similar to the single-issue MIPS and ARM Cortex-M architectures. Three transport buses are used, corresponding to addressing two

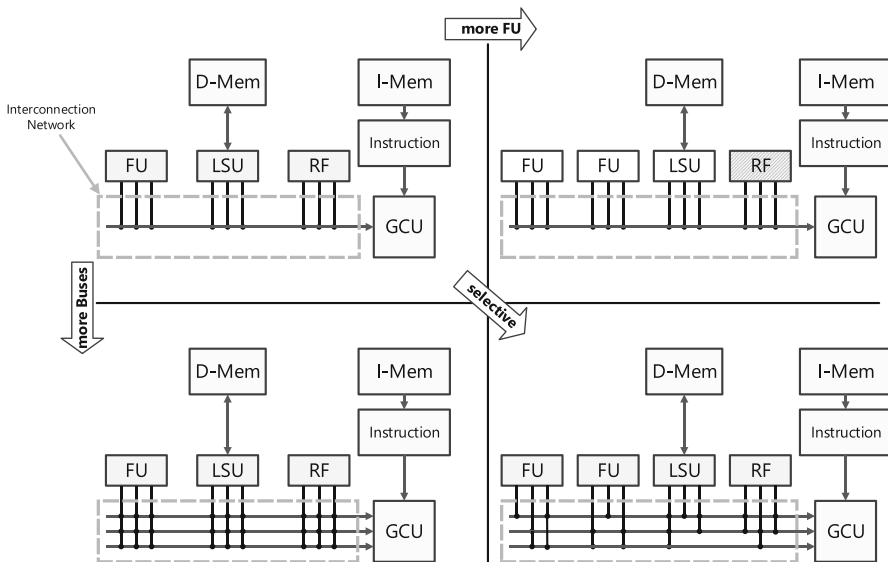


Fig. 4 Different configurations of the transport-triggered architecture (TTA)

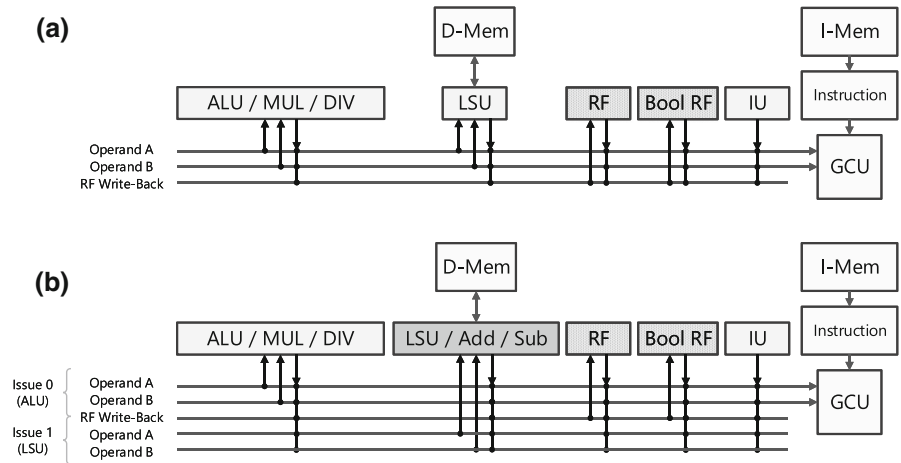


Fig. 5 TTA configurations comparable to **a** single-issue MIPS and ARM data path, **b** dual-issue VLIW-MIPS data path

operands and one result per cycle like in 3-address RISC machines. The result of a FU may be directly fed back onto the operand buses to resemble the forwarding mechanism in pipelined architectures. Furthermore, contrary to the aforementioned RISC architectures, the TTA-specific programmer-visible interconnection network allows the software programmer (or the compiler) to explicitly direct data from one unit to another without touching the register file (register bypassing). Each FU contains two operand registers and/or one result register, which are updated whenever an operation is executed in this particular FU. Moreover, depending on the complexity of the implemented operations, FUs may contain further internal state registers not directly accessible via the TTA programming model. Due to the registered FUs and register bypassing, a register file (RF) is optional for TTA operation, but is commonly attached to the system to store frequently accessed temporal variables and results that have to outlive succeeding operations.

Techniques like register bypassing and operand sharing, i.e., multiple FU ports receive common operands simultaneously from a single source, reduce RF port pressure on TTA register files. To illustrate this, Listing 2 shows a disassembly example of a TTA transport schedule for the architecture in Fig. 5a, where each of the three columns represents a transport bus and each line represents a clock cycle. Data transports on a specific bus are represented in the form `FU.SourcePort -> FU.DestinationPort`, empty bus cycles (NOPs) without data transports are indicated by three dots (...). A source port may be either a RF read port with register address, an immediate value encoded in the instruction, or the result output of a FU (`out1`). A destination port may be either a RF write port with register address, the trigger operand input of a FU that also defines the transport-triggered operation by an opcode (`in1t.OP`), or the second operand input of a FU (`in2`). In the example of Listing 2, one operation is triggered every cycle on the first bus (`LDQ, ADD, MUL, ADD`), but only one operand is read from the RF per cycle. The other operand is obtained via register bypassing from the registered result output of other FUs, and no RF writes are

necessary. A conventional RISC architecture like MIPS or ARM Cortex-M, however, would require 2 operand reads and 1 result write from/to the RF per cycle. Due to that, the TTA concept offers the flexibility to merge hardware resources like buses and RF ports that are never or not often used in parallel, which reduces the circuit complexity and improves resource usage efficiency [17]. An example for this is the configuration in Fig. 5b that corresponds to the dual-issue VLIW-MIPS, with the ALU/MUL/DIV FU resembling the first issue, i.e., arithmetic operations, and the LSU/ADD/SUB FU resembling the second issue, i.e., load, store, addition and subtraction operations. Here, only five buses are required because the two result buses back to the RF could be merged due to register bypassing. Furthermore, for both configurations, register bypassing allows to use a RF with a single read port only.

Listing 2: Disassembly of register bypassing on TTA

```
ALU.out1 -> LSU.in1t.ldq, LSU.out1 -> ALU.in2, ... ;
RF.19 -> ALU.in1t.add, ALU.out1 -> ALU.in2, ... ;
RF.12 -> ALU.in1t.mul, ALU.out1 -> ALU.in2, ... ;
RF.20 -> ALU.in1t.add, LSU.out1 -> ALU.in2, ... ;
```

The basic FUs required for compilation and execution of C code are the Global Control Unit (GCU) which controls the overall program flow and performs jumps and function calls, a 32-bit wide RF with at least 8 entries or more, a 1-bit wide Boolean RF, the ALU, as well as the memory Load-Store Unit (LSU). To avoid the excessive increase of instruction word width with the number of buses, the encoding of immediate operands on each bus is shortened. If longer immediate values should be required, an Immediate Unit (IU) is used to combine multiple short immediate values from several buses to form a 32-bit immediate value [18].

Listing 3: Disassembly of TTA NOP schedule due to functional unit latency

```
ALU.out1 -> ALU.in1t.mod, ..., ... ;
..., ..., ... ;
..., ..., ... ;
..., ..., ... ;
..., ..., ... ;
..., ..., ... ;
RF.9 -> ALU.in1t.add, ALU.out1 -> ALU.in2, ... ;
```

Listing 4: TTA disassembly of sparsely filled LSU data fetch latency

```
!BOOL.0 ALU.out1 -> LSU.in1t.ldw, ..., !BOOL.0 ALU.out1 -> RF.3 ;
..., ..., !BOOL.0 RF.1 -> RF.6 ;
..., ..., ... ;
..., ..., LSU.out1 -> RF.5 ;
```

Listing 5: TTA disassembly of sparsely filled GCU jump latency

```
IU_1x32.0 -> gcu.pc.jump, ..., ... ;
..., ..., ... ;
RF.12 -> LSU.in1t.stq, LSU.out1 -> LSU.in2, ... ;
7 -> ALU.in1t.add, RF.9 -> ALU.in2, ... ;
```

Regarding the control flow, the TTA shows linear behavior and sequentially executes one instruction line from memory each cycle. The GCU is very simple, and the hardware complexity and critical path are mostly determined by the FUs and the bus interconnect network. Each instruction encodes each data transfer for all buses in that particular cycle, therefore the instruction width is directly dependent on the particular interconnection network configuration. Latency and resource access schedule of each FU are automatically considered by the retargetable TTA compiler backend, which allows the use of standard single-cycle as well as multi-cycle and pipelined FUs. It should be noted that no hazard detection and resolution are implemented in hardware. Therefore, there are no hardware-generated *stalls*, and the instruction schedule generated at compile time must not contain any data hazards, which is accomplished either by filling in data-independent transfers or empty bus cycles (NOP). Due to that, increased latency on functional units could result in sparsely filled schedules and ineffective instruction memory utilization. An example is shown in Listing 3. The same applies to pipelined synchronous memory read accesses to the data memory by the LSU (Listing 4) and to the instruction memory by the GCU (Listing 5), which have three cycles data fetch or *branch delay* due to address registration, synchronous memory read, and read-back value processing and registration. If the schedule cannot be filled otherwise, loading from memory, jumping, branching, and function calling is expensive in terms of performance and instruction memory utilization. In order to reduce the amount of expensive control flow, every bus transport can be executed conditionally by using status flags in the Boolean RF, which are generated by ALU comparison operations. By *conditional execution*, control flow is converted into data flow, avoiding jumps by the GCU. Additionally, conditional branches are realized by conditionally executing a jump operation.

3.5 Fully Configurable Divider and Multiplier Co-processors

When processing divisions or multiplications purely in software, the actual operation is performed by several compare, add/subtract and shift instructions. Obviously, this increases the processing time. As a design parameter for application-specific microcontrollers, a dedicated co-processor for integer multiplication or division can massively speed up the calculation of these operations. Since both operations are based on iterative shift-and-add algorithms, a systolic array is suitable for an efficient hardware implementation [26]. This array can be directly implemented as a pipeline structure (Fig. 6a) or can be folded in order to reduce the silicon area requirements (see Table 3 and Fig. 6b, c), which is important in large technology nodes like the 180 nm high-temperature SOI technology used later in this paper. However, contrary to a pipelined architecture, the resulting folded array cannot start a new operation every clock cycle due to resource conflicts.

As an example, Fig. 6 shows the projection of a 4×4 -bit non-restoring divider array to a folded structure [26]. In Fig. 6a, the division is performed by a fully pipelined array in four stages. This implementation has the shortest critical path and can process a new operation every cycle at a result latency of four cycles. However, due to the pipelined structure, the silicon area for this implementation is the largest.

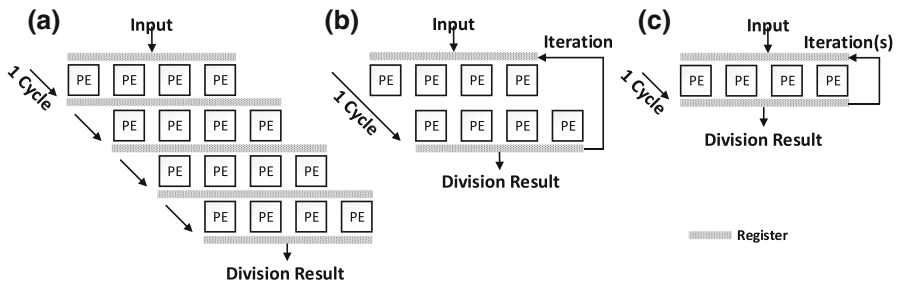


Fig. 6 Different configurations of an exemplary 4×4 -bit non-restoring divider array. **a** Fully pipelined structure, **b** two-level folded structure and **c** one-level folded structure

Table 3 Trade-off between 32-bit multiplier/divider configurations for a 180 nm high-temperature SOI standard cell technology

Latency/levels	Frequency (MHz)		Structure	Area (mm ²)	
	MUL	DIV		MUL	DIV
32 cycles/1 level	107.5	123.5	Folded	0.051	0.047
			Pipelined	0.372	0.537
16 cycles/2 levels	102.0	90.1	Folded	0.057	0.056
			Pipelined	0.275	0.439
8 cycles/4 levels	92.6	58.5	Folded	0.069	0.076
			Pipelined	0.226	0.391
4 cycles/8 levels	76.0	34.0	Folded	0.089	0.113
			Pipelined	0.202	0.366
2 cycles/16 levels	54.6	18.3	Folded	0.130	0.189
			Pipelined	0.190	0.354
1 cycle/32 levels	34.8	9.5	Folded	0.184	0.348
			Pipelined	0.184	0.348

The folded configurations use an iterative multi-cycle processing scheme, e.g., by implementing only two levels or one level of this divider array (Fig. 6b, c) used for two or four clock cycles per operation, respectively. For Fig. 6c, the number of result latency cycles and the critical path length are therefore equal to the pipelined implementation in Fig. 6a. The silicon area of folded array architectures is significantly reduced compared to pipelined implementations (see Table 3). However, the folded structure does not allow overlapped execution of multiple operations because of computation resource conflicts due to the truncation of the full divider array. The folding technique can also be applied to array multipliers in the same way [26], which allows to trade off multiplication/division latency, silicon area, and maximum clock frequency for a specific processor architecture and application.

In real applications, the use of a pipelined or folded architecture depends on the application code characteristic. The number of multiplication/division operations in the code, as well as data dependencies, influence the maximum number and utilization of multiplier/divider hardware units. Furthermore, architectural decisions

and properties of a particular processor data path play an important role. On the one hand, MIPS32 and ARM Cortex-M are single-issue architectures and include single multiplication and division functional units in the ALU. In the dual-issue VLIW-MIPS, single multiplier and divider array units are included in the arithmetic ALU and can therefore only be used by the first issue slot. On the other hand, the TTA generally allows more multiplier and divider units to work in parallel due to exploited parallelism by several buses. However, the implemented number of units has to match the application code characteristics to implement an area- and power-efficient architecture.

To illustrate the performance impact on the target application (see Sect. 4.1) when varying the multiplier level configuration, an evaluation of a VLIW-MIPS and MIPS with different folded array configurations is presented in Fig. 7. As shown in Fig. 7a, the maximum throughput for the VLIW-MIPS is achieved by the use of a 16-level multiply unit. This is the optimal trade-off between operating frequency (limited by long timing paths which pass through the architecture including the co-processor arrays) and processing latency (required clock cycles for the application). For the MIPS, the optimum number of levels in the multiply unit is eight, as the performance reaches its maximum there, as seen in Fig. 7b. In the MIPS, the decrease of the critical path from the 16-level to the 8-level multiply unit, which directly increases the maximum operating frequency, has a high positive impact on the total processing performance. This also applies to an exemplary TTA configuration with four identically leveled multipliers (Fig. 7c), which shows a comparable behavior except that the absolute performance metric is higher due to exploited functional unit parallelism. In contrast, the increase of the cycles on the VLIW-MIPS processor is higher than the frequency gain (which is quite small due to the critical path leading through the architecture now), so the processing performance is reduced by an 8-level multiplier.

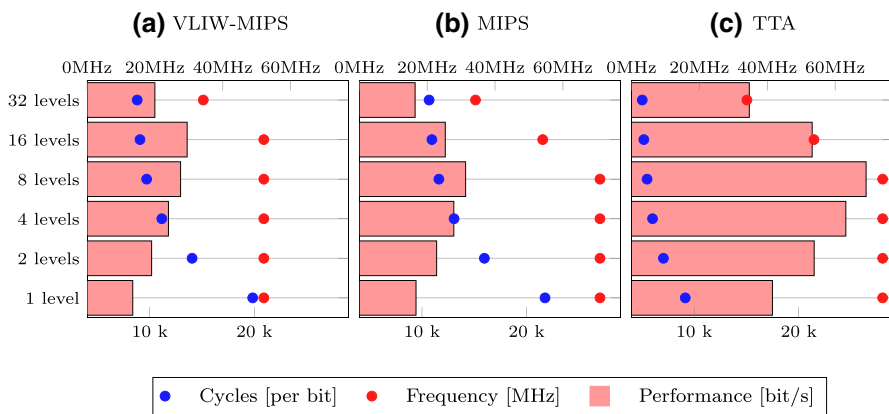


Fig. 7 Performance of the encode (ROBO: 1216 bit, see Sect. 4.1) and maximum operating frequency using different folded multiply unit configurations in **a** the VLIW-MIPS, **b** the MIPS, and **c** an exemplary TTA configuration

From this evaluation, the following conclusions can be drawn: As long as the critical path is located in the multiplier/divider unit, a reduction of the array levels increases the performance, because the increased clock frequency overcompensates the additional latency cycles of the units. When the clock frequency reaches its maximum, the performance optimum is found, because latency cycles on smaller array levels are not compensated anymore. These conclusions influence the selection of TTA configurations shown in Sect. 4.2, which are based on 32-level, 16-level and 8-level multiplier units.

4 Evaluation

In this section, the different processor architectures described in Sect. 3 are compared in terms of silicon area, power consumption, and processing performance. For the HDL synthesis, Cadence Encounter RTL Compiler (RC14.28) was used to create a gate-level netlist. An SOI CMOS technology, capable of high-temperature usage and based on transistors with a gate length of $0.18 \mu\text{m}$, was used at a corner case of 175°C at 1.62 V to determine the silicon area requirements and the critical path of the synthesized circuit. The processing performance is measured in simulation for a specific algorithm, i.e., HomePlug encoder, by multiplying the number of executing cycles with the maximum achieved frequency of the synthesized netlist. The switching activity, obtained from gate-level simulation using Questa Sim (10.6a) running the target application, was used to estimate the power consumption using Synopsys PrimeTime (2019.03) for the execution of the application.

4.1 Target Application

The target application is the PHY layer encoder of the HomePlug Alliance 1.0.1 powerline communication standard [15]. Figure 8 gives a brief overview of the application, showing blocks for header and payload data encoding. The PHY layer contains analog and digital parts, down to a mechanical interface specification for the bus connector. The scope of this work focuses on the digital domain up to a frame buffer FIFO which holds a completely encoded packet that is ready for analog transmission. Header and payload data are randomly generated input bit and are compared to the encoded result of a C reference executed on a x86 platform.

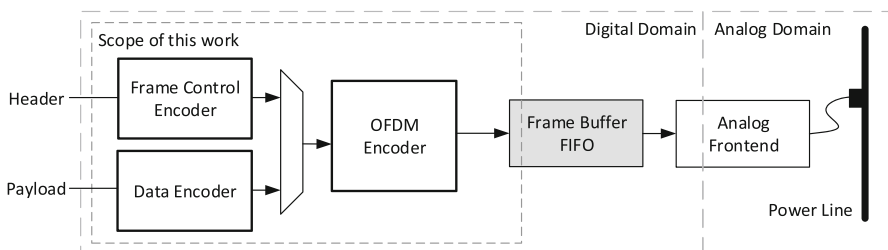


Fig. 8 Data processing block structure of the PHY encoder

The *Frame Control Encoder* block encodes header and trailer information, which have a fixed width of 25-bit and contain for example the packet type, flow control information and transmission status. The *Data Encoder* block encodes the payload data including destination and source address of a packet. The length of the payload data encoded in one transfer is variable and depends on the modulation type, the redundancy encoding for error correction and the useable frequency carriers. Here, a Reed–Solomon and convolutional encoder are used to add redundant information to help to correct bit and burst errors at the receiver. Additionally, two modulation schemes are supported, i.e., Differential Binary Phase Shift Keying (DBPSK) and Differential Quadrature Phase Shift Keying (DQPSK). The latter encodes two bit per frequency carrier and DBPSK encodes only one bit per carrier.

The last block in Fig. 8 is the OFDM Encoder. The input multiplexer sequentially passes encoded header, payload and trailer information to this block. Inside this block, data are mapped onto frequency carriers and an Inverse Fast Fourier Transform (IFFT) converts them into a time domain signal which is then stored inside the Frame Buffer FIFO. This buffer is read by the Analog Frontend, converting the digital samples into analog ones. It is ensured that the Frontend is reading the buffer faster than the encoder can write it.

The standard defines a more redundant robust OFDM (ROBO) transmission mode. It induces redundancy by time- and frequency-shifting the encoded bits four times. Therefore, a block of ROBO-encoded payload data can only have a quarter of the information bits compared to a block of regular encoded data, uses DBPSK and encoders configured to add maximum redundancy. The advantage of this additional redundancy is the possibility to recover highly distorted packets due to a very poor transmission channel characteristics, e.g., burst noise or frequency fading transmission channels. The ROBO mode is of particular interest for this work, since burst noise and frequency fading often occur under harsh environmental conditions [28]. Since the HomePlug 1.0.1 standard is designed for in-house multimedia streaming, the ROBO mode was added to ensure communication even during bad channel conditions. Since in harsh environment situations these conditions could be constant over a long period of time, we focus on this mode.

The maximum theoretical throughput, assuming no delay between two packets, is 817 kbit/s (with 1216 bit input data per packet), which is still significantly larger than typically required data rates that are at a few kbit/s. Therefore, the goal of this work is less about reaching these maximum possible throughput and more about having a reliable, stable and robust communication.

4.2 TTA Configurations

In order to cope with the large and fine-grained design space, two systematic approaches, i.e., a bottom-up and a top-down design strategy, are used to generate TTA configurations for the comparative evaluation in this paper. A selection is shown in Table 4, which presents details on the performed configuration process. According to the latency and critical path analysis of multiplier (MUL) and divider (DIV) units in Sect. 3.5, MUL/DIV level parameterizations are balanced to achieve the lowest cycle counts per target frequency in order to maximize the performance.

Table 4 Different configurations of the TTA

No.	Buses	Instr. width	Num. of inst. (Mem. size)	MUL Conf.	DIV Conf.	Additional FU	Register file (32-bit)	Clock Freq. (MHz)	Cycles ROBO	Runtime ROBO (ms)
A01	3	48 b	4874 (48 kB)	1 × 32Lev	1 × 8Lev	–	32	33.8	11,298,470	333.8
A02	3	48 b	4562 (48 kB)	1 × 32Lev	1 × 8Lev	–	32 (2r1w)	33.9	10,342,047	305.2
A03	4	64 b	4427 (64 kB)	1 × 32Lev	1 × 8Lev	1 Add	32	33.8	10,261,137	303.3
A04	4	64 b	3953 (32 kB)	1 × 32Lev	1 × 8Lev	1 Add + 1 Shift	32	33.8	9,509,078	280.9
A05	5	80 b	3882 (40 kB)	1 × 32Lev	1 × 8Lev	1 Add	32	33.8	9,487,654	280.5
A06	5	80 b	3670 (40 kB)	1 × 32Lev	1 × 8Lev	1 Add + 1 Shift	32	33.9	8,972,745	264.9
A07	5	80 b	3891 (40 kB)	1 × 32Lev	1 × 8Lev	1 Add	32 (2r1w)	34.0	8,670,779	255.0
A08	6	88 b	3617 (44 kB)	1 × 32Lev	1 × 8Lev	1 Add	32 (4r2w)	34.0	8,106,321	238.4
A09	6	88 b	3575 (44 kB)	2 × 32Lev	2 × 8Lev	1 Add	32 (4r2w)	33.6	7,613,784	226.5
A10	7	96 b	3410 (48 kB)	3 × 32Lev	2 × 8Lev	2 Add	32 (4r2w)	33.6	6,907,775	205.6
A11	7	104 b	3366 (52 kB)	3 × 32Lev	2 × 8Lev	2 Add + 1 Shift	32 (4r2w)	33.7	6,808,392	202.3
A12	8	112 b	3351 (56 kB)	3 × 32Lev	2 × 8Lev	2 Add + 1 Shift	32 (4r2w)	33.8	6,739,149	199.3
B01	3	48 b	5005 (48 kB)	1 × 16Lev	1 × 4Lev	–	32	54.4	11,264,537	206.9
B02	3	48 b	4737 (48 kB)	1 × 16Lev	1 × 4Lev	–	32 (2r1w)	54.6	10,669,448	195.3
B03	4	64 b	4613 (64 kB)	1 × 16Lev	1 × 4Lev	1 Add	32	54.0	10,554,404	195.5
B04	4	64 b	4162 (64 kB)	1 × 16Lev	1 × 4Lev	1 Add + 1 Shift	32	54.0	9,910,327	183.4
B05	5	80 b	4107 (80 kB)	1 × 16Lev	1 × 4Lev	1 Add	32	53.8	9,944,969	184.9
B06	5	80 b	3914 (40 kB)	1 × 16Lev	1 × 4Lev	1 Add + 1 Shift	32	54.2	9,433,843	174.0
B07	5	80 b	4045 (40 kB)	2 × 16Lev	1 × 4Lev	1 Add	32 (2r1w)	53.5	8,461,184	158.1
B08	6	88 b	3864 (44 kB)	1 × 16Lev	1 × 4Lev	1 Add	32 (4r2w)	54.5	8,296,435	152.1
B09	6	88 b	3779 (44 kB)	2 × 16Lev	2 × 4Lev	1 Add	32 (4r2w)	53.4	7,693,623	144.0
B10	7	96 b	3588 (48 kB)	3 × 16Lev	2 × 4Lev	2 Add	32 (4r2w)	53.2	6,986,380	131.4
B11	7	104 b	3584 (52 kB)	3 × 16Lev	2 × 4Lev	2 Add + 1 Shift	32 (4r2w)	52.9	6,943,868	131.3
B12	8	112 b	3568 (56 kB)	3 × 16Lev	2 × 4Lev	2 Add + 1 Shift	32 (4r2w)	52.8	6,820,757	129.2

Table 4 continued

No.	Buses	Instr. width	Num. of inst. (Mem. size)	MUL Conf.	DIV Conf.	Additional FU	Register file (32-bit)	Clock Freq. (MHz)	Cycles ROBO	Runtime ROBO (ms)
<i>C01</i>	<i>3</i>	<i>48 b</i>	<i>5474 (48 kB)</i>	<i>1 × 8Lev</i>	<i>1 × 2Lev</i>	<i>–</i>	<i>32</i>	<i>76.0</i>	<i>11,804,596</i>	<i>155.3</i>
<i>C02</i>	<i>3</i>	<i>48 b</i>	<i>5182 (48 kB)</i>	<i>1 × 8Lev</i>	<i>1 × 2Lev</i>	<i>–</i>	<i>32 (2r1w)</i>	<i>76.0</i>	<i>11,508,717</i>	<i>151.4</i>
<i>C03</i>	<i>4</i>	<i>64 b</i>	<i>5089 (64 kB)</i>	<i>1 × 8Lev</i>	<i>1 × 2Lev</i>	<i>1 Add</i>	<i>32</i>	<i>75.4</i>	<i>11,149,210</i>	<i>147.9</i>
<i>C04</i>	<i>4</i>	<i>64 b</i>	<i>4678 (64 kB)</i>	<i>1 × 8Lev</i>	<i>1 × 2Lev</i>	<i>1 Add + 1 Shift</i>	<i>32</i>	<i>76.0</i>	<i>10,564,695</i>	<i>139.0</i>
<i>C05</i>	<i>5</i>	<i>80 b</i>	<i>4659 (80 kB)</i>	<i>1 × 8Lev</i>	<i>1 × 2Lev</i>	<i>1 Add</i>	<i>32</i>	<i>76.0</i>	<i>10,599,395</i>	<i>139.5</i>
<i>C06</i>	<i>5</i>	<i>80 b</i>	<i>4452 (80 kB)</i>	<i>1 × 8Lev</i>	<i>1 × 2Lev</i>	<i>1 Add + 1 Shift</i>	<i>32</i>	<i>75.7</i>	<i>10,088,453</i>	<i>133.2</i>
<i>C07</i>	<i>5</i>	<i>80 b</i>	<i>4477 (80 kB)</i>	<i>2 × 8Lev</i>	<i>1 × 2Lev</i>	<i>1 Add</i>	<i>32 (2r1w)</i>	<i>74.0</i>	<i>9,118,956</i>	<i>123.1</i>
<i>C08</i>	<i>6</i>	<i>88 b</i>	<i>4336 (88 kB)</i>	<i>1 × 8Lev</i>	<i>1 × 2Lev</i>	<i>1 Add</i>	<i>32 (4r2w)</i>	<i>76.0</i>	<i>9,273,170</i>	<i>122.0</i>
<i>C09</i>	<i>6</i>	<i>88 b</i>	<i>4049 (44 kB)</i>	<i>4 × 8Lev</i>	<i>2 × 2Lev</i>	<i>1 Add</i>	<i>32 (4r2w)</i>	<i>73.5</i>	<i>8,394,866</i>	<i>114.2</i>
<i>C10</i>	<i>7</i>	<i>96 b</i>	<i>3976 (48 kB)</i>	<i>3 × 8Lev</i>	<i>2 × 2Lev</i>	<i>2 Add</i>	<i>32 (4r2w)</i>	<i>74.6</i>	<i>7,581,973</i>	<i>101.6</i>
<i>C11</i>	<i>7</i>	<i>104 b</i>	<i>3975 (52 kB)</i>	<i>3 × 8Lev</i>	<i>2 × 2Lev</i>	<i>2 Add + 1 Shift</i>	<i>32 (4r2w)</i>	<i>74.3</i>	<i>7,483,641</i>	<i>100.7</i>
<i>C12</i>	<i>8</i>	<i>112 b</i>	<i>3984 (56 kB)</i>	<i>3 × 8Lev</i>	<i>2 × 2Lev</i>	<i>2 Add + 1 Shift</i>	<i>32 (4r2w)</i>	<i>74.3</i>	<i>7,360,815</i>	<i>99.0</i>

Italicized rows show those configurations selected for analysis in Sect. 4.3. All MUL/DIV units use a folded structure

Therefore, 32-level MUL and 8-level DIV, 16-level MUL and 4-level DIV, and 8-level MUL and 2-level DIV units are used together for maximum clock frequencies of 34.0 MHz, 54.6 MHz, and 76.0 MHz, respectively (see Table 3). These target clock frequencies are represented by the three groups A, B, and C in Table 4, and within each group, the entries are sorted by their amount of buses and their instruction width in bit in ascending order and are denoted by #01 to #12 in the following. Higher clock frequencies and higher performance are limited by paths through the TTA ALU and interconnection network, so lower-level MUL/DIV configurations do not achieve higher performance and are therefore not considered.

For the bottom-up approach, MIPS-like and VLIW-MIPS-like resource configurations (#01 and #05, see Sect. 3.4) are used as a starting point and are iteratively extended with more functionality in order to increase the performance, i.e., decrease the number of clock cycles. Based on the utilization of certain parts of the hardware, bottlenecks were found and removed with the TTA toolchain from [19], e.g., by adding further parallel buses and replicating frequently used adder and shifter resources by additional functional units (configurations #03, #04, #06). Immediate operands are shortened to 6-bit per bus, except for one bus that provides 13-bit jump target addresses to the GCU.

The top-down approach is adapted from [17] and is characterized by starting from a canonical VLIW-style connected TTA, i.e., for each operand and result port of a functional unit, a dedicated transport bus and a RF read or write port are reserved. This allows to exploit any possible parallel operation execution offered by the available resources, but also has the largest instruction width and RF port and interconnection overhead. For the initial VLIW configuration, the cycles required for memory accesses and computations are balanced by adding an additional adder or combined adder/shifter FU. Then, in order to trade off performance against architectural area and power efficiency, bus resources and RF ports that are never or rarely used in parallel are merged (configurations #02, #07, #08, #09, #10, #11 and #12). Therefore, for configurations #10 to #12, only four-read two-write port (4r2w) RF instances are used, whereas a comparable three-issue VLIW machine would require a six-read and three-write port RF. The relaxed port requirements improve RF timing as well as area and power efficiency.

In general, multiple of the folded multi-cycle MUL and DIV units can be used to increase performance. The specific number of units is limited by the amount of parallelism offered by the application. In this paper, four independent multiplication and two division operations can be executed in parallel overlapping execution in the IFFT block and the Reed–Solomon block of the application, respectively. However, the exploitable parallelism of the TTA architecture configuration and the resulting application code schedule affect the number of units effectively used in parallel. For the configurations in Table 4, the number of MUL and DIV units is selected to obtain a high silicon area efficiency, i.e., the increase in total processor area due to additional units is overcompensated by the performance gain in terms of cycle count drop. When considering the parameterization of MUL and DIV levels, the configurations with 8-level MUL and 2-level DIV units provide the highest performance, as the higher clock frequency overcompensates the increased number of execution cycles. For example, C01 requires $1.05\times$ the ROBO mode clock

cycles compared to B01, but has a $1.4\times$ higher clock frequency, which in turn results in $1.33\times$ higher performance due to reduced ROBO mode runtime. Therefore, for conducting the trade-off analysis between high performance, small silicon area and high energy efficiency in Sect. 4.3, the TTA configurations with the highest possible performance are selected from group C, i.e., C09 to C12. The baseline configurations C01 and C02, which are comparable to other single-issue processors, are included in order to span the TTA configuration corner cases. When trading off for small silicon area, configurations can be selected from groups A and B due to more MUL and DIV array levels, which correspond to less latency cycles and denser instruction schedules. Therefore, smaller memory blocks with only 4096 instruction lines instead of 8192 are required. This reduces the instruction memory size to 32 kB (A04), 40 kB (B06, B07) and 44 kB (B08, B09) compared to the baseline configurations C01 and C02, which require 48 kB. Additionally, A10 and B10 are included for trade-off analysis because these configurations feature the highest energy efficiency of the evaluated architectures.

Finally, it should be mentioned that TCE offers great flexibility to add specialized custom FUs to TTAs according to the ASIP concept. Especially for the FFT/IFFT kernel also used in the application in this paper, custom accelerators for TTA have been realized in [25, 37], which exploit streaming-like data flow in the FFT calculation scheme for even higher performance. However, such specializations are out of the scope of this paper in order to allow an undistorted comparison with the other processor architectures with non-custom operations only.

4.3 Trade-Off Analysis

In the following, the performance, area and power trade-offs are analyzed for the ARM, MIPS, VLIW-MIPS and TTA processor architectures. Furthermore, a comparison of those configurations that show Pareto-optimal power and silicon area efficiency is performed. The performance and power relations between the architectures were found using the aforementioned robust mode of the target application in Sect. 4.1.

Figure 9 depicts the maximum performance and the maximum clock frequency for each architecture configuration. Following conclusions can be drawn:

- The highest performance is provided by the TTA configurations in group C using an 8-level multiplier and 2-level divider units. The clock frequency is approximately 75 MHz, which is the highest, together with the MIPS with the same multiplier/divider configuration, among the evaluated architectures.
- To achieve higher performance, however, higher frequency is not the only design point. For example, TTA configuration C01 has comparable data path resources to the MIPS and a similar clock frequency but 30% more performance. The performance comprises not only operating frequency, but also the number of executed cycles for encoding one data block, which is reduced on the TTA due to register bypassing and temporal data registration in functional units.
- The single-issue RISC processor implementations ARM CM3 and MIPS show a similar performance, which emphasizes the similarities in their ISA, pipeline

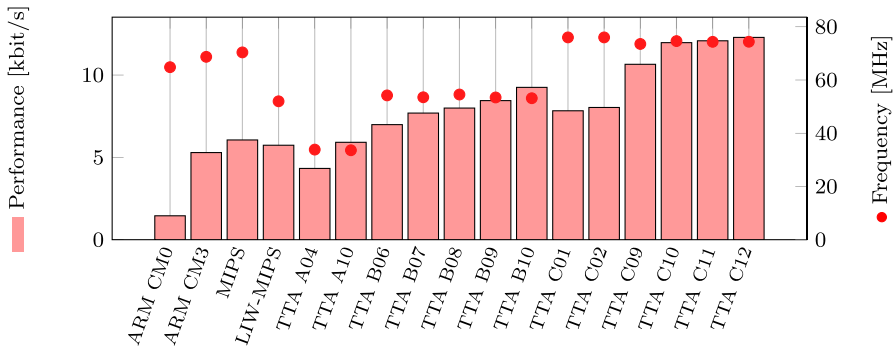


Fig. 9 Performance and frequency of the evaluated architectures

concept, and control flow realization. Due to the lack of a dedicated hardware division unit, the ARM CM0 has to emulate divisions in software, which causes a significant performance loss ($8.5\times$ smaller than TTA configuration C12). Regarding the control flow, 76.3 % of MIPS *branch delay slots* are filled in the static code analysis, and the MIPS dynamic Instructions per Cycle (IPC) is at 0.81 with critical path-balanced multiplier and divider latency (see Sect. 3.5).

- Despite the second issue slot, the VLIW-MIPS does not achieve performance improvements compared to MIPS, because both issue slots cannot be efficiently used in parallel by the application code. With critical path-balanced multiplier and divider latency, the dynamic IPC of the VLIW-MIPS reaches 0.95 and thus, is only 17% higher than for the MIPS, which is mitigated again by the lower clock frequency due to more complex inter-issue slot forwarding mechanism and hazard detection. Also, the presence of a second, mostly unfilled *branch delay slot* has a negative impact on the IPC. With a single *branch delay slot*, the IPC could be increased to 1.00. Furthermore, an improved instruction ordering algorithm in the currently used VLIW-MIPS instruction scheduler could increase the number of parallel executed operations.
- Among the group C of the TTA configurations, there is a variance of up to $1.6\times$ in performance at approximately the same clock frequency, which is caused by the increased number of functional units and interconnection buses. More resources achieve higher performance, but also increased circuit complexity.
- Despite the lower cycle count, TTA groups A and B show lower performance than the C group because the multiplier/divider folded arrays with more levels unnecessarily limit the clock frequency due to longer critical paths. This supports the conclusion from Sect. 3.5 that for any architecture organization, the parameterization of multi-cycle arithmetic co-processors should be carefully balanced.

The required silicon area for implementing the different processor architectures is shown in Fig. 10. Following conclusions can be drawn:

- The core logic area of the architectures is comparable and mostly varies with the number and configuration of the large arithmetic units, i.e., multiplier and

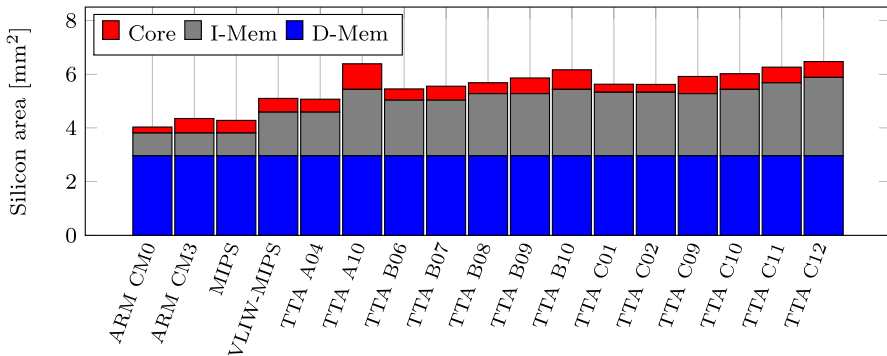


Fig. 10 Silicon area of instruction memory, data memory and core

divider arrays, as indicated in Table 3. The smallest core is the ARM CM0 with no hardware divider and a 32-level multiplier. The largest core is the TTA configuration A10 with two 8-level dividers and three 32-level multipliers, which has a $4.26\times$ larger core area.

- The instruction memory (I-Mem) and the data memory (D-Mem) of the processors are implemented using multiple single-port RAM macroblocks from the technology library. The size of the I-Mem is defined by the architecture's instruction word width and the number of instructions in the application code and, therefore, varies with the architectures. On the contrary, the D-Mem size is related to the 32-bit data path width and the processor-independent data storage required by the application, which is fixed to 64 kB for all evaluated architectures.
- In all processor configurations, I-Mem and D-Mem require more than 80% of the total silicon area. Therefore, the total silicon area differs only by a factor of $1.6\times$ between ARM CM0 and TTA A10, although the core size difference is $4.26\times$. The ARM processors provide a high code density with 16-/32-bit wide instructions, therefore the I-Mem size is small (16 kB) compared to the D-Mem (64 kB). The TTAs, however, need large instruction memory sizes of up to 56 kB due to the wide instruction words (48- to 112-bit). Methods like variable length encoding [33] could be applied to mitigate the effect of wide instructions on the memory size.
- VLIW-MIPS instructions are 64-bit wide, so the I-Mem size needs to be doubled compared to the MIPS. Due to the poor parallel usage of both issue slots, the code density in the VLIW-MIPS application code is lowered and an I-Mem size similar to the MIPS cannot be achieved.
- In general, the cycle- and bus slot-based TTA instruction encoding provides less code density than RISC-style encoded instructions. For example, the central IFFT loop of the application requires 70 mixed 16-/32-bit instructions on the ARM CM3 architecture, which correspond to 200 bytes of code. On the ARM CM3-comparable TTA configuration C01, 76 48-bit instructions are required, resulting in 456 bytes of code and thus, $2.25\times$ less code density, which is directly related to three times larger I-Mem over the complete application. In

total, 13% more total silicon area is consumed due to increased instruction memory requirements. Because the TTA does not use a *stalling* mechanism, 9 out of the 76 IFFT instructions are NOP because of multiplier latency. In the presence of multi-cycle functional units like multipliers and dividers, the implementation of such a *stall* mechanism would avoid empty NOP instructions in the schedule and improve the code density. However, this also requires modifications of the current TCE scheduling concept in the compiler, which were not available for this paper.

In Fig. 11, the power requirements of the evaluated architectures are shown. These are obtained from switching activity simulation of the application in ROBO mode at the maximum operation clock frequency per architecture. Following conclusions can be drawn:

- The power consumption shows a close correlation between the frequency and the silicon area of the processor, but also with the achieved performance. The TTA configuration C12 shows the highest performance and also the highest power consumption, which is $3.6\times$ higher than the ARM CM0 power consumption. Absolute power values might be an important consideration for power-constrained systems, e.g., due to limited heat dissipation. An evaluation of the computational energy efficiency (power normalized to performance) is given further below in this section.
- Although memories require more than 80% of the silicon area, the internal processes in the I-Mem and D-Mem SRAM blocks during reading/writing memory contribute only less than 60% to the total power consumption. The core logic requires a significant amount of power for computations and data transport within the data path, e.g., between RF and ALU.
- Both ARM CM0 and ARM CM3 have a substantially lower I-Mem power consumption than the other architectures. This is due to the possibility of

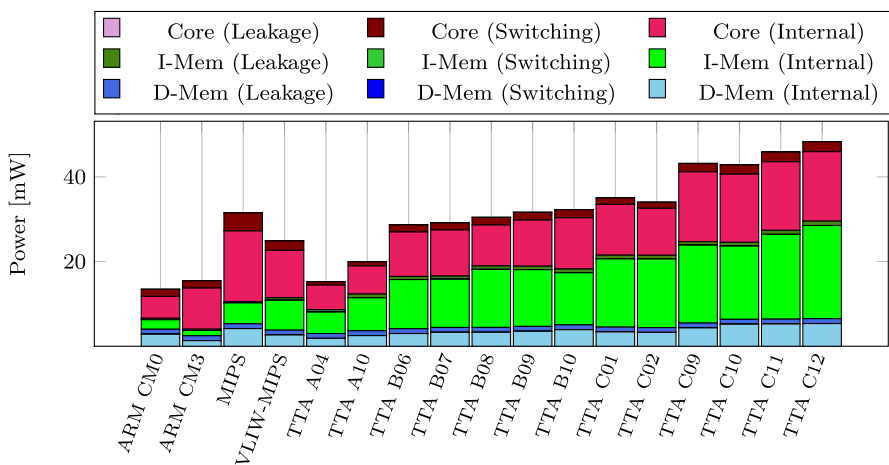


Fig. 11 Power Consumption of instruction memory, data memory and core for ROBO

fetching two 16-bit short Thumb instructions with one memory access, but also due to instructions like `ldmia` and `stmia` (see Sect. 3.1), which combine multiple operations running over multiple cycles and require to fetch only a single instruction. Therefore, the I-Mem access activity and thus the power consumption is minimized.

- At the maximum operation frequency, the power consumption is totally dominated by cell-internal dynamic power and dynamic switching power on the connection nets. Even though the values have been generated for a corner case of 175 °C, virtually no leakage appears, which is one of the strengths of SOI technology in high-temperature harsh environments.

To better compare the different performances of the processor architectures, an area efficiency metric is calculated by normalizing the silicon area requirements to the reached throughput for the different processor architectures. For energy efficiency, the power requirements are normalized to the reached throughput, resulting in the energy requirements for encoding a single bit. There are other, more general architecture-related metrics commonly applied to quantify efficiencies, e.g., energy efficiency in MOPS/mW. However, the identical application is the relevant benchmark for all architecture organizations in this paper and thus, it was decided to keep a direct relation to the application use case. Results are shown in Fig. 12 with the optimum in the bottom left corner. The performance increases or power and area requirements drop the closer an architecture gets to the optimum. It should be mentioned that Fig. 12 contains all TTA configurations generated for this paper in order to allow a valid Pareto analysis. The configurations not highlighted in Table 4 are depicted as gray dots (TTA others).

The TTA configuration C10 yields the highest area efficiency. With this configuration, the hardware is utilized with a high processing efficiency and fewer idling components than by configurations A04 and A10, which show a low area efficiency. This is due to the different level configurations of the multiply and divide units, achieving higher frequencies for the more area-efficient architectures. This results in a high parallelization of the evaluated algorithm with high utilization of functional units and low idle cost for other components during the calculation delay of the co-processors. The advantage of the TTA against ARM and MIPS cores is the exposed parallelism of the evaluated application. The functional units of the TTA are controlled by short encodings for the corresponding sockets and the total instruction length increase is small for multiple multiply and divide units. For the MIPS, additional issue slots would allow more parallelism, but excessively increase the instruction memory size and therefore the silicon area requirements. The implemented multiply and divide units build up a range of different versions for MIPS and VLIW-MIPS with different advantages, as performance, area and power requirements can exchange for each other. The comparison between MIPS and ARM CM3 also shows a trade-off of area and power efficiency. As the area efficiency drops, the power efficiency increases for some configurations. In Fig. 12, there are three Pareto-optimal TTA configurations regarding area and energy efficiency. These are the TTA A10, TTA B10, and TTA C10 where no improvement on one dimension can be achieved without trading for another

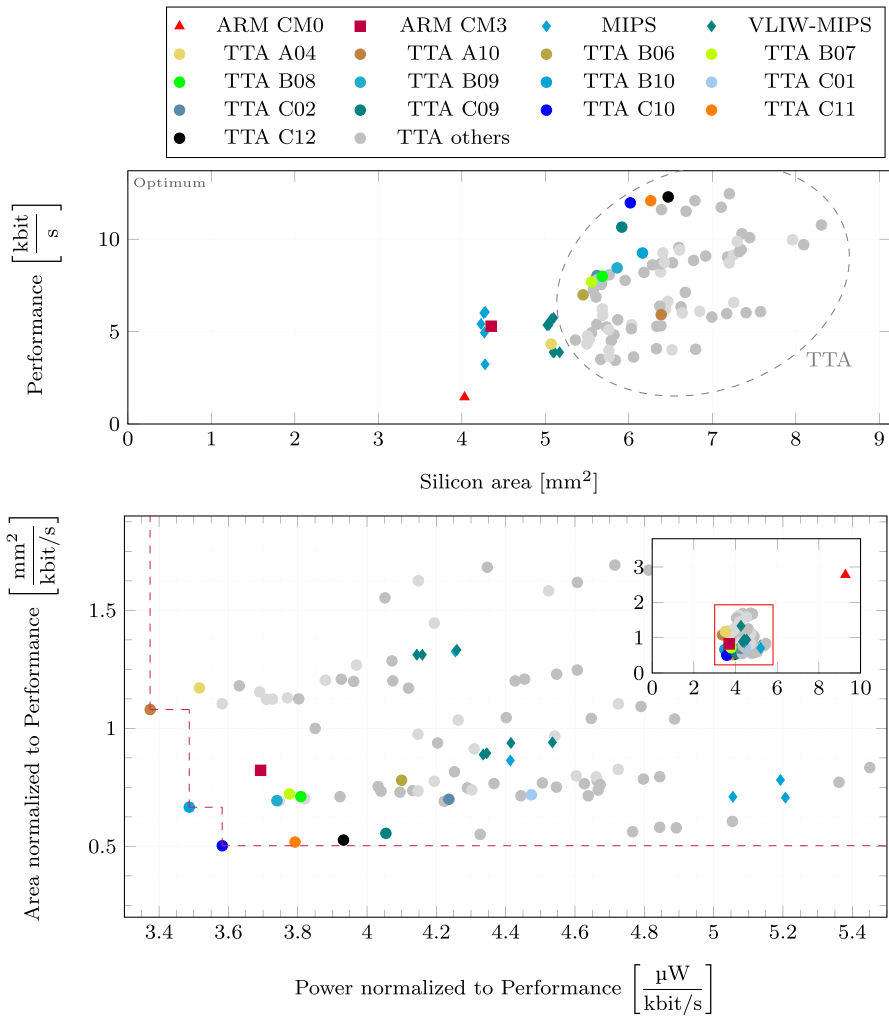


Fig. 12 Top: Silicon area and processing performance of the evaluated architectures. Bottom: Efficiency: Silicon area and power normalized to performance. Top Right gives an overview including the ARM CM0

drawback. These configurations differ in their multiplier and divider level configuration. Less levels lead to a higher clock frequency, more performance and area efficiency (C10), whereas more levels lead to a higher energy efficiency at reduced clock frequency and performance (A10).

5 Conclusion

In harsh environments, the design space for microcontrollers is large and allows to achieve different goals like minimum silicon area, minimal energy consumption or high processing performance. A design space exploration for four RISC processor architectures, i.e., ARM Cortex-M0, ARM Cortex-M3, MIPS, VLIW-MIPS, as well as different configurations of a transport-triggered architecture (TTA) has been performed in this paper. Using a 0.18 μm SOI CMOS technology for high temperature and a powerline physical layer encoder application as a case study, the different processor architecture organizations have been compared regarding the efficient use of silicon area and power. While the RISC architectures mainly differ in the number of parallel issue slots and pipeline stages, as well as in their control flow properties (*branch delay* and *stall cycles* etc.), the TTA presents an exposed data path concept, which allows to optimize the usage efficiency of register file and interconnection resources for the target application.

The corner cases of the design space exploration are the ARM Cortex-M0 and the TTA configuration C12. On the one hand, the ARM processor requires the smallest silicon area and power consumption. Due to a high code density, the instruction memory size is small compared to other architectures, but the performance is also the lowest due to the lack of a dedicated hardware division unit. On the other hand, the TTA has the highest performance due to the better exploitation of the application code parallelism and parallel utilization of processing resources, e.g., using multiple multiplier and divider units in overlapping execution. As a drawback, the TTA programming model requires large instruction widths of up to 112 bit, increasing the silicon area of the instruction memory and the power consumption. However, mechanisms like instruction register files, variable length encoding, and loop buffers [23, 24, 33] could be applied in future work to mitigate the effects on area and power consumption. Compared to the ARM Cortex-M0, the transport-triggered architecture configuration C12 has an $8.5\times$ higher performance and $2.4\times$ higher computational energy efficiency at a $1.6\times$ larger total silicon area, showing the considerable range of design trade-offs for different architectures. It can be concluded that the flexible configuration of the TTA data path is more suitable to generate a large variety of trade-off possibilities and opens up a fine-grained design space to meet the desired application constraints.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Amirshahi, P., Navidpour, S.M., Kavehrad, M.: Performance analysis of uncoded and coded OFDM broadband transmission over low voltage power-line channels with impulsive noise. *IEEE Trans. Power Deliv.* **21**(4), 1927–1934 (2006)
2. ARM Limited: Cortex-M3 Technical Reference Manual, ARM DDI 0337E edn. (2006)
3. ARM Limited: Cortex-M0 Technical Reference Manual, ARM DDI 0432C edn. (2009)
4. ARM Limited: ARM v7-M Arch. Reference Manual, ARM DDI 0403E edn. (2014)
5. ARM Limited: ARM v6-M Arch. Reference Manual, ARM DDI 0419D edn. (2017)
6. Bannatyne, R., Gifford, D., Klein, K., McCarville, K., Merritt, C., Neddermeyer, S.: Creation of an ARM® Cortex®-M0 microcontroller for high temperature embedded systems. In: *HiTEN*, vol. 2017, pp. 31–35 (2017)
7. Braun, S., Kordas, N., Utz, A., Kappert, H., Kokozinski, R.: Fully integrated sensor electronics for inductive proximity switches operating up to 250 C. In: *Additional Conference (Device Packaging, HiTEC, HiTEN, & CICMT) 2019(HiTen)*, pp. 112–116 (2019)
8. Chen, W., Sadana, D.K., Taur, Y.: SOI CMOS structure, patent (1998)
9. Corporaal, H.: *Microprocessor Architectures: From VLIW to TTA*. Wiley, Hoboken (1997)
10. Freescale: MC9S12G (2017). <https://www.nxp.com>
11. George, T., Son, K.A., Powers, R.A., Del Castillo, L.Y., Okojie, R.: Harsh environment microtechnologies for NASA and terrestrial applications. In: *IEEE Sensors* (2005)
12. Gesper, S.: *Implementierung eines VLIW-MIPS-Prozessors für Hochtemperaturanwendungen mit Compilerunterstützung*, Master Thesis, Leibniz Universität Hannover. Master's thesis (2018)
13. Gesper, S., Weißbrich, M., Nolting, S., Stuckenberg, T., Jääskeläinen, P., Blume, H., Payá-Vayá, G.: Evaluation of different processor architecture organizations for on-site electronics in Harsh environments. In: *Embedded Computer Systems: Architectures, Modeling, and Simulation*. Springer (2019)
14. Hennessy, J.L., Patterson, D.A.: *Computer Organization and Design: The Hardware/Software Interface*. Elsevier, Hoboken (2014)
15. HomePlug Powerline Alliance: HomePlug 1.0 Specification. Standard, HomePlug Power Alliance, United States, CA (2001)
16. Honeywell: HT 83C51 (2011). <https://aerospace.honeywell.com>
17. Jääskeläinen, P., Tervo, A., Payá-Vayá, G., Viitanen, T., Behmann, N., Takala, J., Blume, H.: Transport-triggered Soft Cores. In: *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 83–90 (2018)
18. Jääskeläinen, P., Kultala, H., Viitanen, T., Takala, J.: Code density and energy efficiency of exposed datapath architectures. *J. Signal Process. Syst.* **80**, 49–64 (2015)
19. Jääskeläinen, P., Viitanen, T., Takala, J., Berg, H.: HW/SW Co-design toolset for customization of exposed datapath processors. In: *Computing Platforms for Software-Defined Radio*. Springer (2017)
20. Lele, C., Siohan, P., Legouable, R., Javaudin, J.: Preamble-based channel estimation techniques for OFDM/OQAM over the powerline. In: *2007 IEEE International Symposium on Power Line Communications and Its Applications*, pp. 59–64 (2007)
21. Microchip: PIC24HJ32GP202/204 & PIC24HJ16GP304 (2011). <https://www.microchip.com>
22. MIPS Technologies: *Programmers Volume II-A: The MIPS32 Instruction Set*
23. Multanen, J., Kultala, H., Jääskeläinen, P.: Energy-delay trade-offs in instruction register file design. In: *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pp. 1–7. IEEE (2018)
24. Multanen, J., Viitanen, T., Linjamäki, H., Kultala, H., Jääskeläinen, P., Takala, J., Koskinen, L., Simonsson, J., Berg, H., Raiskila, K., et al.: Power optimizations for transport triggered SIMD processors. In: *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pp. 303–309. IEEE (2015)
25. Patyk, T., Guevorkian, D., Pitkänen, T., Jääskeläinen, P., Takala, J.: Low-power application-specific FFT processor for LTE applications. In: *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pp. 28–32. IEEE (2013)
26. Pirsch, P.: *Architekturen der digitalen Signalverarbeitung*. Springer, Berlin (2013)
27. Stuckenberg, T., Gottschlich, M., Nolting, S., Blume, H.: Design and optimization of an ARM cortex-M based SoC for TCP/IP communication in high temperature applications. In: *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 169–183. Springer, Cham (2019)

28. Tan, Z., Cheng, Y., Liu, H., Ma, Z.: Noise modelling for power line communication in Harsh environment. In: 2017 4th International Conference on Information Science and Control Engineering (ICISCE), pp. 1559–1563 (2017)
29. Tekmos Inc.: TK89h51 Microcontroller (2018). <https://www.tekmos.com>
30. Texas Instruments: TI SM320F2812-HT (2011). <https://www.ti.com>
31. Texas Instruments: TI SM320f28335-HT (2014). <https://www.ti.com>
32. Texas Instruments: TI SM470r1b1m-HT (2015). <https://www.ti.com>
33. Viitanen, T., Helkala, J., Kultala, H., Jääskeläinen, P., Takala, J., Zetterman, T., Berg, H.: Variable length instruction compression on transport triggered architectures. *Int. J. Parallel Program.* **46**, 1283–1303 (2018)
34. Vorago: VA10800 (2018). <https://www.voragotech.com>
35. Waterman, A.S.: Design of the RISC-V instruction set architecture. Ph.D. thesis, UC Berkeley (2016)
36. XFab Silicon Foundries: 0.18 Micron Modular BCD-on-SOI Technology (2020)
37. Žádník, J., Takala, J.: Low-power programmable processor for fast Fourier transform based on transport triggered architecture. In: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1423–1427. IEEE (2019)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.