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

Sven Gesper[1][0000–0002–3570–1638], Moritz Weißbrich[1][0000–0002–6647–5215], Stephan Nolting[1], Tobias Stuckenberg[1][0000–0002–9245–3167], Pekka Jääskeläinen[2][0000–0001–5707–8544], Holger Blume[1][0000–0002–0640–6875], and Guillermo Payá-Vayá[1][0000–0003–3503–8386]

[1] Institute of Microelectronic Systems, Leibniz Universität Hannover, Appelstr. 4, 30167 Hannover, Germany
gesper@ims.uni-hannover.de
[2] Customized Parallel Computing group, Tampere University, Tampere, Finland

Abstract. Microcontroller units used in harsh environmental conditions are manufactured using large semiconductor technology nodes in order to provide reliable operation, even at high temperatures or increased radiation exposition. These large technology nodes imply high gate propagation delays, drastically reducing the system's performance. When reducing area costs and power consumption, the actual processor architecture becomes a major design point. Depending on the application characteristics (i.e., inherent data parallelisms, type of arithmetic, ...), several parameters like data path width, instruction execution paradigm, or other architectural design mechanisms have to be considered. This paper presents a design space exploration of five different architectures implemented for a 0.18 µm SOI CMOS technology for high temperature using an exemplary case study from the fields of communication, i.e., Reed-Solomon encoder. For this algorithm, an application-specific configuration of a transport-triggered architecture has 37.70 x of the performance of a standard 8-bit microcontroller while the silicon area is increased by 4.10 x.

Keywords: ASIC · Application Specific Processors · Design Tradeoff Analysis · Harsh Environment · MIPS · Processor Architecture Organization · Transport-Triggered Architecture · VLIW

1 Introduction

Automotive and aerospace applications with on-site microcontroller-like systems, which provide continuous maintainability and thus, flexibility, are an emerging field of control engineering. Exemplary systems are motor control units for cars, full authority digital engine controls for piston engines, or satellite arbitration systems. The electronic components have to ensure reliable operation even in harsh environmental situations, such as high temperature or increased radiation. Hence, the integrated circuits are manufactured using very large technology

nodes and silicon on insulator (SOI) stacks to reduce leakage current and latch-up effect probability [1]. As a drawback, these large technology nodes only provide a moderate operating frequency, reducing the overall system performance. Due to the large silicon structure, the number of transistors and consequently the circuit complexity on a die is limited. Furthermore, the power consumption is restricted for embedded applications. Because of these limitations, the according system's processor architecture organization has a high impact on the overall efficiency. The design space of those architectures includes numerous parameters. Some of the most significant parameters are data path width and microarchitecture organization. The 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, which are also directly connected to the architectural design organization, i.e., RISC- or CISC-like design concept. A completely different design concept is presented by the transport-triggered architecture [3]. This architecture performs instructions as side effects to move operations. All of these parameters highly influence the processor's performance as well as silicon area and energy requirements.

In this paper, five different processor architecture organizations were implemented in VHDL and optimized for an exemplary 0.18 µm high-temperature CMOS technology. The resulting implementations, including an application-optimized transport-triggered architecture, were compared in terms of processing performance, silicon area and power consumption using an exemplary case study, i.e., Reed-Solomon encoder. This paper is organized as follows: In Section 2, exemplary commercial architectures for harsh environment are compared. Section 3 describes the features of the implemented architectures. The evaluation of these architectures using the aforementioned case study is given in Section 4. Finally, a conclusion is drawn in Section 5.

2 Related Work

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 done by a multi-cycle structure. Deeper pipelined architectures 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 [20] also offers a floating-point unit. However, the dynamic range and resolution of floating-point operations is rarely required in embedded applications. The available space for programs in the instruction memory ranges from 2 kB up to 4 MB when using external Flash memory. This exemplary portfolio shows 8-, 16- and 32-bit architectures, allowing a wide spectrum of applications being efficiently implemented.

Table 1: Commercial Architectures for Digital Signal Processing in High Temperature Environment (T = Temperature, MUL = Multiplier, DIV = Divider, FPU = Floating-Point Unit).

Company	Model	Features	T_{max}[°C]	f_{max} [MHz]	Instruction Memory
TI	[20]	32-bit, RISC, 8-stage pipeline, MAC, FPU	125/210 °C	150/100	Flash 512 kB
TI	[19]	32-bit, RISC, 8-stage pipeline, MAC	220 °C	150	128 kB (ext. Flash 4 MB)
TI	[21]	16-/32-bit, RISC, 3-stage pipeline, MUL	220 °C	60	64 kB (ext. Flash 1 MB)
Tekmos	[17]	8-bit, RISC, 2 to 3-stage pipeline, DIV & MUL	210 °C	16	2 kB (ext. 64 kB)
Honeywell	[8]	8-bit, RISC, 2 to 3-stage pipeline	225 °C	16	64 kB
VORAGO	[22]	32-bit, RISC, 3-stage pipeline, MUL	200 °C	50	128 kB
Microchip	[11]	16-bit, CISC, 2-stage pipeline, DIV & MUL	150 °C	80	16 kB / 32 kB
Freescale	[4]	16-bit, RISC, multi-cycle, DIV & MUL	150 °C	25	16 kB

Table 2: Distinctive Architectural Features of the Presented Cores (Regs/R = Register(s), M = Memory, FU = Functional Unit).

CORE	Data Word Size, Design	Regs.	Type	Instruction Execution	Multiplier/Divider Unit (Latency)
AVR8	8-bit, RISC	32	R ⇔ R	pipelined (2 stages)	No (-)
NEO430	16-bit, CISC	16	R/M ⇔ R/M	multi-cycle (4..11 cycles)	Yes (16 cycles)
MIPS32	32-bit, RISC	32	R ⇔ R	pipelined (5 stages)	Yes (1..32 cycles)
VLIW-MIPS	32-bit, RISC	32	R ⇔ R	pipelined (4 stages)	Yes (1..32 cycles)
TTA	32-bit (variable)	var.	R/FU ⇔ R/FU	transport-triggered	Yes (1..32 cycles)

3 Processor Architecture Organizations

In this paper, five different processor architectures were implemented in VHDL and evaluated for the purpose of representing a wide variety of different design concepts. These are an AVR8-compatible processor, the NEO430, a MIPS32-compatible processor, a VLIW-MIPS processor with two issue slots. Moreover, different configurations of a transport-triggered architecture (TTA) [10] were also evaluated. The main features of these five cores are summarized in Table 2. The architecture organizations vary in their data width (8/16/32-bit), instruction set architecture (ISA) principles (RISC/CISC), instruction execution paradigm (pipeline/multi-cycle), number of registers and presence of a dedicated hardware accelerator for integer division and multiplication. Each processor is programmable in C language and comes with an according LLVM or GCC based toolchain for compiling and assembling.

3.1 2-Stage Pipeline AVR8 (8-bit) Processor

The implemented AVR8 processor features a 8-bit architecture with an ISA compatible to the AT90S8515 [12]. The small amount of different instructions

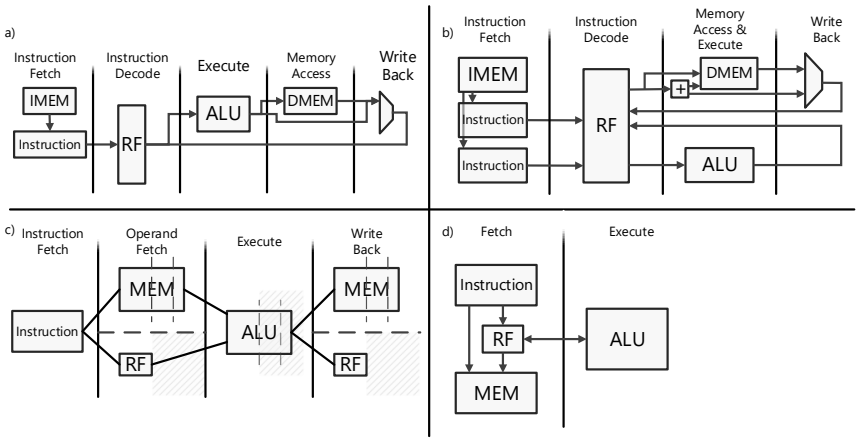

Fig. 1: Simplified architecture overview: a) MIPS32, b) VLIW-MIPS, c) NEO430, and d) AVR8.

defines a RISC architecture with 16-bit instruction words. These are executed in a two-stage pipeline (see Fig. 1d). Each instruction allows up to two operands, making the AVR a two operand machine. The first operand is always one of 32 registers and serves as data source and destination for the actual operation (e.g., R1 + R2 → R1: `ADD R1, R2`). The second operand may also be an immediate (e.g., R1 + Imm → R1: `ADIW R1, 0xF`). Complex operations like division or multiplication have to be emulated in software due to the lack of dedicated hardware. The processor implements a Harvard architecture, so the memories for storing data and instructions use separated buses, memories and address spaces. Data and instruction memory can be up to 16 kB and 8 kB, respectively.

3.2 Multi-Cycle NEO430 (16-bit) Processor

The MSP430 [18] compatible NEO430 [14] implements a 16-bit data path. The 27 instructions of the processor can directly operate on data from the data memory as well as data from the internal register file (16 entries). Just like the AVR8, the NEO430 is a two operand machine (e.g., M[R1] + R2 → R2, R1 = R1 + 1: `ADD @R1+, R2`). The instruction encoding is variable as one instruction is built of one to three 16-bit words. Due to the multi-cycle architecture, instruction execution is split into 4 to 11 cycles (see Fig. 1c). The complex operand addressing modes and the variable instruction encoding and execution cycles make the processor a CISC-like architecture. The processor includes a serial multiplier and divider unit which takes 16 cycles per operation. Because of its Von-Neumann architecture organization, the processor has a shared bus for data and instruction memory using a unified address space. However, data is stored in separated memory

instances, e.g, ROM for I-Mem and RAM for D-Mem. Data and instruction memory can be up to 28 kB and 32 kB, respectively.

3.3 5-Stage Pipeline MIPS32 (32-bit) Processor

The implemented MIPS32, based on [13], is a 32-bit architecture and has 5 pipeline stages which support hazard resolution (see Fig. 1a). 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. 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 → R3: `ADD R3, R1, R2`). A multiply and accumulate unit as well as a divider unit are available in the execution stage. Those hardware units are implemented using a configurable processing latency of 1 to 32 cycles at design time (see Section 3.6). Data and instruction memory use independent 32-bit address spaces with byte-wise alignment.

3.4 4-Stage Pipeline VLIW-MIPS (32-bit) Processor

The VLIW-MIPS processor is derived from the aforementioned MIPS32 [6]. In contrast to the MIPS32, the VLIW-MIPS uses two parallel issue-slots (see Fig. 1b). Thus, the 64-bit instruction word contains two separate instructions based on the ISA of the MIPS32. Instructions allocated in the first issue-slot are capable of performing memory accesses, while the other one is designated for arithmetic and logic operations. However, the first issue-slot is still able to perform `add` or `sub` instructions by using the adder which was initially only implemented for calculation of base-offset memory addresses. The Execute stage and Memory-Access stage of the MIPS32 are combined in the first issue-slot, while the Memory-Access stages is omitted in the second issue-slot. With this reduction down to 4 pipeline stages, the complexity of forwarding is reduced. The parallel execution of memory accesses along with arithmetic or 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 parallelize-able memory instructions is about 35% (based on SPEC CPU2006 benchmark [7]). Because of data dependencies within an application, the two issues cannot always be fully utilized.

3.5 Transport-Triggered Architecture (32-bit) Processor

The transport-triggered architecture (TTA) can be described as an exposed data path processor. Functional units are connected through a programmable interconnect, which creates a programmable data path [3, 10]. In contrast to describing specific computational operations, the instruction words contain configurations for the interconnections between the functional units. Operands are transferred to a functional unit through programmable sockets, which are basically switches to connect a unit to a set of buses. A unit's operation is triggered

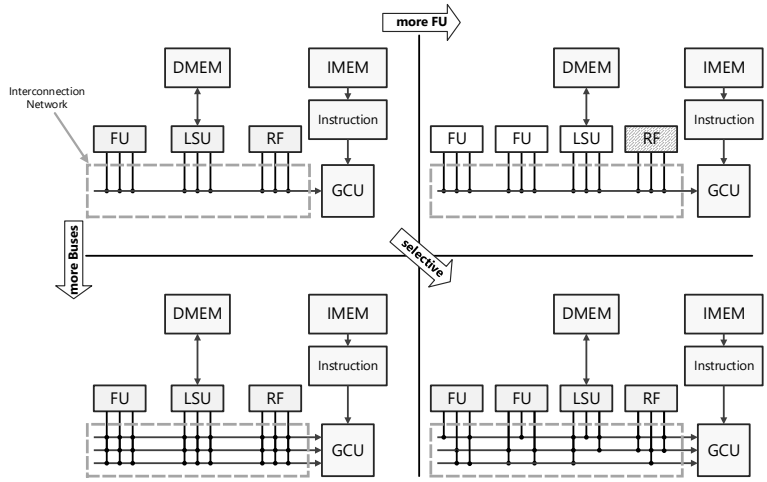

Fig. 2: Different configurations of the transport-triggered architecture (TTA).

as a side-effect of writing data to it. Due to the programmer-visible interconnection network, data can be directed from one unit to another without the need to store intermediate data back to a register file. Thus, the register file is optional but can be attached to the system to store data temporarily. The overall program flow is controlled by the global control unit (GCU), which performs jumps and function calls. A basic setup of the TTA is shown in Fig. 2. The design allows to add functional units (FU) with custom function units, buses and the selective configuration of sockets in order to have a trade-off between flexibility and complexity, taking profit of the processing characteristics of the target application.

3.6 Fully Configurable Divider Co-Processor

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 [15]. This array can be directly implemented on a pipeline structure (Fig. 3.a) or can be folded in order to reduce the silicon area requirements (see Table 3 and Fig. 3.b,.c). However, the resulting folded array unit can not start a new operation every clock cycle and its latency constraints the maximum number of parallel divider units that can be used by a single issue-slot pipeline architecture.

Fig. 3 shows the projection of a 4x4-bit non-restoring divider array to a folded structure [15]. In Fig. 3a, the division is performed by a fully pipelined

Table 3: Trade-off between Divider Configurations for 32-bit Division.

Latency	Frequency	Structure	Levels	Area	A single Issue-slot can fully utilize up to
32 cycles	101.5 MHz	folded	1 level	0.055 mm^2	32 divider units
32 cycles	101.5 MHz	pipelined	1 level	0.570 mm^2	1 divider units
16 cycles	73.0 MHz	folded	2 levels	0.063 mm^2	16 divider units
16 cycles	73.0 MHz	pipelined	2 levels	0.473 mm^2	1 divider units
8 cycles	46.1 MHz	folded	4 levels	0.086 mm^2	8 divider units
8 cycles	46.1 MHz	pipelined	4 levels	0.424 mm^2	1 divider units
4 cycles	26.6 MHz	folded	8 levels	0.130 mm^2	4 divider units
4 cycles	26.6 MHz	pipelined	8 levels	0.400 mm^2	1 divider units
2 cycles	14.2 MHz	folded	16 levels	0.212 mm^2	2 divider units
2 cycles	14.2 MHz	pipelined	16 levels	0.387 mm^2	1 divider units

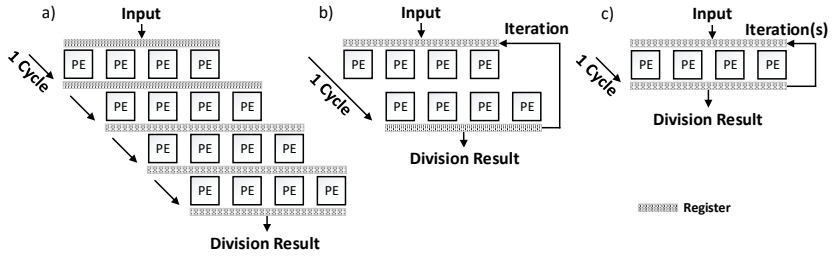

Fig. 3: Different configurations of the 4x4-bit non-restoring divider array. (a) Fully pipelined structure, (b) Two-level folded structure and (c) One-level folded structure.

array. This implementation has the shortest critical path and is able to process a new operation every cycle. However, due to the pipelined structure, the silicon area for this implementation is the largest. The folded configurations use an iterative processing scheme, e.g., by using two or one array levels (Fig. 3b/Fig. 3c). This leads to increased latency cycles and short critical paths as well. The silicon area is significantly reduced (see Table. 3) but the folded structure does not allow overlapped execution of multiple divisions. In real applications, the use of a pipelined or folded architecture depends on the application code characteristic. The number of division operations in the code as well as data dependencies influences the maximum number of parallel utilizable divider units. In the VLIW-MIPS and MIPS32, the divide array unit is included in the ALU and can therefore only be used by the first issue-slot. The second issue-slot of the VLIW-MIPS works in parallel but only processes `load-`/`store` and simple `add-`/`sub` instructions. Generally, the TTA allows more divider units to work in parallel due to exploited parallelism by several buses, but for an area efficient processor architecture, the implemented number of divider units should agree to the application code characteristics. The NEO430 already includes an optimized divider unit which is folded due to its multi-cycle execution paradigm, while

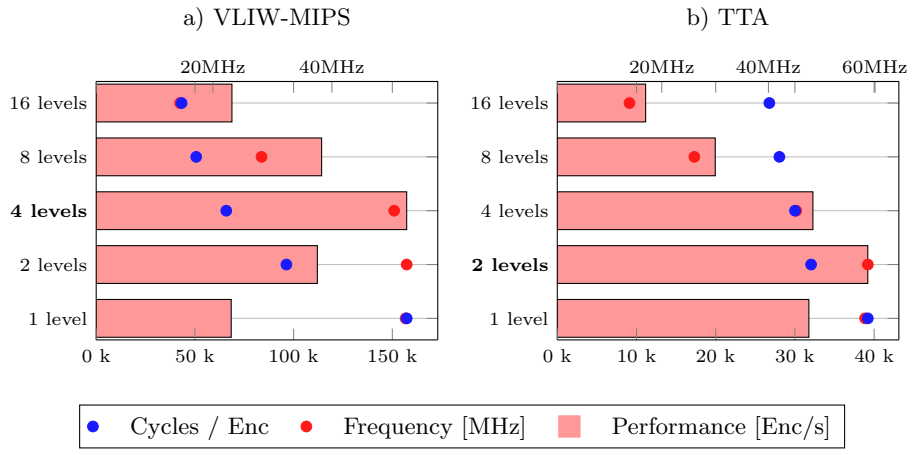

Fig. 4: Performance and maximum operating frequency using different folded divider unit configurations in (a) the VLIW-MIPS and (b) the TTA (see TTA configurations `#05` to `#09` in Table 4).

the AVR8 uses software emulation for division (which results in an significant increase of the number of executed instructions on this processor).

To illustrate the performance impact on the target application (see Section 4.1) when varying the divider level configuration, an evaluation of a VLIW-MIPS and TTA with different folded divider array configurations is presented in Fig. 4. As shown in Fig. 4a, the maximum throughput for the VLIW-MIPS is achieved by the use of a four-level divider unit. This is the optimal trade-off between operating frequency (limited by long timing paths which pass through the divider) and processing latency (required clock cycles for the division operation). For the TTA, the optimum number of levels in the divider is two, as the performance reaches its maximum there, as seen in Fig. 4b. In the TTA, the decrease of the critical path from the four-level divider to the two-level divider unit, which directly increases the maximum operating frequency, has a high positive impact on the total processing performance. In contrast, the increase of the cycles on the VLIW-MIPS processor is higher than the frequency gain, so the processing performance is reduced by a two-level divider. In this paper, only one divider unit with a throughput optimized number of levels is implemented in the MIPS32 and the first issue-slot of the VLIW-MIPS. Multiple divider units can not be efficiently used due to other arithmetic operations scheduled in the same issue-slot. In the TTA, more dividers can be used in parallel by increasing the number of buses. However, the utilization efficiency also depends on the exploitable application code parallelism.

4 Evaluation

In this section, the different processor architectures described in Section 3 are compared in terms of silicon area, power consumption and processing perfor-

mance. For the VHDL synthesis, Cadence Encounter RTL Compiler (RC14.28) was used to create a gate-level netlist. A SOI CMOS technology, capable of high temperature usage and based on transistors with a gate length of 0.18 µm, 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., Reed-Solomon 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 (2017.06) for execution of the application.

4.1 Target Application

The Reed-Solomon algorithm is a forward error correction (FEC) encoder/decoder for correction of transmission errors in communication applications. It calculates redundancy symbols to be appended to the transmitted data. The configuration of the algorithm used for this paper calculates 16 parity symbols from 239 information symbols which results in 255 codeword symbols in each block [2]. Each symbol has 8-bit, i.e., represents one byte of data. The code is able to correct 8 corrupted symbols per codeword. Configurations with higher redundancy are used for more defective channels, e.g., for the Voyager mission of the NASA [23].

An analysis of the C implementation [16] of the Reed-Solomon algorithm shows the use of 3943 modulo operations in one encode block. These are part of the Galois-Field's arithmetic and could be processed by dedicated Galois functional units (e.g., see ASIP implementation in [5]). In order to offer a flexible implementation of the algorithm for a non-specialized processor, just modulo operations are used as basic operations in the c code for this paper. The modulo operation itself uses the division function.

4.2 TTA Configurations

Due to the large design space of TTA configurations, a minimum configuration was used as a starting point and it was iteratively extended with more functionality in order to increase the performance (i.e., decrease the amount of clock cycles per encode-block run). The set of selected configurations for further evaluation is shown in Table 4. Configuration `#00` shows the minimal C compiler supported hardware setup consisting of one bus, one ALU supporting basic instructions (`add`, `sub`, `and`, `or`, `shift`,... [10]) and a small register file. Based on the utilization of certain parts of the hardware, bottlenecks were found and removed with the TTA toolchain from [10] (e.g., 4 buses in contrast to 1 in configuration `#02`). To avoid the excessive increase of instruction word length, immediate values are shortened to 8-bit. A long immediate unit combines multiple short immediates from several buses to form a longer 32-bit immediate (LImm) [9]. From configuration `#00` to `#04` on, the number of buses, register files or functional units is

increased which results in a reduced number of cycles but also in an increase of the instruction word width.

From configuration `#05` to `#09` the number of divider units and their configurations regarding the implemented array-levels (see Section 3.6) were varied to reduce the number of cycles, which increases the applications performance. With increased number of divider units, the number of cycles drops due to exploited parallelism in the application. The number of divider units for evaluation in Section 4.3 is selected by the configuration's silicon area efficiency, e.g., the silicon area efficiency of the 2-level configuration with 6 instances is higher (normed to performance) than by use of 7 instances.

As mentioned in Section 3.6, the configuration with two divider levels provides highest performance for the TTA, as the higher frequency compensates increased number of execution cycles in comparison to divider units with cycle-based faster execution.

4.3 Trade-off Analysis

The instruction memory (I-Mem) and the data memory (D-Mem) of the processors are implemented using multiple single-port RAM macro blocks from the technology library. The bit width of the I-Mem is defined by the architecture's instruction word width while the D-Mem bit width is equal to the actual data path width.

In most of the evaluated architectures, the memory defines the majority of the required silicon area (more than 50 %), which is shown in Fig. 5 separated for both, the memories and the core. Due to the wide instruction words (42- to 89-bit), the TTA configurations need large instruction memory sizes.

To achieve higher performance, higher frequency is not necessarily the major design point. The NEO430 operates at about 90 MHz but provides lower performance than the MIPS32, due to the multi-cycle (NEO430) versus the pipelined (MIPS32) instruction execution paradigm (see Fig. 6). Among the configurations of the TTA, there is a variance of up to 17 x in performance, which is caused by the major differences in number and type of functional units, size of the register file and interconnection complexity. The differences regarding the resulting silicon area of the TTA configurations are up to 1.84 x as shown in Fig. 5.

In Fig. 7, the power requirements of the evaluated architectures are shown. The AVR8 has the lowest requirement, while the TTA `#05` demands up to 6.4 x the amount of power. This is related to silicon area requirements and operating frequency of the architectures. When comparing processing performance and silicon area (see top of Fig. 8) of the different processor architectures, three clusters can be defined. The smallest architectures (AVR8 and NEO430) also provide the lowest performance. The MIPS32 and VLIW-MIPS provide higher performance with a linear increase in silicon area requirements. The largest architectures with the most variation in performance are presented by the TTA configurations. Not every TTA processor outperforms the MIPS32-based architectures or even the small 8- and 16-bit microcontrollers. The highest performance is provided by the TTA `#06` using 6 two-level divider units, which is clocked at approximately

Table 4: Different Configurations of the TTA. Colored rows show those configurations selected for later analysis. All divider units use a folded structure. Bus configuration reads as follows: No. of Buses (Immediate Width, No. of Long Immediate Units).

No	Instr. Width	Instr. Count (Mem. Size)	Bus Config.	Divider Conf.	ALU - FU	Register File (32-bit)	Cycles
00	42-bit	3 780(4k)	1 (32-bit, -)	-	1 ALU	1x5	825 577
01	42-bit	3 672(4k)	1 (32-bit, -)	**1x2level**	1 ALU	1x5	249 911
02	58-bit	1 024(1k)	**4** (8-bit, 1x)	**2x**2level	1 ALU+**1 Add**	**3x8**	64 698
03	68-bit	861(1k)	**5** (8-bit, 1x)	2x2level	1 ALU+**2 Add**	**3x16**	53 053
04	89-bit	831(1k)	**6** (8-bit, 1x)	2x2level	1 ALU+2 Add	**2x16+1x32**	50 664
-	89-bit	-	6 (8-bit, 1x)	**6x1level**	1 ALU+2 Add	2x16+1x32	45 443
-	89-bit	-	6 (8-bit, 1x)	**8x**1level	1 ALU+2 Add	2x16+1x32	40 445
05	89-bit	877(1k)	6 (8-bit, 1x)	**10x**1level	1 ALU+2 Add	2x16+1x32	39 201
-	90-bit	-	6 (8-bit, 1x)	**12x**1level	1 ALU+2 Add	2x16+1x32	39 058
-	89-bit	-	6 (8-bit, 1x)	**3x2level**	1 ALU+2 Add	2x16+1x32	42 038
-	89-bit	-	6 (8-bit, 1x)	**4x**2level	1 ALU+2 Add	2x16+1x32	35 614
-	89-bit	-	6 (8-bit, 1x)	**5x**2level	1 ALU+2 Add	2x16+1x32	33 708
06	89-bit	751(1k)	6 (8-bit, 1x)	**6x**2level	1 ALU+2 Add	2x16+1x32	32 042
-	89-bit	-	6 (8-bit, 1x)	**7x**2level	1 ALU+2 Add	2x16+1x32	31 804
-	89-bit	-	6 (8-bit, 1x)	**1x4level**	1 ALU+2 Add	2x16+1x32	53 334
-	89-bit	-	6 (8-bit, 1x)	**2x**4level	1 ALU+2 Add	2x16+1x32	35 010
07	89-bit	697(1k)	6 (8-bit, 1x)	**3x**4level	1 ALU+2 Add	2x16+1x32	30 010
-	89-bit	-	6 (8-bit, 1x)	**4x**4level	1 ALU+2 Add	2x16+1x32	29 298
-	89-bit	-	6 (8-bit, 1x)	**6x**4level	1 ALU+2 Add	2x16+1x32	29 060
-	89-bit	-	6 (8-bit, 1x)	**1x8level**	1 ALU+2 Add	2x16+1x32	37 842
08	89-bit	670(1k)	6 (8-bit, 1x)	**2x**8level	1 ALU+2 Add	2x16+1x32	28 042
-	89-bit	-	6 (8-bit, 1x)	**3x**8level	1 ALU+2 Add	2x16+1x32	27 328
-	89-bit	-	6 (8-bit, 1x)	**4x**8level	1 ALU+2 Add	2x16+1x32	27 330
-	89-bit	-	6 (8-bit, 1x)	**1x16level**	1 ALU+2 Add	2x16+1x32	30 341
09	89-bit	647(1k)	6 (8-bit, 1x)	**2x**16level	1 ALU+2 Add	2x16+1x32	26 773
-	89-bit	-	6 (8-bit, 1x)	**3x**16level	1 ALU+2 Add	2x16+1x32	26 820

60 MHz. The processing performance is 37.70 x the performance of the AVR8 with only an increase in silicon area of 4.10 x.

The efficiency of the implementations is calculated by normalizing the silicon area requirement to the reached throughput (Reed-Solomon encodes per second) for the different processor architectures. Results are shown at the bottom of Fig. 8. The optimum is in the bottom left corner. The TTA configurations `#06` and `#07` yield the highest efficiency. This is due to the exposed parallelism of the evaluated application. For the VLIW-MIPS, a single issue-slot constrains further possible parallelism of the applications code. More issue-slots would allow more parallelism and divider units but excessively increase the instruction memory size and therefore the silicon area requirements. For the TTA, the units are controlled by short encodings for the corresponding sockets and the total instruction length increase is small for multiple divider units.

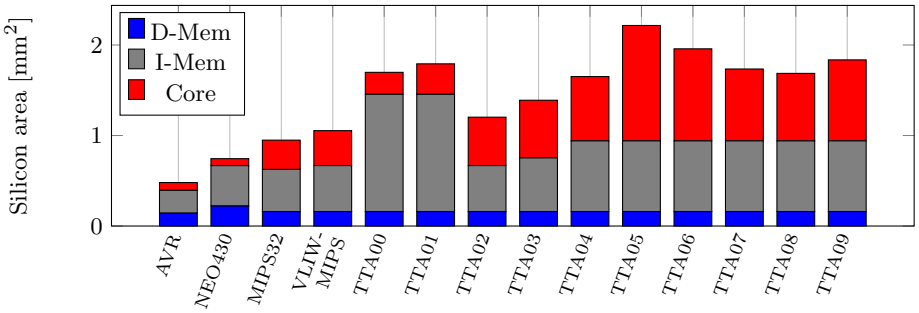

Fig. 5: Silicon area of instruction memory, data memory and core.

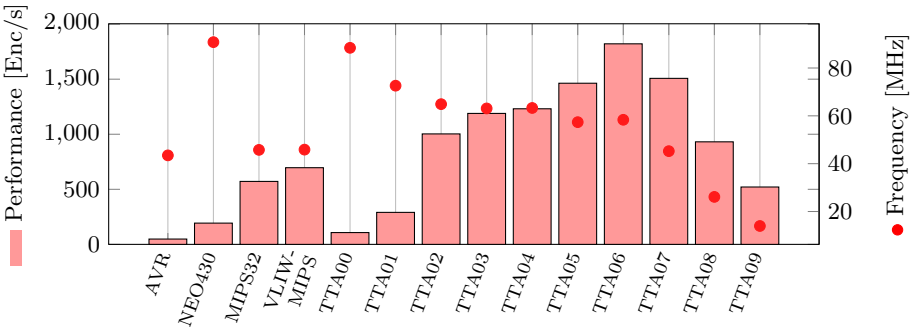

Fig. 6: Performance and frequency of the evaluated architectures.

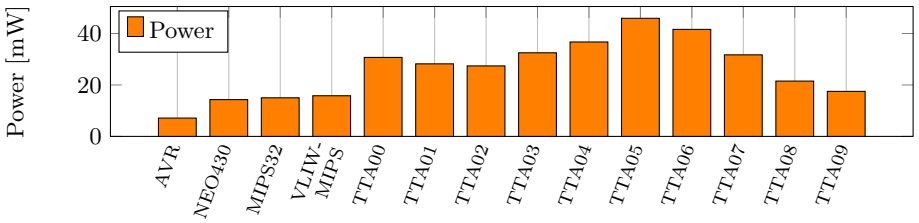

Fig. 7: Power requirements of the processors architectures for executing the Reed-Solomon encoder.

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. Different processors architecture organizations differ regarding the efficient use of silicon area and power. However, the efficient use of the processor's resources highly depends on the target application. In this paper, a Reed-Solomon encoder algorithm was used to evaluate different processor architecture organizations. Processors with minimum silicon area or energy consumption (AVR8 and NEO430) are not the most efficient architectures in

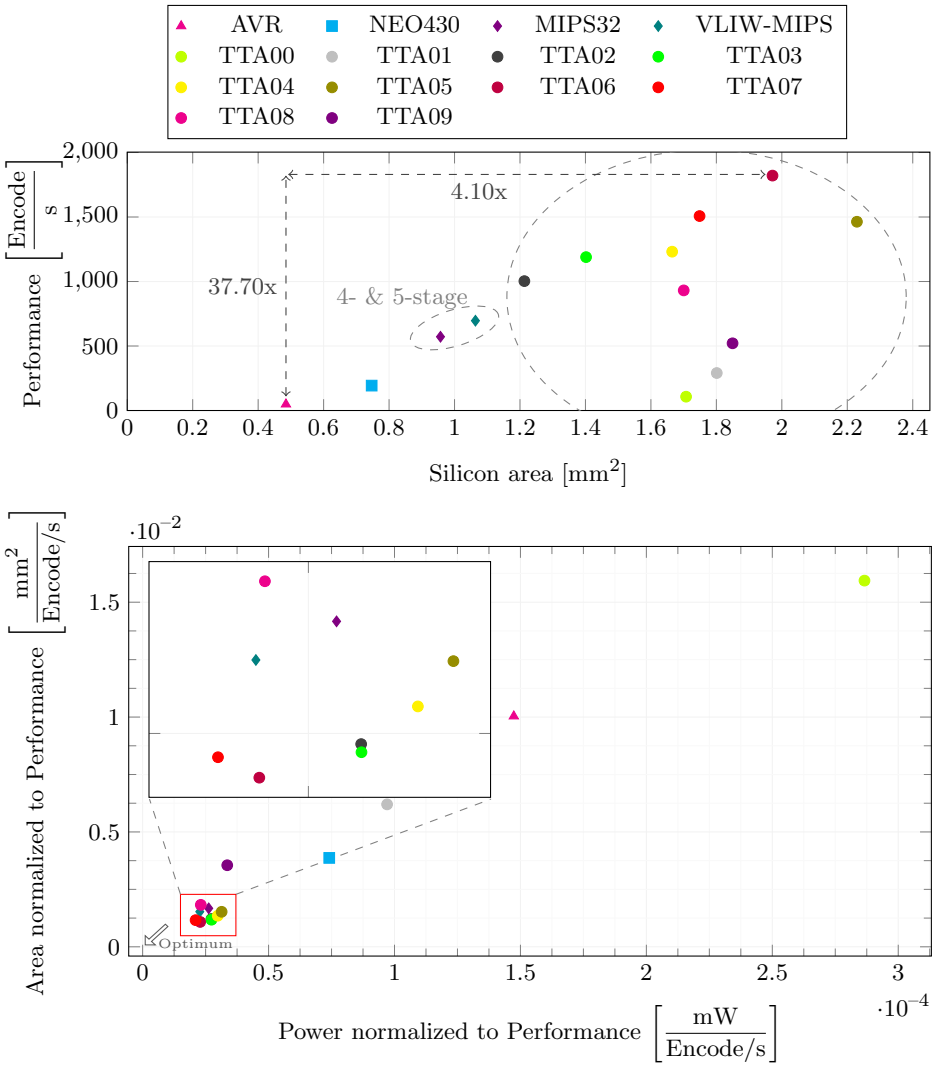

Fig. 8: Top: Silicon area and processing performance of the evaluated architectures. Bottom: Efficiency: Silicon area and power normalized to performance. Zoom of red box in top left of diagram.

terms of energy consumption vs. performance and silicon area utilization vs. performance. Larger cores, like VLIW-MIPS and variants of the TTA, provide higher efficiencies due to the better exploitation of the application code parallelism and the pipelined execution of the instructions. The parallel utilization of the available processing resources (i.e., functional units) is constrained by the application code and hardware implementation (i.e., sufficient data parallelism or number of parallel instructions). Due to the flexible customization, the TTA

processor provides the highest efficiency in silicon area utilization and energy consumption per Reed-Solomon encode block. The proposed configuration `#06` of the TTA processor using 6 divider units provides the best silicon area efficiency results. In comparison with the VLIW-MIPS, the TTA better exploits the application code parallelism without a strong increase of the instruction memory requirement. This makes it an efficient choice for use in area-constrained and performance-limited high-temperature technologies.

References

1. Chen, W., Sadana, D.K., Taur, Y.: SOI CMOS structure, patent (1998)
2. Clarke, C.K.P.: Reed-solomon error correction. BBC R&D White Paper (2002)
3. Corporaal, H.: Microprocessor Architectures: From VLIW to TTA. John Wiley & Sons, Inc. (1997)
4. Freescale: MC9S12G (2017), https://www.nxp.com
5. Genser, A., Bachmann, C., Steger, C., Hulzink, J., Berekovic, M.: Low-Power ASIP Architecture Exploration and Optimization for Reed-Solomon Processing. In: 20th IEEE Int. Conf. on Appl.-specific Sys., Architectures and Processors (2009)
6. Gesper, S.: Implementierung eines VLIW-MIPS-Prozessors für Hochtemperaturanwendungen mit Compileunterstützung, Master Thesis, Leibniz Universität Hannover, Institute of Microelectronic Systems (2018)
7. Hennessy, J.L., Patterson, D.A.: Computer Organization and Design: The Hardware / Software Interface. Elsevier (2014)
8. Honeywell: HT 83C51 (2011), https://aerospace.honeywell.com
9. Jääskeläinen, P., Kultala, H., Viitanen, T., Takala, J.: Code Density and Energy Efficiency of Exposed Datapath Architectures. Journal of Signal Processing Systems pp. 49–64 (2015)
10. 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 Int. Publishing (2017)
11. Microchip: PIC24HJ32GP202/204 and PIC24HJ16GP304 (2011), https://www.microchip.com
12. Microchip (former Atmel): AVR AT90S8515, https://microchip.com
13. MIPS Technologies: Programmers Volume II-A: The MIPS32 Instruction Set
14. Nolting, S.: The NEO430 Processor (2019), https://github.com/stnolting/neo430
15. Pirsch, P.: Architekturen der digitalen Signalverarbeitung. Springer-Verlag (2013)
16. Rockliff, S.: Reed-Solomon (RS) codes (1989), http://eccpage.com/
17. Tekmos Inc.: TK89h51 Microcontroller (2018), https://www.tekmos.com
18. Texas Instruments: MSP430, http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html
19. Texas Instruments: TI SM320F2812-HT (2011), https://www.ti.com
20. Texas Instruments: TI SM320f28335-HT (2014), https://www.ti.com
21. Texas Instruments: TI SM470r1b1m-HT (2015), https://www.ti.com
22. Vorago: VA10800 (2018), https://www.voragotech.com
23. Wicker, S.B., Bhargava, V.K.: Reed-Solomon codes and their applications. John Wiley & Sons (1999)