Microelectronics Journal 98 (2020) 104753

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Siwa: A custom RISC-V based system on chip (SOC) for low power medical applications

Ronny Garcia-Ramirez [a,*], Alfonso Chacon-Rodriguez [a], Roberto Molina-Robles [a], Reinaldo Castro-Gonzalez [a], Egdar Solera-Bolanos [a], Gabriel Madrigal-Boza [a], Marco Oviedo-Hernandez [a], Diego Salazar-Sibaja [a], Dayhana Sanchez-Jimenez [a], Melissa Fonseca-Rodriguez [a], Johan Arrieta-Solorzano [a], Renato Rimolo-Donadio [a], Alfredo Arnaud [b], Matias Miguez [b], Joel Gak [b]

[a] *Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica, Costa Rica*
[b] *Departamento de Ingeniería Eléctrica, Universidad Católica del, Uruguay*

A R T I C L E I N F O

Keywords:
Implantable medical devices
CMOS HV
RISC-V
System on chip

A B S T R A C T

This work introduces the development of Siwa, a RISC-V RV32I 32-bit based core, intended as a flexible control platform for highly integrated implantable biomedical applications, and implemented on a commercial 0.18 μm high voltage (HV) CMOS technology. Simulations show that Siwa can outperform commercial micro-controllers commonly used in the medical industry as control units for implantable devices, with energy requirements below the 50 pJ per clock cycle.

1. Introduction

There is a wide range of requirements for electronic devices in the realm of medical applications: for instance, in the analysis of medical images or computerized axial tomography scanners, the use of high performance, high power microprocessors is standard, whereas in implantable devices there is a need for small, low power, low performance but reliable processing units. Being reliability a primary concern, the use of discrete commercial, mature, market-tested micro-controllers is common in the latter, as in the case of pacemakers. However, the assembly on a printed circuit board (PCB) of independent processors, communication interfaces (IO), sensors, and actuators requires extra validation for the final product, as well as higher power consumption to drive the wires between the discrete components and the PCB. Integrating both the processing units, the sensors and the IO interfaces would cut heavily on such costs. As such, a good alternative is the use of field programmable arrays (FPGAs) [1], but these devices are often not able to meet the requirements of small form factor and capacity at the same time, and in general, achieve worse power consumption and speed performance compared to an application specific integrated circuit (ASIC) fabricated on the same CMOS node [2,3].

Now, even though there is plenty of micro-controller intellectual property (IP) available for the development of ASICs, the use of these blocks is tied to restrictive and expensive licenses; these IP blocks are also typically not open for customization, making them unsuitable for small development teams with restricted design targets and budgets. This is why RISC-V [4], an initiative from Berkeley University, aiming at the development of a scalable open instruction set architecture (ISA) CPU for research and the industry has spurred a growing interest (see for instance Refs. [5–8]) with several companies already offering design tools and IPs based on RISC-V [9,10], and even commercial processors [11].

In this paper, the design of a custom low power system on chip (SoC) using a 32-bit reduced instructions set computer (RISC) micro-controller [12,13], targeted for implantable/wearable applications, is discussed in detail. The processor core is based on the RISC-V RV32I ISA and is implemented using a centrally controlled non-segmented micro-architecture. It includes the control for an integrated sensing and stimulation interfaces as well as standard communication interfaces like a Universal Asynchronous Receiver/Transmitter (UART), eight general purpose input/output ports (GPIO) and a serial peripheral interface (SPI).

* Corresponding author.
E-mail address: rgarcia@tec.ac.cr (R. Garcia-Ramirez).

https://doi.org/10.1016/j.mejo.2020.104753
Received 1 May 2019; Received in revised form 24 January 2020; Accepted 2 March 2020
Available online 5 March 2020

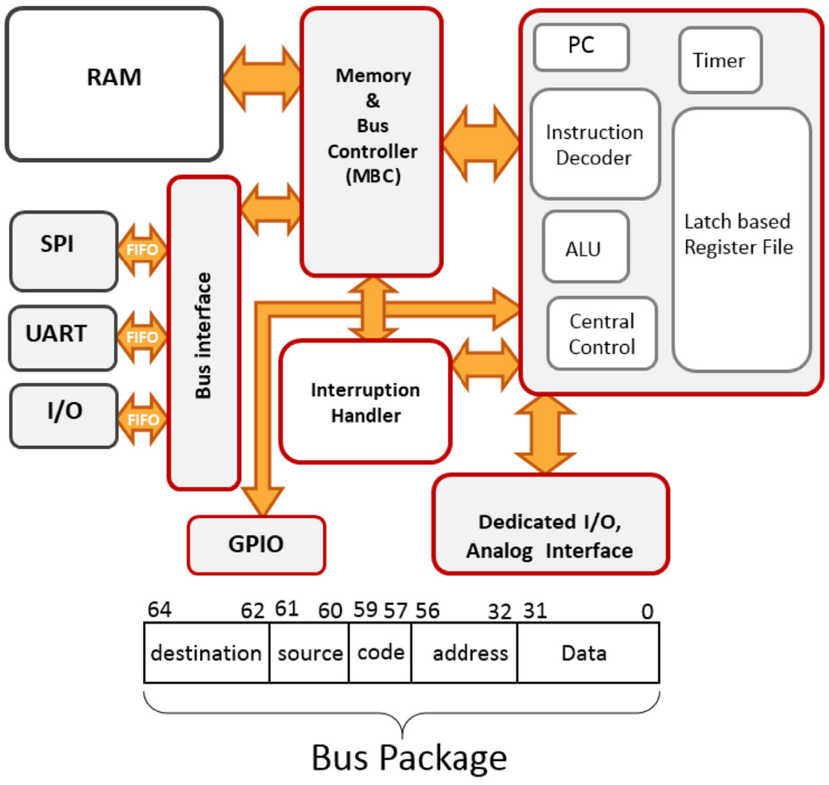


Fig. 1. Block level description of the proposed low power System on Chip (SoC).

This paper is organized as follows: in section 2, an overview of all the functional details of Siwa's[1] proposed architecture is provided as well as a description of the implementation for the main components of its micro-architecture. In section 3, a comparison of the proposed micro-architecture with regards to other RISC-V implementations and other processors commonly used in implantable medical devices is presented. Finally, in section 4, some conclusions of this work, as well as some prospects of its development, are shared.

2. Proposed micro-architecture

This section details the micro-architectural implementation of a low power system on chip (SoC) for biomedical applications centered around Siwa, a RISC-V RV32I based central processing unit (CPU), a universal asynchronous receiver-transmitter interface, a serial peripheral interface, eight general purpose input/output interfaces, and an analog block intended for biological stimulation (details about this application can be found in Ref. [12]). Fig. 1 represents the main functioning blocks of the system; here, the cardiac stimulation and sensing interfaces are represented by the block called "Analog Interface"; a basic description of each unit in the micro-architecture, as well as the details of the programming model required for its correct utilization, are provided next (details about the RISC-V open platform can be found in Ref. [14–16]).

2.1. Bootloading and programming model

The SoC has a hardwired bootstrap subroutine, that takes control of the SPI interface after a reset, and loads the system's program stored in an external 16 MB flash memory into the micro-controller's 8 kB static random access memory (SRAM). Once the bootstrap process is finished, the CPU executes a fetch micro-instruction, and the flash memory driver becomes accessible for normal write and read operations, serving as a secondary storage device. A hand-shaking protocol is provided for the bootstrap process in order to avoid an overflow in the bus access queues, due to differences in clock speed between Siwa and the flash memory.

2.1.1. Control and status registers (CSR)

Siwa employs a custom implementation of the RISC-V RV32I standard, with several of its CSR's modified from their original definition in Ref. [15]. This saves area and power, and some registers from the specification were not even implemented at all; more details can be found in Ref. [17]. This must be taken into account by the programmer while using the standard RISC-V tool-chain. A general pre-processing script has been generated to help the programmer handle most of these variations. A main configuration register (MCR) is used to configure some of the basic features of the system, such as the masking of external or internal interrupts. From here, the programmer can also activate/deactivate the memory and bus controller (MBC) in order to save power, configure the general input/output registers (GPIO), and check the status of the internal timers.

A machine exception program counter (MPEC) holds the value of the program counter while an interrupt is serviced and two machine exception cause registers, (mcauseA) and (mcauseB) are used to direct the interrupt handler to the appropriate service. A machine trap-vector base-Address (mvtec) is used to point to the address where the interrupt handling routine is located. The PC jumps there every time an interrupt/exception is detected. By default, this register points to address zero after reset and must be configured by the basic input/output system (all the code in Siwa runs at machine privilege level, so it is possible for the main code to directly modify this register). Two extra registers are used to control the internal timer: the Machine Timer Fence register (TMRFNC) holds the maximum count value at which the system trig-

[1] Siwa: From the Bribri, a Costa Rican indigenous language, meaning wisdom or knowledge.

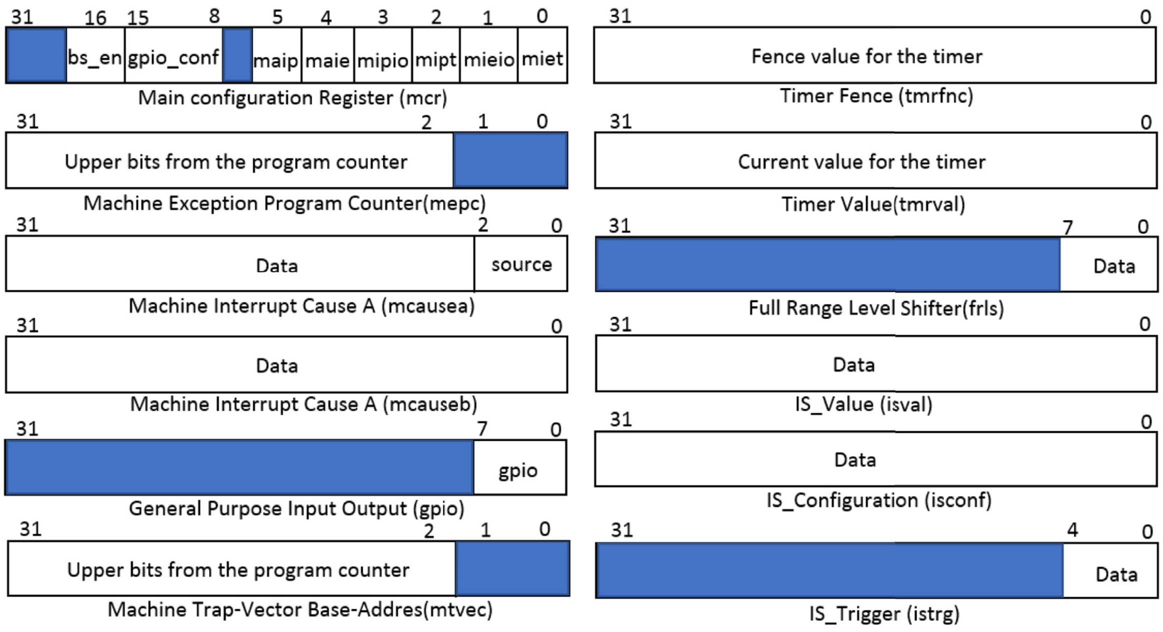

31 16 15 8 5 4 3 2 1 0
bs_en
gpio_conf
maip
maie
mipio
mipt
mieio
miet
Main configuration Register (mcr)
Upper bits from the program counter
Machine Exception Program Counter(mepc)
Data
source
Machine Interrupt Cause A (mcausea)
Machine Interrupt Cause A (mcauseb)
gpio
General Purpose Input Output (gpio)
Machine Trap-Vector Base-Addres(mtvec)
Fence value for the timer
Timer Fence (tmrfnc)
Current value for the timer
Timer Value(tmrval)
Full Range Level Shifter(frls)
IS_Value (isval)
IS_Configuration (isconf)
IS_Trigger (istrg)

Fig. 2. Bit fields detail for each CSR implemented in Siwa's Micro-Architecture.

gers an interrupt; the Machine Timer Value register (TMRVAL) holds the current value of the internal timer. Fig. 2 shows a detailed description of the positions of each field for the implemented CSRs. The solid blue spaces in the registers of Fig. 2 represent read-only reserved bits which are tied to zero in this implementation (specific details of each register and their configuration bits can be found in Refs. [17]).

2.1.2. *Memory map*

Both the UART and the SPI are managed as memory mapped devices (MMIO) instead of port mapped devices (PMIO). The memory map is hard-coded instead of being programmable in order to save area.

The CPU's memory and bus controller (MBC) directs every access in the range from 0 to 8 kB to the SRAM memory, while the accesses in the range from 8 MB to 32 MB are directed to the SPI and the UART; any attempted access to the region from 8 kB to 8 MB triggers a bad address interruption and it is managed by the interrupt handler software.

2.1.3. *Interrupts*

There are two sources of non-maskable interrupts in the system: those generated by an invalid instruction in the instruction decoder and those generated by an invalid address transaction. There are three sources of maskable interruptions: those coming from external devices connected to the bus, SPI or UART, those coming from the internal timer, and those coming from an external pin (the latter are called analog interruptions in this document, as they are intended to be used by the cardiac stimulation device). The central control unit of the CPU checks for interruptions after loading the next instruction in the program counter (PC) and before the fetch of the instruction from memory and into the instructions decoder (ID). The behavior of the machine, whenever an interruption is detected, is the same for both maskable and non-maskable interrupts: The current PC is saved into MEPC, and all the maskable interrupts are turned off for the execution of the interrupt controller routine.

Once the interrupt handler routine is loaded, it must save all the CSR and GPR registers in the stack before checking the mcauseA and mcauseB CSRs in order to deduce which interrupt is requesting service. The interrupt handler routine identifies the interrupt by checking the source vector field in the MCAUSEA register; extra information about the interrupt is provided in the rest of the vector fields. Servicing an

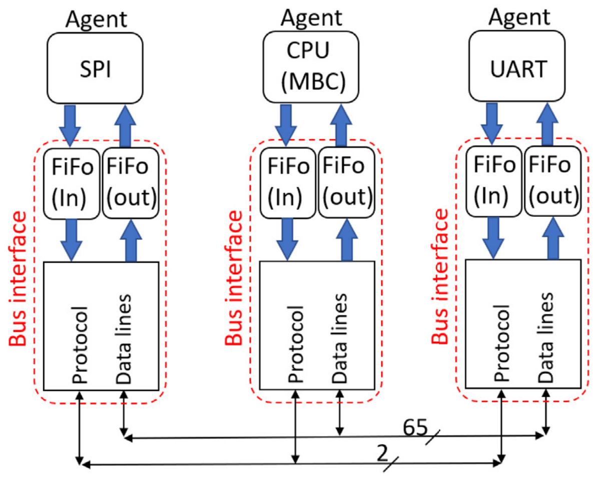
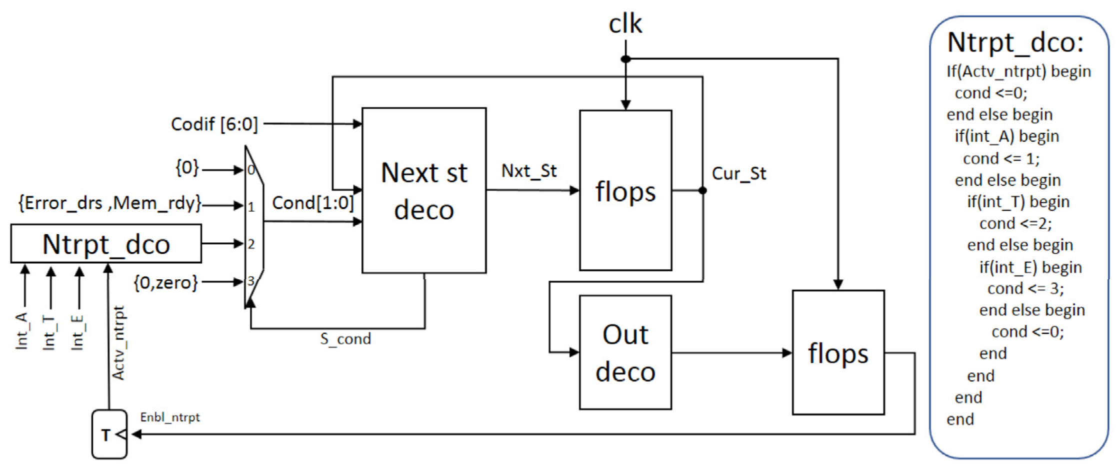


Fig. 3. Siwa's bus micro-architecture. FIFOs are used to synchronize data among devices with different speeds. Instead of having a central arbitration block for the bus, this function is distributed among each bus controller making the bus more robust against failures in a central point.

interrupt disables the handling of further interrupts until the service routine is completed and a return to normal executions is performed, except for system exceptions that are always accepted.

2.2. *System bus*

All of the SoC's data agents capable of generating and consuming data, such as the UART, the SPI, and the MBC, are communicated via a 64-bit parallel bus with distributed arbitration. Two FIFOs are used to interface any devices that connect to the bus, as illustrated in Fig. 3; these FIFOs avoid flow control issues deriving from speed differences between the units connected to the bus. In order to keep FIFOs from overflowing, a handshaking protocol is used. To save area, FIFOs have a depth of only two messages in this implementation, but this option is re-configurable for future, performance oriented versions of Siwa. The fact that the bus has a distributed arbiter instead of a central one is important because it reduces the number of connections that would be



```
Ntrpt_dco:
If(Actv_ntrpt) begin
 cond <=0;
end else begin
 if(int_A) begin
  cond <= 1;
 end else begin
  if(int_T) begin
   cond <=2;
  end else begin
   if(int_E) begin
    cond <= 3;
   end else begin
    cond <=0;
   end
  end
 end
end
```

Fig. 4. Basic block diagram of the FSM for the central control unit of the CPU.

otherwise necessary between the arbiter and the bus interfaces, at the expense of larger bus controller interfaces. This option also removes the arbiter as a critical point of failure, because of the distributed redundant bus arbitration.

The data package presented in Fig. 1 represents the format used to transmit information in the system's bus. Here, the destination and the source of the package of information are encoded using particular IDs, and specific codes are defined for each device protocol.

The CPU services each communication with the agents in the system as an interrupt. The interrupt handler interprets each field in the message received from the bus agents, except for the data field (bits 31 to 0), which is transparent for the handler and must be interpreted at a higher software level of abstraction.

2.3. General central processing unit micro-architecture

Siwa's CPU is based on a RISC-V R32I programming architecture. As depicted in Fig. 1, five general blocks are implemented: the memory and bus controller (MBC), in charge of interfacing the CPU with the general bus and the system's memory; an instruction decoder (ID) that translates instructions into micro-coded data going to the ALU and the RF, and command fields serving as the input to the CPU's controller finite state machine (FSM); a 32 word, 32-bit data register file (RF) that also includes all the special CSRs of the system; a 32-bit arithmetic and logic unit (ALU) capable of signed addition, word rotation and standard the boolean logic functions defined by the R32IV specification; and the central micro-coded finite state machine FSM which coordinates all the CPU (not shown in Fig. 1).

Choosing a centrally controlled micro-architecture instead of a pipelined one allowed for a smaller footprint, without extra segmentation registers or feedback units used to handle data dependencies common in pipelined architectures. The impact in degraded processing performance is an expected trade-off considering the intended application in implantable devices, where area and power performance are the key factors, for systems that any way, handle very slow biological signals (in the order of several Hz to a few KHz). Constraints in budget and area limited the system to only one memory level (used as main memory for both data and program) and a secondary flash storage device connected to the SPI interface. The internal memory in this first implementation is limited to 8 kB of SRAM due to space reasons; the micro-controller had to leave space for the analog stimulus section, typically consuming a major section of area for the handling of high voltage drivers and fine tuned current sources. However, the processor's general design allows for a much larger amount of memory, with at least an extra level of cache. A diagram of the implementation of the controlling FSM is presented in Fig. 4.

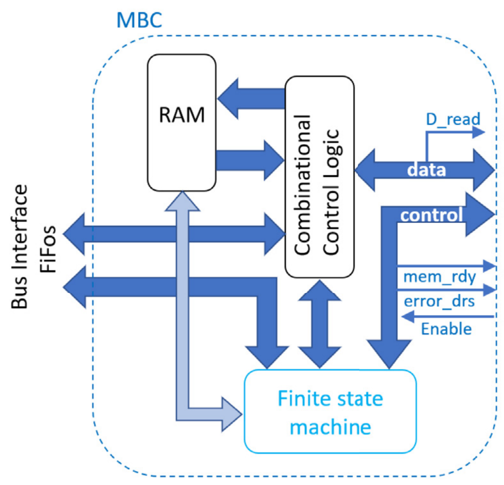


Fig. 5. Detailed micro-architecture for the memory and bus controller (MBC). Here the SRAM block is an IP provided by the foundry.

2.3.1. Memory and bus controller (MBC)

As previously mentioned, the MBC interfaces the bus with the CPU. This unit also controls transactions to the SRAM. Fig. 5 describes the general MBC's micro-architecture. Like most other blocks in the design, Siwa's MBC may be configured, in this case, for handling higher memory spaces if required.

Since the RV32I specification allows for the read and writes of individual bytes, half words and full words, the MBC transactions have different latencies depending on the instruction and the final target of the transaction.

From the CPU's point of view, the MBC flags its readiness to execute a transaction if the mem_rdy bit is asserted, and starts transactions with a pulse in the enable bit. While the transaction is being executed, the mem_rdy bit is low and when the transaction finishes the mem_rdy pin is asserted again, meaning that data is ready in the D_Read line and the package of information has been written in memory or that the package of information has been sent to the appropriate device through the bus.

In the case of an invalid instruction, the Error_drs bit is asserted and the MBC waits for a pulse in the Enable bit in order to become functional for the next transaction.

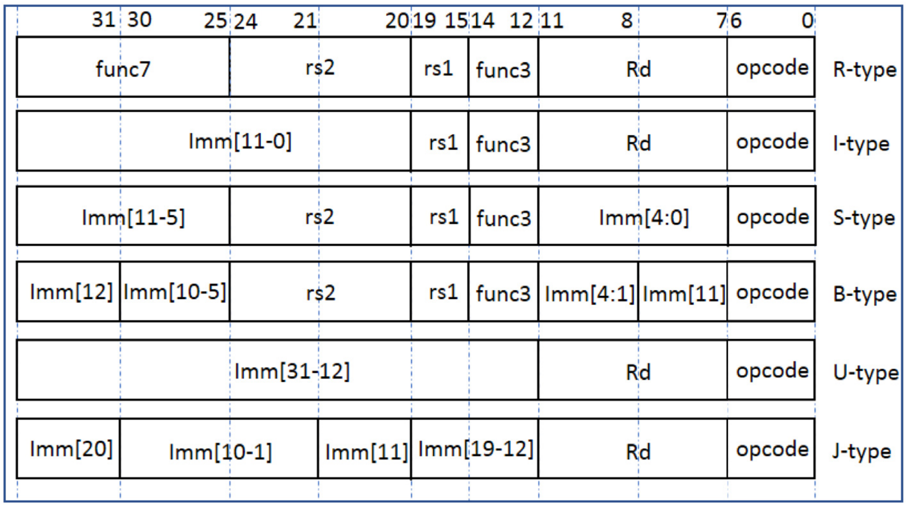


Fig. 6. Description for each one of the instruction formats specified in the RISC-V architectural description [14].

2.3.2. *Instruction decoder (ID)*

This combinational block is in charge of decoding the instructions stored in memory and provides the system with the information required for the correct execution of the code, as well as to identify illegal instructions. RISC-V RV32I is a fixed instruction length architecture, with six different instruction formats available for the user and privileged mode [14]: R-type for register to register operations, I-type for short immediate arguments and loads, S-type for stores, B-type for conditional branches, U-type for long immediate arguments and J-type for unconditional jumps. The detail for the format of each kind of instruction is presented in Fig. 6. Based on these instruction formats, RISC-V provides 47 instructions which are available from user mode [14]. In this first version, SIWA does not implement the Fence and Fence.i instructions, being a single physical/logical thread architecture. When an instruction is not recognized, the code returned to the control unit has its bits all set to 1, being thus equivalent to have an illegal instruction exception. For each one of the supported instructions this block decode the information required to execute the instruction correctly; for instance, in some cases, the immediate output (IMM) is sign extended while in others it is zero extended.

The ID unit function is straightforward: once the next instruction is ready in the input port of the block (D_Read), the instruction is captured with a pulse in the ld_id pin, and is subsequently decoded, providing thus the central FSM with the identity of the fetched instruction, and all of the necessary bit fields to execute it. Specifically, the fields returned by ID block are: a unique 7-bit codification (CODIF) that corresponds unequivocally to a particular instruction, and function fields particular to each instruction type, as presented in Fig. 6.

The other fields returned by the instruction decoder correspond with the fields described in Fig. 6: immediate operand (IMM), destination register (RD), register source 1 (RS1), register source 2 (RS2) and control ans status register identifier (CSR). To get a better understanding of the use of each field for each instruction please refer to [16].

2.3.3. *Arithmetic and logic unit (ALU)*

The functions provided by SIWA's ALU are: signed and unsigned addition, subtraction and comparison, logical AND, logical OR, logical XOR, and logical and arithmetic shifts (both to the right and to the left). Two's complement codification is used for the signed variables. From the control's point of view, the operation of the ALU is straightforward: the operands are processed according to a control code.

The ALU's boolean and comparison blocks are fast, small and relatively simple to implement via the foundry leaf cells library. On the other hand, the shifter and the adder blocks are large and potentially slow depending on the selected architecture. Several options from the literature were evaluated in terms of area power and delay (see Table 1).

Table 1
Implemented instructions grouped according to their codification type.

Instruction Type	Instruction Mnemonic
U-type	LUI, AUIPC
J-type	JAL
B-type	BNE,BGE,BGEU
	BEQ,BLT,BLTU
S-type	SH, SW, SH
I-type	LH, LW, LBU,LHU,XORI
	LBU,ADDI, SLTIU,SLTI
R-type	SLLI, SRAI,SRLI,SUB
	SLT,XOR,SRA,AND
	ADD,SLL,SLTU,SRL,OR
Machine Mode	EBREAK, CSRRW,CSRRC
I-type	CSRRSI, ECALL,MRET
	CSRRS, CSRRWI,CSRRCI

Several adders were conceptually analyzed, including logarithmic look ahead adders as the Brent Kung, and carry skip adders variations, including a Manchester Carry Chain topology and a Carry Select adder (CSA). Being area and power consumption the most important constraints in this version of the processor, logarithmic look ahead adders were readily taken off the list, while adders such as the ripple carry adder (RCA) were excluded due to their low speed over operands of 8 bits or more [18]. Three architectures were selected after this pre-screening: Constant Width Carry Skip adder (CSK), Manchester Carry Chain adder (MCC), and Variable Width Carry Skip adder (VSCK). The three candidate architectures were logically and physically synthesized in order to determine the best fit for the project. Table 2 gives a summary of the post-layout results; based on these results the CSK topology was selected because of its better power, area and delay metrics (with the MCC and the VSCK being about 20% slower than the CSK, and about 7–10% more energy expensive). Data from Table 2 was generated using a 20 MHz clock, with post-layout extracted data from the three adders, applying random data to their inputs every clock cycle.

Power consumption results in Table 2 may be pessimistic because these assume a new calculation every clock cycle, however, these are only used as a selection metric. The considered activity factor for these estimates is way over the expected ALU real data toggling, as any ALU data execution in this version of Siwa takes only one clock cycle from the multi-cycled completion of any instruction.

Table 2
Summary of post-layout data for selected adders using 32-bit data words. For a better comparison, all data normalized to the CSK results. Clearly, the CSK beats the MCC and the VCSK from 7% to 21% on all metrics.

Adders	Power (mW)	Area (μm^2)	Delay (ns)
CSK	0.7245	1715.18	16.24
MCC	0.7805	1823.45	19.34
VCSK	0.7993	1838.91	19.65
Adders	**Norm. Power**	**Norm. Area**	**Norm. Delay**
CSK	1	1	1
MCC	1.0773	1.0631	1.1909
VCSK	1.1032	1.0721	1.210

Two options were evaluated for the selection of the shifter: a funnel shifter and a barrel shifter implementation [19]. The barrel shifter was selected due to its shorter delay characteristics, being very similar in terms of power, according to Ref. [20].

2.3.4. *Register file (RF)*

The register file is the general block encompassing the status and general purpose registers (GPRs). Due to budget restrictions, the module was implemented using static latches instead of a denser SRAM IP. The latch cells, anyway, allowed for a 50% area saving when compared with a preliminary flip-flop RF version.

The implemented register file block is based on pass-gate multiplexers and allows for the coherent read and write of the same register in one clock cycle by using an intermediate retention latch working in the complementary level.

3. Post-layout evaluation

Fig. 7 shows the diagram of the implemented layout of the processor, with the SRAM block located at the top left corner. This block represents one of the biggest physical constraints of the present implementation in terms of performance, because in restrict the amount of cache memory available in the micro-architecture. A limited cache memory implies that in order to run big programs, (which do not fit in the internal SRAM), it is necessary to access an external memory which significantly reduces the performance in terms of power consumption and executed instructions per time unit. In the case of the present implementation, this external memory block is implemented with a flash memory connected to the bus via the SPI interface.

The custom analog interface for the cardiac stimulus unit is not shown in Fig. 7, and the ports required for this interface are aligned in the right side of the die.

A post-layout estimation of the energy consumed by each one of the implemented instructions is given in Fig. 8. The energy consumption is estimated by running a test with 1000 iterations of each instruction, with random operators, and using the average power consumption from the extracted post-layout model given by Synopsys IC compiler. The bootload process is not taken into account. As expected, because of the memory accesses they must perform, the load/store instruction family are the most expensive in terms of energy consumption. In particular, the subset of load/store byte and half word instructions require the highest energy, because of the costlier unaligned access to memory. These instructions also take more clock cycles to execute; this is why the average power consumption, as given in Fig. 8 remains relatively constant. The introduction of architectural clock gating and block sleep modes were left for subsequent versions of SIWA.

The average clock cycles per instruction (CPI) was calculated with the number of cycles need it for each instruction and the total instructions for each MCU. The information required from commercial MCUs

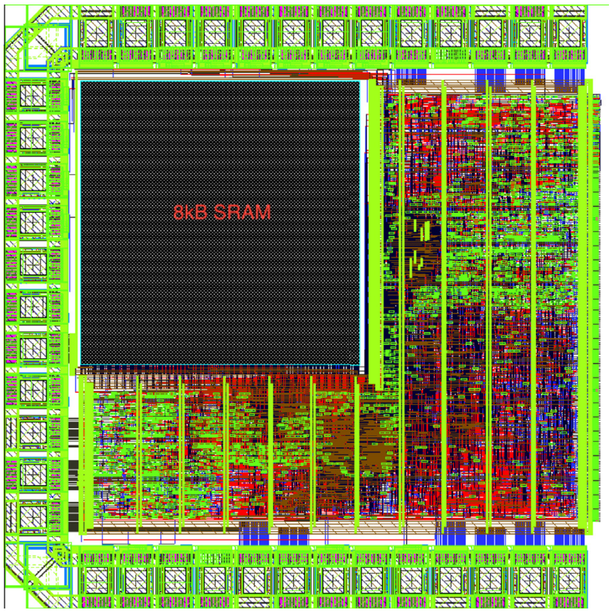


Fig. 7. Physical layout of the proposed Siwa RISC-V SoC. To the right, 3.3 V ports provide interfacing with analog stimulus circuits (not shown).

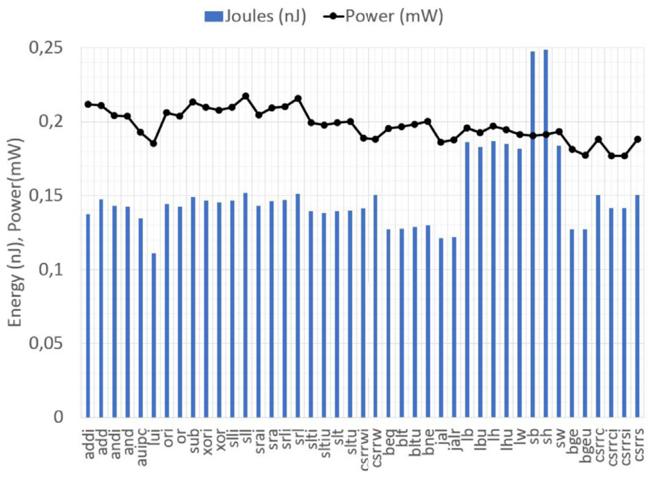


Fig. 8. Estimation of the energy consumed by each implemented instruction. In order to estimate the energy consumed by each instruction, 1000 iterations of the instruction with random operators were executed and averaged, using a clock speed of 20 MHz and 1.8 V_{DD}. Here the names in the axis of the abscissa represent the instructions implemented in SIWA.

was taken from Ref. [21,22] y [23], such that:

$$CPI = \frac{\sum \text{Cycles taken per instruction}}{\text{Total number of instructions}} \tag{1}$$

The energy per cycle metric represents the average energy consumed by cycle for a micro-architecture running a task. This metric can also be found in the equivalent terms of $\mu W/MHz$:

$$\frac{\mu W}{MHz} = \frac{\mu W}{\frac{1000000 \text{ clock cycles}}{1 \text{ s}}} = \frac{\mu W \cdot s}{1000000} = \frac{\text{pJ}}{\text{cycle}} \tag{2}$$

For commercial MCUs, power consumption data was extracted from data sheets where the average total consumption was reported for a specific frequency and voltage. This metric can be calculated as:

$$\frac{\text{Energy}}{\text{cycles}} = \frac{\text{Average Power Consumption}}{\text{Frequency}} = \frac{\text{Joules/s}}{\text{Cycles/s}} \tag{3}$$

Table 3
Comparison between different RISC-V cores, scaled according to Ref. [27]. The data used for the proposed micro-architecture is estimated based on post-layout simulations.

Core	Siwa	Mriscv	Riscy	Zero-riscy	Micro-riscy
Reference	This work	[24,25]	[26]	[26]	[26]
Technology	0.18 μm	0.13 μm	65 nm	65 nm	65 nm
Frequency	20 MHz	160 MHz	Not reported	Not reported	Not reported
ISA	RV32I	RV32IM	RV32IM + DSP	RV32IM	RV32E
Program memory	8 kB	4 kB	Not reported	Not reported	Not reported
Average CPI	4	Not reported	1.27	1.49	1.49
pJ/cycle	48.31	850	63.68	26.12	23.61
pJ/cycle (not scaled)	48.31	167	5.07	2.08	1.88
pJ/Instruction	193.24	Not reported	80.87	38.91	35.17
Core Area (μm^2)	672,146	350,250	703,296	326,592	200,448

Table 4
Comparison of the proposed micro-architecture with other MCU architectures common in the IMDs field.

Core	Siwa	8051-compatible	Atmega328p	PIC16LF1823	MSP430
References	This work	[28,29]	[21]	[22]	[23]
Technology	0.18 μm	0.18 μm	Not reported	Not reported	Not reported
Instruction word size (bits)	32	8–24	16	8	16
Program memory	8 kB SRAM	Not reported	32 kB FLASH	2 kB	8 kB RAM
Freq.	0–20 MHz	13 MHz	0–20 MHz	31 kHz-32MHz	4–16 MHz
Average CPI	4	Not reported	1.62	1.1837	$\approx$ 1
pJ/cycle	48.31	70.6	360	54	803
Average power	48.31	70.61	360	54	803
(μW)	@1 MHz	@1 MHz	@1 MHz	@1 MHz	@1 MHz
VDD (V)	1.8	Not reported	1.8	1.8	2.2

For the calculations of power consumption in the proposed SoC, we used the power estimations obtained through simulation and reported in Fig. 8, assuming a balanced test running all the instructions in the same proportion.

Table 3 compares other reported RISC-V micro-architectures with similar characteristics to SIWA. The first one was developed in a 130 nm technology, and implements a RV32IM ISA, an AXI-LITE, and an APB bus [24,25]. In Ref. [26], three cores are presented: Riscy (RV32IM + DSP), Zero-riscy (RV32IM) and Micro-riscy (RV32E), implemented in a 65 nm technology and embedded on a PULPino platform. The first of these cores was intended for DSP, the second was intended for integer operations, and the last one, with the lowest performance, area, and energy consumption, was customized for administrative tasks only.

Since the technologies used to implement these micro-architectures are different from the one used in the present work, the power consumption and area of these cores was scaled using the method and equations provided in Ref. [27], to approximate their behavior as if these had been fabricated on a 0.18 μm process. It can be noticed from the data in Table 3 that the scaled average energy consumption per clock cycle of the proposed architecture is between 2 and 9 times better with respect to other RISC-V cores considered. However, the average power per instruction might not be favorable, since most of the reported cores are segmented architectures and intended for other applications where performance is more relevant. The fabrication process selection is responsible of the larger area metrics for the proposed architecture, which is justified by fabrication costs, process reliability, and high-voltage device capabilities; these features might not be available in more recent CMOS technologies.

Table 4 provides a comparative basis between the proposed architecture and those commonly used for IMD applications reported in the literature. One of the challenges to compare among different micro-architecture implementations is that they obviously do not have the same features. Moreover, the use of performance standard metrics is not common either in this field of application; for instance, the use of the Drhystone benchmark (DMIPS), pervasive in the measurement of general purpose microprocessors performance, is unfeasible for small micro-controllers (MCUs) due to memory requirements.

Nevertheless, the comparative data in Table 4 shows that the proposed architecture has a lower energy per cycle while running at much higher frequency and having a wider word size. Customization capabilities and the possibility of monolithic integration with other functionalities are again other clear advantages of the proposed architecture in the field on IMD applications.

It is expected that future versions of Siwa may have better metrics in terms of area and power consumption. Next iterations might include power states, which reduce the power consumption by adding the capability of turning off specific hierarchical blocks depending on the code that is being executed. Regarding the total area, further improvements can include the optimization of the bus architecture with a simplified central arbiter.

4. Conclusions and future work

The use of external intellectual property for the development of SoCs is tied to restrictive and expensive licenses; also, the IP blocks are not open for customization, which makes them unsuitable for small development teams with particular design targets and restricted budgets. An original SoC based on the RISC-V R32I architecture and intended for low power medical devices was presented and evaluated against other RISC-V implementations and other processors used in the literature for implantable medical devices.

Even this first version of the core processor, based on post-layout analysis, presents the lowest energy per cycle metric of all the evaluated commercial micro-architectures and RISC-V cores when escalated using the method presented in Ref. [27] to compensate the differences from fabrication technologies.

This initiative demonstrates the feasibility of designing a customized architecture with a small development team, enabling the possibility of implementing flexible low-power and low-area solutions for robust and cost-effective ASICs.

Future work can address further customization and optimization of the architecture for low-power and low-area implementations in new SoC designs. Validation of the fabricated hardware and formal benchmarking against other architectures is also planned in the near future for further evaluation of the proposed SoC.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Ronny Garcia-Ramirez: Investigation, Project administration, Software, Conceptualization, Methodology, Validation, Writing - original draft. **Alfonso Chacon-Rodriguez:** Funding acquisition, Investigation, Project administration, Software, Conceptualization, Supervision, Methodology, Writing - review & editing. **Roberto Molina-Robles:** Software, Validation. **Reinaldo Castro-Gonzalez:** Software. **Egdar Solera-Bolanos:** Investigation, Software, Validation. **Gabriel Madrigal-Boza:** Investigation, Software. **Marco Oviedo-Hernandez:** Investigation, Software. **Diego Salazar-Sibaja:** Software. **Dayhana Sanchez-Jimenez:** Software. **Melissa Fonseca-Rodriguez:** Software. **Johan Arrieta-Solorzano:** Software. **Renato Rimolo-Donadio:** Funding acquisition, Project administration, Conceptualization, Supervision, Writing - review & editing. **Alfredo Arnaud:** Conceptualization, Funding acquisition, Writing - review & editing. **Matias Miguez:** Writing - review & editing. **Joel Gak:** Writing - review & editing.

Acknowledgments

Work supported by ANII – Uruguay grant FMV_1_2017_136543. Thanks to Universidad Católica del Uruguay and Instituto Tecnológico de Costa Rica. Thanks also to the Europractice IC Service, and Synopsys Electronic Design University program.

References

[1] G.I. Alkady, I. Adly, H.H. Amer, T.K. Refaat, Mitigation of soft and hard errors in FPGA-based pacemakers, in: 2018 13th International Conference on Computer Engineering and Systems (ICCES), 2018, pp. 284–289.

[2] I. Kuon, J. Rose, Measuring the gap between FPGAs and ASICs, in: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, FPGA '06, ACM, New York, NY, USA, 2006, pp. 21–30.

[3] A. Boutros, S. Yazdanshenas, V. Betz, You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference, ACM Trans. Reconfigurable Technol. Syst. (TRETS) 11 (2018) 20:1–20:23.

[4] D. Patterson, A. Waterman, The RISC-V Reader: an Open Architecture Atlas, Strawberry Canyon, 2017.

[5] M. Gautschi, et al., Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices, IEEE Trans. VLSI Syst. 25 (2017) 2700–2713.

[6] R. Uytterhoeven, W. Dehaene, A sub 10 pJ/cycle over a 2 to 200 MHz performance range RISC-V microprocessor in 28 nm FDSOI, in: 44th European Solid-State Circuits Conference (ESSCIRC, 2018, pp. 236–239.

[7] P. Chiu, C. Celio, K. Asanović, D. Patterson, B. Nikolić, An out-of-order RISC-V processor with resilient low-voltage operation in 28nm CMOS, in: IEEE Symposium on VLSI Circuits, 2018, pp. 61–62.

[8] C. Salazar-Garcia, R. Castro-Gonzalez, A. Chacon-Rodriguez, RISC-V based sound classifier intended for acoustic surveillance in protected natural environments, in: 2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS), 2017, pp. 1–4.

[9] Ultrasoc Technologies Ltd, 2019.

[10] Codasip Ltd., 2019.

[11] Sifive, 2019.

[12] A. Arnaud, M. Miguez, J. Gak, R. Puyol, R. Garcia-Ramirez, E. Solera-Bolanos, R. Castro-Gonzalez, R. Molina-Robles, A. Chacon-Rodriguez, R. Rimolo-Donadio, A risc-v based medical implantable soc for high voltage and current tissue stimulus, in: 2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS), 2020.

[13] R. Garcia-Ramirez, A. Chacon-Rodriguez, R. Castro-Gonzalez, A. Arnaud, M. Miguez, J. Gak, R. Molina-Robles, G. Madrigal-Boza, M. Oviedo-Hernandez, E. Solera-Bolanos, D. Salazar-Sibaja, D. Sanchez-Jimenez, M. Fonseca-Rodriguez, J. Arrieta-Solorzano, R. Rimolo-Donadio, Siwa: a risc-v 32i based micro-controller for implantable medical applications, in: 2020 IEEE 10th Latin American Symposium on Circuits Systems (LASCAS), 2020.

[14] A. Waterman, K. Asanovic, The Risc-V Instruction Set Manual-Volume I: User-Level Isa-Document Version 2.2, RISC-V Foundation, 2017. May 2017.

[15] A. Waterman, Risc-v Privileged Architecture, 2017.

[16] D. Patterson, A. Waterman, The RISC-V Reader: an Open Architecture Atlas, Strawberry Canyon, 2017.

[17] R. Garcia Ramirez, Development of Integrated Electronics Subsystems for Biomedical Applications, Ph.D. thesis, Instituto Tecnologico de Costa Rica, Costa Rica, 2019.

[18] C. Nagendra, M.J. Irwin, R.M. Owens, Area-time-power tradeoffs in parallel adders, IEEE Trans. Circuit. Syst. II: Analog Dig. Sign. Process. 43 (1996) 689–702.

[19] N.H. Weste, D.M. Harris, Integrated Circuit Design, Pearson, 2011.

[20] S. Huntzicker, M. Dayringer, J. Soprano, A. Weerasinghe, D.M. Harris, D. Patil, Energy-delay tradeoffs in 32-bit static shifter designs, in: 2008 IEEE International Conference on Computer Design, 2008, pp. 626–632.

[21] Z.T. Irwin, D.E. Thompson, K.E. Schroeder, D.M. Tat, A. Hassani, A.J. Bullard, S.L. Woo, M.G. Urbanchek, A.J. Sachs, P.S. Cederna, W.C. Stacey, P.G. Patil, C.A. Chestek, Enabling low-power, multi-modal neural interfaces through a common, low-bandwidth feature space, IEEE Trans. Neural Syst. Rehabil. Eng. 24 (2016) 521–531.

[22] S. Rodriguez, S. Ollmar, M. Waqar, A. Rusu, A batteryless sensor ASIC for implantable bio-impedance applications, IEEE Trans. Biomed. Circuit. Syst. 10 (2016) 533–544.

[23] C.S. Mestais, G. Charvet, F. Sauter-Starace, M. Foerster, D. Ratel, A.L. Benabid, Wimagine: wireless 64-channel ECoG recording implant for long term clinical applications, IEEE Trans. Neural Syst. Rehabil. Eng. 23 (2015) 10–21.

[24] C. Duran, D.L. Rueda, G. Castillo, A. Agudelo, C. Rojas, L. Chaparro, H. Hurtado, J. Romero, W. Ramirez, H. Gomez, J. Ardila, L. Rueda, H. Hernandez, J. Amaya, E. Roa, A 32-bit RISC-V AXI4-lite bus-based microcontroller with 10-bit SAR ADC, in: 2016 IEEE 7th Latin American Symposium on Circuits Systems (LASCAS), 2016, pp. 315–318.

[25] C. Duran, L.E. Rueda, G, A. Amaya, R. Torres, J. Ardila, L. Rueda, D, G. Castillo, A. Agudelo, C. Rojas, L. Chaparro, H. Hurtado, J. Romero, W. Ramirez, H. Gomez, H. Hernandez, E. Roa, A system-on-chip platform for the internet of things featuring a 32-bit RISC-V based microcontroller, in: 2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS), 2017, pp. 1–4.

[26] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, L. Benini, Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications, in: 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation, PATMOS, 2017, pp. 1–8.

[27] A. Stillmaker, B. Baas, Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm, Integration 58 (2017) 74–81.

[28] X. Zhang, H. Jiang, L. Zhang, C. Zhang, Z. Wang, X. Chen, An energy-efficient ASIC for wireless body sensor networks in medical applications, IEEE Trans. Biomed. Circuit. Syst. 4 (2010) 11–18.

[29] X. Zhang, H. Jiang, B. Zhu, X. Chen, C. Zhang, Z. Wang, A low-power remotely-programmable MCU for implantable medical devices, in: 2010 IEEE Asia Pacific Conference on Circuits and Systems, 2010, pp. 28–31.