A Verified RISC-V I Based Processor with an External Debugging Capability

Hanssel Morales, Director PhD Elkim Roa
Department of Electrical and Electronics Engineering,
Universidad Industrial de Santander, Colombia.

***Abstract*—Reaching low-energy consumption is one of the critical challenges in low-cost, and battery-powered internet of things (IoT) sensor nodes. These nodes are usually coordinated by a processor which collects, operates, and transmits data from multiple sensors to the cloud. IoT processors must manage workloads characterized by intermittent bursts of compute-intensive operations mixed with prolonged periods of low activity. Dynamic power minimization through low switching microarchitectures has been the focus of recent solutions. Though this can lead to low execution efficiency increasing the execution time, and therefore, leading to high energy loss. This work explores the addition of soft-acceleration techniques to enhance the computing efficiency in the design of Arcabuco, a RISC-V I/IM based single-issue in-order processor. Which achieves 2.7 CoreMark/MHz and 1.13 DMIPS/MHz, with a power consumption of 84.46μW/MHz. Providing an alternative solution to low-energy limitations.**

I. Introduction

Battery-constrained IoT processors design usually bases on low switching architectures to reduce the overall power consumption. For example, the ARM Cortex-M0 [1] is a single-issue in-order core with three pipeline stages that implements the Armv6-M instruction set architecture (ISA) and presents 2.33 CoreMarks [2] per MHz in conjunction with 66μW per MHz. In [3] C. Duran et. al. Present a RISC-V based processor called mRISC-V which comprises an area-optimized implementation of the integer and iultiplicative ISA extensions that exposes 97μW per MHz and 0.305 MIPS [4] per MHz. The design of these processor designs focuses on low-area and low-power although neglecting their computing efficiency.

Fig. 1 Illustrates a qualitative example of the energy consumption in an IoT sensor node where most of the time the devices are in a "sleep-mode" and only "wake-up" when an operation is required. Although core two has lower dynamic power consumption (red), it spends more time running a certain program. Conversely, core one presents a higher dynamic power consumption (green) but with a lower execution time for the same workload. Finally, the core with higher power consumes less energy due to an appropriate design that prioritizes the use of the data-path to extract more computation per cycle, allowing to achieve execution efficiency.

To address the energy reduction problem, this work explores the addition of soft-acceleration techniques (i.e, pipelining, branch prediction, forward unit, direct memory access(DMA)) to enhance the computing efficiency in the design of a RISC-V based processor. To measure computing efficiency this work utilises two embedded processor benchmarks, Dhrystone [4] and Coremark [2]. In addition, a set of verification strategies (formal verification and coverage-driven test generation) were applied in order to obtain a reliable design. Besides, to accomplish an optimal post-silicon verification, this processor comprises a debug interface. The implemented debug interface enables the to control and monitor internal processor execution states even after silicon chip fabrication trough a debug platform [5].

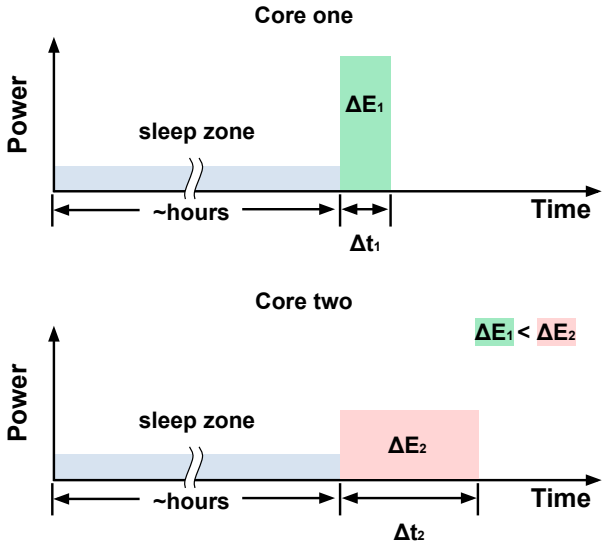

Fig. 1. Qualitative example of the energy consumption in an IoT processor

II. Processor Microarchitecture

Fig. 2 presents the microarchitecture implemented, a single issue in order (SIIO) five-stage pipelined processor. This implementation was based on the computer architecture book [6] which shows the architectural design process from a higher-level of abstraction. The five stages are divided by their functional purpose: Instruction Fetch, Instruction Decode, Execute, Memory, and Write-Back stages.

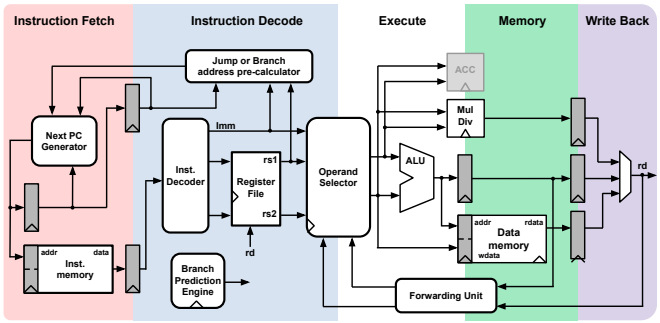

Fig. 2. Arcabuco Microarchitecture Block Diagram.

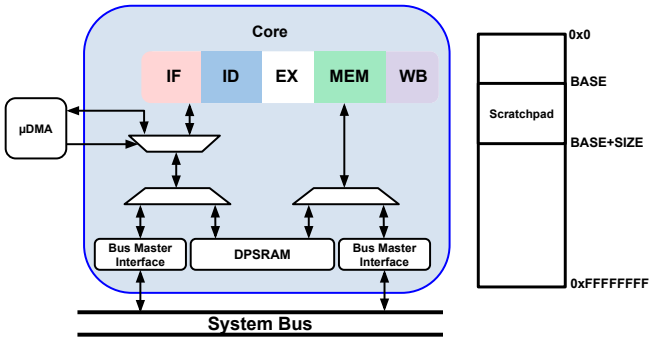

Fig. 3. Memory Access subsystem Block Diagram

A. Instruction Fetch

The design of an instruction fetch stage should focus on high instruction throughput and low latency in order to maximize the usage of the following stages. As long as the bus performance will variate depending on the number of masters connected in the System-on-Chip (SoC) and the number of transactions required by the application. The instruction throughput and latency will be imposed by the external bus performance.

The design reduces this bus performance dependency by implementing a memory hierarchy technique called scratchpad [7], which consists of fast access dedicated memory addressed on the SoC's memory map but only accessed by the processor. The implemented scratchpad architecture in Fig. 3 uses a true dual-port SRAM bank from TSMC 180nm library. The SRAM counts with one cycle latency, 32 bits of word-length, and parametrizable depth, which gives us on each port a reading bandwidth of 32bits/cycle. The external bus is a 32bits AHB-lite based implementation [8] which achieves maximum bandwidth of 32bits/cycle as long as the master interface takes control of the system bus. The bus and scratchpad word lengths were selected, considering that the processor will execute RISC-V 32bits instructions and counts with a single execution issue. With the word-length selection and the parallel generation of the program counter value, the instruction fetch achieves one instruction per cycle of throughput when accessing the scratchpad. The instruction fetch has one penalty cycle on the latency side when there is a jump in the program counter and the target address is located in the scratchpad. The penalty occurs due to the inherent latency of the memory and the register used to avoid merging combinational paths in the SRAM and the instruction decoder.

B. Instruction Decode

The instruction decoding stage is responsible for general purpose registers management in the registers file and the identification of the previously fetched instruction as the RISC-V ISA specification [9] requires. After the identification, the stage selects which operands from the register file or the immediate value or the forwarding unit should pass to the execution stage to be operated. In the case of conditional jumps instructions, the instruction decoding stage uses the branch prediction engine to known the most likely result of the branch comparison based on the previous branch results. This prediction is generated by a finite state machine known as saturation counter exposed in Fig. 4. The saturation counter changes its state depending on the previous result of the branch comparison, and the output converges to the most used decision on the previous instructions. The assumption that the most previously used result will be most likely to occur is based on iterative loops' behavior when the decision to jump or not jump will be repeated several iterations until the loop breaks. If there is a conditional jump that will be taken or an unconditional jump, the instruction decoding stage calculates the jump target address based on the immediate value and the actual program counter or based on a register, depending on the jump type to reduce stalling in jump instructions.

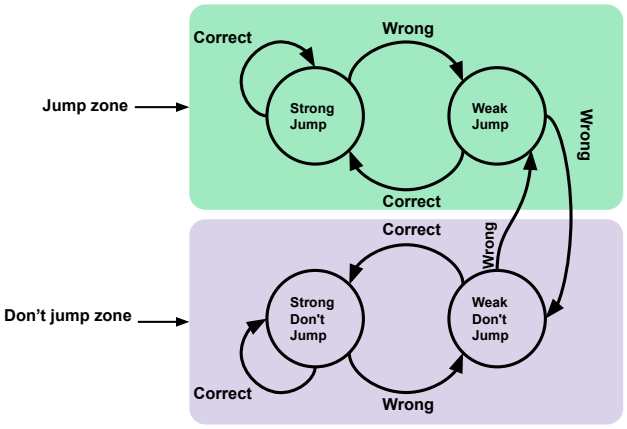

Fig. 4. Saturation counter FSM

C. Execute

The third stage of the processor contains the ALU and other execution units. The implementation of the execution stage has been structured to be extended with additional instruction accelerators. Using a standard handshake interface, we can select different multiplication and division units with variable latency or low area radix based multiplication [10]. As the processor has five pipeline stages, instruction dependency hazards that

occur when an instruction in the execute stage depends on a result that hasn't be stored yet in the register file. Stall and wait for the result is a common practice. However, it implies three cycles of penalty in the worst case, which will reduce the instructions per cycle (IPC) metric drastically. To avoid IPC reduction, the forwarding unit handles the dependency hazards, by monitoring the instructions destine registers in memory and write-back stages. If it matches with one operator in the execution stage, it replaces the value selected from the register file with the correct value.

D. Memory

The transactions with the scratchpad or the system bus are launched from the memory stage and pass through a scope selector in the scratchpad interface, which selects based on the destination address if the transaction goes to the fast access SRAM or the bus master interface. The master interfaces from the core are instantiated from a unique module description and can be changed to support different bus protocols. Due to the merge of the sampling cycle of the SRAM and the input pipe, the fast access transactions do not need to stall the core. Conversely, there will be a stall in bus transactions if the transaction takes more than two cycles, depending on the target slave.

E. Write-Back

Finally, the write-back stage is in charge of the instruction retirement from the pipeline by writing the register file with the instruction's corresponding result that could come from the memory, an instruction accelerator, or the ALU.

III. SoC Implementation

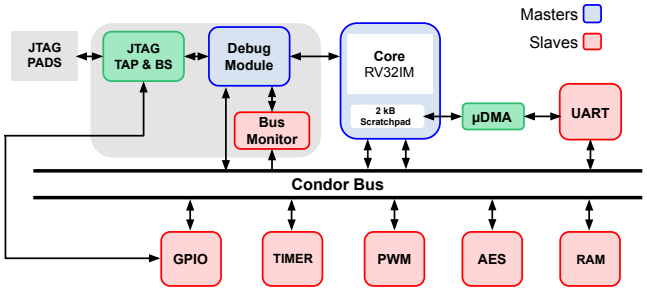

Fig. 5. Testing SoC Block Diagram

Fig. 5 exposes an SoC platform employed to test the processor under embedded applications. The platform communications between masters and slaves are managed by an AHB-lite based bus [8]. A JTAG based debug platform [5] controls and monitor the processor and the bus. An UART interface loads programs through the direct memory access Controller μDMA [11]. A general-purpose input-output GPIO controller enables the core to access the pads. Other peripherals connected in the SoC are a timer, a pulse modulated width PWM controller, and an advanced encryption standard AES accelerator.

A. Debug Mechanism

Post-silicon testing has become a mandatory step of the design flow in modern SoC. This process implies high effort due to the complexity of the integrated circuits [12], the integration of debugging systems to monitor and control the processor in simulation and after fabrication has demonstrated being a reliable practice [13], which increments productivity during testing and increase the probability of the detection of unexpected flaws. The support for a flexible debugger [5] in the processor requires the implementation of adaptation circuits for a debug interface as it is shown in Fig. 6. These implementations enable the users to emulate interruptions, stop the processor with or without breakpoints, insert instructions through the program buffer and, control and monitor the general-purpose registers.

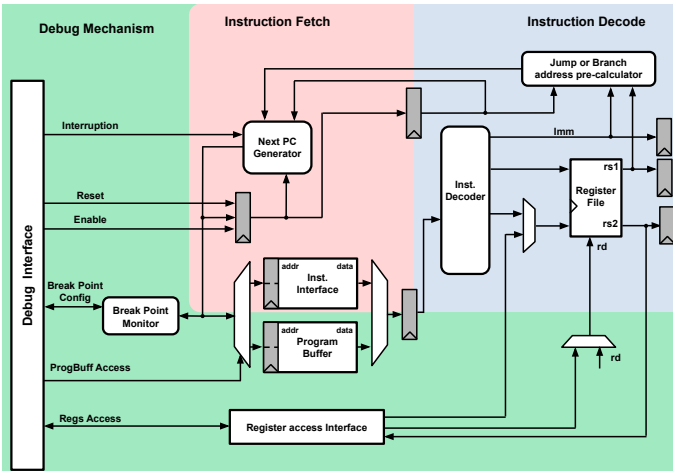

Fig. 6. Debug Interface

B. Programming Mechanism

Booting sequence is a critical consideration in the design of a computing system. There are several options for loading a program into the memory. The most common ways are by adding support for an external memory interface or using the debug interface to load instructions to the system memory [14]. In this case, we add a third way by using a μDMA subsystem based on the low area DMA presented in [11], able to load instructions from a UART interface directly into the scratchpad. The μDMA intervenes in the instruction port by sending nop operations to the processor while writing to the scratchpad. This instruction load method minimizes the processor movement of instructions during the boot-loader execution and reduces the usage complexity Fig. 7. Shows how a program is loaded into the system after its compilation. The program's binary source is sent as an argument for a script that uses a serial converter driver to send it to the SoC.

IV. Verification and Evaluation

Verification is a crucial process in the design of an integrated circuit. In recent years multiple paradigms of verification have taken relevance in contrast with traditional simulation-based

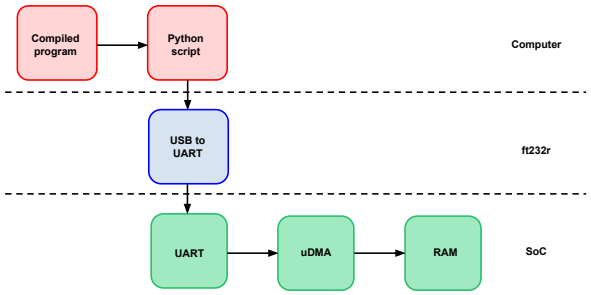

Fig. 7. UART Programming Flow

test-benches [15]. This work uses a combined scheme that integrates coverage-driven and formal verification with FPGA Emulation to produce a reliable design.

A. Coverage Driven Verification

Coverage analysis consists of evaluating the percentage of the circuit excited in a simulation. It groups a set of metrics that can be used to evaluate the quality of the simulation:

- *Block coverage*: measures the number of procedural blocks excited during the simulation, usually correlated with the number of flops excited, depending on the hardware description style.
- *Branch coverage*: evaluates the "if-else" conditions reached.
- *Statement coverage*: counts the number of assignments excited during the simulation.
- *Expression coverage*: the amount of arithmetical and logical expressions used.
- *Toggle coverage*: expresses the percentage of signals that have changed from one to zero or inversely.
- *FSM coverage*: covers the percentage of detected states reached in the simulation.
- *Total coverage*: is a weighted average of the above.

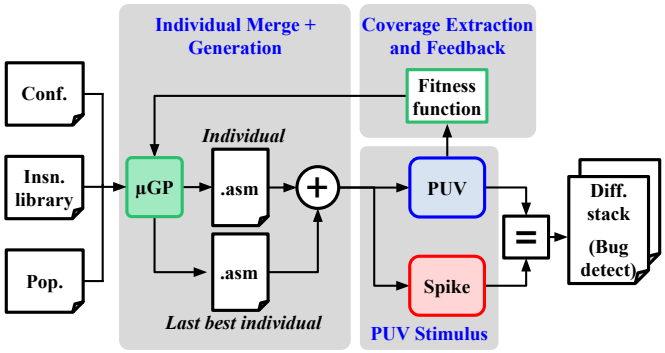

Fig. 8. Test Generation In Coverage Driven Verification

Convectional verification of processors starts with the design of test programs and simulation test-benches after this, the verification engineer evaluates the testing program with the coverage reached, then the program is modified. This process is repeated until a desired coverage is achieved.

The used methodology [15] diverges from conventional because it automatizes the process by using an evolutionary algorithm called μGP, which generates assembly test programs and treats them as individuals with a fitness function of its coverage metrics. This optimization loop is represented in Fig. 8. While this process is running, the individuals are simultaneously executed in a reference model the RISC-V instruction set simulator *spike* [16], this to compare the internal registers with the processor under verification (PUV). Fig. 9. depicts a comparison of the code coverage reached executing with Arcabuco, a widely used cryptography algorithm RSA, the official RISC-V torture unit tests [17] and an assembly program generated by the *μGP* framework.

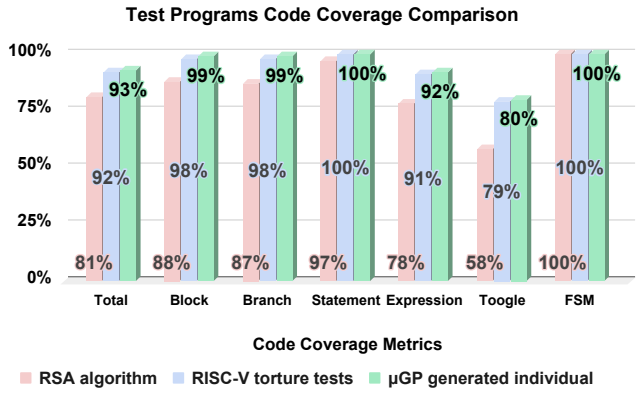

Fig. 9. Code coverage metrics comparison between two C test programs and the best individual generated by the *μGP* framework.

B. Formal Verification

Formal verification is a mathematical technique based on automata theory, discrete event dynamic systems, and graph theory [18]. This technique creates a graph of dynamic states based on the fact that every digital circuit can be expressed as a finite state machine. The verification engineer defines a set of formal properties, e.g., assertions and assumptions, which describes the subspace of valid states. The mathematical engine using Boolean satisfiability solvers searches in the states graph if the formal properties can be violated. The costly part when applying formal is the property description. It implies high engineering time and model checking of the reference ISA. In order to verify the correctness of our implementation against a formal model of the ISA specification, we use *RISC-V formal*.

RISC-V formal is a non-invasive processor-independent formal verification framework of RISC-V based processors [19]. It consists of a formal description of the RISC-V ISA and the specification for the *RISC-V formal* interface (RVFI). Fig. 10 exhibits a simplified diagram of the interconnection using the *RISC-V formal* interface (RFVI) to verify the processor

formally. The RVFI must carry the state of execution of all the instruction to a formal environment, which requires synchronizing all the necessary signals from any stage of the processor architecture to the write-back stage. In this way, the formal properties inside the *RISC-V formal* environment compare the instructions specifications against the processor's internal final-states.

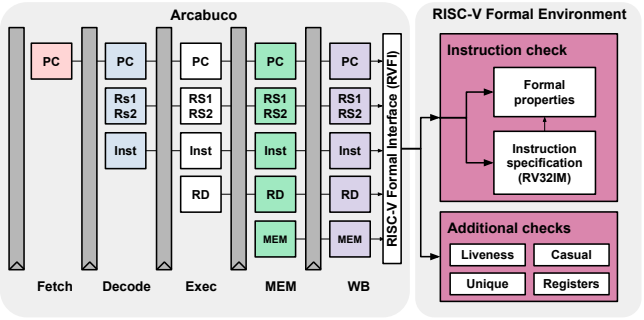

Fig. 10. RISC-V formal frameworok

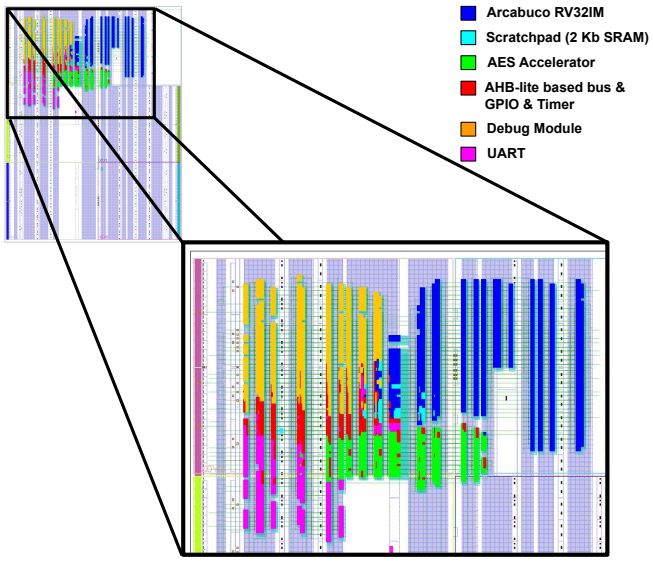

Fig. 11. Placed and Routed Design Using an A7-35T FPGA

C. FPGA Prototyping

Prototyping in FPGA allows earlier testing of digital integrated circuits, therefore with the purpose of earlier behavioral validation, benchmarking, and unit testing. The design showed in Fig.5 was synthesized with Vivado FPGA toolchain and mounted in an Arty A7-35T FPGA Development Board. In order to achieve behavioral emulation, Verilog models of technology-dependent-IP such as SRAM were described using the dual-port Block RAMS (BRAM) from the Arty FPGA [20]. The RISC-V torture unit tests [17] and other program demos were loaded into the design in the FPGA in order to check functional correctness in the emulated processor.

TABLE I
UTILIZATION RESULTS USING AN A7-35T FPGA

Circuit	LUT (20800)	Registers (41600)	BRAM (50)
Arcabuco(RV32I)	1839	1458	1
Arcabuco(RV32IM)	2107	1618	1
Test SoC (RV32IM)	5645	4819	2

To achieve a higher correlation between syntheses reports and place and route final reports. A compact floor-plan and structured IO planning is required to avoid enormous net delay due to parasitic capacitance in sparse routes across the FPGA chip. Fig. 11 shows the place and route result of the implemented prototype in the Arty FPGA. The utilised IO were selected in order to occupy the upper tiles of the FPGA chip and the floor-plan was structured to get a smooth data-flow from the UART and the Debug Module (left) to the processor core(right).

D. Benchmarking

Two widely used industry benchmarks for embedded processors Coremark [2] and Dhrystone [4], were ported to the core to evaluate the performance and compare it against commercial and academic embedded processors. Both benchmarks contain integer arithmetic and control code operations, which represent commonly used programs in embedded applications. But Coremark complements the integer arithmetic with matrix operations, and the Embedded Microprocessor Benchmark Consortium promotes cormark as a compiler independent benchmark. Both benchmarks export results as a value calculated dividing the number of executed iterations on time spent in the execution of the workload; therefore, a common practice is to normalize the value with the core frequency to evaluate execution efficiency.

The FPGA prototype exposed in Fig. 11 was utilized to characterize the performance of Arcabuco, running both benchmarks. The memory size was adjusted depending on the benchmarks requirements (18Kb for Dhrystone and 36Kb for CoreMark), and the timer peripheral was used to obtain the corresponding execution time. The benchmarking results are exposed in Table II, and compared against the results exposed in the ARM-Cortex-M Specifications [1]. Arcabuco achieves higher computational performance than an Arm cortex-M0+, and lower performance than an Arm cortex-M3. An hypothesis for the drop in performance in the coremark benchmark between Arcabuco RV32IM and RV32I implementations. Is that the matrix operations increase the amount of multiplication and division instructions utilized, leading to higher time spent on the software emulation of the multiplication extension instructions.

V. PHYSICAL DESIGN

TSMC 0.18μm technology provides two standard cells libraries 1.8V (low area/high frequency/low dynamic-power) and 3.3V (low leakage). Each library comes with a total of six voltage, process, and temperature (PVT) characterization

TABLE II
BENCHMARKS, POWER AND AREA COMPARISON

Processor	Drystone (DMIPS/MHz)	CoreMark (CoreMarks/MHz)	Area (μm^2)	Power Efficiency (μW/MHz)
Arm cortex-M3	1.25	3.34	350000@180nm	141
Arcabuco RV32IM	**1.13**	**2.7**	**250000@180nm**	**84.45**
Arm cortex-M0+	0.95	2.46	98000@180nm	47.4
Arm cortex-M0	0.87	2.33	110000@180nm	66
Arcabuco RV32I	**0.87**	**1.2**	**200000@180nm**	**69.1**
mRISC-V RV32IM	0.305	-	120776@130nm	97

corners showed in Table III. The synthesizer uses these libraries to map RTL designs to standard cells net-list. And then this net-list is analyzed to get power, timing, and area information. With the results the synthesizer performs optimizations depending on the specified targets in the synthesis process. These optimizations will variate depending on the implemented flow, and the strategies enabled.

TABLE III
1.8V LIBRARY CHARACTERIZATION CORNERS

Corner	Voltage(V)	Temperature (C)	Process
BC	1.98	0	FF
LT	1.98	-40	FF
ML	1.98	125	FF
TC	1.80	25	TT
WC	1.62	125	SS
WCL	1.62	-40	SS

In the case of Arcabuco, the 1.8V library was used in conjunction with Cadence low power synthesis flow, which includes a clock gating strategy. Clock gating consists of the automatic identification of enable logic to zero the clock when the flip-flops are disabled to reduce switching activity in the design. The results of the synthesis applied are exposed in Table IV. As transistor-level models of the standard cells are not provided, the power and operating frequency results that we are able to extract are limited to the characterization corners. The synthesis also provides an area estimation of 478x478μm^2 for the processor core, including cells and nets area.

TABLE IV
SYNTHESIS RESULTS

Corner	WC	TC	BC	LT	ML	WCL
Operating Frequency [MHz]	122	200	273	286	231	149
Power Efficiency [μW/MHz]	66.55	84.45	110.51	108.21	117.51	62.14
Leakage Power [μW]	4.426	0.58	1.24	0.32	127	0.18

In digital Integrated circuit design flow, the place and route is the process that generates layout based on the results from the synthesis step. Fig. 12 illustrates the final layout from the Arcabuco core. This layout occupies an Area of 500x500 μm^2. The increase in the area against the synthesis report is 8%. The hypothesis for that discordance, is that this increase is related to the power ring area and the solution of some

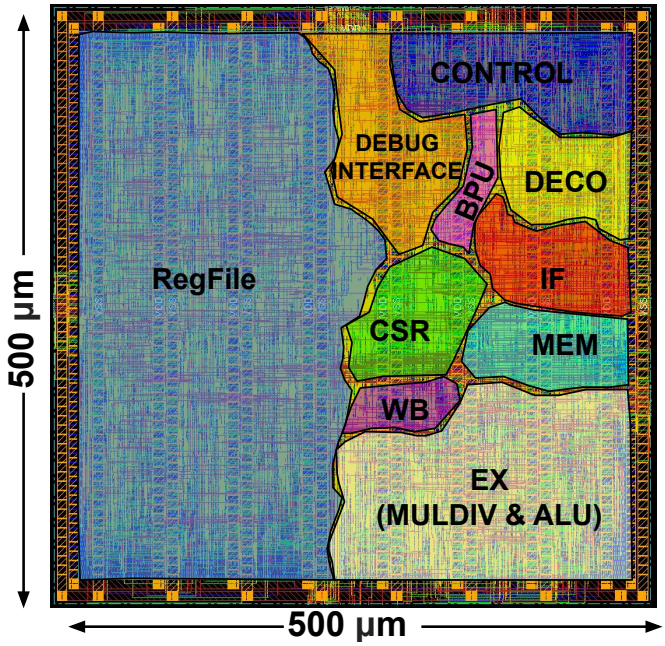

Fig. 12. Arcabuco's layout in TSMC 180nm technology node

routing problems not considered in the synthesis estimation. The sub-module with the highest area consumption in the core is the register file. The Register File module has 31x32-bit registers. A relevant area reduction could be obtained by using an SRAM block for the register file. This could introduce additional latency and timing issues, which could be addressed in future works.

VI. FURTHER CONTRIBUTIONS

- The processor, test-benches, and the FPGA implementation presented in this paper were utilized as a reference design in the laboratories of the computer architecture elective, in the electronics engineering program, at the Universidad Industrial de Santander a long semester 2020-1. This project enabled the students to use, explore and modify the processor design. With the purpose to add custom instructions and peripherals to the testing system.

- Three master's theses are using the Arcabuco core to test their projects. The first thesis explore processor's advanced verification methodologies for embedded systems. The second is related to the design of bus communication protocols in low-power systems. And in the third thesis Arcabuco helped to verify a feed-forward non-harmonic oscillator model

- The designs involved in this project were presented in an application to obtain funds for manufacturing, to the ASIC Program of the Society of Electronic Devices (EDS) Region 9 (Latin America). This request was approved and Arcabuco will be sent to manufacturing in May 2021. Additionally, the post-silicon measurements that will be performed must be reported at an IEEE EDS event, accordingly, to the commitments accepted.

- The work depicted in this document was developed from conclusions gathered in:
  - A paper presented as first-author in the 2019 IEEE 10th Latin American Symposium on Circuits and Systems (LASCAS) tittled "A Low-Area Direct Memory Access Controller Architecture for a RISC-V Based Low-Power Microcontroller" [11].
  - A paper presented as co-author in the 2020 IEEE Custom Integrated Circuits Conference (CICC) titled "An Energy-Efficient RISC-V RV32IMAC Microcontroller for Periodical-Driven Sensing Applications" [14].
  - A paper presented as co-author in the 2020 IEEE International Symposium on Circuits and Systems (ISCAS) titled "Simulation and Formal: The Best of Both Domains for Instruction Set Verification of RISC-V Based Processors" [15].
  - A journal presented as co-author in the IEEE Transactions on Circuits and Systems I: Regular Papers (Volume: 67, Issue: 4, April 2020) titled "On the Cross-Correlation Based Loop Gain Adaptation for Bang-Bang CDRs" [21].
  - Finally one recently accepted coauthored paper in the 2021 IEEE International Symposium on Circuits and Systems (ISCAS) titled "A Low-Cost Bug Hunting Verification Methodology for RISC-V-based Processors"(not published yet).

  These papers belong to the fields of low-power, high-frequency digital integrated circuits design, and verification.

VII. Summary

This work demonstrated a RISC-V IM based processor synthesized and place and routed in CMOS 180nm technology with 200MHz of maximum operating frequency and a power consumption of 84.46μW per MHz in nominal conditions. The design was tested in FPGA and verified among formal verification and with a coverage driven framework. The processor achieves 2.7 CoreMarks/MHz and 1.13 DMIPS/MHz, occupying an area of 500x500 μm^2.

References

[1] ARM, "Cortex-m specs," *Arm Developer*, 2021. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-m

[2] EMBC, "Coremark benchmark," *Embedded Microprocessor Benchmark Consortium*, 2021. [Online]. Available: https://www.eembc.org/coremark/

[3] C. Duran *et al.*, "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.

[4] R. P. Weicker, "Dhrystone: A synthetic systems programming benchmark," *Commun. ACM*, vol. 27, no. 10, p. 1013–1030, Oct. 1984. [Online]. Available: https://doi.org/10.1145/358274.358283

[5] W. Ramirez, M. Sarmiento, and E. Roa, "A flexible debugger for a risc-v based 32-bit system-on-chip," in *2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)*, 2020, pp. 1–4.

[6] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[7] R. Banakar *et al.*, "Scratchpad memory: Design alternative for cache on-chip memory in embedded systems," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, ser. CODES '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 73–78. [Online]. Available: https://doi.org/10.1145/774789.774805

[8] J. Romero, N. Cuevas, and E. Roa, "Energy efficient peripheral and system buses for low-area and low-power soc applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 866–870, 2020.

[9] A. Waterman *et al.*, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[10] D. Guevorkian *et al.*, "A radix-8 multiplier design and its extension for efficient implementation of imaging algorithms," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, T. D. Hämäläinen *et al.*, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 324–333.

[11] **H. Morales**, C. Duran, and E. Roa, "A low-area direct memory access controller architecture for a risc-v based low-power microcontroller," in *2019 IEEE 10th Latin American Symposium on Circuits Systems (LASCAS)*, 2019, pp. 97–100.

[12] H. B. Carter and S. G. Hemmady, *Metric Driven Design Verification*. Springer Science Business Media, LLC, 2007.

[13] F. Refan, B. Alizadeh, and Z. Navabi, "Bridging presilicon and post-silicon debugging by instruction-based trace signal selection in modern processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 7, pp. 2059–2070, 2017.

[14] **C. Duran** *et al.*, "An energy-efficient risc-v rv32imac microcontroller for periodical-driven sensing applications," in *2020 IEEE Custom Integrated Circuits Conference (CICC)*, 2020, pp. 1–4.

[15] C. Duran and **H. Morales** *et al.*, "Simulation and formal: The best of both domains for instruction set verification of risc-v based processors," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–4.

[16] RISC-V Foundation, "Spike RISC-V ISA Simulator," https://github.com/riscv/riscv-isa-sim, 2019.

[17] ——, "RISC-V Tortures Unit Tests," https://github.com/riscv/riscv-tests, 2020.

[18] C. Kern and M. R. Greenstreet, "Formal verification in hardware design: A survey," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 2, p. 123–193, Apr. 1999. [Online]. Available: https://doi.org/10.1145/307988.307989

[19] Symbiotic EDA, "RISC-V Formal Verification Framework," https://github.com/SymbioticEDA/riscv-formal, 2019.

[20] Xilinx, "Artix-7 FPGAs Data Sheet," https://www.xilinx.com, 2020.

[21] J. Ardila, **H. Morales**, and E. Roa, "On the cross-correlation based loop gain adaptation for bang-bang cdrs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1169–1180, 2020.