A 32-BIT RISC-V MICROCONTROLLER IN 130 nm CMOS TECHNOLOGY

CKRISTIAN RICARDO ESTEBAN DURAN BLANCO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD INGENIERÍAS FISICOMECANICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
MAESTRIA EN INGENIERIA DE TELECOMUNICACIONES
BUCARAMANGA
2017

A 32-BIT RISC-V MICROCONTROLLER IN 130 nm CMOS TECHNOLOGY

CKRISTIAN RICARDO ESTEBAN DURAN BLANCO

Proyecto de grado para optar al titulo de Magister en Ingenieria de Telecomunicaciones

Director
ELKIM FELIPE ROA FUENTES
PhD. en Ingenieria

Co-Director
HÉCTOR IVÁN GÓMEZ ORTIZ
MSc en Electronica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD INGENIERÍAS FISICOMECANICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
MAESTRIA EN INGENIERIA DE TELECOMUNICACIONES
BUCARAMANGA
2017

ACKNOWLEDGMENTS

In this section, will be mentioned all the people involved in the realization of this microcontroller.

Hector Gomez. Mixed signal adaptation.
Giovanny Castillo and Anderson Agudelo. AXI4-Lite and APB GPIO interface.
Camilo Rojas and D. Luis Rueda. AXI4-Lite and APB ADC and DAC interface.
Juan Romero. AXI4-Lite to APB converter.
Hugo Hernandez. ADC and DAC analog modules.
Javier Ardila, Luis Rueda, Giovanny Castillo, Anderson Agudelo, Camilo Rojas, D. Luis Rueda, Luis Chaparro, Henry Hurtado and Wilmer Ramirez. Powering, chip sign-off and circuit verification.
Pamela Gomez, Pablo Rendon and Felipe Remolina. ALU module.
Jonathan Torres and Juan Mart´ınez. Instruction Decoder module.
Laura Galvan and Andres Prada. Multiplication module.
Giovanny Castillo and Anderson Agudelo. AXI4-Lite Memory Interface module.
Erika Cruz and Luis Bohorquez. Utility module.
Juan Romero and Oscar Diaz. Collaborated IRQ module.
Hanssel Morales, Alejandro Pulido, Andres Florez, Brayan Herrera, Alex Mantilla and Daniel Cardenas. Online JavaScript processor simulator.
Hector Gomez. Final PCB layout for circuit testing.

CONTENTS

Page.

LIST OF FIGURES

Page.

LIST OF TABLES

Page.

RESUMEN

TITULO: Un microcontrolador de 32 Bits RISC-V en tecnología CMOS de 130nm[*]

AUTOR: Ckristian Ricardo Esteban Duran Blanco[**]

PALABRAS CLAVE: Microcontrolador, Procesador, RISC-V, CMOS, arquitectura, computadores

La quinta generación de procesadores con Set de Instrucciones de Cómputo Reducido (RISC-V por sus siglas en inglés) han presentado un numero gránde de ventajas en comparación a los procesadores de Set de Instrucciones de Cómputo Complejo (CISC por sus siglas en inglés) durante los últimos años. En este trabajo una completa implementación y diseño de un microcontrolador de 32-bits en 130nm totalmente sintetizable es presentada. Este es el primer microcontrolador ofreciendo el set de instrucciones de código abierto RISC-V montado através de buses AXI4-Lite y APB para procesos de comunicación. El microcontrolador contiene una RAM de 4kB, una interfaz SPI esclabo AXI para verificación, y una interfaz SPI esclavo APB para comprobar el correcto funcionamiento del puente APB. Todos los periféricos son controlados por un procesador RISC-V y una interfaz SPI maestro AXI que es usada para programar el dispositivo y comprobar el flujo de datos através de todos los esclavos. Una densidad total de potencia es reportada como 167µW/MHz a 100 MHz y el area de este microcontrolador RISC-V tiene una reducida huella de 798µm x 484µm. Además de esto, un microcontrolador de 32-bits probado y medido, usado como una plataforma de código abierto para el Internet de las Cosas es presentado. El Sistema en Chip (SoC por sus síglas en inglés) ocupa una area de 2.1mm x 2.1mm en una tecnología de 130nm CMOS. El SoC ha sido probado a una máxima velocidad de 160MHz. Este es el primer microcontrolador de 32-bit probado en silicio con un núcleo RISC-V.

[*] Proyecto de grado

[**] Facultad Ingenierías Fisicomecanicas Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones Maestria en Ingenieria de Telecomunicaciones Director: Elkim Felipe Roa Fuentes Co-Director Héctor Iván Gómez Ortiz

ABSTRACT

TITLE: A 32-bit RISC-V microcontroller in 130 nm CMOS technology[*]

AUTHOR: Ckristian Ricardo Esteban Duran Blanco[**]

KEYWORDS: Microcontroller, Processor, RISC-V, CMOS, architecture, computers

The fifth generation of Reduced Instruction Set Computer (RISC-V) processors have presented a large number of advantages in comparison to Complex Instruction Set Computer (CISC) processors over the last years. In this work a complete implementation and design of a fully-synthesized 32-bit microcontroller in a 130nm CMOS technology is presented. This is the first microcontroller featuring the open source RISC-V instruction set all mounted through AXI4-Lite and APB buses for communication process. The microcontroller contains a 4kB-RAM, an SPI AXI slave interface for output verification, and an SPI APB slave interface for checking the correct behavioral of the APB bridge. All peripherals are controlled by a RISC-V and an SPI AXI master interface that is used for programming the device and checking the data flowing through all the slaves. A total power density is reported as 167µW/MHz at 100 MHz and the area for this RISC-V microcontroller has a reduced footprint of 798µm x 484µm. Moreover, a tested and measured 32-bit microcontroller used as an open-source platform for Internet of Things is presented. The implemented SoC occupies a 2.1mm x 2.1mm area in a 130nm CMOS technology. The SoC has been tested at maximum speed of 160MHz. It is the first silicon-proven 32-bit microcontroller sporting a RISC-V core.

[*] Project of grade

[**] Faculty of Electrical and Mechanical Engineering School of Electrical, Electronics and Telecommunications Engineering Master's Degree in Telecommunications Engineering Director: Elkim Felipe Roa Fuentes Co-Director Héctor Iván Gómez Ortiz

INTRODUCTION

Nowadays commercial architecture chip vendors, such as ARM Holdings and MIPS Technologies, charge substantial license fees to use their patents. The architecture se- crecy in these processors interferes with the legitimate public and educational use as well as security auditing, performance analysis, and the development of public, inex- pensive compilers and operating systems in an open-source free software. In order to face this dilemma, the RISC project was created[1] for academic purposes.

One of the main disadvantages of today's commercial processors relies on the fact that the majority of programs and compilers are not aimed to use the processor's set of instructions entirely. These useless instructions can be avoided exploiting the remained area to implement new functions in order to improve the processor performance [2].

Historically, the Berkeley RISC design was commercialized as the SPARC processor setting stones to the MIPS architecture, and inspiring the ARM architecture which powers most mobile phones. The RISC-V architecture is the fifth generation from RISC processor designs [3]. The configurability and efficiency of the RISC-V, and the great impact on power consumption, shows great compatibility with most of the IoT applications.

The Berkeley RISC processors reduce the instruction list by removing all unneces-sary decoding and calculating circuitry. The modified instruction set also adds speed functions and includes improvements such as the addition of registers that, in contrast with normal memories, can be accessed at a negligible cost. In addition,

[1] Developed originally at UC Berkeley, dated back since 1980s [1]

simplification of the instructions for each different operation is improved making the RISC proces- sor a single instruction-variant architecture. Finally, this instructions set provides the possibility to assign constants directly to operations as well as loading data to registers using less memory for sections that have constant data. The above improvements re- sult on lower memory interaction for common processor operations indicating a better performance and making all instruction executions closer to the system clock.

The RISC-V is planned to become an open-source instruction set architecture (ISA), as an open industry standard under the governance of the RISC-V Foundation[2]. By making this architecture generation as an open ISA, the goal is to enhance collabora- tive work between organizations and improve the development of both academic and industrial sectors. In addition, the resulting developments of organizations such as compilers and operating systems can be shared as well as the implementations that use the architecture.

For organizations, the open-source model brings advantages such as: 1) Access to the last improvements (with its respective discussion); 2) The opportunity to develop new and custom functions, such as security and advanced calculations (directly within the core) and communicating them through a bus; 3) The possibility to use this architecture on chip, which will reduce the chip cost dramatically as organizations do not have to pay additional fees for licensing (RISC-V is under the BSD Open Source license [4]).

In this research, the first 32-bit RISC-V microcontroller, using RV32I Basic and RV32M instruction set using a 130 nm CMOS standard technology is presented [5]. The implementation and design are equivalent to commercial microcontrollers implemented with an ARM-M0 core. Features of this microcontroller are: AXI4-Lite and APB buses for interfacing communication between the core and all the

[2] http://riscv.org/

peripherals attached to it, a 4kB-RAM, Serial Peripheral Interface (SPI) slaves for output, a GPIO module, a SAR Analog-to-Digital converter and a Digital-to-Analog converter. The circuit was designed using 130nm CMOS technology and tested using RISC-V tool-chain along with an efficient test algorithm in bus. In addition, a SPI master has been implemented in order to explore all the status of the peripherals while the microprocessor is still executing its program.

Additional to this research, testing and measurements of this microcontroller in transistor level were performed. The implemented SoC occupies a 2.1mm x 2.1mm area in 130nm CMOS technology with a dynamic power of 167µW measured from a 1.2V when the core is clocked at 100MHz. The testing proves SoC can be run at maximum speed of 160MHz and it is the first silicon-proven 32-bit microcontroller sporting a RISC-V core. A demonstration was performed using an online web server capable of running small auto-generated user-defined demos running in the microcontroller. This demo was launched at the 5th RISC-V Workshop and reached a total of 410 users.

Giving the importance of Open Hardware, code about implementations of the RISC- V core, buses and peripheral adapters are exposed in a public GitHub[3]. Verilog code is human-readable and fully-synthesizable in any transistor-gated process.

[3] https://github.com/onchipuis/

1. RISC-V PROCESSOR ARCHITECTURE

Reduced Instruction Set Computer (RISC) is a type of microprocessor architecture that uses a small, highly-optimized, constant cycle, set of instructions, rather than a more specialized set of instructions often found in other types of architectures. Some of the characteristics of a RISC processor are: 1) One or constant cycle execution time. 2) Pipelining. 3) Large number of registers. The optimization of RISC processors are focused on register operations only, making tasks such as storing and loading through memory totally independent. To perform an operation that involves variables saved on memory it is needed to load these information into registers, then the processor operates (i.e. arithmetic, bit shift and bitwise operations) and stores the result in memory. On Complex Instruction Set Computer (CISC) processors these operations are hardware-based, integrating communication and operation in a single instruction.

Due to low-level instructions of the RISC architecture, more RAM is needed to store the assembly level compared to CISC, thus leading to an increment on the complexity of programs on RISC processors. However, the RISC strategy also brings some very important advantages: constants are loaded into the instruction itself, instead of using sections on the memory that carry with these information (read-only sections). The reduced instructions require less transistor area, allowing more general purpose registers and making pipelining possible. Due to the fact that each instruction can be executed in a constant number of clock cycles, a program implemented on a RISC processor can be performed nearly in the same amount of time than using CISC.The RISC-V is the fifth generation of Berkeley RISC, featuring all of the mentioned characteristics plus open-source hardware and software tools.

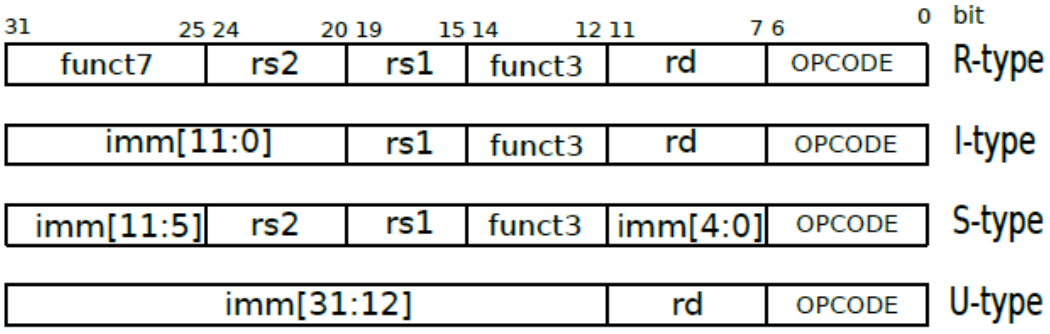
1.1 RISC-V INSTRUCTION SET ARCHITECTURE (ISA)

RISC-V is a new open Instruction Set Architecture (ISA) designed by the Berkeley Architecture Group with the aim to support architecture research and education [5]. RISC-V is fully available to public and has advantages such as a smaller footprint size, compatibility for highly-parallel multi-core or many-core implementations [6], variable- length instructions to support an optional dense instruction, and energy efficiency. Moreover, RISC-V presents improvements in different characteristics over another open ISAs as shown in the Table 2.1.

Table 1. Comparison of the Open ISA

PARAMETER/ISA	SPARCV8	OpenRISC	RISC-V
BASE+EXT	NO	NO	YES
Compact Code	NO	NO	YES
Quad FP	NO	NO	YES
32-bit	YES	YES	YES
64-bit	NO	YES	YES
128-bit	NO	NO	YES
GCC	YES	YES	YES
LLVM	YES	YES	YES
32-bit	YES	YES	YES

Figure 1. RISC-V base instruction formats



The base ISA is clean and suitable for direct hardware implementation. The Fig. 2.1 shows the four core instruction formats (R,I,S,U). R-type format is used for

several arithmetic instructions with one or two source operands, and for the atomic memory op- eration (AMO) instructions that perform read-modify-write operations for multiproces- sor synchronization. In addition, R-type is used for computational instruction register to register. I-type format is used for system instructions to access system functionality that might require privileged access and computational instructions register-immediate. Load and store instructions transfer a value between the registers and memory. Loads are encoded in the I-type format and stores are S-type. U-type format is used for JAL instructions or immediate instructions as LUI (load upper immediate) and AUIPC (add upper immediate to pc). All formats are fixed 32-bit in length, and keep the source (rs1 and rs2) and destination (rd) registers at the same position to simplify decoding. There is also a immediate codification, which MSB is always treated as the sign bit. This can be seen as the embedded constant attached to the instructions.

The RISC-V base integer ISA must be present in any implementation, plus optional extensions to the base ISA. Each base integer instruction set is characterized by the width of the integer registers and the corresponding size of the user address space. The used primary base integer variant, RV32I, provide 32-bit user-level address space. The base integer ISA can be extended with one or more optional instruction-set extensions, but the base integer instructions cannot be redefined. The base integer ISA is named “I” (prefixed by RV32). The standard integer multiplication and division extension is named “M”. Additional info about bit codification is in [5].

1.2 PROGRAMMERS' MODEL FOR BASE INTEGER SUBSET

Figure 2. RISC-V 32-bit user-level base integer register state.

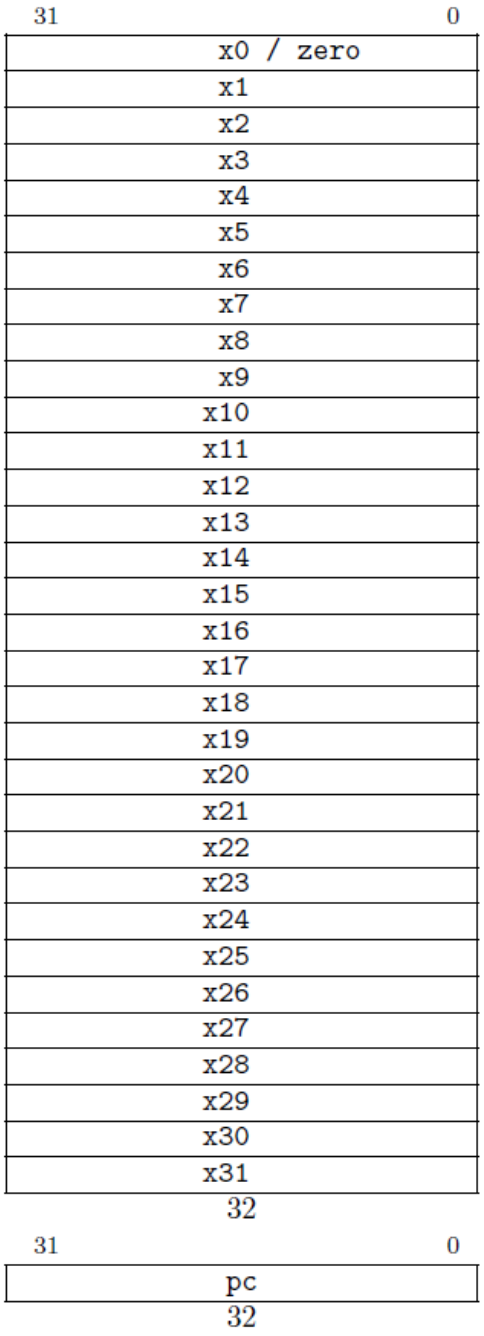


Figure 2.2 shows the user-visible state for the base integer subset. There are 31 general- purpose registers x1–x31, which hold integer values. Register x0 is hardwired to the constant 0. There is one additional user-visible register: the program counter pc holds the address of the current instruction.

1.3 BASE INTEGER INSTRUCTION SET (RV32I)

This is the 32-bit base RISC-V ISA. It is composed by 47 instructions and contains arith- metic instructions, logic instructions, shift instructions, branch instructions, memory instructions, integer stores, misc instructions, and system instructions. This instruction set is mandatory for all RISC-V implementations. In the following subsections a brief abstract of each instruction is described. Codification and more extended implementa- tion definitions are in [5].

1.3.1 Arithmetic instructions. In table 2.2 are listed main Arithmetic instructions. Arithmetic calculations ignores overflow bit, but overflow detection is possible using Branches. ADDI x0, x0, 0 is im- plemented as a software NOP. There are some other instructions like Branches and Memory which always depend of arithmetic operations for calculating results. Func- tional module can be a simple adder with second-operator negate controlled. LUI instruction is listed here but it implements an ALU bypass.

Table 2. List of Arithmetic instructions in RV32I

Inst	Operands	Type	Operation
lui	rd, imm	U-Type	Load upper immediate.
auipc	rd, imm	U-Type	Add upper immediate to pc.
addi	rd, rs1, imm	I-Type	Add immediate.
add	rd, rs1, rs2	R-Type	Add.
sub	rd, rs1, rs2	R-Type	Substract.

1.3.2 Logic instructions Logic instructions are listed in table 2.3. Module definition is only defined by logic gates. Doing an XORI x1, x1, -1 is the same as implementing a NOT x1, x1. For pipeline hazard bubbles, instruction decoder should use a hardware NOP different to the Arithmetic version using typically Logic operations.

Table 3. List of Logic instructions in RV32I

Inst	Operands	Type	Operation
xori	rd, rs1, imm	I-Type	Bitwise XOR immediate.
ori	rd, rs1, imm	I-Type	Bitwise OR immediate.
andi	rd, rs1, imm	I-Type	Bitwise AND immediate.
xor	rd, rs1, rs2	R-Type	Bitwise XOR.
or	rd, rs1, rs2	R-Type	Bitwise OR.
and	rd, rs1, rs2	R-Type	Bitwise AND.

1.3.3 Bit-shift instructions Bit shifting can be performed logically or arithmetic. Certain implementations of this instructions in the table 2.4 can involve an immediate shifter (which solves the operation in 1 clock cycle but frequency is affected. Generally, an implementation always use counters and shift registers. Arithmetic versions of this instructions only appends the 31st bit (sign) to the bit shifting when is a right operation.

Table 4. List of Shift instructions in RV32I

Inst	Operands	Type	Operation
slli	rd, rs1, shamt	I-Type	Shift logical left immediate.
srli	rd, rs1, shamt	I-Type	Shift logical right immediate.
srai	rd, rs1, shamt	I-Type	Shift arith right immediate.
sll	rd, rs1, rs2	R-Type	Shift logical left.
srl	rd, rs1, rs2	R-Type	Shift logical right.
sra	rd, rs1, rs2	R-Type	Shift arith right.

1.3.4 Branch instructions Branch operations in table 2.5 are capable of taking decisions or store the result accord- ing to the desired operation. JAL and JALR are considered unconditional branches. All jumps support 4kB displacement from the pc register, but JAL supports 1MB displacement maximum.

Table 5. List of Branch instructions in RV32I

Inst	Operands	Type	Operation
jal	rd, imm	UJ-Type	Jump and link.
jalr	rd, rs1, imm	I-Type	Jump and link register.
beq	rs1, rs2, imm	SB-Type	Cond branch equal.
bne	rs1, rs2, imm	SB-Type	Cond branch not equal.
blt	rs1, rs2, imm	SB-Type	Cond branch less. Signed
bge	rs1, rs2, imm	SB-Type	Cond branch greater. Signed
bltu	rs1, rs2, imm	SB-Type	Cond branch less. Unsigned
bgeu	rs1, rs2, imm	SB-Type	Cond branch greater. Unsigned
slt	rd, rs1, rs2	R-Type	Set less than.
sltu	rd, rs1, rs2	R-Type	Set less than. Unsigned.
slti	rd, rs1, imm	I-Type	Set less than immediate.
sltiu	rd, rs1, imm	I-Type	Set less than immediate. Unsg.

1.3.5 Memory instructions Listed memory instructions in table 2.6 are capable of save/load from/to a register, using absolute addresses from the desired address register, and relative immediate to this address register. Memory operations can be performed in 3 sizes: Byte, Half-Word and Word. For 32-bit system, the word is 4 bytes.

Table 6. List of Load/Store instructions in RV32I

Inst	Operands	Type	Operation
lb	rd, rs1, imm	I-Type	Load byte. Sign Extended
lh	rd, rs1, imm	I-Type	Load half word. Sign Extended
lw	rd, rs1, imm	I-Type	Load word.
lbu	rd, rs1, imm	I-Type	Load byte. Zero Extended.
lhu	rd, rs1, imm	I-Type	Load word. Zero Extended.
sb	rs1, rs2, imm	S-Type	Store byte.
sh	rs1, rs2, imm	S-Type	Store half word.
sw	rs1, rs2, imm	S-Type	Store word.

1.3.6 System instructions Most of the System instructions listed in table 2.7 needs a multi-layer instance. In simulations or implementations, sbreak is always used to report to an external debugger some failure. Behavioral explanation of the other instructions are beyond this book.

Table 7. List of System instructions in RV32I

Inst	Operands	Type	Operation
sbreak		Constant	Return to debug environment.
fence		Constant	Memory fence.
fence.I		Constant	Free Memory fence.
scall		Constant	Send request to OS.

1.3.7 Misc instructions Those instructions in table 2.8 are only in charge of returning to rd the desired Control and Status register (CSR). Typically are used to measure performance, but can also be used to calculate execution time from reset state.

Table 8. List of Misc instructions in RV32I

Inst	Operands	Type	Operation
rdcycle	rd	I-Type	Read number of cycles
rdcycleh	rd	I-Type	Read number of cycle. High
rdinstret	rd	I-Type	Read number of instructions.
rdinstreth	rd	I-Type	Read number of instr. High
rdtime	rd	I-Type	Read integer from RTC.
rdtimeh	rd	I-Type	Read integer from RTC. High

1.4 EXTENSION FOR INTEGER MULTIPLICATION AND DIVI- SION (RV32M)

This is a 32-bit extension for RISC-V ISA. It is composed by 8 instructions and adds operations to multiply and divide values held in the integer registers.

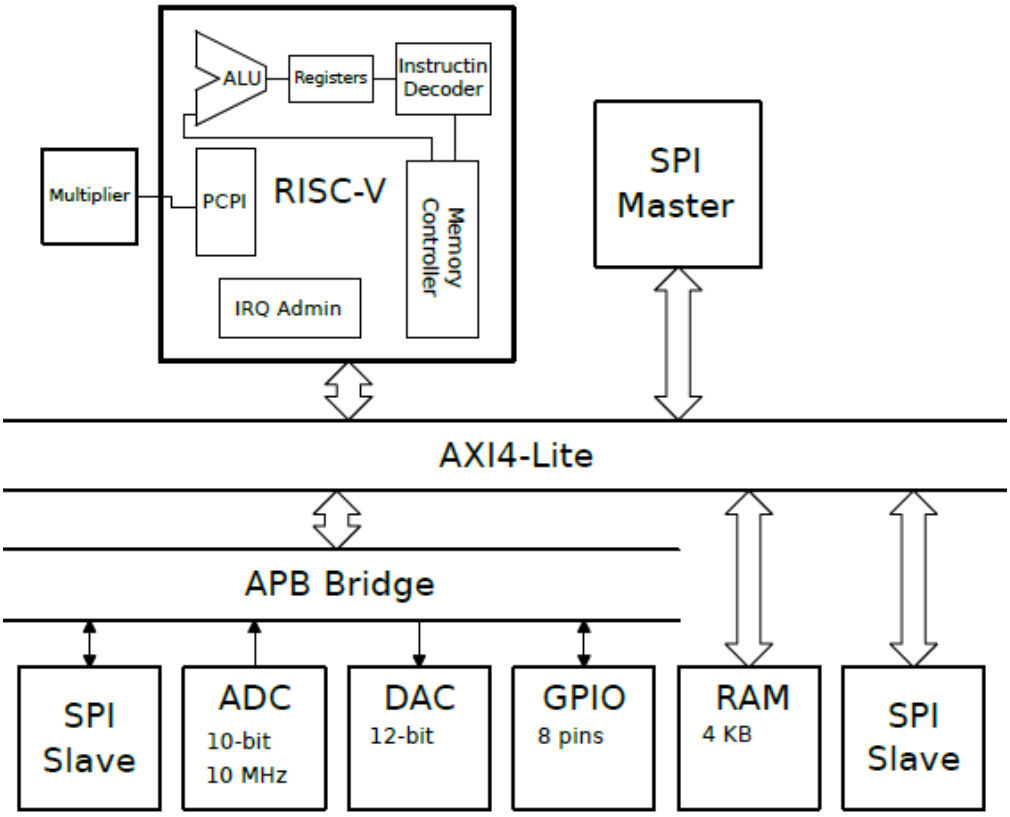
Table 9. List of instructions in RV32M

Inst	Operands	Type	Operation
mul	rd, rs1, rs2	S-Type	Multiply. Ret lower.
mulh	rd, rs1, rs2	S-Type	Multiply. Ret high. sigXsig
mulhu	rd, rs1, rs2	S-Type	Multiply. Ret high. UsigXusig
mulhsu	rd, rs1, rs2	S-Type	Multiply. Ret high. SigXusig
div	rd, rs1, rs2	S-Type	Divide. Sig/Sig
divu	rd, rs1, rs2	S-Type	Divide. Usig/Usig
rem	rd, rs1, rs2	S-Type	Remainder. Sig/Sig
remu	rd, rs1, rs2	S-Type	Remainder. Usig/Usig

1. A 32-BIT RISC-V MICROCONTROLLER IN 130NM CMOS TECHNOLOGY

In this microcontroller, a RISC-V core architecture, communication buses and periph- erals are implemented. A brief architecture of this implementation can be appreciated in figure 3.1. In this implementation, communication buses like AXI4-Lite and APB are used, along with several peripherals described in chapter 4. This implementation is henceforth called ”Open-V”.

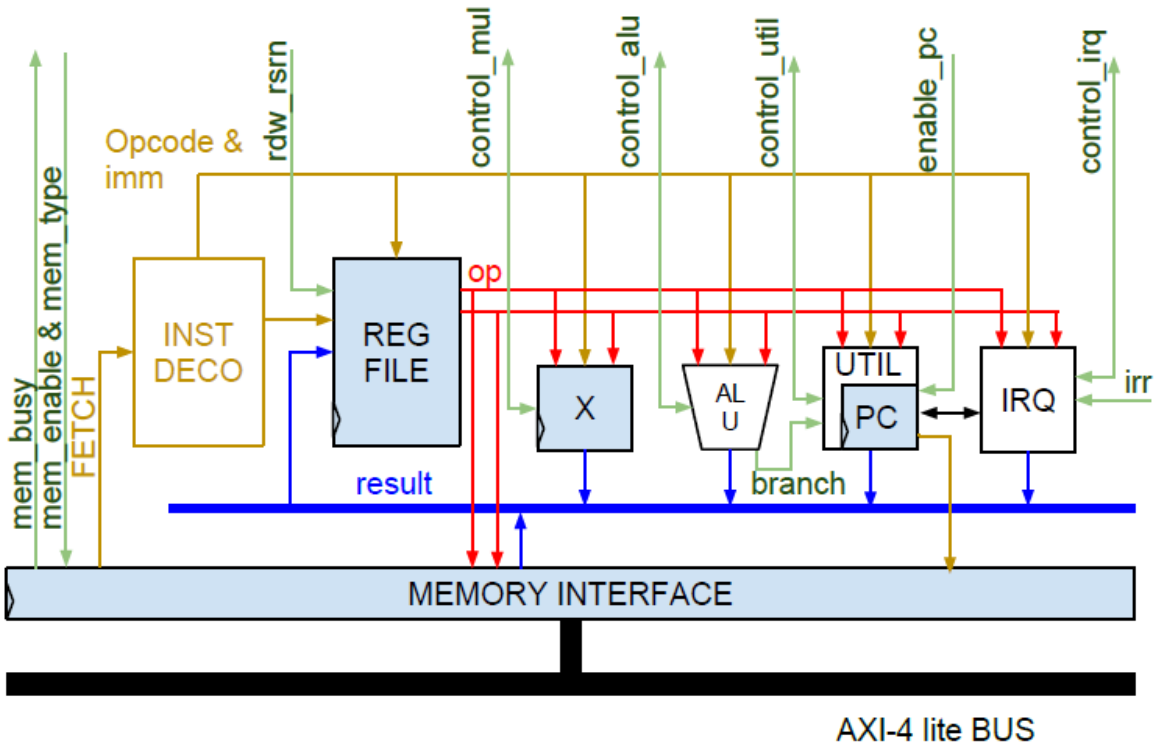
Figure 3. Open-V block diagram.



2.1 OPEN-V CORE ARCHITECTURE

The Open-V implements the RV32I basic instruction set [5], composed by 43 of 47 in- structions omitting FENCE[.I] and SCALL (because these are operative system/multi- core instructions); the instruction MUL[H[U|SU]] of the set M; and custom instructions for handling interrupts . According to the implemented Open-V core architecture in figure 3.2, the pipeline process is divided in 3 stages: Fetch, Exec and Memory; whose are further explained in subsection 3.1.1. Every register access (also named operands) can pass through the ALU, X module, Memory interface (via address for accessing data and retrieving), and Misc modules like Util/IRQ handling and setting. Further expla- nation of implemented modules inside the core architecture are explained in section 3.1.2

Figure 4. Open-V RV32IM architecture



2.1.1 Pipeline stages All pipeline stages are controlled with an external Control module, which is in charge of controlling execution of all pipeline stages. According to each stage, the Control module does an specific action which are further explained in the following subsections. For achieving the desired behavioral, communication between core modules from datapath and the control path is possible through control signals. In figure 3.2 all control signals are indicated with green color.

Fetch:

Fetch is designed to get an instruction from memory, decode the obtained instruction, then put them ready to Exec stage. Results from Fetch stage are indicated in yellow in the figure 3.2.

The instruction fetching is a process that is executed entirely inside the Memory Interface. This behavior was designed as described because memory operations out-side the core are totally asynchronous. If an instruction is not ready to be decoded, the pipeline is stopped through the Control module for avoiding executing the past instruction.

Exec:

Exec decodes the desired instruction using and execution module such as ALU, X or Util. This stage takes the data from Register File module and put the desired operands in every execution module. In figure 3.2, operand signals are in red and results are in blue. Those modules are enabled when a supported operation is recognized through the decoded Operation code (Opcode). If there is a result, the value is passed to the Memory pipeline stage for Write Backs.

For certain types of operations (such as MUL or Shift operations inside ALU) can last longer than 1 clock cycle. Execution modules report this through a ready signal. For preventing data hazards, the pipeline is stopped until the current operation has finished. Calculations of the Program Counter (PC) are executed also here. Further explanation of execution modules are in subsection 3.1.2.

Memory:

Memory stage does the Write Back to the registers and does memory operations through the Memory Interface. A write requests are sent in the first clock cycle. Write Back is sent in the first clock cycle to the Register file module, but if the desired operation is to read data from Memory, memory request is sent in the first cycle, then Write Back is enabled when the Memory Interface has valid data. The Control module is in charge of all memory arbitration between instructions and data (by switching between Fetch or Memory-data commands). As in Fetch stage, if memory operations are not ready when request is sent, pipeline is stopped.

2.1.2 Core modules Core modules can be divided in two major groups: Execution modules, whose do all execution operations; and Functional modules, whose execution are Pipeline stage and Control specific. Most of these modules were developed by undergraduate students through the Computer Architecture I course or enthusiasts inside the Onchip Investi- gation Group.

Execution modules

- Arithmetic-Logic Unit (ALU): The ALU module have all the Arithmetic, Logic and Bit Shift operations performed by the RISC-V ISA. Auxiliary operations for non-related instructions are implemented in other execution modules (like immediate sum to relative addresses for memory operations). Arithmetic according to the second operand. For Shift operations, first the value is

displaced by 4 every clock cycle if second operand is grater than 4. After this sub-operation or if second operand is less than 4, the value is displaced by 1. Additional clock cy- cles are issued for detecting overload or null-shifting by an internal state machine. A Bit-Shifting operation can last between 1 and 15 clock cycles. Implementation of this module was done by undergraduate students.

- Multiplication (X): Multiplication module is a separate digital process which solves a multiplication using Karatsuba Algorithm [7] [8] [9] and Booth Algorithm [10]. For a 32-bit implementation, the combination of this two algorithms calcu- lates the result for any M instruction in 18 clock cycles. Implementation of the algorithm and compatibility with the RISC-V M ISA was done by undergraduate students.

- Utility (UTIL): The Utility module manages all PC register interactions and some additional features such as mandatory RV32I Control and Status Regis- ters (CSR) management for the rd[cycle,instr,time[h]] instructions [5] [11]. The PC calculation is based on the instruction and control bits from other Execution modules such as ALU (for branches) and IRQ (for external interrupts). Addi- tional register which Utility module manages are: clock cycle counter, instruction counter and a basic 1 kHz time counter. Implementation of this module was done by undergraduate students.

- Interrupt request (IRQ): Interrupt request module makes requests to the Con- trol module for external interrupts. Manages special instructions not contained in the RISC-V ISA specification for obtaining current state of the request, interrupt vector and PC restore. This module is capable of returning an End of interrupt (EOI) for returning external modules when interrupt execution is done. All in- terrupt management and responses is software-programmable. Implementation of this module was done in association with undergraduate students and enthusiasts from Onchip investigation group.

Functional modules

- Register File (REG FILE): Register file module is an implementation wrapper of a multi-port 1-Write 2-Read single-clock RAM memory. Because RISC-V ISA specification, wrapper reads zero and ignores writes if address is zero. This RAM was implemented using Flip-Flops, but can be easily replaced by a true dual-port RAM. Addresses for writing and reading comes directly from the instruction fetch, but write enable is only activated if an execution module request a result Write Back, and this action occurs when recognized opcode for an intended module also recognizes saving to register destination (rd).

- Instruction Decoder (INST DECO): Instruction decoder module retrieves the Fetch result from Memory Interface and decodes the Opcode, register read addresses, register destination address, and immediate constants according to the RISC-V ISA. It detects 43 of 47 RV32I instructions omitting operative system/multi- core/cache, 4 RV32M instructions and 7 custom interrupt instructions not in- cluded in the RISC-V ISA. If there is a bit combination that is not inside listed instructions of current implementation, an Illegal Instruction is triggered to the Control module. Implementation of this module was done by undergraduate stu- dents.

- Memory Interface: Memory interface is in charge of do the AXI4-Lite master protocol signaling for requests from Fetch or Memory pipeline stages. If memory requests lasts one clock cycle, this interface is capable of retrieving results from the bus in one clock cycle too, but protocol in general, and this module, are asynchronous. This module holds the value of the last Fetch requested, as well as perform execution-like requests for read/write data. Implementation of this module was done by undergraduate students.

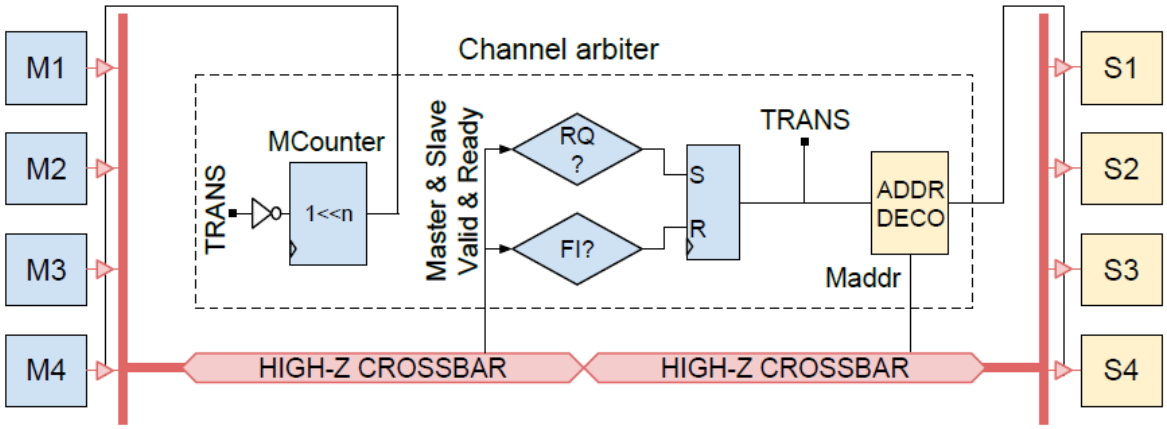
2.2 COMMUNICATION BUSES

Implementation of an efficient microcontroller requires a reliable and fast communica- tion between masters and slaves blocks. Nowadays, many bus-based communication architecture standards are found. In this work, the AMBA and APB protocols are used such that we can compare with microcontrollers based on ARM-M0 cores. The block diagram that follows this connections of the Open-V is shown in Fig. 3.1. The Open-V core in section 3.1 and the SPI master in sub-section 3.2.2 are attached to the AXI4-Lite bus described in subsection 3.2.1, which controls all peripherals. There is also an APB bridge in order to lowering the power consumption in communication, which is explained in subsection3.2.3.

2.2.1 AXI4-Lite The AMBA AXI4 is an ultra-high performance protocol bus standard [12] developed by ARM for easy application in small scale SoCs. AXI4 has different forms to be implemented and this work uses the AXI4-Lite protocol, which has two data channels of 32-bit and control signals for communication between Masters (SPI M, RISC-V) and slaves (RAM, SPI S, APB) [13].

Communication between the masters and the slaves is done through an Interconnect module. This module is in charge of arbitrating all AXI4-Lite transactions between masters and provide full communication between a selected master and a decoded slave. The hierarchy functional description is shown in figure 3.3. In this hierarchy is visible that the crossbar is divided in two: read requests and write requests. This architecture can be separated by this requests because the AXI4 channel structure, allowing to do two independent arbitrations. If certain request is blocked, the other one remains active, featuring asynchronous read / writes from different master / slaves.

Figure 5. AXI4-Lite interconnect functional diagram.



An arbiter basically is composed by a series of enables for controlling High-Z buffers which will be connected to the crossbar, a counter which chooses the desired master, and an address decoder which chooses the desired slave. Preserving by default idle signals for all masters and slaves, a logic detects when is a request for transaction is pending for the selected master through the counter. When a request is detected, a flag is triggered (IS TRANS) and the counter stops counting. The master is connected to the High-Z crossbar and the address decoder uses the requested address and connects the slave to the same crossbar, allowing the selected master and slave continue their transaction.

For addressing decoding per each slave, an address range is given. Figure 3.4 shows the current memory address space implemented inside the decoders. The number of multi-layered interconnected masters / slaves and all the address space are configurable.

2.2.2 SPI Master A SPI is used as a master AXI4-Lite interface for controlling all slaves attached to the core. This interface has a 66-bit data instruction: 32-bit for data, 32-bit for address and 2-bit to define an action such as write, read; to put the core reset; and check the last request. According to the time diagram in fig. 3.5, command is composed by two bits, whose meaning is exposed in table 3.1. A write

or read request is not issued until all the handshake is completed, meaning that the command can be interrupted any time for checking status and reading data after a read request without issuing any write-like commands.

Figure 6. Memory map for Open-V.

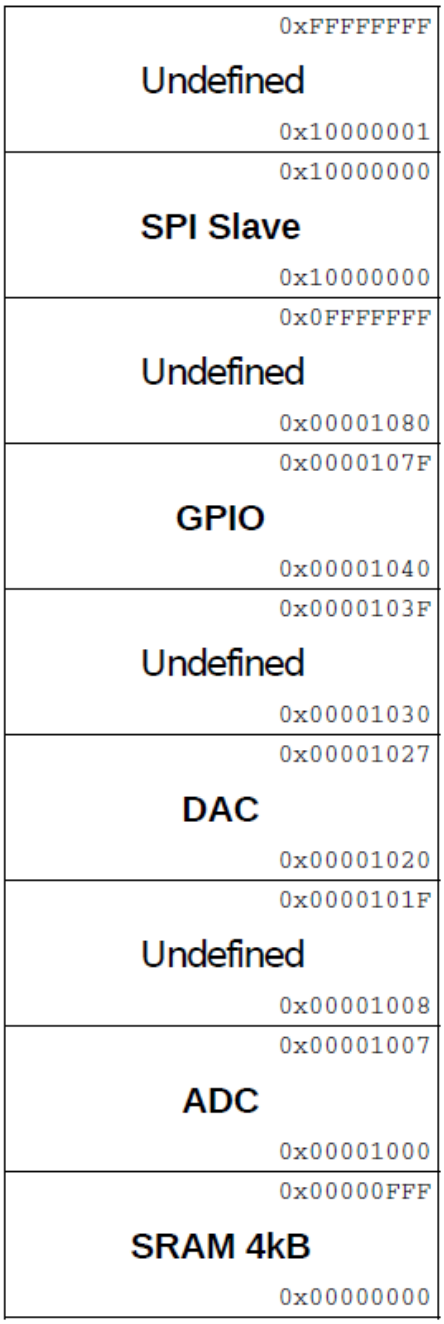


Figure 7. SPI Master timing diagram

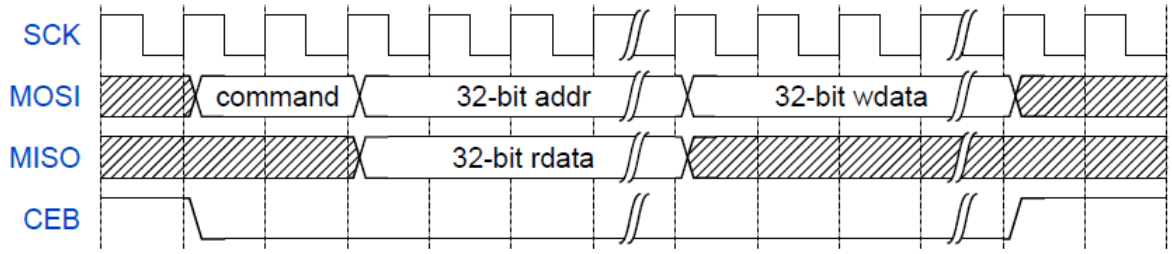


Table 10. SPI Master command usage.

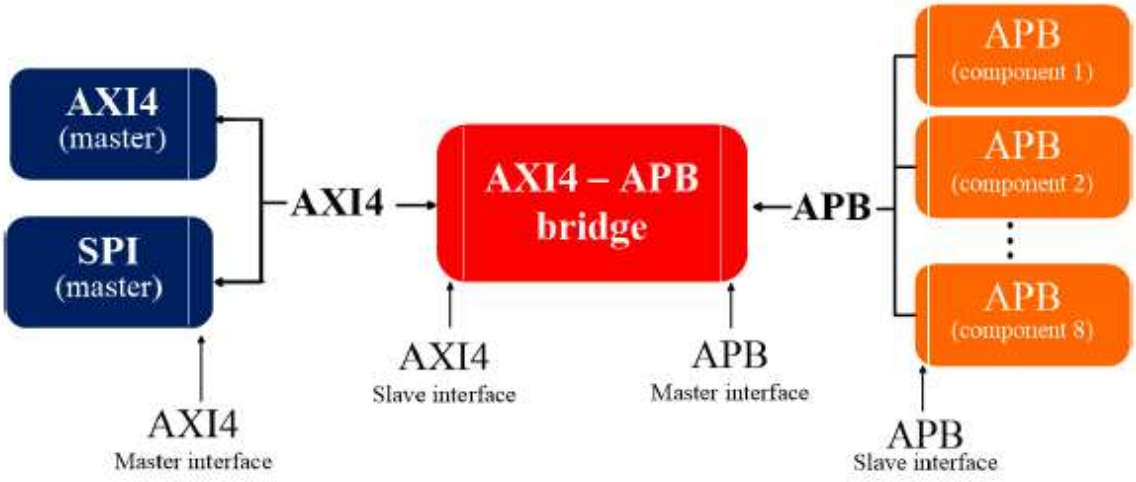
command	Meaning	Input	Output
"00"	Check bus status	None	3-bit busy status
"01"	Request read	Address	None
"10"	Request write	Address & Data	None
"11"	Set reset & retrive read data	Reset status in Data	Data from read request

2.2.3 APB The Advanced Peripheral Bus (APB) is part of AMBA specification, being a low- cost interface that is optimized for minimal power consumption and reduced interface complexity. The APB protocol is not pipelined and connects low-bandwidth peripherals that do not require the high performance of high-speed buses. Every transfer takes at least two clocks cycles[14]. APB bridge was implemented as a the secondary bus on chip responsible for managing low power peripherals. The proposed APB bridge has a low complexity protocol and less speed clock than the principal bus (AXI4-lite), allowing easier interfaces between protocols, reducing power consumption and area.

Usually, the core is assigned as the controller chip (master) and the peripheral is assigned as slaves. In the typical configuration, the AMBA bus handles one or more masters controlling the entire chip and the APB module which is seen as a slave in the AXI4 protocol. This last module acts as a master within the APB protocol, that controls the peripherals connected to it.

In Fig. 3.6 is shown a block diagram of an AXI4-lite to APB bridge used in a simple configuration with two masters and eight APB slaves. The bridge provides an interface between the high-speed AXI4-lite domain and the low-power domain. Read and write transfers on AXI4-lite are converted into the corresponding transfers on the APB and vice versa. The implementation of the bridge was developed by undergraduate students and is explained in [15].

Figure 8. AXI4-Lite to APB bridge blog diagram



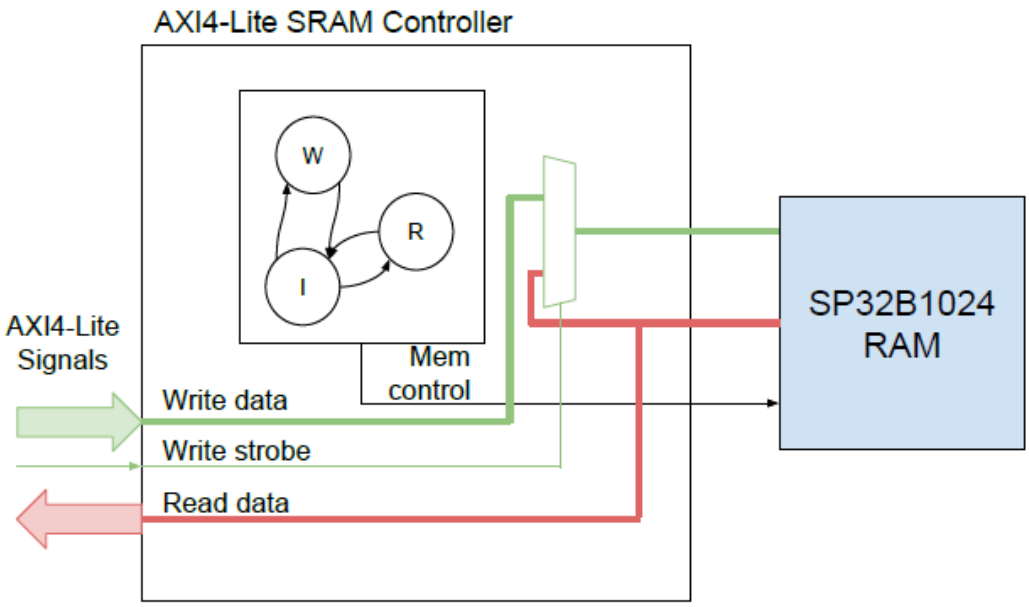
3. PERIPHERALS

Several peripherals are integrated in the microcontroller for giving extended function- ality. This work includes several of them, featuring: SRAM controller for allocating code & data, GPIO controller, and ADC & DAC controllers. Every module have its respective AXI4-Lite or APB interface, mostly developed by undergraduate students.

3.1 SRAM CONTROLLER

The RAM is used to store the program code and the output data or generated data that Open-V loads or stores, which makes one of the most important peripheral in the implementation. The desired memory block to instantiate manages 32-bit wide data and 1024 address (4kB). Figure 4.1 shows the digital module behavioral. One of the difficulties to make this conversion is to manage the byte-wide write strobes (WSTRB) inside the AXI4 specification. The intended memory block is not compatible of managing strobes and there is no way to instance 4 byte-wide memory blocks. A little state-machine is enabled to do the write transactions in order to read first the entire word, then muxes the write word with the desired bytes according to the strobes.

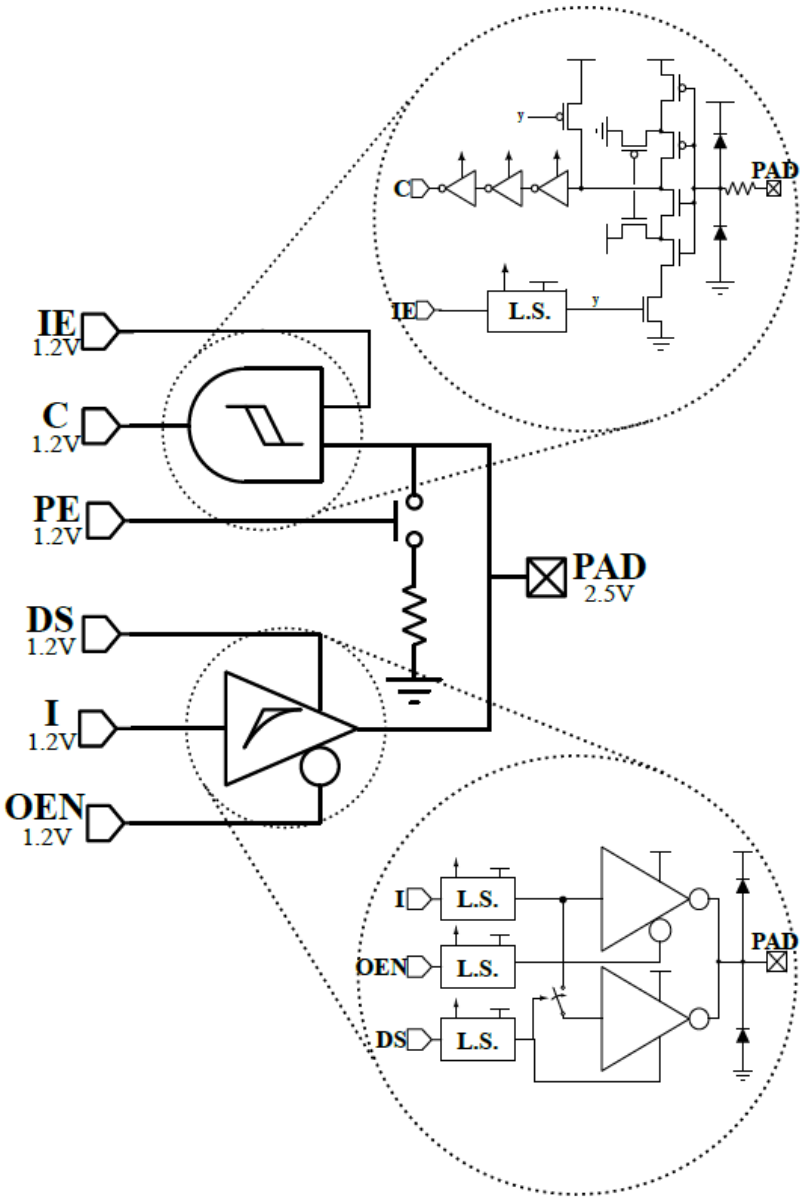
Figure 9. AXI4-Lite SRAM Controller



3.2 GPIO

General-purpose inputs/outputs (GPIOs) are crucial for a variety of microcontroller applications used as digital inputs or outputs. Figure 4.2 shows the block diagram of communication between the core and the pad containing the GPIO, which perform the connection with the APB bridge protocol. This block is controlled by a digital control system designed for speed and low power consumption according to the APB Bridge protocol. The implemented GPIO have slew-rate control capability for out- put and Schmit-Trigger windows for input. The 8-port GPIO is able to drive up to 25mA per output pin. This project was leaded by undergraduate students, and the full implementation details can be found in [16].

Figure 10. GPIO block diagram

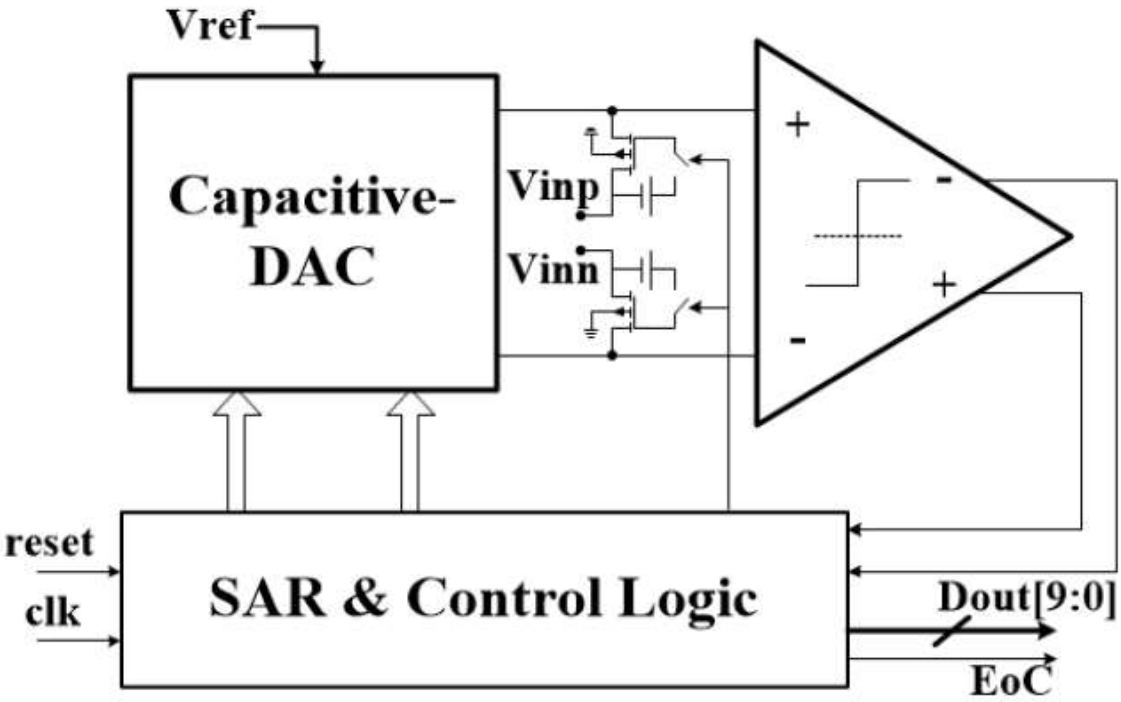


3.3 ANALOG AND DIGITAL CONVERTERS

In order to provide an interface between analog and digital signals in the microcontroller, Open-V incorporates an analog-to-digital converter (ADC) and a digital-to-

analog converter (DAC) which communicate with the coret through APB bridge. The implemented ADC is a successive approximation register (SAR) ADC and its block diagram is shown in Figure 4.3. The SAR ADC implemented is a 10-bit resolution converter, and operates at a maximum sampling frequency of 10MHz, with a maximum peak-to-peak input differential voltage of 1.9Vref. On the other hand, the DAC imple- mented is based in a R2R structure with 12-bit resolution, rail to rail output voltage and a typical settling time of 100ns. ADC and DAC analog modules were donated by Ph.D Hugo Hernandez while working at Universidad Industrial de Santander. ADC and DAC AXI4-Lite/APB interfaces were developed by undergraduate students.

Figure 11. SAR ADC block diagram



4. MEASUREMENTS AND RESULTS

4.1 VERIFICATION

Using the RISC-V core and the SPI master, verification of peripherals has been per- formed. The SPI master is used to program the memory and read or write any pe- ripherals attached to the AXI-4 bus and the APB bridge. Figure 5.1 shows the used initialization and verification tasks. This initialization consists of programming an ini- tial program, which typically is a RISC-V ISA tester, then executes it until program reach a high-level stop signal. After program execution, program is reseted and executed again, while external peripherals are tested at time.

Signal generation and communications transactions are shown in Figure 5.2. The RISC-V core generates AXI4-Lite 1 transactions, while SPI master sends AXI4-Lite master 2 transactions. A verilog methodology is described in order to perform automatic verification for random handshake generation in the AXI master 1 and AXI master 2. As the first step, the AXI master choose a slave, then the transaction type is selected (read or write) depending on the transaction type supported by the slave. Then the data is sent to the bus and the AXI scoreboard registers the incoming information flow between the master and the slave. The data is kept and compared to verify that the transaction is correct. In case of protocol violation, the verification process is paused and the error is reported to the prompt.

Figure 12. Initialization and testing setup for Open-V.

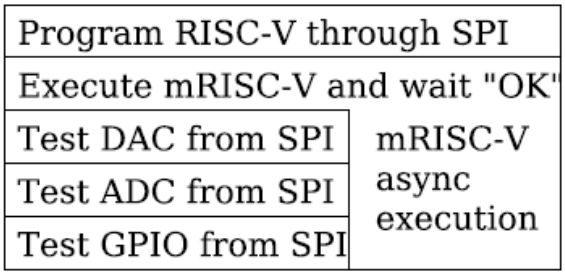
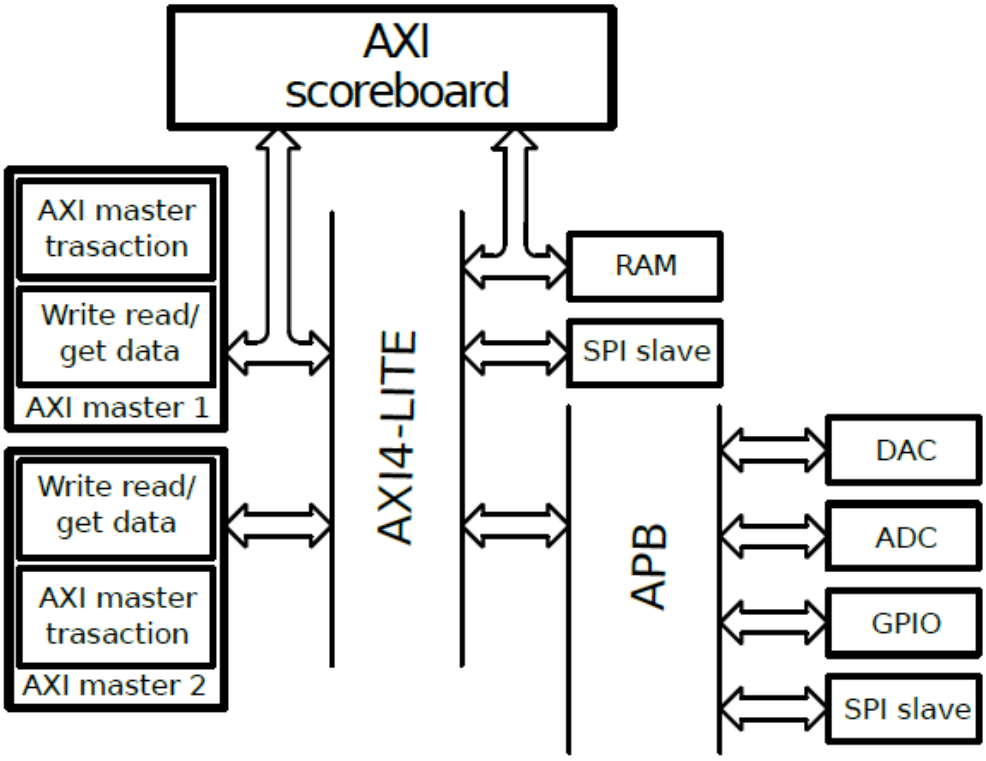


Figure 13. Testbench signaling and verification architecture.



4.2 TSMC 130NM SYNTHESIS

The microcontroller was fully synthesized in 130nm CMOS technology. Synthesis results in Table 5.1 for each peripherals and for whole system with the AXI-APB implemen- tation. The 4 kB RAM module occupies almost the same area as core

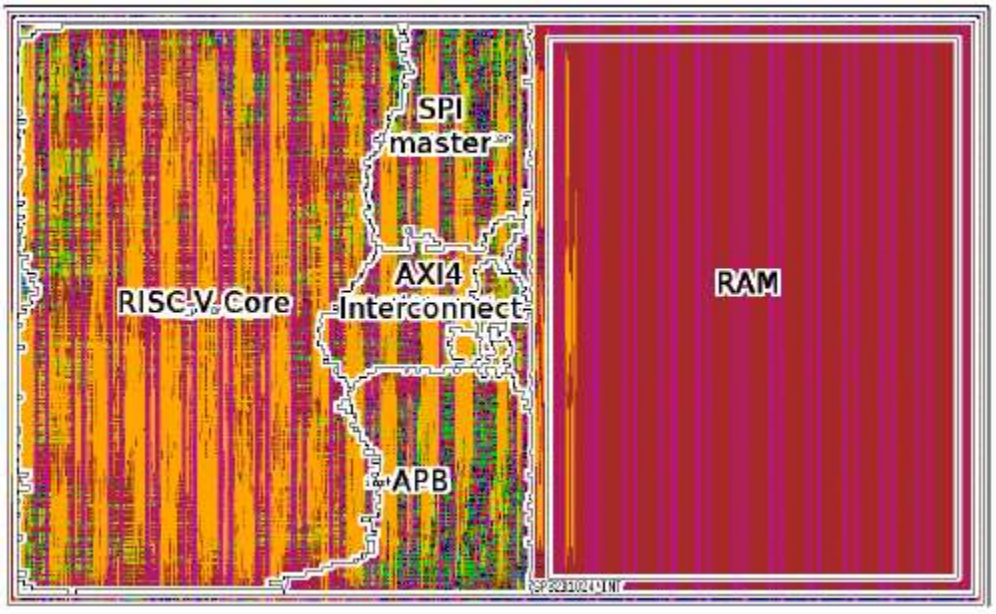
and peripheral controllers both together. The highest power consumption density comes from the RISV-V processor. In synthesis production, maximum operation frequency is deter- mined by the RAM which operates at 100MHz despite the core been able to operate at higher frequency. Some cores, especially the AXI-4 interconnect, uses Hi-Z addressing instead of multiplexer to optimize area.

Table 11. Power, timing and area breakout of the Open-V

Core	Power [nW/MHz]	Time Slack [ps @ 100MHz]	Area [μm^2]
AXI-4 interconnect	5284.66	6093	11830
Open-V core	96952.67	2143	120776
SPI AXI master	13532.69	3998	19627
AXI-RAM	2617.00	3602	2580
RAM	18997.73	N/A	168708
APB TOP	11703.14	3664	23794
SPI AXI slave	2176.93	8085	1899
All	166841.47	1185	349233

The final layout is exposed in the Fig. 5.3. Each instance is highlighted to expose the area breakout. As expected, the RAM block occupies a significant area with a footprint close to 50% of the full chip. A final area of 798µm×484µm and an energy consumption of about 167µW/Hz shows the feasibility of using the proposed Open-V-with additional sensor circuitry- for low-cost and low-power applications.

Figure 14. Final layout for AXI-APB implementation. Area:798µm×484µm



4.3 CHIP PROTOTYPE

The microcontroller, fabricated in the previously synthesized general purpose 130nm TSMC process, has a total chip area of approximately 2.1mm x 2.1mm and its die is observed in Figure5.4 where the peripherals and core are highlighted. Changes on the supply voltages due to high frequency digital signals need to be avoid. To do this, full control of the supplies inductance loops (<2nH) is mandatory. To reach that, the die is directly bonded to the testing board for physical verification. The Figure 5.5 shows the first functional prototype of the platform described in this work.

Figure 15. Die photograph.

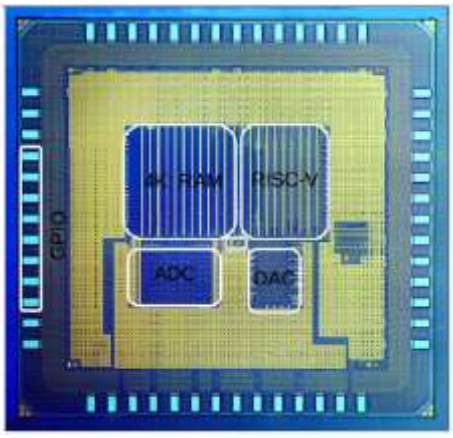
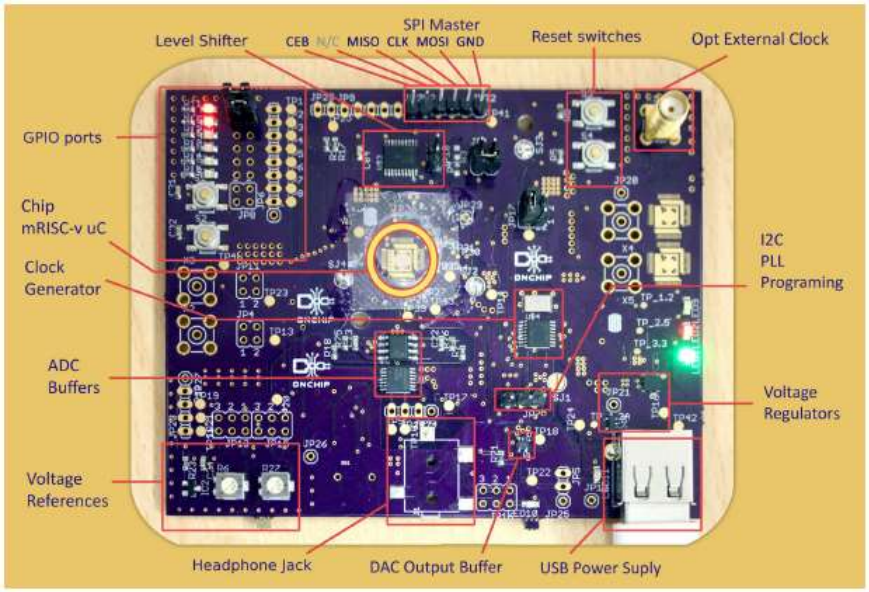


Figure 16. Test Board (PCB).



A summary of the microcontroller performance is presented in Table 5.2. Table 5.3 shows the measured current for different frequencies. The core voltage and the I/O voltage supplies are 1.2V and 2.5V, respectively. The maximum operation frequency tested is 160MHz (by correct algorithm functionality), while the core dynamic power is 167µW/MHz at 100MHz. The dynamic power measurement was performed by running three while loops executed from SRAM, accessing all registers and with all clocks peripheral disabled. Compared to low-power 32-bit

ARM-M0-based microcon- trollers, aside with other commercial ones, the proposed microcontroller presented one of the lowest power consumption considering that the measurement is conditioned to a 100MHz clock, as indicated in Fig. 5.6.

Table 12. Platform performance.

Specification	Value
ISA	RV32IM
Architecture	Single-Issue In-Order 3-stage
Process	TSMC 130nm GP
Die Area	2.1mm x 2.1mm
μP-Core Area	$0.12mm^{2*}$
Max. Frequency	$160MHz^{*}$
Core Voltage	1.2V
I/O Voltage	2.5V
Core Dynamic Power	$167\mu W/MHz^{**}$
1.2V Current @2.5MHz	1.8mA

* Fully constrained clock and loads for worst case (Slow transistors, low resistance, high supply and high temperature). Zero skew Clock Tree Synthesis.

** Measured at 100MHz while running three while loops.

Table 13. Current consumption per clock frequency.

Clock Frequency [MHz]	Total Current [mA] (1.2V and 2.5V supplies)
2.5	2.5
5	4.62
8.8	5.01
10	5.12
20	6.13
40	8.1
60	10.2
80	12.18
100	14.13
120	16.14
140	18.09
160	20.06
180	21.89
200	23.76

Although the GPIO pin count is reduced compared to popular microcontroller families, data converter peripheral performs compared to complete commercial SoCs. Com- paring the dynamic power and maximum clock frequency operation product (Fig. 5.6), the microcontroller out-stands over commercial ARM-M0+ -based microcontrollers, in- dicating the potential of having an available complete open-source platform for IoT applications with similar or better performance than the ones currently available off- the-shelves.

Figure 17. Low-Power Microcontroller performance comparison

	Product	SPI, I2C, SDIO, JTAG	ISA	Dyn. Power [µA/MHz]	Max Speed [MHz]	ADC	DAC	GPIO pin count
This Work	mRISC-V	✓	RISCV	167@100MHz	160	10-bit 10MS/s	12-bit	16
ST	STM32L	no SDIO	ARM M0+	240@4MHz	32	12-bit 1.1MS/s	12-bit	51
Microchip	PIC32MX	no SDIO	MIPS M4K	700@80MHz	80	10-bit 1MS/s	No	64
Atmel	SAMD21	✓	ARM M0+	179@48MHz	48	12-bit 350MS/s	12-bit	52
Silicon Labs	EFM32Z	✓	ARM M0+	115@24MHz	24	12-bit 1MS/s	12-bit	37
NXP / Freescale	LPC812M	✓	ARM M0+	110@30MHz	30	No	No	18
TI	MSP430G	no SDIO	16-bit RISC	506@1MHz	16	10-bit 200Ksps	No	16
Atmel/Microchip	ATMEGA328P	no SDIO	megaAVR 8-bit	700@8MHz	20	10-bit 77KS/s	No	23

4.4 ONLINE PROGRAMMING SERVER

An easy-to-program web interface demo generator with its respective programming server was developed[4]. This web features a dynamic way to create different demo programs in RISC-V assembly code, C native code, Arduino C++ code[5], or Google Blockly functional blocks[6]. The web interface integrates 3 demo levels. The first level demo creates a program that blinks a LED in a desired position at chosen time between ON/OFF transitions. The second level demo runs a user-defined LED sequence with variable length and frame time transition. The final level demo

[4] http://onchip.uis.edu.co
[5] https://github.com/onchipuis/arduinoReady
[6] https://developers.google.com/blockly/

is similar to the second one, but creates two sequences which can be switched using buttons included in the PCB.

For programming the board, the back-end server accepts requests from the front-end site. Basically the user generates some code in the editor[7], then the server receives the text and stores it in a temporal folder, for later compilation. Binary text file is later converted in plain text hexadecimal format and sent back to the user interface. If the user decides to program the compiled program, the same hexadecimal plain text is sent again to the server. This back-end is designed to program by one user at time. Using Google Firebase Library[8], an specific user can link a Google account to the site and try programming the board a maximum of 5 times per day. The server has an scheduled ticket list for all users requesting to program. The duration of any user ticket is 60 seconds, but can be server-configurable. In this time, front-end web is allowed to send hexadecimal programs to the board via Open-V programmer[9]. Interactions between the front-end web and back-end server are possible via POST requests. Server back-end was written in NodeJS, mostly for running system commands easily.

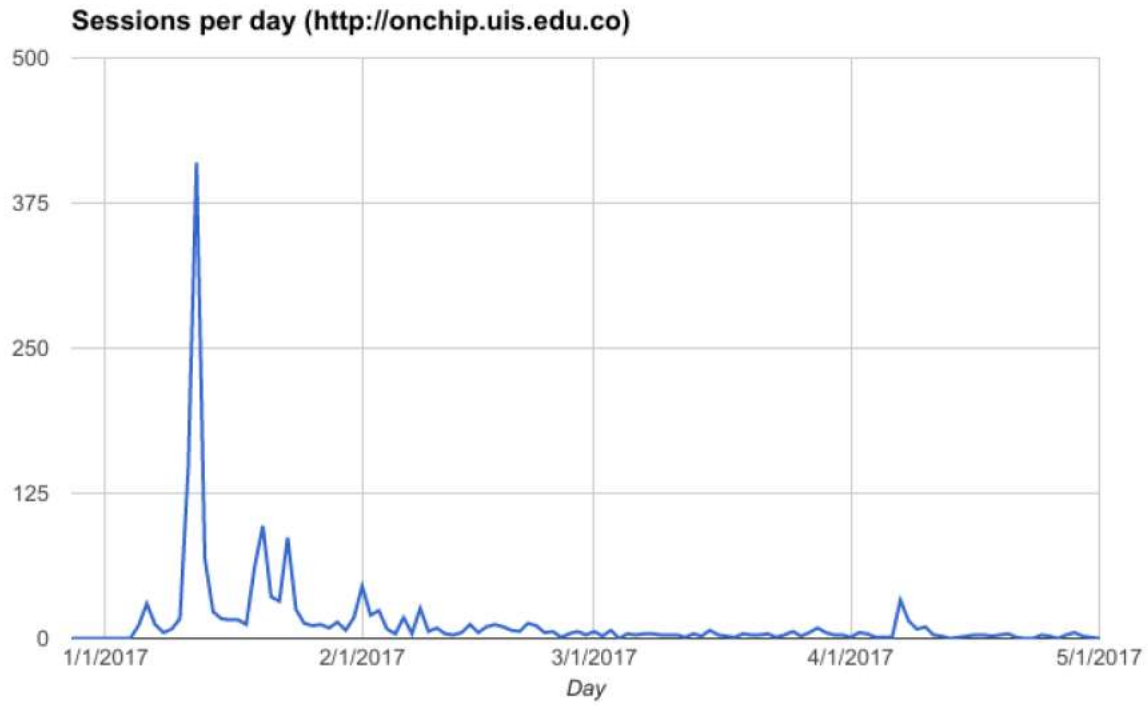
The Online programming server also has a JavaScript RISC-V simulator. It emulates the same implemented instruction set (RV32IM) with debug support and live response after compiling a program. It is only capable of running hexadecimal plain text from the server. The development of this simulator was done by undergraduate students.

This site is intended for any enthusiast to program and control this microcontroller live. The demo was launched in the 5th RISC-V Workshop at Google[10]. Figure 5.7 shows stats of the site since this launchment. The Open-V online programmer

[7] https://codemirror.net/, with highlights for RISC-V assembly
[8] https://firebase.google.com/
[9] https://github.com/onchipuis/mriscvprog
[10] https://youtu.be/XW5SL1JAuYM?t=11m15s

popularized very well in the first days, and once per site feature update (such as adding Blockly programming, Arduino compatibility and Online simulator).

Figure 18. Amount of sessions per day for demo web server (generated using Google Analytics).



5. SUMMARY

A synthesized microncontroller with collaborative analog circuits based on RISC-V architecture on 130nm CMOS technology has been presented. Many verified peripherals are included. The proposed methodology to verify the correct operation is to perform random handshake operations in AXI masters. The proposed architecture shows the interconnection between the RISC-V, the SPI AXI master, and all the peripherals attached to the AXI4-Lite and APB buses, explaining details of the implementation.

The proposed Open-V is the first RISC-V microcontroller designed with enough peripheral to perform common microcontroller tasks. Power and area results show that a reduced RISC-V architecture can be used to replace ARM-M0 based microcontrollers with similar peformance. Considering the advantage of the growing RISC-V community and the existing tool-chain and software around this new instruction set, the Open-V paves the way of future implementations for specific and general applications in the world of IoT, and with open source devices.

A platform for IoT is presented and explained, where the protagonist is a fully-synthesized 32-bit microcontroller in a 130nm CMOS technology. High level modules of the microcontroller architecture are presented including the buses protocols. The maximum frequency tested of the system was 160MHz, spending a total current of 24mA, and demonstrating full functionality. The complete die, including pads, presents an area of 2.1mmx2.1mm.

6. CONTRIBUTIONS

Publishing is an important measure in academia and an imperative way to contribute to the public knowledge. In this chapter, published works related to this microcontroller are shown. Published articles will be listed in section 7.1, presentations in section 7.2, and appearances in influent news media in section 7.3.

6.1 RELATED ARTICLES

C. Duran et al. A 32-bit 100MHz RISC-V Microcontroller with 10-bit SAR ADC in 130nm CMOS (LASCAS 2016) [17]

In this paper a complete implementation and design of a fully-synthesized 32-bit microcontroller in a 130nm CMOS technology was presented. This paper was published as the first microcontroller featuring the open source RISC-V instruction set all mounted through AXI4-Lite and APB buses for communication process. The presented work contains all specifications described in previous sections; and reports for area, layout and power consumption. This paper was presented and published by 2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS).

C. Duran et al. A System-on-Chip Platform for the Internet of Things fea- turing a 32-bit RISC-V based Microcontroller (LASCAS 2017) [18]

A tested and measured 32-bit microcontroller used as an open-source platform for Internet of Things was presented in this paper. This paper was published as the

first silicon-proven 32-bit microcontroller sporting a RISC-V core. A brief description about the implemented system was shown, along with reports of the implemented SoC such as area, real measurements of power consumption and maximum frequency. It is shown that the use of AXI-4 lite bus for communication considers the possible implementation of accelerators for specialized applications. This paper was presented and published by 2017 IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS).

6.2 RELATED PRESENTATIONS

E. Roa A 32-bit 100MHz RISC-V Microcontroller with 10-bit SAR ADC in 130nm GP CMOS (3rd RISC-V Workshop) [19]

The goals of the RISC-V workshops are to bring the community together to share information about recent activity in the various RISC-V projects underway around the globe, and build consensus on the future evolution of the instruction set. In one of those workshops the first RISC-V microcontroller was presented. A brief description of the project, all reports, comparison with other implementations and fabrication process were exposed. Presentation was leaded by Prof. Elkim Roa at 2016 3rd RISC-V Workshop.

E. Roa YoPuzzle: A mRISC-V development platform for next generations (5th RISC-V Workshop) [20]

In this presentation an idea for full PCB implementation and platforming of the mRISC-V (or Open-V) microcontroller (named YoPuzzle) was proposed. YoPuzzle is a way to do dynamic and fun ways to do programming aimed to young people. The purpose of the Puzzle is encouraging young people to use Open platforming to do projects, allowing to understand all functionality from the minimum detail. Also,

the fabricated silicon and test PCB was presented, along with a live demo featuring different kind of tests with LEDs. Presentation was leaded by Prof. Elkim Roa and Ckristian Duran (demo only) at 2016 5th RISC-V Workshop.

E. Roa RISC-V Community needs Peripheral Cores (5th RISC-V Workshop) [21]

This presentation shows ideas of solution about the disadvantages about Intellectual Property (IP) peripheral silicon implementation inside RISC-V platforms. It was exposed that Open-Hardware is the main way to avoid different non-standardized prob- lems. These problems are such as quality Open Linux driver coding, solving bottlenecks about information throughput, and standardize analog devices such as GPIO. As exam- ples of developing work-in-process circuitry were presented the Open-V microcontroller, Analog-to-Digital converter, Clock Data Recovery (CDR), Phase-Locked Loop oscilla- tor (PLL), USB 3.1 Gen 2 and its different analog modules, Pseudo-Random Bit Shift (PRBS) generation and checking, lpDDR3 interface controller using VIP and UVM to strong verification, Trusted Platform Module (TPM) as key handshaker, True-Random Noise Generator (TRNG), and Non-Volatile RAM (NVRAM). Presentation was leaded by Prof. Elkim Roa at 2016 5rd RISC-V Workshop.

6.3 APPEARANCE IN NEWS

A university group in Colombia developing a RISC-V-based microcontroller is a sign of the country’s intention to make its mark in technology (EETIMES) [22]

EE Times connects the global electronics community through news, analysis, education, and peer-to-peer discussion around technology, business, products and design. In this magazine an article talking about this microcontroller and its features was pub- lished.

The Open-V, World’s First RISC-V-based Open Source Microcontroller (MAKE:) [23]

”MAKE:” is an American bimonthly magazine which focuses on do it yourself (DIY) and/or DIWO (Do It With Others) projects involving electronics. Make magazine is considered a central organ of the maker movement. In this magazine an article about the Open-V microcontroller and future features based on this work was published.

7. FUTURE WORK

We are currently extending the current architecture to achieve low power consumption and good processing and communication performance, and preliminary results are encouraging. Based on this work, improvements will be implemented for achieving a commercial-competent microcontroller. In this chapter, works in development and future planned works are presented.

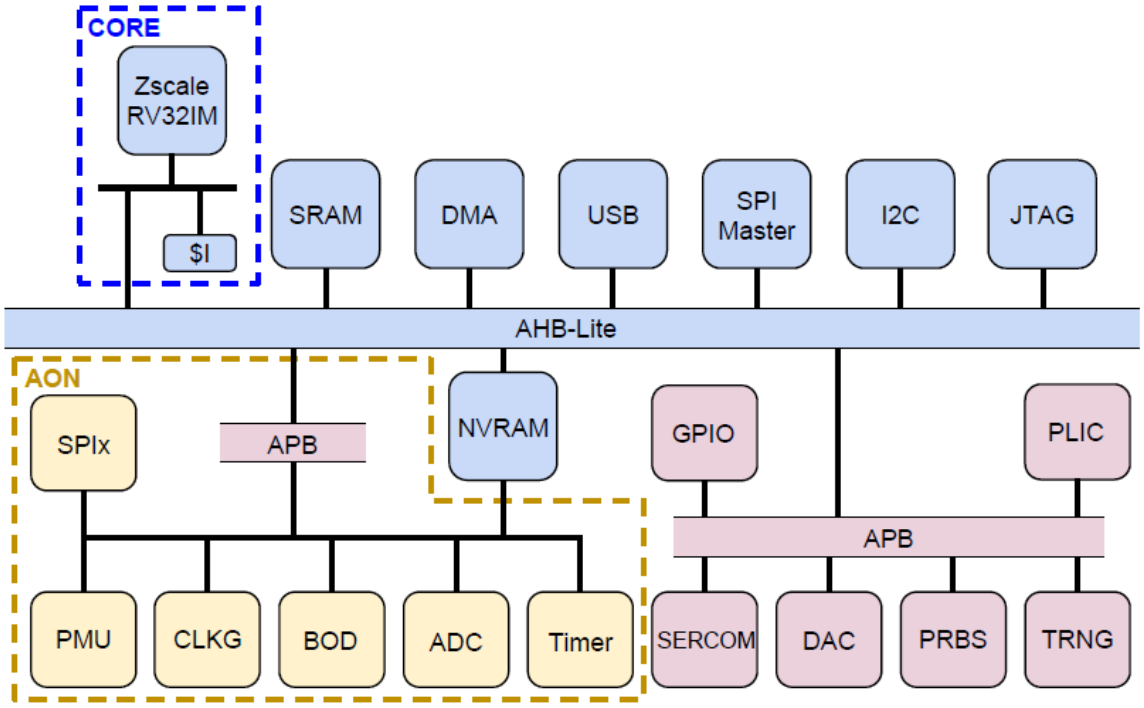
7.1 OPEN-V V.2

A second version of Open-V is being developed. In asociation with SiFive Inc.[11], this mi- crocontroller will improve features compared to the presented in this research. Some of the new features will be: AHB-Lite & AHB Buses, RISC-V core with RV32IM instruc- tions and implementation of privileged instruction set (RV32S) [11], separate scratch pad memory and SRAM for instructions and data, Direct Memory Access (DMA) con- troller, low-end USB controller, additional I2C Master (in contrast with SPI Master) for send requests to all slaves attached to AHB-Lite crossbar, JTAG debugger capable of control the RISC-V core and do memory operations according to RISC-V External Debug Support (draft) [24], Platform-Level Interrupt Controller (PLIC) in charge of bring hardware interrupts information, configuration and triggering, NVRAM storing a bootloader and permanent configurations for analog circuitry inside the microcon- troller, Always-ON (AON) module in charge to control chip powering startup, clock-/timer manipulation and Analog-to-Digital (ADC) conversion, Serial Communications (SERCOM) controller capable of send/receive through SPI, UART and I2C protocols, General-Purpose Input Output (GPIO) multiplexer controller with Pulse

[11] https://www.sifive.com/

Width Mod- ule (PWM) generator, Pseudo-Random Bit Shifter (PRBS) and True Random Noise Generator (TRNG, based in work [25]) providing randomized numbers, and a Digital- to-Analog Converter (DAC). A brief proposal of the architecture for this microcontroller is shown in Figure 8.1.

Figure 19. Open-V v.2 proposed architecture.



7.2 LEVERAGING A RISC-V ISA IN A LOW-POWER 32- BIT MICROCONTROLLER

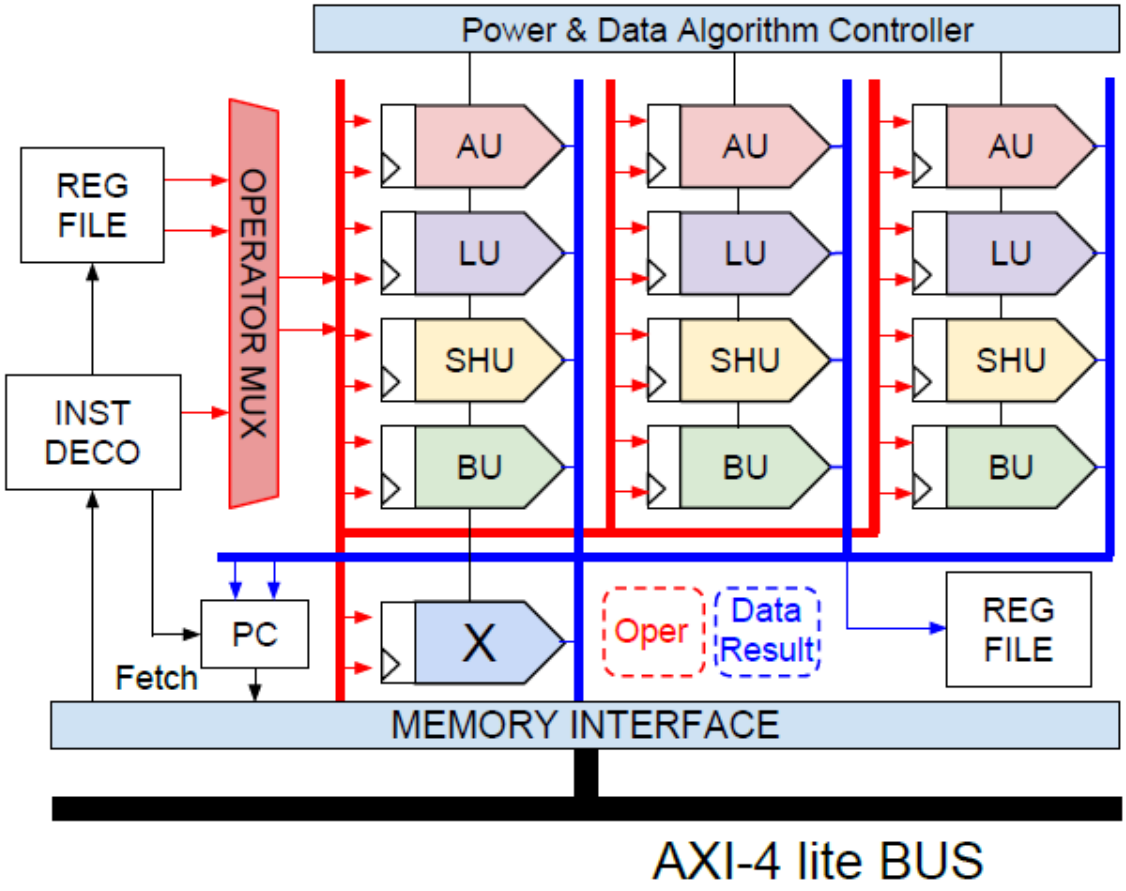
The energy optimization in high-level synthesis for low-power embedded processing applications of the RISC-V ISA for integer and multiplication extensions is still far away when compared with energy consumption of current embedded low-power pro- cessors. We propose a scheme to reduce excessive power by allocating special paths of constrained instructions between high-speed

customized cells and low power cells. In addition, we propose a programmable arbitration logic for the AXI-4 and APB buses by configuring their latencies. These approaches make it possible to tolerate latency penal- ties incurred during transition between low power and high performance paths. The proposed schemes were implemented in the first RISC-V based open microcontroller implemented in 130nm CMOS technology (described previously in this work).

7.2.1 Proposed Architecture Related works such as [26] and [27] present functional modules which power is measured to control their activation an deactivation. A microarchitecture control constructs exe- cution pipelines from a distributed poll of execution resources, which activation can be controlled for managing energy consumption. Moreover, ReDEEM uses a scheduling system where every module remains active until the power manager decides to deac- tivate it due to inactivity in a period of time, always executing 16 instructions [26]. Viper instead executes a group of instructions with certain limit criteria, activating the necessary modules to execute all the instruction group at once [27].

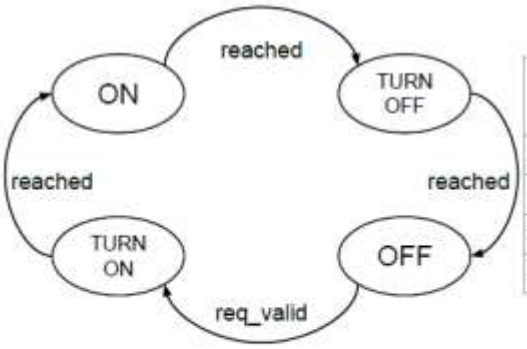
The proposed microcontroller based on RISC-V ISA addressed the optimization by implementing a modularization of the ALU considering the four groups of inte- ger operations: Arithmetic, Bitwise Logic, Shift Logic and Branches [5]. A dynamic modularization of the ALU is implemented for achieving optimal execution and power management for a limited subsequent instructions. In this paper, those modules are named as such: AU (Arithmetic Unit), LU (Logic Unit), SHU (Shift Unit) and BU (Branch Unit). Dividing these operations is an advantage because there is no need to implement any additional decode process because RISC-V codification process can be resumed using simple muxers to any operation. Figure 8.2 shows the accurate modu- larization of the previous microcontroller architecture.

Figure 20. Proposed Power-Optimized Processor Architecture.



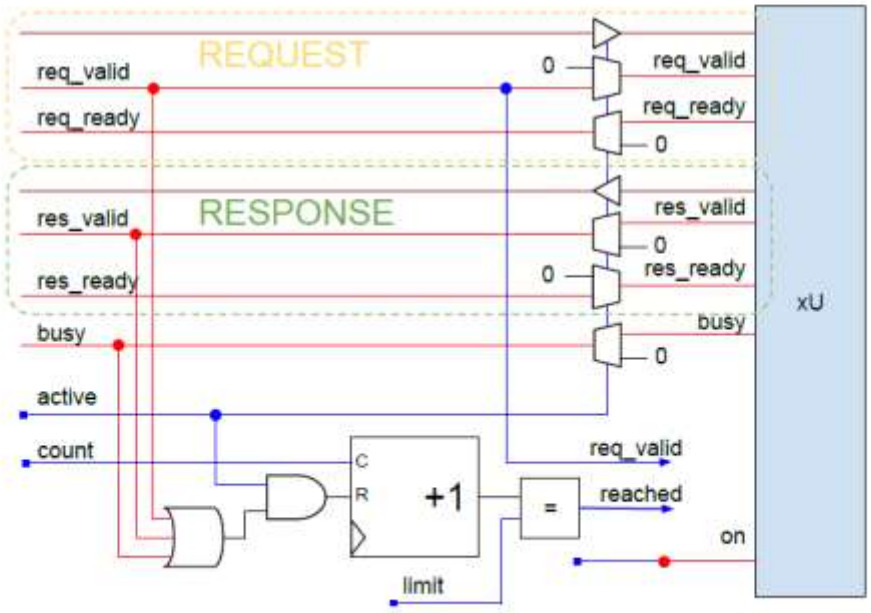
The power management includes different units controlling the power on/off of each module. Figure 8.3 shows this Simple unit power manager. Intended operation is to turn on the device when a request is issued, then turn it off when certain elapsed time has passed if the module remains inactive. Using this power manager is transparent to the protocol. The only remaining part is to manage tasks in the architecture.

Figure 21. Single unit power manager. a) State machine. b) Datapath.



State	Outputs			
	active	on	count	limit
ON (default)	1	1	1	timeinc
TURN_OFF	0	0	1	timeoff
OFF	0	0	0	x
TURN_ON	0	1	1	timeon

(a)

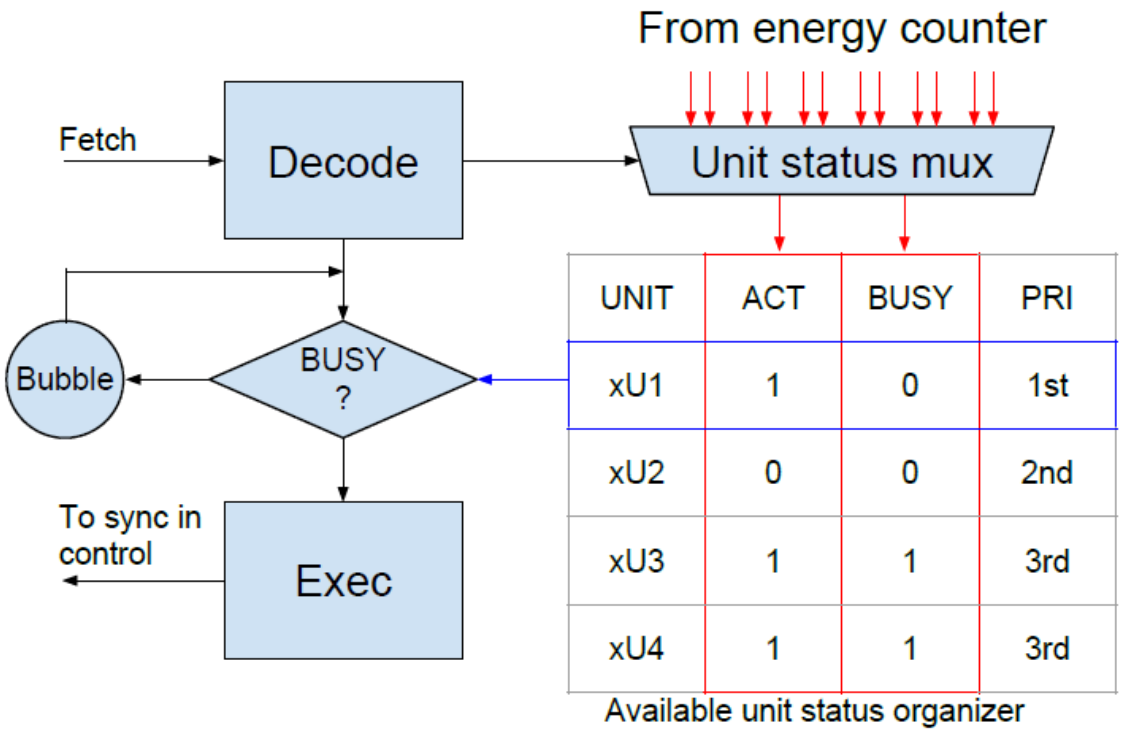


(b)

An implemented Simple task management can be appreciated in Figure 8.4. Accord- ing to current state of the power-managed Units about the classified decoded instruc- tion, a priority is given for each one, and stacked to a sorted list. The first priority is given to the active units without being busy, second priority for the inactive units, then the last priority for the active & busy units. Choosing the highest order in the priority list, the busy state is again evaluated (this is the case when all units are busy). If the selected unit is not busy, the data is committed, then

the respective power management unit is in charge of doing the respective task transaction for the intended instruction.

Figure 22. Flowchart of implemented task assignment.

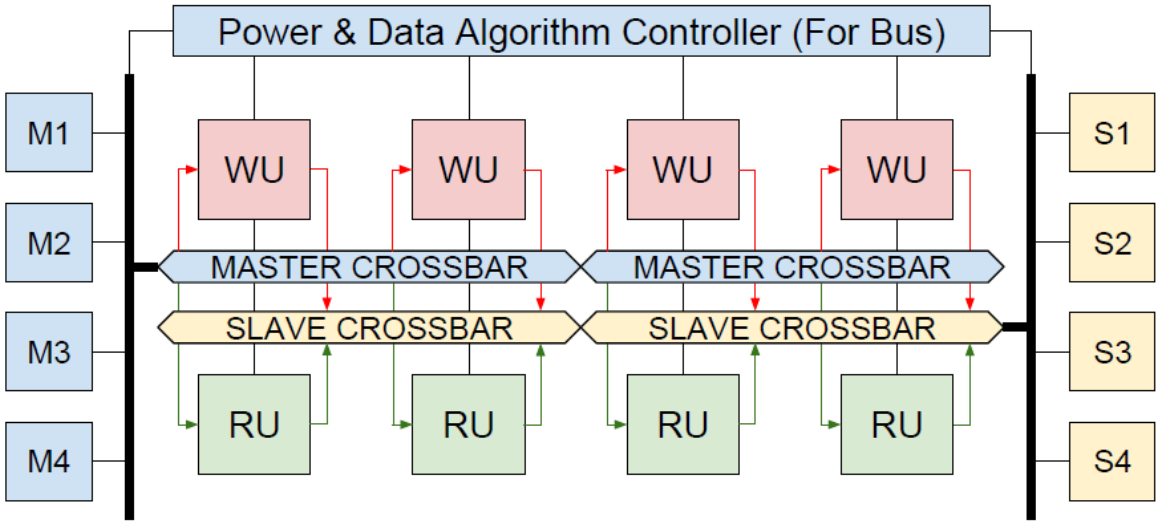


7.2.2 Bus Architecture The power reduction in buses operation is focused in Read and Write operations using the same concept as in the processor for the power management of the modules. For each master in the AXI4-Lite specification certain communication module can be assigned for each task (WU for write unit, and RU for read unit) [13]. A bus crossbar allows the intercommunication between all the slaves and masters with all implemented read/write units. Figure 8.5 shows the proposed bus architecture. The design must ensure that the number of modules connected to the crossbar is less than the number of masters.

The control arbiter is in charge of selecting an available communication unit with its respective master and slave. For either transaction, the communication unit is

activated (or remains active) and an internal transaction control inside the unit ensures communication is complete for the intended requested task. At the end of the task, the communication unit deactivates itself if power-off time has elapsed. Dynamics about this architecture are expected to remove operation into not active persistent operations in a single master.

Figure 23. Proposed Power-Optimized Bus Architecture



BIBLIOGRAPHIC REFERENCES

[1] C. H. S´equin and D. A. Patterson, “Design and implementation of RISC I,” University of California at Berkeley, Tech. Rep. CSD-82-106, July 1982. [Online]. Available: http://digitalassets.lib.berkeley.edu/techreports/ucb/text/ CSD-82-106.pdf

[2] D. Bhandarkar and D. W. Clark, “erformance from architecture: Comparing a RISC and a CISC with similar hardware organization,” September 1991, pp. 310–319.

[3] T. Chen and D. Patterson, “RISC-V geneology,” University of California at Berkeley, Tech. Rep. UCB/EECS-2016-6, Jan 2016. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-6.html

[4] “BSD open source license agreement,” University of California, Berkeley. [Online]. Available: https://opensource.org/licenses/BSD-3-Clause

[5] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi´c, “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

[6] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanovic, and K. Asanovic, “A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators,” in European Solid State Circuits Conference (ESSCIRC), ESSCIRC 2014 - 40th, Sept 2014, pp. 199–202.

[7] A. Karatsuba and Y. Ofman, “Multiplication of Many-Digital Numbers by Automatic Computers,” Proceedings of the USSR Academy of Sciences, p. 293–294, Aug 1962.

[8] A. A. Karatsuba, “The Complexity of Computations,” Proceedings of the Steklov Institute of Mathematics, pp. 169–183, Jan 1995. [Online]. Available: http://www.ccas.ru/personal/karatsuba/divcen.pdf

[9] K. D.E., “The Art of Computer Programming. v.2,” Addison-Wesley Publ.Co., p. 724, Jan 1969.

[10] A. D. Booth, “A Signed Binary Multiplication Technique,” Quarterly Journal of Mechanics and Applied Mathematics 4, Aug 1950. [Online]. Available: http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee241s00/PAPERS/archive/booth51.pdf

[11] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanovi´c, “The risc-v instruction set manual volume ii: Privileged architecture version 1.7,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-49, May 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-49.html

[12] S. Pradeep and C. Laxmi, “Design and verification environment for AMBA AXI protocol for SOC integration,” NCRIET-2014, vol. 3, no. 3, pp. 338–343, may 2014.

[13] ARM, “AMBA AXI and ACE protocol specification,” 2011, pp. 1–121. [14] ——, “AMBA APB protocol version: 2.0,” 2010, pp. 1–27.

[15] J. Romero, R. Torres, and E. Roa, “An AXI4-Lite to APB Bridge based on Advanced Microcontroller Bus Architecture,” Integrated Systems Research Group Onchip, Universidad Industrial de Santander, Bucaramanga, Colombia, August 2016.

[16] G. Castillo, A. Agudelo, and E. Roa, “An 8-bit General Purpose IO for a 32-bit mi- crocontroller,” Integrated Systems Research Group Onchip, Universidad Industrial de Santander, Bucaramanga, Colombia, August 2016.

[17] 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, and 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), Feb 2016, pp. 315–318.

[18] C. Duran, L. Rueda, J. Ardila, D. L. Rueda, G. Castillo, A. Agudelo, C. Ro- jas, L. Chaparro, H. Hurtado, J. Romero, W. Ramirez, H. Gomez, H. Hernandez, J. Amaya, and 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), Feb 2017, pp. 298–301.

[19] E. Roa, “A 32-bit 100MHz RISC-V Microcontroller with 10-bit SAR ADC in 130nm GP CMOS,” in 3rd RISC-V Workshop, Jan 2016, pp. 1–11. [Online]. Available: https://riscv.org/wp-content/uploads/2016/01/Wed0945-mriscv.pdf

[20] E. Roa, “YoPuzzle: A mRISC-V development platform for next generations,” in 5th RISC-V Workshop, Nov 2016, pp. 1– 34. [Online]. Available: https://riscv.org/wp-content/uploads/2016/12/Wed1430-Yopuzzle-Roa-Universidad-Industrial-de-Santander.pdf

[21] E. Roa, “RISC-V Community needs Peripheral Cores,” In 5th RISC-V Workshop, Nov 2016, pp. 1– 24. [Online]. Available: https://riscv.org/wp-content/uploads/2016/12/ Wed1445-RISC-V-Peripherals-Roa-Universidad-Industrial-de-Santander.pdf

[22] A. Sampayo, “A university group in Colombia developing a RISC-V-based microcontroller is a sign of the country’s intention to make its mark in technology,” EETimes, Sep 2016. [Online]. Available: http: //www.eetimes.com/author.asp?section id=36&doc id=1330445

[23] G. Branwyn, “The Open-V, World’s First RISC-V-based Open Source Microcontroller,” MAKE:, Nov 2016. [Online]. Available: http://makezine.com/ 2016/11/30/open-v-worlds-first-risc-v-based-open-source-microcontroller/

[24] T. Newsome, “RISC-V External Debug Support,” SiFive Inc., Apr 2017, This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard. [Online]. Available: https://dev.sifive.com/documentation/ risc-v-external-debug-support/

[25] J. Cartagena, H. Gomez, and E. Roa, “A fully-synthesized TRNG with lightweight cellular-automata based post-processing stage in 130nm CMOS,” in 2016 IEEE Nordic Circuits and Systems Conference (NORCAS), Nov 2016, pp. 1–5.

[26] B. Mammo, R. Parikh, and V. Bertacco, “ReDEEM: A heterogeneous distributed microarchitecture for energy-efficient reliability,” in 2015 IEEE/ACM Interna- tional Symposium on Low Power Electronics and Design (ISLPED), July 2015, pp. 297–302.

[27] A. Pellegrini, J. L. Greathouse, and V. Bertacco, “Viper: Virtual pipelines for enhanced reliability,” in 2012 39th Annual International Symposium on Computer Architecture (ISCA), June 2012, pp. 344–355.

BIBLIOGRAPHY

ARM, “AMBA AXI and ACE protocol specification,” 2011, pp. 1–121

AMBA APB protocol version: 2.0,” 2010, pp. 1–27.

BHANDARKAR D. and CLARK D. W., “erformance from architecture: Comparing a RISC and a CISC with similar hardware organization,” September 1991, pp. 310–319.

BOOTH A. D., “A Signed Binary Multiplication Technique,” Quarterly Journal of Mechanics and Applied Mathematics 4, Aug 1950. [Online]. Available: http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee241s00/PAPERS/archive/booth51.pdf

BRANWYN G., “The Open-V, World’s First RISC-V-based Open Source Microcontroller,” MAKE:, Nov 2016. [Online]. Available: http://makezine.com/2016/11/30/open-v-worlds-first-risc-v-based-open-source-microcontroller/

CARTAGENA J., GOMEZ H., and ROA E., “A fully-synthesized TRNG with lightweight cellular-automata based post-processing stage in 130nm CMOS,” in 2016 IEEE Nordic Circuits and Systems Conference (NORCAS), Nov 2016, pp. 1–5.

CASTILLO G., AGUDELO A., and ROA E., “An 8-bit General Purpose IO for a 32-bit mi- crocontroller,” Integrated Systems Research Group Onchip, Universidad Industrial de Santander, Bucaramanga, Colombia, August 2016.

CHEN T. and PATTERSON D. , “RISC-V geneology,” University of California at Berkeley, Tech. Rep. UCB/EECS-2016-6, Jan 2016. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-6.html

DURAN C., RUEDA D. L., CASTILLO G., AGUDELO A., ROJAS C., CHAPARRO L., HURTADO H., ROMERO J., RAMIREZ W., GOMEZ H., ARDILA J., RUEDA L., HERNANDEZ H., AMAYA J., and ROA E., “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), Feb 2016, pp. 315–318.

DURAN C., RUEDA L., ARDILA J., RUEDA D. L., CASTILLO G., AGUDELO A., ROJAS C., CHAPARRO L., HURTADO H., ROMERO J., RAMIREZ W., GOMEZ H., HERNANDEZ H., AMAYA J., and ROA E., “A System-on-Chip Platform for the Internet of Things fea- turing a 32-bit RISC-V based Microcontroller,” in 2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS), Feb 2017, pp. 298–301.

K. D.E., “The Art of Computer Programming. v.2,” Addison-Wesley Publ.Co., p. 724, Jan 1969.

KARATSUBA A. A., “The Complexity of Computations,” Proceedings of the Steklov Institute of Mathematics, pp. 169–183, Jan 1995. [Online]. Available: http://www.ccas.ru/personal/karatsuba/divcen.pdf

KARATSUBA A. and OFMAN Y., “Multiplication of Many-Digital Numbers by Au- tomatic Computers,” Proceedings of the USSR Academy of Sciences, p. 293–294, Aug 1962.

LEE Y. , WATERMAN A. , AVIZIENIS R. , COOK H. , SUN C. , STOJANOVIC V. , and ASANOVIC K., “A 45nm 1.3GHz 16.7 double-precision

GFLOPS/W RISC-V pro- cessor with vector accelerators," in European Solid State Circuits Conference (ES- SCIRC), ESSCIRC 2014 - 40th, Sept 2014, pp. 199–202.

MAMMO B., PARIKH R., and BERTACCO V., "ReDEEM: A heterogeneous distributed microarchitecture for energy-efficient reliability," in 2015 IEEE/ACM Interna- tional Symposium on Low Power Electronics and Design (ISLPED), July 2015, pp. 297–302.

NEWSOME T., "RISC-V External Debug Support," SiFive Inc., Apr 2017, This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard. [Online]. Available: https://dev.sifive.com/documentation/ risc-v-external-debug-support/

PELLEGRINI A., GREATHOUSE J. L., and BERTACCO V., "Viper: Virtual pipelines for enhanced reliability," in 2012 39th Annual International Symposium on Computer Architecture (ISCA), June 2012, pp. 344–355.

PRADEEP S. and LAXMI C., "Design and verification environment for AMBA AXI protocol for SOC integration," NCRIET-2014, vol. 3, no. 3, pp. 338–343, may 2014.

ROA E., "A 32-bit 100MHz RISC-V Microcontroller with 10-bit SAR ADC in 130nm GP CMOS," in 3rd RISC-V Workshop, Jan 2016, pp. 1–11. [Online]. Available: https://riscv.org/wp-content/uploads/2016/01/Wed0945-mriscv.pdf

ROA E., "RISC-V Community needs Peripheral Cores," In 5th RISC-V Workshop, Nov 2016, pp. 1– 24. [Online]. Available: https://riscv.org/wp-content/uploads/2016/12/ Wed1445-RISC-V-Peripherals-Roa-Universidad-Industrial-de-Santander.pdf

ROA E., “YoPuzzle: A mRISC-V development platform for next generations,” in 5th RISC-V Workshop, Nov 2016, pp. 1– 34. [Online]. Available: https://riscv.org/wp-content/uploads/2016/12/Wed1430-Yopuzzle-Roa-Universidad-Industrial-de-Santander.pdf

ROMERO J., TORRES R., and ROA E., “An AXI4-Lite to APB Bridge based on Advanced Microcontroller Bus Architecture,” Integrated Systems Research Group Onchip, Universidad Industrial de Santander, Bucaramanga, Colombia, August 2016.

SAMPAYO A. , “A university group in Colombia developing a RISC-V- based microcontroller is a sign of the country’s intention to make its mark in technology,” EETimes, Sep 2016. [Online]. Available: http: //www.eetimes.com/author.asp?section id=36&doc id=1330445

SEQUIN C. H. and PATTERSON D. A. , “Design and implementation of RISC I,” University of California at Berkeley, Tech. Rep. CSD-82-106, July 1982. [Online]. Available: http://digitalassets.lib.berkeley.edu/techreports/ucb/text/ CSD-82-106.pdf

UNIVERSITY OF CALIFORNIA, BERKELEY “BSD open source license agreement,”. [Online]. Available: https://opensource.org/licenses/BSD-3-Clause

WATERMAN A., LEE Y., AVIZIENIS R., PATTERSON D. A., and ASANOVIC K., “The risc-v instruction set manual volume ii: Privileged architecture version 1.7,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-49, May 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/ 2015/EECS-2015-49.html

WATERMAN A., LEE Y., PATTERSON D. A., and ASANOVIC K., “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