A Survey of RISC-V CPU for IoT Applications

Dr. Manoj Sharma[ab*], Ekansh Bhatnagar[ac], Kartik Puri[ad], Amitav Mitra[ae], Jatin Agarwal[af]

[a] *Bharati Vidyapeeth's College of Engineering, Paschim Vihar, Delhi-110063, India*
[b] *manojs110281@gmail.com*
[c] *ekanshbhatnagar@outlook.com*
[d] *kartikpuri99@gmail.com*
[e] *amitavmitra28@gmail.com*
[f] *aggarwaljatin16@gmail.com*

Abstract: With rise in innovation, comes rise in cost of production and design. A major piece of this cost comes from costly licenses for locked Instruction Set Architecture. Thus, rise of RISC-V as a viable and robust ISA is crucial for low-cost applications. RISC-V is free and open-source, thus enabling CPU designs which are free of any proprietary IP. As, the demand of IoT capable hardware is increasing. Developing RISC-V based boards with capabilities tailored towards IoT application is becoming much more feasible. Though, there is extensive work done in the development of open-source CPU cores, there is lack of focus on IoT specific features. Most openly available cores either lack proper documentation or are targeted towards specialised use. This survey paper studies and summarises findings about various open-source CPUs and SOCs built with RISC-V architecture and their applications in IoT domain. It also proposes a RISC-V SoC architecture made for IoT applications. It discusses the features needed for an ideal open-source IoT node. The need for research and development in supporting specific protocols is also discussed as the future scope.

1. Introduction

IoT is bringing in a significant shift in the way people work, live, or entertain themselves. The number of IoT devices are increasing exponentially day by day in consumer as well as industrial sectors and are expected to reach worldwide projected amount of about 30.9 billion units by 2025. The data generated by these IoT devices would be enormous and hence would require a processor to efficiently process the data. Current options for developing CPUs include ARM, x86 and RISC-V, however ARM and x86 ISAs are not in public domain and require a huge licensing fee and are highly complex. This makes it hard for small organisations and individuals to develop a custom chip solely focused on IOT application (Patterson, D., and Hennessy, J. 2017).

Considering the open standard nature of the RISC-V ISA, it makes an alluring option for developing custom IoT focused chips without the barriers of entry present in ARM and x86 ISA's. RISC-V is an instruction set architecture (ISA) based on reduced instruction set principles and was started as a university project at the university of California, Berkeley in 2010. It is currently maintained by SiFive inc. that handles all repositories associated with the RISC-V ISA and its development is sustained by its open-source nature. Like x86, it supports both 32-bit and 64-bit and even 128-bit address variants however due to its impracticality in the current CPU addressing needs, the 128-bit addressing variant is not used in the mainstream development of RISC-V processors. RISC-V ISA has a frozen basic instruction set known as RV32I (For 32-bit processors). With already available specifications we can add specific instructions like Compressed (C), Floating point (P), double precision floating point (D), vector calculations (V) and many more for different applications based on requirements and design specifications. Because of its extensions-based ISA, developers with limited knowledge of computer architecture do not have to study about the RISC-V's internal architecture and focus on the real objective of the project (Gautschi, M., Schiavone, P., 2017).

This paper studies the available implementations and proposes a RISC-V CPU core which would be ideal as an Internet of Things node. Its features include:

- Simple and scalable RISC-V core
- 32 GPIO pins with 4 state support.
- RV32IM Instruction set
- Support for compilation and execution of C code.

Section 2 describes existing open-source RISC-V based CPUs available. Their applications in IoT domain, their capabilities, and their drawbacks are discussed in Section 3. Section 4 then proposes a RISC-V based SoC architecture for IoT applications. Section 5 discusses the societal implications of a freely available RISC-V core for IoT. Section 6 concludes the discussions and discusses future scope of work in the field.

2. Existing Work

2.1. PicoRV

PicoRV is a CPU implemented via the RISC-V ISA that combines the RV32I (base integer), RV32M (multiplication) and RV32C (compressed) extensions (RV32IMC) and can also be configured to work with RV32E, RV32IM, RV32IC and RV32IMC combinations. It is meant to be a co-processor to a FPGA or ASIC but can also be used as a standalone CPU core. The core comes with the following bus configurations:

- picorv32 with a simple bus multiplexer interface.
- picorv32 axi with a AXI4-lite interface
- picorv32 wb with a wishbone master interface
- picorv32 pcpi_mul that implements multiplication instructions and supports addition of custom instructions

The base PicoRV32 CPU has the following specifications:

- Support for RV32IMC instruction sets
- 32KB RAM.
- Parametric design for enabling or disabling select CPU features.
- A selection of memory interfaces (mux, AXI4).

The CPU has been implemented on several FPGAs (Xilinx Kintex and Virtex series) and uses a maximum of 2019 LUTs for the CPU and 88 LUTs for the memory and it can run at clock speeds of up to 769 MHz This makes PicoRV32 highly customizable and favourable for implementing custom CPUs and it is even the base for a major part of

RISC-V based SOCs. The official repository even contains a full-fledged SOC containing a GPIO controller, SPI flash controller, SRAM, and UART. The SOC even supports the addition of custom blocks such as extra communication protocol controllers or memory extensions. This CPU has been extensively used in various design over the years. It has been tested in various commercial designs such as RavenSoC and PicoRio (Miura, J., Miyazaki, H., 2020), thus proving its robustness. Though many alternate designs exist like Miura et al. RISCSoC (Ahmadi-Pour, S., Herdt, V., and Drechsler, R. 2021) most of the popular implementations use Clifford Wolf's PicoRV, due to its accessibility (Tan, Z., Zhang, L., Patterson, D., and Li, Y. 2020).

2.2. MicroRV

Primarily intended for educational and research purposes, MicroRV32 also known as RV32 is an open-source RISC-V platform which integrates 32-bit RISC-V core to several peripherals via a generic bus system. It is implemented in modern SpinalHDL language and has capability to run the FREERTOS operating system. The top-level architecture includes the RV32I core, which is interfaced with peripherals like Memory, GPIO, UART via a simple bus interface.

The core interacts with the other peripherals through an interface defined by an address, a command, and data. A valid ready handshake bus interface is chosen by them in which the bus master (the CPU core) interacts with peripherals by asserting a valid signal to notify the bus slaves (the other peripherals) that there is a payload on the bus. On the top level the peripherals are addressed mapped and hence depending on the memory address the transaction packet is then routed to their respective peripheral. The peripherals then respond back to the transaction request after one clock cycle. If the peripheral doesn't respond back within one clock cycle the CPU would be stalled.

For testing the proposed architecture execution time for different CPP programs like Fibonacci, greatest common divisor (GCD) etc in an RTL simulator is observed. Along with that the final SOC is then tested on Lattice Semiconductor HX8K Development Board, and it seen to be operated at a maximum clock frequency fmax of 28.61 MHz with a device utilization of 55%. The use of 78% of BRAM Cells on the FPGA will vary with the program used to initialize in the memory (Waterman, A., Lee, Y., 2011). CPUs like MicroRV could have various uses, as they get tested in the real world. The low complexity of MicroRV enables it to be easier for demonstration and learning, it has extensive educational uses.

2.3. RavenSoC

The RavenSoC works upon the already available and popular PicoRV RISC-V core, developed by Clifford Wolf. The core though provides a basic SoC for testing purposes, it is not sufficient for any professional work. The PicoRV CPU core used in the design of RavenSoC was also proven to work with various popular FPGA implementations. Raven provides a first open-source complete SoC implementation for the PicoRV Core. The SoC has sufficient on-chip 32x1024 SRAM for simple applications. It is tested for 100 MHz clock rate, which doesn't seem much but the light overhead due to the highly optimized ISA, enables it to work much more efficiently.

As shown in figure 1, Raven SoC also provides a plethora of I/O capabilities. It has a very useful GPIO bank (16 pins) which can provide as interface for many Arduino/Raspberry Pi modules. The input/output functions and interrupts are programmable on the GPIO pins individually. For Program memory it uses a special SPI interface, which can be used to load up programs for the Core, within raven spi it also has spi ₋memory which can be used as intermediate storage or temporary storage for the program instructions. It also has support for 2 Analog-to-digital convertor and a Digital-to-analog convertor. The raven soc module instantiated inside top module for the board, has a UART controller built in which allows modules to directly interface with the SoC using a standard UART interface. As a special feature it also supports an over-temperature alarm which is not found on many open-source implementations studied by us. It protects the SoC from overheating and provides it extra robustness. Overall, this implementation of RISC-V board is the closest in term of IoT friendliness, as it provides many of the features that are crucial for such work. This can be used as general-purpose reference for many similar designs as the code is robust and easily extendable (Sadeeq, M., Zeebaree, S., 2018).

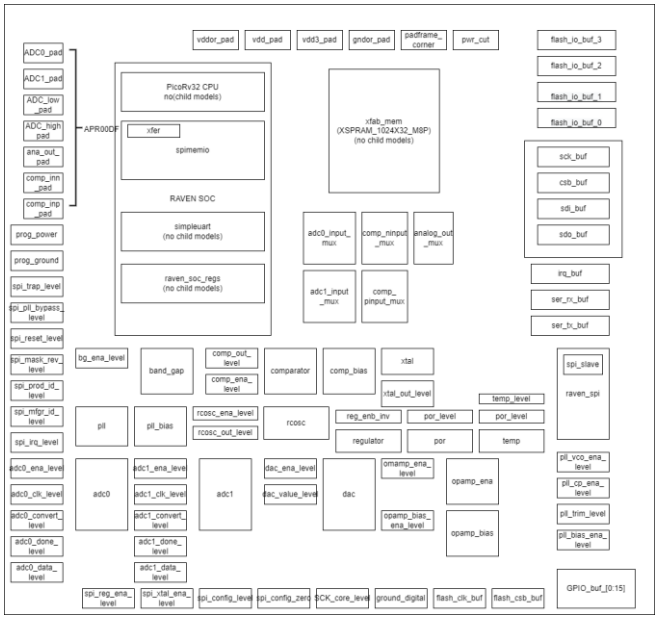

Figure 1: Raven SoC Block Diagram

2.4. Ultraembedded RISC-V

This implementation features a 32-bit RISC-V CPU core which is available in the open-source communities under the Apache Licensing scheme. It implements Integer(I), multiplication and division (M) and CSR extensions collectively written as RV32IMZicsr. The architecture of the CPU Core consists of a five-stage pipeline mechanism which can also be configured according to design specifications as follows

- Instruction Fetch: This stage reads the next instruction using the address returned by Program Counter and feeds it to the next stage.
- Instruction Decode: This stage decodes the incoming instruction and updates the necessary signals for the upcoming stages.

- Execute: This stage executes the instruction and produces a result from the ALU unit inside it.
- Memory: This stage is used for memory read/write operations.
- Writeback: This stage writes the results into the register file of the CPU core.

The core also supports instructions/data cache, bus interfaces like AXI and TCMs (Tightly Coupled Memory). The core is designed to be able to boot a basic open-source operating system (stock Linux 5.0.0-rc8 in this case) efficiently.

3. Discussion

As discussed in the above sections, each implementation is unique and optimized for different objectives. When studied for the purpose of IoT applications, following is the comparison of features of each implementation.

Table 1: Comparison of the studied SoCs

Implementation	PicoRV	MicroRV	RavenSOC	Ultraembedded RISCV
Memory	32KB	32KB	32KB	64KB
Bus Interface	MUX, AXI, Wishbone	MUX	AXI	AXI
Instruction Set	RV32IMC	RV32I	RV32IMC	RV32IMicsr
Pipelining	None	None	None	6-7 Stage
Peripheral	SPI, GPIO, UART	GPIO, UART	SPI, GPIO	None
FPGA Implementation	Xilinx Kintex Series, Xilinx Artix Series	Lattice HX8K	ASIC	Xilinx Artix 7
Operating Frequency	714 MHz	28.61 MHz	100 MHz	>50 MHz

Table 2: Achievements and Drawback of the studied SoCs

Implementations	Achievements	Drawbacks
PicoRV32	Can run C code, Support for multiple bus configurations such as AXI4 and MUX	Insufficient Documentation
MicroRV	Supports bare metal applications and FreeRTOS	No SPI flash memory, No proper documentation
RavenSOC	Fully implemented ASIC, includes ADCs, DACs, and temperature alarm	Insufficient Documentation
Ultraembedded RISCV	Can boot Linux, Has cache and TCM options	No peripherals for interfacing

For a hardware design to be IoT friendly, it must have few of these features:

- Low-power usage
- Small Size
- Support for High-Level Languages (C/C++)
- Interface with standard sensors
- Support for simple network interfaces

Many ARM implementations already exist for such IoT application, but as discussed previously these implementations require costly licensing fees to modify and develop. But increasing support for RISC-V architecture, which the completion for GNU/Linux libraries for RISC-V assisted, has allowed design of complete RISC-V systems for specific IoT needs.

PicoRV (Section 2.1) though provides a basic design for a general-purpose low power CPU, it lacks interfacing for sensors. Also, its support of only RV32IMC extensions could also be improved for specific tasks such as cryptography support, which is sometimes essential for IoT uses (Efabless, 2021). Also, many IoT devices are nowadays used to pre-process the data from the sensors. Thus, adding support for DSP extensions can also be useful, as seen in Gautschi et al. (Ultraembedded,2020).

An ideal IoT centric hardware must balance the size and features of the implementation. Also, a major concern is the power usage as well. While studying the implementations in Section 2, these parameters were studied.

Though most of the implementations are very robust for general purpose use, many lack these IoT specific features. Also, the design methodology was selected with keeping in mind the needs of a general-purpose CPU. This means added complexity where cuts could be made to decrease power usage. The following section proposes our implementation for said architecture.

4. Proposed RISV-V System Architecture

Our Implementation of a SOC based RISC-V ISA focuses on IOT interfacing with sensors and devices via protocols like UART and GPIO serial communication peripherals. The SOC consists of a modular structure with each block communicating with the CPU via a bus multiplexer. The SOC can run C code with compilation through the GNU RISC-V toolchain and a custom compilation process. The GNU RISC-V toolchain makes use of several utilities to translate the input C language code into the final hexadecimal file which is fed to the memory.

- The gcc utility compiles the input C code and create an assembly file (. s format).
- The assembly file is further worked upon using bash scripting and sent to the assembler which return a .out file.
- The .out file acts as an input for the linker to create .elf file which is utilised by the objcopy utility to finally create a .hex file.
- At the end this .hex file is loaded into the program memory which is then fetched by the preprocessor and perform the necessary instruction.

Features of the SOC:

- Support for RV32IM instruction set
- 32KB Memory block
- 32 32-bit GPIO pins
- UART
- Bus multiplexer

- Dedicated timer block for clocks provided to each block
- Capable of running C code

More blocks can be added via interfacing them to the bus multiplexer.

5. Societal Implication

The IoT is the next technological revolution we are going to witness soon. But the overall induction of this technology still has some challenges (Sinha, B.B et al. 2022). These challenges include but are not limited to following:

- Final implementation costs
- Region Specific changes in need and feature set.
- Development Costs

Developing an open-source processors for such application can solve many of these problems. Development costs can be kept low by using open ISAs like RISC-V as they do not entail any licensing costs. Also, the flexibility of RISC-V ISA enables us to make region specific changes to the feature set. An urban node might need better bandwidth and data storage, whereas a rural node might need error correcting and overall stability. Different applications might also have different interface needs. This all can be accomplished using a modular approach to SoC designing and conditional gating of not used modules.

Rural areas in developing countries like India, have specific needs. The cost of the device will decide the bar for availability in such areas. Thus, keeping the costs low must be the most important factor while designing for such applications. Availability of IoT hardware which is both powerful and cost-effective could lead to a revolution in internet space for the rural areas. This could help in increase of yields and income and could provide a healthy boom to agriculture and related activities.

6. Conclusion and Future Scope

Internet of Things is a highly dynamic field, having advancements almost daily. With these advancements the need for the hardware also keeps evolving. Thus, to keep up with the times, open-source hardware designs need to be updated regularly. An implementation which will provide a complete solution for needs of a common IoT node could be very successful. The proposed designs strive to tread that balance of area and complexity intelligently. The nature of the field demands continuous improvements. Following suggestions can be made during the future implementations. The CPU core and the SOC can be verified using a complete verification environment using UVM tools. Since the current implementation of the proposed design does not feature any graphical interface to interact with the SOC, a video controller like VGA, HDMI can be incorporated in the design. Cryptography (for security purposes) and Digital Signal Processing is extensively used in IoT devices nowadays, therefore separate extensions for them would result in elevated processing speeds. Efficient pipeline algorithms play a big role when it comes to processor design. An efficient pipe lining mechanism would prove to be highly constructive for the core's processing speeds at the cost of extra hardware as long as it meets the minimum design requirements.

REFERENCES

Patterson, D., and Hennessy, J. 2017. . Computer Organization and Design RISC-V Edition: The Hardware Software Interface. Morgan Kaufmann Publishers Inc..

Gautschi, M., Schiavone, P., Traber, A., Loi, I., Pullini, A., Rossi, D., Flamand, E., Gürkaynak, F., and Benini, L. 2017. Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(10), p.2700–2713.

Miura, J., Miyazaki, H., and Kise, K. 2020. A portable and Linux capable RISC-V computer system in Verilog HDL. arXiv preprint arXiv:2002.03576.

Ahmadi-Pour, S., Herdt, V., and Drechsler, R. 2021. MircoRV32: an open source RISC-V cross-level platform for education and research. In Proceedings of the Workshop on Design Automation for CPS and IoT (pp. 30–35).

Tan, Z., Zhang, L., Patterson, D., and Li, Y. 2020. PicoRio: An open-source, RISC-V small-board computer to elevate the RISC-V software ecosystem. Tsinghua Science and Technology, 26(3), p.384–386.

Waterman, A., Lee, Y., Patterson, D., and Asanovic, K. 2011. The risc-v instruction set manual, volume i: Base user-level isa. EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62, 116.

Sadeeq, M., Zeebaree, S., Qashi, R., Ahmed, S., and Jacksi, K. 2018. Internet of Things security: a survey. In 2018 International Conference on Advanced Science and Engineering (ICOASE) (pp. 162–166).

YosysHQ. YosysHQ/picorv32: PicoRV32 - A Size-Optimized RISC-V CPU.

Efabless. efabless/raven-picorv32: Silicon-validated SoC implementation of the PicoSoc/PicoRV32.

Ultraembedded. ultraembedded/riscv: RISC-V CPU Core (RV32IM).

Sinha, B.B. and Dhanalakshmi, R., 2022. Recent advancements and challenges of Internet of Things in smart agriculture: A survey. Future Generation Computer Systems, 126, pp.169-184.