ICIC Express Letters
Part B: Applications
Volume **14**, Number **7**, July **2023**

ICIC International ©2023 ISSN 2185-2766
pp. **673–683**

DEVELOPMENT OF A RISC-V CPU AND EDUCATIONAL LINUX SYSTEM ON A CHIP FRAMEWORK RV32XSOC

Naoki Abe, Chihiro Koyama and Naohiko Shimizu

School of Information and Telecommunication Engineering
Tokai University
2-3-23, Takanawa, Minato-ku, Tokyo 108-8619, Japan
n2113abe@star.tokai-u.jp; 9bjnm012@fuji.tokai-u.jp; nshimizu@tokai.ac.jp

Received November 2022; accepted February 2023

Abstract. *In recent years, RISC-V has become more popular as a practical instruction set architecture that is royalty-free and open. Both academia and industry have developed various chips using RISC-V processors so far. Furthermore, the RISC-V community has ported many former software assets such as gnu-toolchain and Linux to RISC-V. In this paper, we present Linux-compatible RISC-V System on a Chip, RV32XSoC, and its educational framework. We designed the system-on-chips with minimum requirements to support the Unix-like operating system. Furthermore, we constructed an educational framework by developing the software simulator. We can observe execution flow from both hardware and software perspectives through the software simulator. We can effectively acquire comprehensive knowledge about computer systems through this work and its framework.*
Keywords: RISC-V processor, Linux-compatible System on a Chip, Hardware education

1. **Introduction.** Semiconductor process shrinking has made to build high-performance and multifunctional chips. In recent years, system-on-chips (SoCs) that incorporate many functions on a single chip have become mainstream in the embedded domain. However, such SoCs are large and complex circuits. In large systems, the software scale has also become large. For these reasons, hardware/software co-design is important to construct large systems. Embedded engineers who develop such systems are required to have skills in both hardware and software. Since such engineers are scarce, human resources with these skills are wanted in the embedded domain. The objective of this research is to develop a framework for comprehensive educational exercises in hardware to software for embedded systems, which are becoming complex. For this purpose, we implemented a compact CPU with a minimum ISA that can run Unix-like OS and demonstrate the framework by porting and running xv6 and Linux. We use NSL [1] as a development language. Despite clock accuracy, NSL has a high-level abstraction. We achieved high abstraction and readability by describing all SoC components in NSL. The rest of the paper is organized as follows. Chapter 2 shows related works of this study. Chapter 3 shows the RISC-V processor core implementation in RV32XSoC. Chapter 4 shows the design of RV32XSoC. Chapter 5 shows the verification method and FPGA configuration results. Chapter 6 shows Unix-like OS porting of RV32XSoC. Chapter 7 shows the conclusion of this study.

2. **Related Work.** There are many projects to study LSI. MIPSfpga [2] is learning digital circuit design and computer architecture by implementing MIPS microprocessors using BASYS development boards. The semester-long project of the University of Michigan [3] is learning computer architecture and SystemVerilog through the design and implements

DOI: 10.24507/icicelb.14.07.673

an out-of-order pipelining processor core using RISC-V. However, these projects focus on hardware education, and a few of them include software such as operating systems.

Rocket chip [4] and Boom [5] are previous research on the high-performance RISC-V processors that can run Linux. The Rocket-Chip has the RISC-V core which is implemented almost all ISA, caches, interconnects, and so on. Boom is a RISC-V out-of-order processor, which can decode and issue multiple instructions, and thus has higher performance than Rocket. Both are open-source, and anyone can use them. Both are implemented in Chisel [6]. Chisel is the register transfer level (RTL) language which is an internal domain-specific language (DSL) of Scala. Chisel can compile to Verilog. Chisel achieves high productivity by using Scala-related assets. However, it has a high learning cost because developers learn both Chisel and Scala. In addition, compiling Chisel to Verilog is slow because the process includes Scala language processing. Also, the learning cost is high because of hardware architecture complexity due to multifunctionality.

3. **Design of RISC-V Processor in RV32XSoC.** We designed the RISC-V processor which supports a Unix-like operating system. Table 1 shows extensions of the RISC-V ISA we implemented. We implemented minimum ISA to support Unix-like OS, excluding

Table 1. Implemented RISC-V ISA

Extensions	Features
RV32I	32-bit base integer instruction set
M Extension	Hardware multiplication and division instruction support
A Extension	Atomic memory instruction support
Zifencei	Instruction-Fetch fence instruction support
Zicsr	CSR instruction support
M/S/U-mode	Machine/Supervisor/User mode support
SV32	Page-based 32-bit virtual-memory systems support

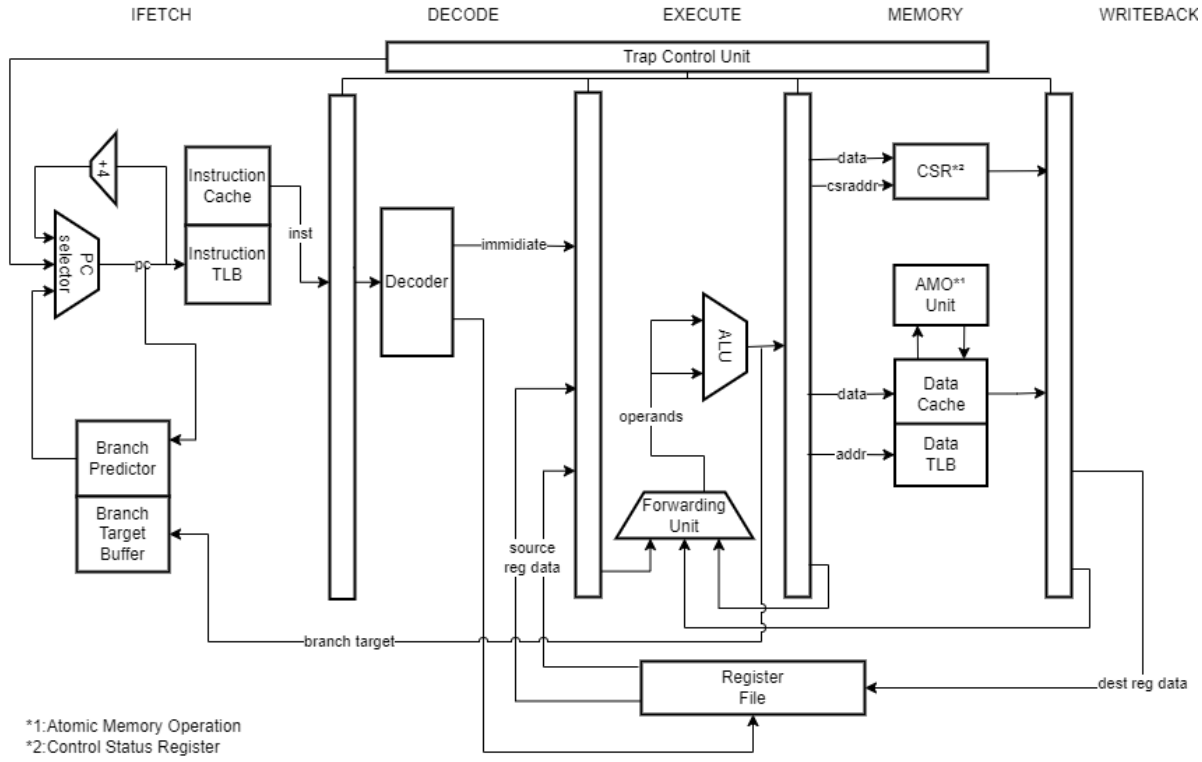

Figure 1. The block diagram of our RISC-V processor core

Table 2. The structures we implemented for acceleration

Structures	Remarks
Pipelining	In-order 5 stage pipeline
TLB	4-entries instruction TLB, and 32-entries data TLB
Cache	4KB direct-mapped instruction cache, and 4KB 2-way set-associative data cache
Branch prediction	2-bit saturating counter with 32-entries branch target buffer (BTB)
Forwarding	Detects read-after-write (RAW) hazard and passes directly the value among the stage

Table 3. Description of pipeline stages

Stage	Features	Hazard detection (excluding trap)
Ifetch	Fetches instruction of current PC, updates PC	Instruction cache miss
Decode	Fetches value of operand by register file, generates immidiate	RAW hazard, instruction at execute stage are taken branch or jump
Execute	Processes calculation in ALU, updates BTB on the branching result	Load hazard (Memory, CSR), multiplication and division
Memory	Accesses memory or CSR, processes AMO instructions	Data cache miss, TLB miss, fence, fence.i, sfence.vma and AMO
Writeback	Updates register-file	None

floating-point extensions and some not critical control status registers (CSRs) to run Unix-like OS from the "general-purpose" ISA [7]. Figure 1 shows an overview of the RISC-V processor. Our processor has an in-order five-stage pipeline. We implemented several structures for acceleration such as pipelining to our processor. Table 2 shows the structures for acceleration we implemented in our processor. The forwarding unit passes the ALU calculation result or read memory value to the instruction in the previous stage if the subsequent instruction has read after write (RAW) hazard. The pipeline stages are composed of Ifetch, Decode, Execute, Memory, and Writeback. Table 3 shows the description of pipeline stages. In addition to those structures, the processor includes a trap-control line. The trap control line stalls the appropriate stages when a trap occurs. After that, the PC updates to the trap vector defined in trap vector CSR.

4. **Design of RV32XSoC.** We designed the RV32XSoC with minimum required circuits to support Unix-like OS. Table 4 shows the SoC components we implemented. All these components are written by NSL. Table 5 shows the NSL code size of the whole components of our SoC. The total code size is about 7000 lines. The real codes of RV32XSoC are less than 7000 lines because these codes include debugging signals handling. Figure 2 shows an overview of RV32XSoC. All these components are memory-mapped access as Figure 2 shows. We implemented UART and SPI Master to use Unix-like OS character/block devices. These modules can interrupt driven. We use platform level interrupt controller (PLIC) and core-local interruptor (CLINT) [8] as interrupt controllers. Figure 3 shows the connection between the processor and the interrupt controller. PLIC handles external device interrupts. PLIC sends interrupt requests from an external device to the processor external interrupt pending (EIP) CSR. The processor handles interrupts of source devices based on information stored in memory-mapped PLIC CSRs. CLINT handles timer interrupt and software interrupt for each hart. CLINT counts time in constant frequency

Table 4. RV32XSoC components

Components	Remarks
RISC-V Processor	Single core. Detailed in Section 3: Design of our RISC-V processor
PLIC	Support 31 IRQ line, 8-level IRQ priority, 2 hart contexts
CLINT	50Mhz, support only one hart context
UART	50Mhz, support 38400 bps, 8-bit data length, no parity check
SPI I/F	25Mhz, MMC compatible

Table 5. NSL files of RV32XSoC

			Filename	Lines
SoC	Core	ALU	adder32.nsl	19
			alu32.nsl	67
			div32.nsl	74
			inc32.nsl	18
			mul32.nsl	45
			munit32.nsl	95
			shifter32.nsl	34
			sub32.nsl	19
			cache.nsl	390
			ptw.nsl	73
			tlb.nsl	69
			btb.nsl	107
			imm_gcn.nsl	32
			amoalu.nsl	29
			inst_dec.nsl	213
			load_store_unit.nsl	426
			ifetch_unit.nsl	234
			reg32.nsl	241
			rv32x5p.nsl	1037
			rv32x_core.nsl	1572
			bootrom.nsl	31
			clint.nsl	55
			fifo.nsl	61
			mmio_dev.nsl	62
			plic.nsl	361
			uart_reciever.nsl	158
			uart_sender.nsl	122
			mmcspi.nsl	992
			rv32x_integration.nsl	337

and generates timer interrupt at the interval set in the timer comparison CSRs. CLINT also generates software interrupt by writing machine software interrupt pending (msip) CSR. The software interrupt completes by clearing the msip CSR of CLINT.

5. Verification.

5.1. **RV32XSoC software simulator.** We developed a software simulator of RV32X-SoC for verification and debugging. Figure 4 shows the simulator verification flow. We use Verilator [9] to generate a C++ RTL simulation model from the compiled Verilog HDL files. Our software simulator verification flow is as follows. First, compile the NSL files of the simulation-only module and whole components of our SoC to Verilog HDL by

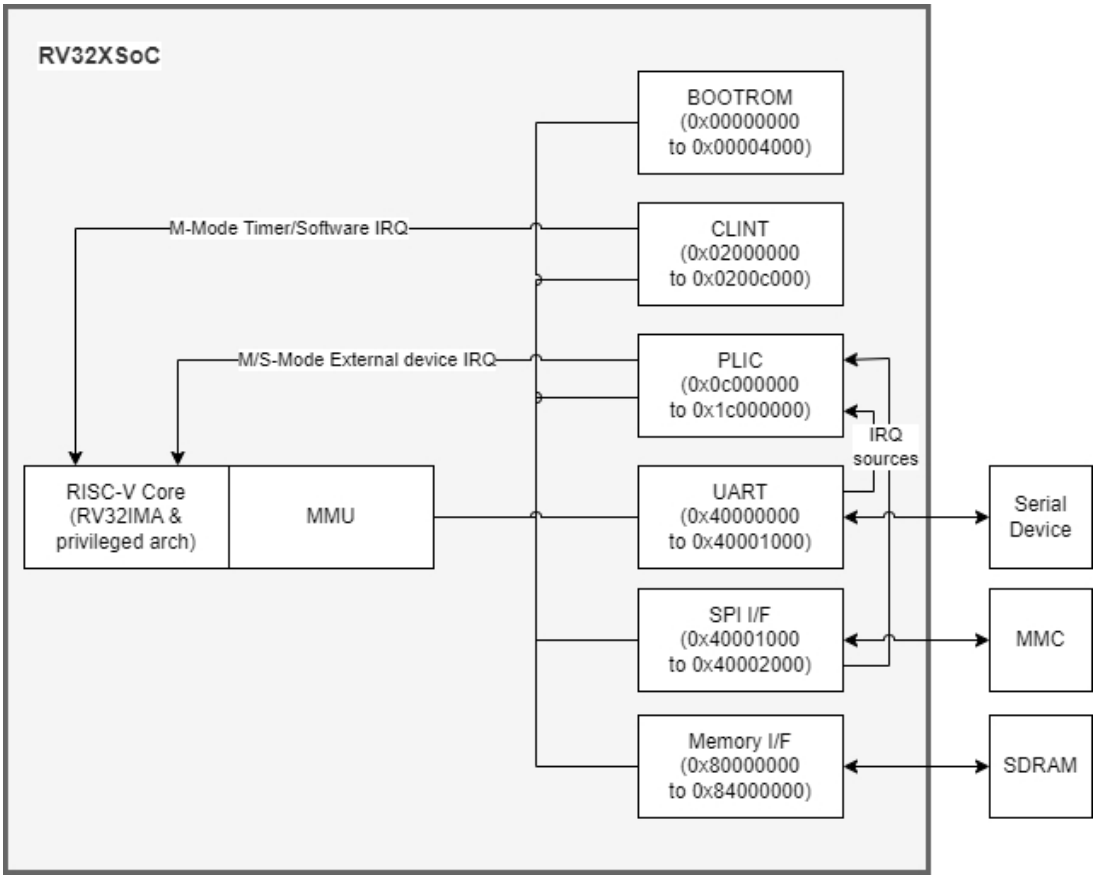

FIGURE 2. An overview of RV32XSoC

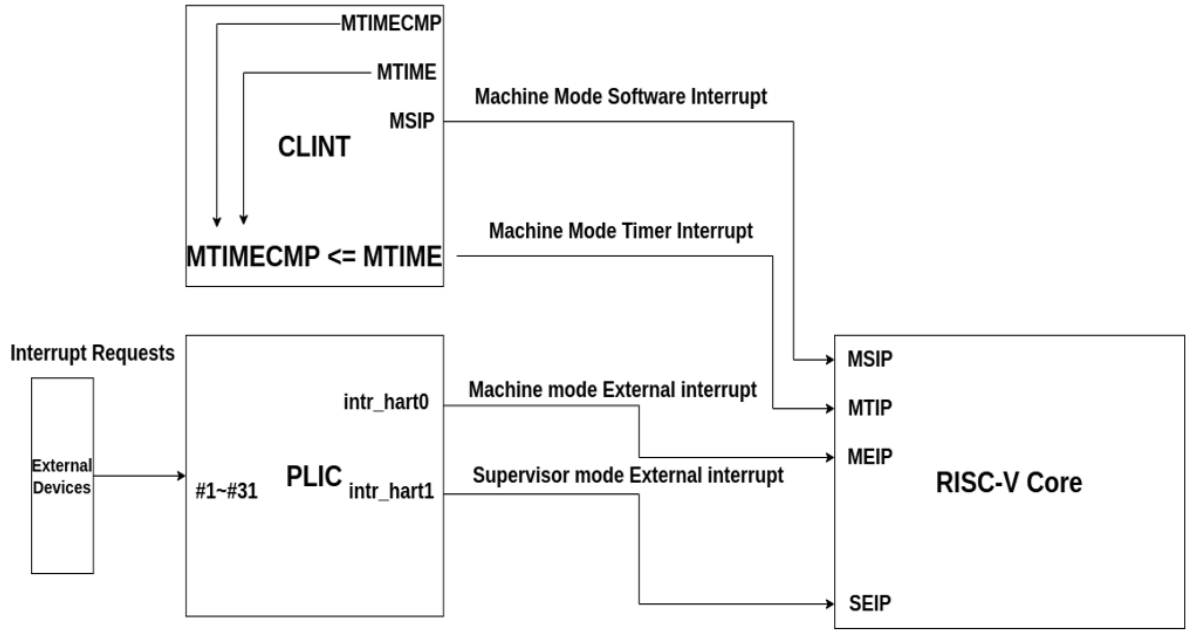

FIGURE 3. Processor and interrupt controller connection

NSL compiler (NSL2VL). Second, generate a C++ RTL simulation model by Verilator. Then, compiling our simulator source and simulation model on gnu C++ compiler (g++) makes the software simulator. Figure 5 shows the interface between the simulator and the RV32XSoC simulation model. The simulator expands the executable linkable format (elf) passed as a command-line argument into the host machine memory and interfaces between the host machine memory and the simulation model. The simulator passes the

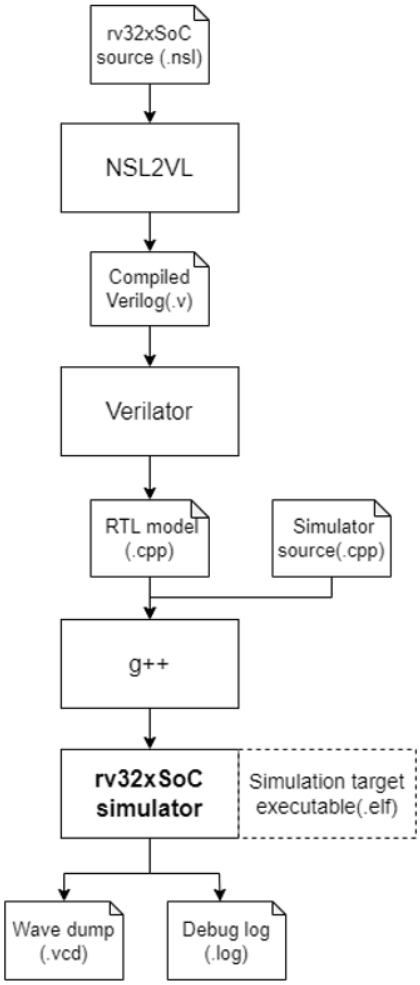

FIGURE 4. Verification flow

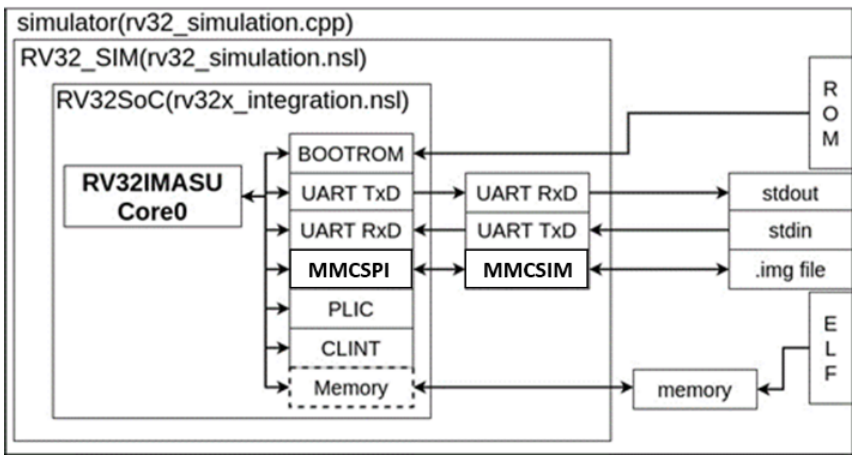

FIGURE 5. Interface of RV32XSoC simulator

host machine standard input to the UART input and passes the UART output to the host machine standard output. The simulator emulates the MMC card as a binary file. Thus, the simulator emulates RV32XSoC and generates a waveform dump and execution log. The waveform dump is a value change dump (VCD) format. We can detect implementation errors and analysis timing-critical paths with VCD observation software. The execution log contains much helpful information for debugging like disassembling, register/memory trace, interrupt information, and procedure/object tracking. Figure 6 shows

```
     s10 <- 002c18a0
core    0: 0x00000000001d530c (0xa20504e3) beqz a0,0x00000000001d4d34
core    0: 0x00000000001d5310 (0x00090593) mv   a1,s2
      a1 <- 00000180
core    0: 0x00000000001d5314 (0x978fd0ef) jal  ra,0x00000000001d248c
      ra <- 001d5318

Depth=6, entry: alloc_perturb(0x1d248c) inst_counter = 123524172
core    0: 0x00000000001d248c (0x2ec1a783) lw   a5,748(gp)  < a5=*(perturb_byte) >
      a5 <- 00000000
core    0: 0x00000000001d2490 (0x00079463) bnez a5,0x00000000001d2498
core    0: 0x00000000001d2494 (0x00008067) ret
core    0: 0x00000000001d5318 (0xa1dff06f) j    0x00000000001d4d34
core    0: 0x00000000001d4d34 (0x07c12083) lw   ra,124(sp)
      ra <- 001d5e24
core    0: 0x00000000001d4d38 (0x07812403) lw   s0,120(sp)
      s0 <- 002bd89c(main_arena)
```

FIGURE 6. Simulator execution log

an example of the execution log. We can analyze RV32XSoC execution flow using the software-level output from the execution log and the register-transfer-level output from the wave dump.

5.2. **FPGA configuration.** We did an FPGA logic synthesis of RV32XSoC using Quartus Prime 21.1. Table 6 shows the configuration result of our SoC and its RISC-V core. We choose Cyclone V 5CEBA4F23C7 and Arria II EP2AGX45DF29I5 as synthesis target devices. Large data blocks such as cache are allocated to block RAM by Quartus Prime RAM inference. We can use more resources for further expansion and acceleration because both targets have surplus logic elements. We did a comparison with other Linux-ready RISC-V SoC. Table 7 shows a comparison between RV32XSoC and other Linux-ready RISC-V SoC design configurations. OpenPiton+Ariane [10] includes the 64-bit RISC-V processor which implements RV64GC (In the configuration, without FPU), UART, SD/SDHC controller, MMU, Ethernet, CLINT, PLIC, and debug module. OpenPiton+Ariane is the largest scale configuration result in the comparison. It is because it has a large data path and most SoC components. RVSoC [11] includes the RISC-V processor which implements RV32IMAC, RISC-V microcontroller which implements RV32I, MMU, VirtIO-based console, and disk. RVSoC is the least scale configuration result in the comparison, but it is because it has no pipeline structure. In comparison, RV32XSoC is the compact configuration result even though pipeline structure.

TABLE 6. RV32XSoC and its RISC-V core FPGA configuration result

	Family	**LUTs**	**FFs**	**Logic utilization**	**Memory blocks**	**DSP block**	**Fmax**
RV32XSoC	Cyclone V	14647	18136	11985/18480 (65%)	16/No data (5%)	6/66 (9%)	48.21Mhz
	Arria II	18291	17405	No data (78%)	17/319 (5%)	4/232 (2%)	71.67Mhz
RISC-V core	Cyclone V	10641	8586	7122/18480 (39%)	14/No data (5%)	6/66 (9%)	47.37Mhz
	Arria II	8515	10360	No data (44%)	15/319 (5%)	4/232 (2%)	71.76Mhz

TABLE 7. FPGA resource usage comparison of Linux-ready RISC-V SoC

	OpenPiton+Ariane [10]	**RVSoC [11]**	**RV32XSoc**
Family	Virtex-7	Artix-7	Cyclone V
LUTs[k]	99	10	14
FFs[k]	73	6	18
BRAM type	RAMB36 (36kb/tile)		M10K (10kb/tile)
BRAM tiles	63	33	16
DSPs	16	No data	6
configured ISA	RV64IMAC	RV32IMAC	RV32IMA
Pipeline	6-stage In-order	None	5-stage In-order

5.3. **Dhrystone benchmark.** We did the Dhrystone benchmark to evaluate the performance of our processor. Table 8 shows the result of the benchmark. We did the benchmark of RV32XSoC processor core implemented on Cyclone V 5CEBA4F23C7 based board. The benchmark program was compiled with GCC 11.1.0 using the -O2 option. Figure 7 shows a comparison of DMIPS/Mhz with other CPUs. Our processor has almost the same performance as Pic 24 and MSP430, which are low-end embedded processors.

TABLE 8. Dhrystone benchmark result

Family	**MHz**	**Dhrystone/sec**	**DMIPS**	**DMIPS/MHz**
Cyclone V	50.0	39406	22.428	0.448

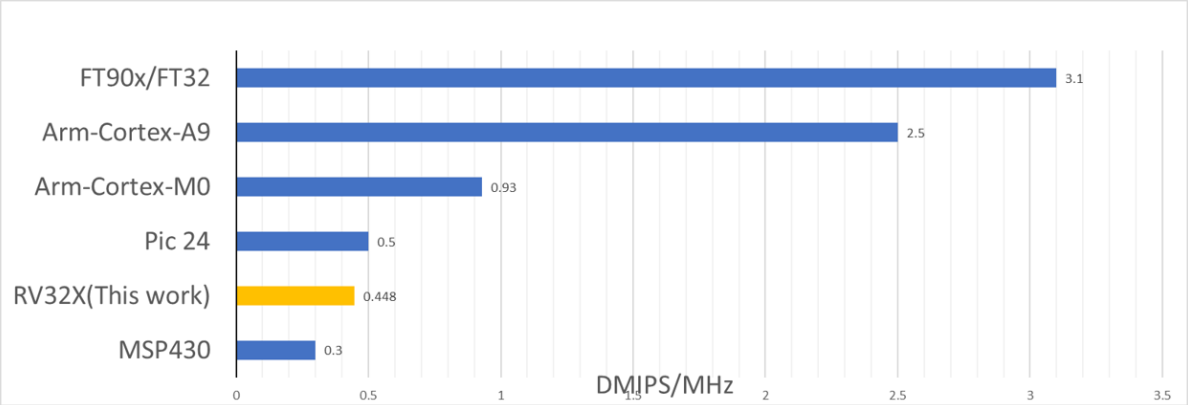

FIGURE 7. Comparison of DMIPS/MHz with other CPUs [12]

6. **Unix-Like OS Support.** We ported the RV32XSoC to xv6 and Linux to make Unix-like operating system practice and teaching environment. We can effectively learn the interface between the computer hardware and OS through the exercise environment we constructed. We did these two OS validations using the RV32XSoC software simulator. Table 9 shows the problems encountered during validation. These problems were not detected in the unit test. We were able to fix these problems early by analyzing the execution logs and waveforms of the software simulator.

6.1. **xv6.** xv6 is RISC-V compatible educational operating system developed by MIT. The pure xv6 is designed to run in QEMU with VirtIO as an external disk and UART16550 as serial communication. Accordingly, we replaced the device driver to make it compatible with the RV32XSoC-specific UART and disk controller.

Table 9. Problems with xv6 and Linux validation

Problem	Cause
xv6 outputs "panic:release" message and fails to boot.	The executed instruction was re-executed after the return from interrupt. As a result, illegal value was entered in the instruction which source and destination are the same.
Linux/init process does not work.	AMOSWAP.W instruction following the FENCE instruction had been executed twice. As a result, the mutex used in the init program had an incorrect value, cause deadlock.
Linux stops working after a few tens of seconds after boot.	When the machine interrupt occurred immediately after the supervisor exception, the supervisor context was destroyed. As a result, it was not possible to recover from the interrupt.

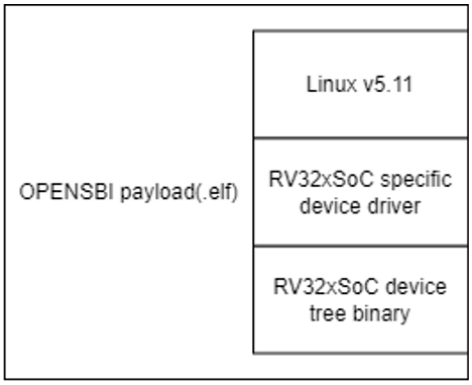

Figure 8. Executable overview

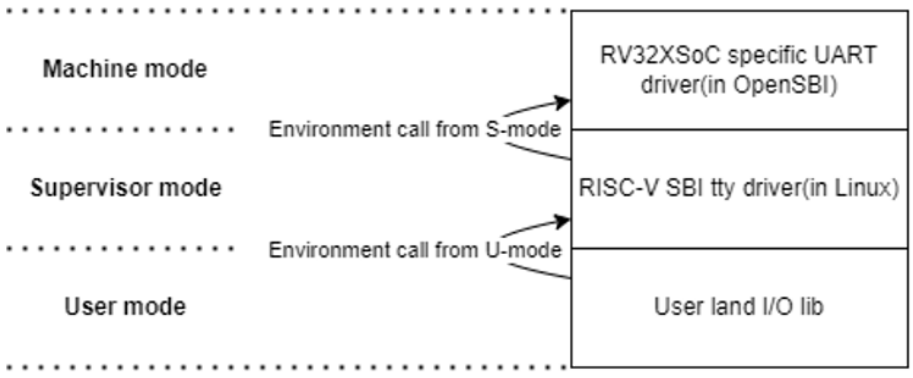

Figure 9. Character device access overview

6.2. **Linux.** We choose Linux version 5.11 for validation. We made a device tree and device driver which are necessary to run Linux on RV32XSoC. The device tree is a format for describing hardware-specific information and is referenced in the Linux kernel. We use OpenSBI as the firmware and boot loader to run Linux. OpenSBI is a supervisor binary interface (SBI) [13] that runs in machine mode. It works as the interface between hardware and the supervisor software by controlling the device driver. In this project, we packed OpenSBI, Linux, the device tree, and the device driver in one executable. Figure 8 shows an overview of the packed executable. The device tree is passed to the Linux kernel by the SBI. Linux kernel accesses the device driver via SBI. Figure 9 shows an overview of Linux character device access using SBI. RV32XSoC-specific hardware I/O process is done by elevating the privilege mode using the ECALL instruction. In addition

```
[    0.028550][    T0] Console: colour dummy device 80x25
[    0.042052][    T0] printk: console [hvc0] enabled
[    0.042052][    T0] printk: console [hvc0] enabled
[    0.071137][    T0] printk: bootconsole [sbi0] disabled
[    0.071137][    T0] printk: bootconsole [sbi0] disabled
[    0.103522][    T0] Calibrating delay loop (skipped), value
[    0.136647][    T0] pid_max: default: 32768 minimum: 301
[    0.157842][    T0] Mount-cache hash table entries: 1024 (or
[    0.177279][    T0] Mountpoint-cache hash table entries: 102
[    0.265467][    T1] clocksource: jiffies: mask: 0xffffffff m
[    0.271995][    T1] futex hash table entries: 256 (order: 0,
[    0.408260][    T1] clocksource: Switched to clocksource ris
[    4.683213][    T1] workingset: timestamp_bits=30 max_order=
[    5.035492][    T1] Block layer SCSI generic (bsg) driver ve
[    5.039827][    T1] io scheduler mq-deadline registered
[    5.043000][    T1] io scheduler kyber registered
[    5.045742][    T1] start plist test
[    5.395589][    T1] end plist test
[    6.708978][    T1] brd: module loaded
[    6.721051][    T1] debug_vm_pgtable: [debug_vm_pgtable
[    6.954787][    T1] Freeing unused kernel memory: 5344K
[    6.959028][    T1] Run /init as init process
[    6.961589][    T1]   with arguments:
[    6.963477][    T1]     /init
[    6.965444][    T1]   with environment:
[    6.967490][    T1]     HOME=/
[    6.969537][    T1]     TERM=linux

Please press Enter to activate this console.
/ # uname -r
5.11.0-00001-g485a42756d45-dirty
/ # echo hello
hello
/ # ls
bin      etc      lib      mnt      sbin     tmp      var
dev      init     linuxrc  proc     sys      usr
/ # ps
```

FIGURE 10. Linux on RV32XSoC

TABLE 10. Gantt chart on developing RV32XSoC

Work \ Time (month)	1	2	3	4	5	6	7	8
RISC-V core implementation	■	■	■	■				
SoC components implemation					■			
Programming simulator	■						■	
Testing ISA	■	■	■	■	■			
Testing SoC components					■			
Porting xv6						■		
Porting OpenSBi, Linux							■	■

to building these interfaces, we set up the root file system using busybox [14] and Linux kernel config for the features we implemented. Thus, we constructed the Linux execution environment. Figure 10 shows Linux running in the execution environment.

7. **Conclusion.** We developed an educational SoC framework with a compact CPU that is highly effective in education. Table 10 shows the development Gantt chart. We were able to understand, develop, and debug the system in a short period by using the high-level

synthesis language NSL. Through this project, we can effectively acquire comprehensive knowledge about computer systems such as computer architecture, operating system, and device driver. We can learn the interface between hardware and software through the framework by supporting Unix-like OS such as Linux and xv6. RV32XSoC is highly readable and compact. For this reason, we can use it as a basic framework for various educational programs, such as CPU speed acceleration, adding I/O, and adding OS support. Therefore, the framework we constructed is effective for learning and teaching both hardware and software.

REFERENCES

[1] NSL Core, *Overtone Corporation*, http://www.overtone.co.jp/en/support/downloads, Accessed on 06/09/2022.

[2] S. L. Harris, R. Owen, E. Sedano and D. C. Martinez, MIPSfpga: Hands-on learning on a commercial soft-core, *2016 11th European Workshop on Microelectronics Education (EWME)*, pp.1-5, DOI: 10.1109/EWME.2016.7496470, 2016.

[3] S. A. Zekany, J. Tan and J. A. Connolly, Teaching out-of-order processor design with the RISC-V ISA, *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*, pp.1-8, DOI: 10.1109/WCAE53984.2021.9707143, 2021.

[4] K. Asanović, R. Avizienis, J. Bachrach et al., *The Rocket Chip Generator*, Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html, 2016.

[5] C. Celio, P.-F. Chiu, B. Nikolic, D. A. Patterson and K. Asanović, *Boom V2: An Open-Source Out-of-Order RISC-V Core*, Technical Report UCB/EECS-2017-157, EECS Department, University of California, Berkeley, 2017.

[6] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek and K. Asanovič, Chisel: Constructing hardware in a Scala embedded language, *DAC Design Automation Conference*, pp.1212-1221, 2012.

[7] A. Waterman, Y. Lee, D. A. Patterson and K. Asanovíc, *The RISC-V Instruction Set Manual*, Volume I: Unprivileged ISA, Document Version 20191213, https://riscv.org/specifications, Accessed on 06/09/2022.

[8] *SiFive Interrupt Cookbook Version 1.2*, https://starfivetech.com/uploads/sifive-interrupt-cookbook-v1p2.pdf, Accessed on 06/09/2022.

[9] *Verilator*, https://www.veripool.org/verilator/, Accessed on 19/11/2022.

[10] J. Balkind, K. Lim, F. Gao, J. Tu, D. Wentzlaff, M. Schaffner, F. Zaruba and L. Benini, OpenPiton+Ariane: The first open-source, SMP Linux-booting RISC-V system scaling from one to many cores, *Proc. of the 3rd Workshop Comput. Archit. Res. RISC-V (CARRV)*, pp.1-6, 2019.

[11] J. Miura, H. Miyazaki and K. Kise, A portable and Linux capable RISC-V computer system in Verilog HDL, *arXiv.org*, https://arxiv.org/abs/2002.03576, Accessed on 06/09/2022.

[12] *Application Note AN 304 FT90x Microcontroller Benchmark Version 1.1*, http://www.ftdichip.com/Support/Documents/AppNotes/AN_304%20FT900%20Microcontroller%20Benchmark.pdf, Accessed on 11/19/2022.

[13] P. Dabbelt and A. Patra, *RISC-V SBI Specification 1.0.0*, https://github.com/riscv-non-isa/riscv-sbi-doc, Accessed on 06/09/2022.

[14] *BusyBox: The Swiss Army Knife of Embedded Linux*, https://busybox.net/, Accessed on 06/15/2022.

[15] R. Cox, F. Kaashoek and R. Morris, *xv6: A Simple, Unix-Like Teaching Operating System*, https://pdos.csail.mit.edu/6.S081/2020/xv6/book-riscv-rev1.pdf, Accessed on 06/09/2022.