Orca: Optimized RISC-V Cryptographic Accelerator

Kosi Nwabueze
Massachusetts Institute of Technology
Cambridge, Massachusetts
kosinw@mit.edu

Haoran Wen
Massachusetts Institute of Technology
Cambridge, Massachusetts
hranwen@mit.edu

ABSTRACT

Hardware acceleration for cryptography is widely supported in modern computer architectures. Intel first added support for Advanced Encryption Standard (AES) instructions to x86 in 2008. ARM added support for AES instructions for ARMv8 in 2011. However, instead of adding instructions to extend the ISA, we propose a different model for improving the performance of AES cryptographic routines: a MMIO-controlled coprocessor. We present Orca, a RISC-V microcomputer implemented on a Xilinx Spartan-7 FPGA which features: a coprocessor providing a hardware implementation of AES-128, a text-mode video card reminiscent of 1980s IBM VGA cards, a PS/2 interface for a keyboard, and an SPI bus for board-to-board network communication. By offloading cycle-intensive cryptographic routines such as AES encryption and decryption to a coprocessor, we expect to observe a massive throughput increase compared to software-implemented alternatives.

1 INTRODUCTION

At a high level, the final microcomputer design consists of the following components:

Central Processing Unit. At the heart of the Orca microcomputer sits a RISC-V [1] core acting as the central processing unit. Our custom core implements the RV32IM ISA with a classic RISC pipeline complete with an interrupt controller, control and status registers (CSRs), and instruction and data caches.

Video Card. Orca can display a matrix of 160×45 character cells on a 720p HDMI display. Software can manipulate each character cell individually by writing to a memory mapped buffer.

Keyboard. Orca supports a keyboard peripheral device which uses the PS/2 protocol. Whenever the microcomputer receives a scancode from the keyboard, the scancode is written in a memory-mapped hardware register and the interrupt controller triggers an interrupt.

SPI Bus. Orca can communicate with other microcomputers over the SPI protocol. Packets are sent in frames and the interrupt controller will be triggered whenever a complete packet is received over the bus.

AES-128 Coprocessor. Orca features a cryptographic coprocessor which implements both encryption and decryption with the AES block cipher. Software can interface with this coprocessor through memory-mapped registers.

Figure 1 shows how the different submodules of the microcomputer interact. Orca has a 32-bit address bus, which is used for ROM, RAM, and I/O. Table 1 shows the physical memory map of the entire microcomputer.

In the following sections, we will discuss the design of each submodule, our implementation progress so far, and further work that needs to be completed before the final report.

Table 1: Physical address space of Orca microcomputer.

Start	End	Description	Notes
00000h	1FFFFh	General Purpose RAM	Manta-programmable
20000h	23840h	Video RAM	
30000h	3007Fh	PS/2 Scancode Buffer	Ring buffer
30080h	3008Fh	PS/2 Control Registers	Reserved for future use
40000h	4FFFFh	AES Reserved	Reserved for future use
50000h	5FFFFh	SPI Reserved	Reserved for future use

2 CENTRAL PROCESSING UNIT

Currently, the implementation of the processor is incomplete. At the moment, we have a single-cycle processor; however, in the final design we expect to use a 5-stage pipeline with basic hazard management. Also, the interrupt controller has not been implemented yet. The central processing unit (CPU) at the core of the Orca computer is founded on a RISC-V design, specifically adhering to the RV32I architecture with the incorporation of the Multiply Extension. The processor's architectural framework is organized into five stages:

- **IF (Instruction Fetch)**: Responsible for fetching instructions.
- **ID (Instruction Decode)**: Concerned with decoding instructions.
- **IE (Instruction Execute)**: Manages the execution of instructions.
- **MEM (Memory)**: Governs memory-related operations.
- **WB (Writeback)**: Handles the writeback process to registers.

Within the "core" folder, there are the essential components of the RISC-V core.

- riscv_core.sv: This file contains the high-level flow of the RISC-V processor, including both memory and writeback functionalities.
- riscv_defs.sv: This file serves as the repository for the definitions of RV32IM instructions and opcodes.
- riscv_decode.sv: Responsible for the decoding of instructions.
- riscv_execute.sv: Manages the execution of instructions.
- riscv_register.sv: Controls the registers.

The memory hierarchy for the general purpose memory on Orca uses a block RAM with an 8-bit data width, 32-bit address width, and a depth of 128 KiB.

3 VIDEO CARD

Currently, the implementation of the video card is complete. The video card is a text-mode display, which treats the content of the

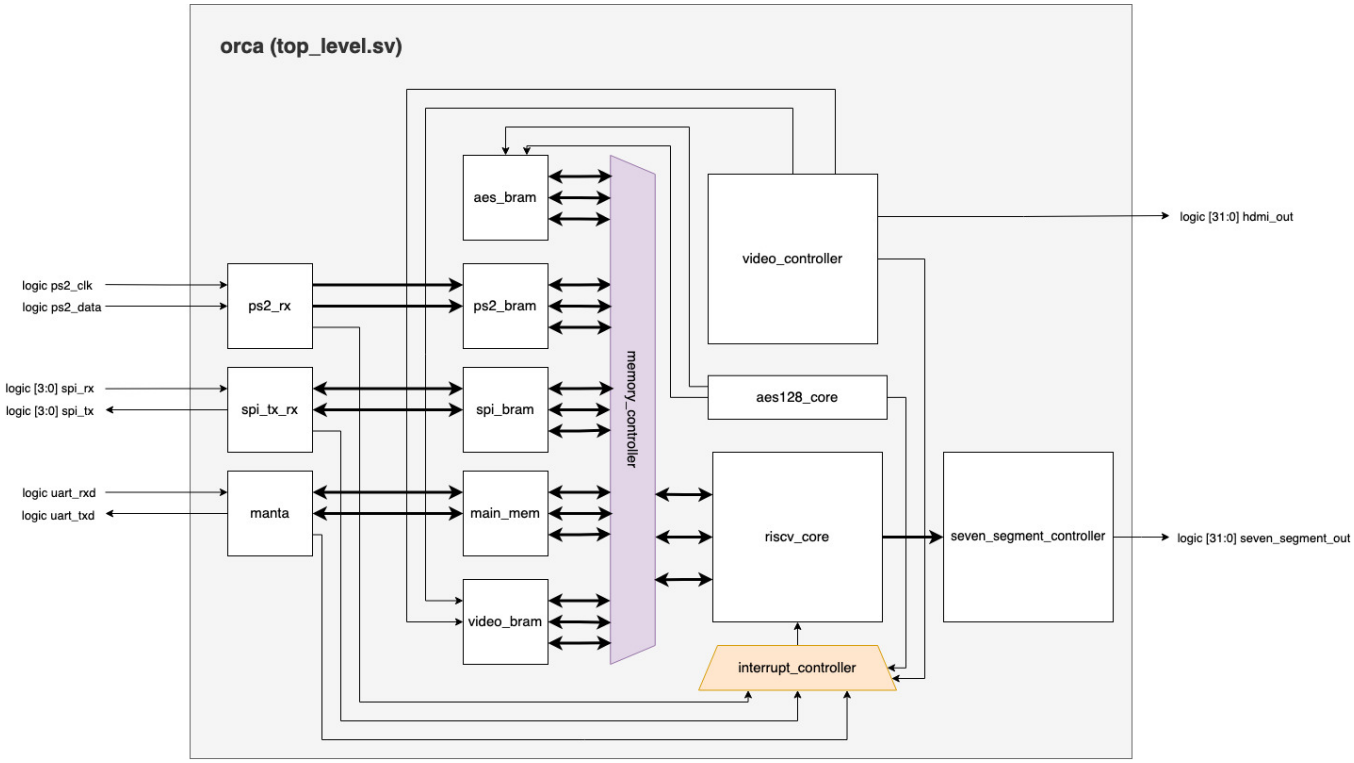

Figure 1: A block diagram of the Orca architecture.

screen in terms of characters instead of individual pixels. The video card displays a 160×45 matrix of character cells on the screen. Each character cell is 8×16 pixels, so in total the video buffer covers 1280×720 pixels, utilizing 100% of the space in 720p HD video.

Figure 2 demonstrates the high-level design of the video card. We reuse the `video_sig_gen.sv` module we wrote in earlier labs to generate a video signal at around a 74MHz clock rate. Using the horizontal count and vertical count signals, the video hardware performs a look-up in software-controlled video memory to determine what character to render at that position on the screen. Then, using the `font_brom.sv` module, the hardware figures out the pattern to draw for a particular character by looking up its dot matrix pattern in memory storing its font. Finally the `attribute_brom.sv` module determines the background color (one of 8 colors based on the original Microsoft VGA Palette), the foreground color (one of 16 colors based on the same palette), and whether or not that character cell is blinking.

For any character cell location (x, y), user software can configure what character is displayed by writing the ASCII value (technically any character from IBM code page 437 works) of the character to memory address $0x3000 + 2 \times (160 \times y + x)$. To manipulate the background and foreground color, write to address $0x3000 + 2 \times (160 \times y + x) + 1$.

The character font and color palette are converted to a synthesizable `.mem` file by a custom Python script. During synthesis, Vivado flashes the read-only block memories with the appropriate `.mem` files.

4 KEYBOARD

Currently, the implementation of the keyboard is complete. Keyboard peripheral support is implemented by the `ps2_rx.sv` module. The keyboard interfaces with a PS/2 connector which is connected to the Urbana Board through a PS/2-to-Pmod breakout board. Figure 3 shows the simple finite state machine which implements the receiver logic for the keyboard.

Scancodes are the fundamental message send by the peripheral device. Each scancode is a serial frame of 10 bits:

- 1 start bit (always 0)
- 8 data bits
- 1 parity bit (odd parity)
- 1 stop bit (always 1)

The PS/2 protocol outputs one scancode whenever a key is pressed an then outputs the same scancode preceded by the byte `0xF0` when the key is released.

The receiver module writes scancodes it receives from the keyboard into module `ps2_bram.sv`, which implements a hardware ring buffer. After the scancode has been written to memory, the receiver module notifies the CPU through a hardware interrupt. Software can then read from the ring buffer and update appropriate status registers to notify the PS/2 hardware which characters have been read.

5 SPI BUS

Currently, we have not started implementation on the SPI bus. In principle, the SPI bus will allow network communication between

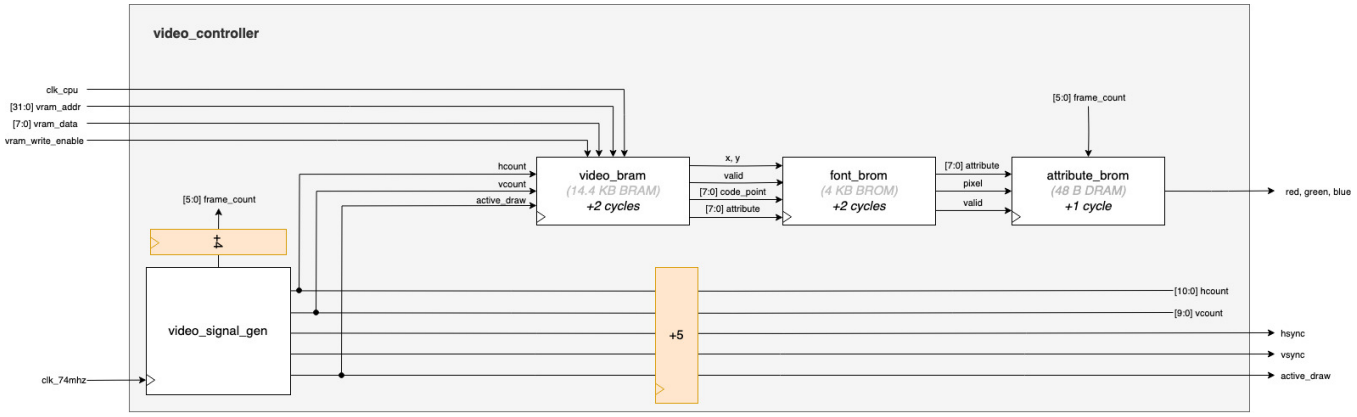

Figure 2: A block diagram of the video pipeline.

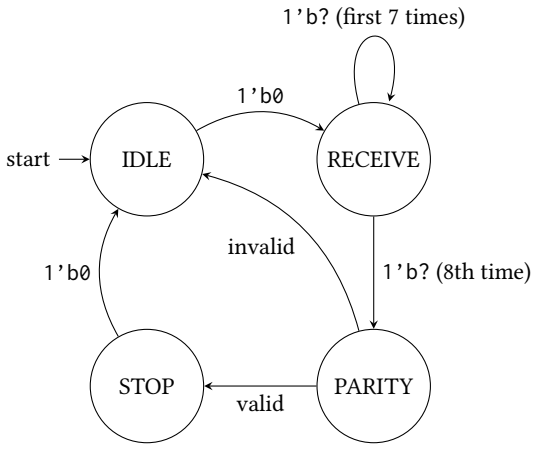

Figure 3: Receiver state machine for the PS/2 protocol.

two Orca computers. The operational sequence is as follows. First, software writes a message into a particular buffer into the SPI hardware. Then, software will tell the SPI hardware to send the message by writing a value into a memory-mapped register. Finally when the other computer fully receives the message, it will trigger a hardware interrupt which will notify the CPU that a message is waiting in the SPI receive buffer. For the end demo, we will use the SPI bus to send message encrypted using our hardware-implemented AES algorithm.

While the actual implementation of this module hasn't been started yet, the core of is will utilize the SPI protocol written during the earlier labs. That will serve as the basis for the subsequent development of the SPI bus functionality within the Orca computer system.

6 AES-128 COPROCESSOR

Currently, we have not started implementation on the AES coprocessor. We need to still refine the exact design specifications for the coprocessor; however, we have built a general understanding of our design based on current literature [2] [3]. First, the coprocessor will implement the 128-bit block size version of the AES standard. Second, the coprocessor will have three memory-mapped buffers, one buffer for the plaintext to be encrypted, one for the cryptographic key, and one for the ciphertext. Third, software can tell the coprocessor to begin encrypting or decrypting by writing to a memory-mapped register. Fourth, when the coprocessor finishes, it will trigger a hardware interrupt on the interrupt controller.

7 SOFTWARE

Currently, we have not started writing any software yet. Within the software components of the Orca computer project, we will focus on the development of a basic monitor program, designed to provide essential functionalities akin to those found in the original Apple 1's WozMon, created by Steve Wozniak. This monitor program will support the display of data located in a specific memory location. Users will be able to input a designated memory address, triggering the program to display the corresponding data. Furthermore, the monitor program facilitates the modification of memory contents by enabling users to specify both the memory location and the new data for overwriting.

In addition, the software component extends to the implementation of an encryption and transmission protocol. Specifically, we are crafting software modules that uses AES for message confidentiality. The encrypted messages will be transmitted via the SPI bus to a recipient Orca computer. Subsequently, the receiving computer will execute a corresponding program to decrypt the message.

As a stretch goal, we can implement cool software such as Tetris, Chess, or a Mandelbrot Set viewer to show that our processor can run a wide variety of general purpose programs.

REFERENCES

[1] Cui, Li, and Wei. 2022. RISC-V Instruction Set Architecture Extensions: A Survey. *IEEE Xplore* (2022). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10049118.

[2] Gomes. 2023. FAC-V: An FPGA-Based AES Coprocessor for RISC-V. 12, 4 (2023), 50. https://www.mdpi.com/2079-9268/12/4/50.

[3] Poncino. 2022. *Design and Programming of a Coprocessor for a RISC-V Architecture.* Master's thesis. Politecnico di Torino. https://webthesis.biblio.polito.it/6589/1/tesi.pdf.