Customized RISC-V in a simple game console

Zdenek Prikryl and Pavel Snobl

Codasip

→ Use case for customization: NES

Nintendo Entertainment System

- Introduced in 1983 in Japan, later in other regions
- 8-bit video game console
- Built around the Ricoh 2A03/7 CPU, a variant of the MOS Technology 6502

MOS Technology 6502

- 8-bit microprocessor with 16-bit address width
- 6 registers (A, X, Y, SP, PC, Flags)
- 56 instructions
- 13 addressing modes

ADC	AND	ASL	BCC	BCS	BEQ	BIT
BMI	BNE	BPL	BRK	BVC	BVS	CLC
CLD	CLI	CLV	CMP	CPX	CPY	DEC
DEX	DEY	EOR	INC	INX	INY	JMP
JSR	LDA	LDX	LDY	LSR	NOP	ORA
PHA	PHP	PLA	PLP	ROL	ROR	RTI
RTS	SBC	SEC	SED	SEI	STA	STX
STY	TAX	TAY	TSX	TXA	TXS	TYA

→ NES emulator: InfoNES*

- Emulates MOS Technology 6502, PPU, APU, etc.
- Written in C++
- Bare metal application: no operating system is needed
- Portable to many platforms including x86 and Arm
- We ported it to RISC-V

*https://github.com/jay-kumogata/InfoNES

→ Codasip Studio

Codasip Studio is a unique collection of tools for fast and easy designing or modification of processors. The design language is used to describe both the ISA and the microarchitecture. Key capabilities include:

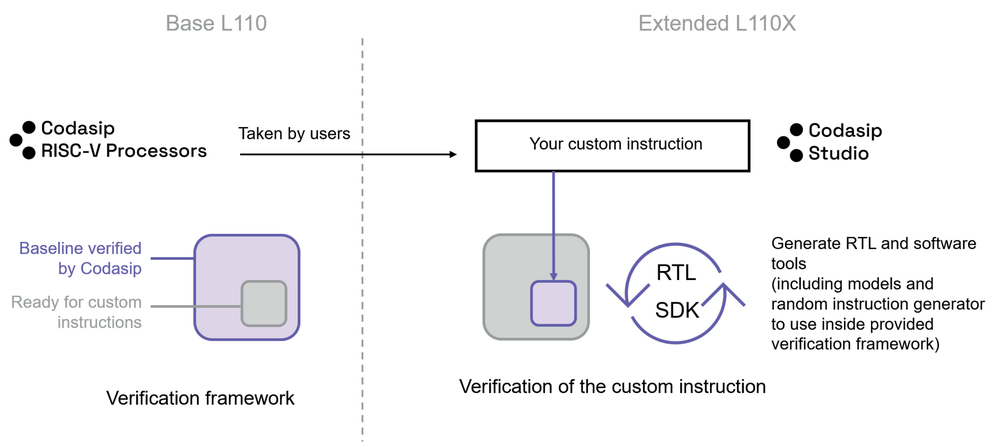
- Rapid architecture exploration and prototyping
- Automated generation of a custom compiler that understands your custom hardware
- Automated generation of power and area-optimized synthesizable, human-readable RTL

→ Codasip L110 processor

Pipeline	In-Order, 3-stage, single-issue pipeline(*)
Memory Interface	Harvard Architecture – 2 Separate AHB Interfaces (Data and Instruction)
Memory Protection	Physical Memory Attributes
Multiplication & Division	Individually Supported
Multiplication	Parallel and Sequential Mode
CLIC	Fast Vectored Interrupt – RISC V standard

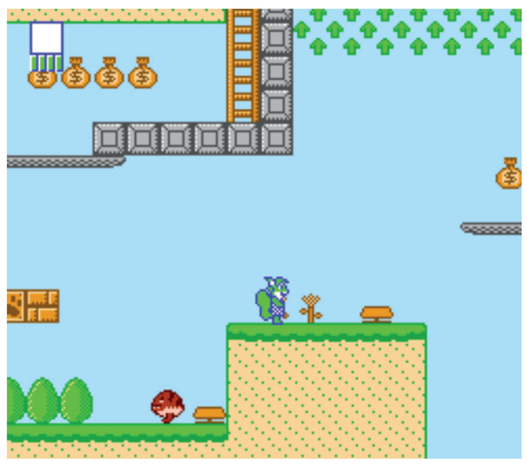
*Optional/configurable

→ Bounded Customization



→ Finding places that deserved optimizations

- We used Codasip Studio to profile the overall performance of the NES emulator (InfoNES) and find hot spots
- We measured data for the homegrown game Nova the Squirrel*
- Three functions take most of the execution time per frame:
  - InfoNES_LoadFrame takes 27% of the execution time
  - InfoNES_DrawLine takes about 7%
  - **K6502_Step takes 52% of the execution time and is the best candidate for optimization!**



*https://github.com/NovaSquirrel/NovaTheSquirreltools

→ Example of added instruction

NES emulator

```
case 0x20: // JSR Abs
  // read immediate from memory
  wA0 = (K6502_Read(PC++) |
        (WORD)K6502_Read(PC) << 8);
  // store PC to stack
  K6502_Write(BASE_STACK + SP--, PC >> 8)
  K6502_Write(BASE_STACK + SP--, PC & 0xff)
  // jump to the new address
  PC = wA0;
  // progress internal time
  g_wPassedClocks += 6;
  break;
```

```
// module that implements M6502
$$(user_cbi_modules += [[0, “m6502”, “SINGLE”, [“jsr”]])
// instruction mnemonic and encoding
$$(user_cbi_encodings += [[“jsr”, “CBI_0DR_1SR”]])
```

CodAL instruction

```
module m_6502_t {
  // resources of M6502
  register uint16 pc;
  register uint8 sp;
  register uint32 clk;
  register_file uint8 stack { size = 256;
    dataport w0, w1 { flag = W; } … };
  …
  // Instructions CBI_ODR_1SR
  export always static void
    compute_0dr_1sr(uint32 op1, uint32 opc) {
    switch (opc) {
      case $$(GET_CBI_ENCODING[“jsr”]):
        stack.w0[sp] = pc[15..8];
        stack.w1[sp - 1] = pc[7..0];
        sp -= 2;
        pc = op1;
        clk += 6;
      break;
      …
```

→ Results

- **Performance gain per frame is 28%**
- Most of the area increase is in the register files used for Stack and Zero Page data storage
- No impact on the frequency, L110 and L110X runs on 50MHz

The energy consumption per frame dropped by 13% even though the design is 18% bigger. Why? Less cycles needed to do the computation and clock gating that Codasip Studio inserts.