MOTIUS

WE R&D.

NEORV32 Tutorial

Motius GmbH

November 19, 2025 20:31 (ccd1cd0)

NEORV32 Tutorial & Customization

Educational & Open-Source Contribution (Q3 2021)

> ✓ **Merged into Official Repository**
>
> Tutorial successfully merged into the official NEORV32 repository, demonstrating custom peripheral integration methodology.

Project Overview

Timeline: Q3 2021

Type: Open-source contribution / Educational

Objective: Build a comprehensive tutorial on how to add custom IP modules to the open-source NEORV32 microcontroller

Status: ✅ Complete and published

Project Goal

Create a tutorial demonstrating custom peripheral integration for the NEORV32 RISC-V processor:

- Show how to design custom IP modules
- Explain integration with NEORV32 architecture
- Document the complete development process
- Contribute to the open-source RISC-V community
- Provide reference for future custom IP development

NEORV32 Overview

What is NEORV32?

- Open-source RISC-V microcontroller
- Highly customizable processor system
- Extensive peripheral set

- Well-documented architecture
- Active community support

Why NEORV32?

- Perfect for learning RISC-V concepts
- Modular design makes adding peripherals straightforward
- Open-source allows complete transparency
- Good starting point for custom SoC development

Custom IP Module: CRC32 Engine

What We Built

Hardware-Accelerated CRC32 Module

A complete Cyclic Redundancy Check (CRC32) module demonstrating custom peripheral integration through the NEORV32's Custom Functions Subsystem (CFS).

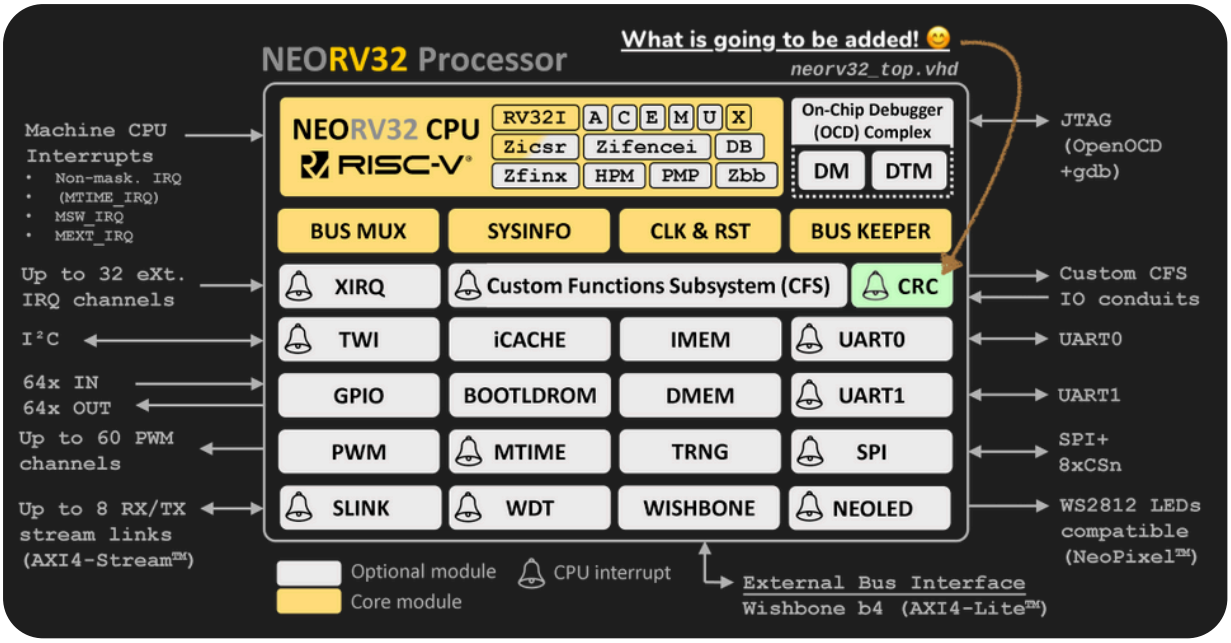
Purpose: - Hardware-accelerated CRC32 calculation for error detection - Common requirement in communication protocols and data integrity - Demonstrates custom peripheral design methodology - Shows integration with processor memory-mapped I/O

Technical Features: - **CRC32 polynomial** - Standard IEEE 802.3 polynomial - **Parallel computation** - Combinatorial XOR logic (not iterative) - **8-bit input interface** - Process data byte-by-byte - **32-bit result output** - Full CRC32 checksum - **Memory-mapped registers** - Direct CPU access - **Runtime detection** - SYSINFO capability flags

---

Technical Implementation

NEORV32 Processor Architecture



The NEORV32 processor with its NEORV32 CPU core, bus infrastructure, and comprehensive peripheral set. Our CRC32 module integrates through the Custom Functions Subsystem (CFS).

Key Components:

- **NEORV32 CPU** - 32-bit RISC-V core with configurable extensions
- **BUS MUX** - Central interconnect with Wishbone bus protocol
- **SYSINFO** - System information and capability flags
- **Custom Functions Subsystem (CFS)** - Integration point for custom peripherals
- **Memory Systems** - IMEM (instruction), DMEM (data), BOOTROM
- **Standard Peripherals** - UART, SPI, GPIO, PWM, TWI, MTIME, WDT, TRNG, etc.
- **Debug Infrastructure** - On-chip debugger (OCD) with JTAG interface

CRC32 Module Integration

Integration Point: Custom Functions Subsystem (CFS)

NEORV32 provides the CFS as a template for custom peripherals. Our CRC32 module occupies dedicated address space outside the standard CFS region.

Memory-Mapped Register Architecture:

Register	Address	Width	Purpose
CRC32_INPUT	`0xFFFFFE78`	8-bit	Data input (write-only)

CRC32_OUTPUT	`0xFFFFFE7C`	32-bit	Computed CRC (read-only)

Address Space Allocation:

```
0xFFFFFE00 ─────┐
                │ Custom Functions Subsystem (CFS)
0xFFFFFE74 ─────┤ 30 registers (reduced from 32)
                │
0xFFFFFE78 ─────┤ CRC32_INPUT (8-bit)
0xFFFFFE7C ─────┤ CRC32_OUTPUT (32-bit)
                │
0xFFFFFE80 ─────┘ PWM and other peripherals
```

Hardware Computation: - Parallel XOR chains process all 32 bits simultaneously - Combinatorial logic (no clock cycles wasted on iteration) - Results available immediately after final byte written - Implements standard CRC32 (IEEE 802.3) polynomial

---

FPGA Platform

Component	Technology
FPGA	AMD Xilinx Artix-7
Board	Nexys A7 development board
Processor	NEORV32 open-source microcontroller
Custom IP	CRC module (designed by Motius)

Integration Process

Files Modified (5 Core Components)

The integration required careful modifications across the NEORV32 system:

neorv32_package.vhd

- Define CRC32 address constants
- Add component declarations
- Create `IO_CRC32_EN` generic flag
- Declare port signals

neorv32_sysinfo.vhd

- Add feature bit (bit 29)
- Enable runtime capability detection
- Allow software to query hardware features

neorv32_top.vhd

- Instantiate CRC32 module
- Route response bus connections
- Add synthesis reporting
- Connect I/O signals

neorv32_test_setup_bootloader.vhd

- Map I/O signals
- Assign LED outputs for visual debugging
- Configure test harness

nexys_a7_test_setup.xdc

- Define physical pin assignments
- Map 8-bit LED output display
- Configure FPGA constraints

Integration Workflow

Recommended Approach:

1. **Prototype in CFS** - Test functionality in Custom Functions Subsystem first
2. **Duplicate Template** - Copy CFS structure for new module
3. **Modify Address Space** - Reduce CFS from 32 to 30 registers (free 8 bytes)
4. **Integrate Response Bus** - Connect module to processor bus architecture
5. **Add SYSINFO Flag** - Enable software capability detection
6. **Validate on Board** - Test with FPGA hardware

Software Driver & Usage

C Driver Implementation

The driver provides simple memory-mapped macros for hardware access:

```c
// Memory-mapped register definitions
#define CRC32_BASE    0xFFFFFE78
#define CRC32_INPUT   (*(IO_REG32 (CRC32_BASE + 0)))
#define CRC32_OUTPUT  (*(IO_REG32 (CRC32_BASE + 4)))

// Example usage: compute CRC32 of a string
void compute_crc32(const char *data) {
    // Write data byte-by-byte
    while (*data) {
        CRC32_INPUT = (uint8_t)*data;
        data++;
    }

    // Read final result
    uint32_t result = ~CRC32_OUTPUT;  // Invert for standard CRC32

    printf("CRC32: 0x%08X\n", result);
}
```

Runtime Capability Detection

```
// Check if CRC32 module is available at runtime
if (SYSINFO_FEATURES & (1 << SYSINFO_FEATURES_IO_CRC32)) {
    // CRC32 hardware is available
    use_hardware_crc32();
} else {
    // Fall back to software implementation
    use_software_crc32();
}
```

Tutorial Content

What the Tutorial Covers

1. Architecture Overview - NEORV32 system architecture - Custom Functions Subsystem (CFS) interface - Wishbone bus protocol - Memory mapping strategies

2. CRC32 Module Design - VHDL implementation with parallel XOR logic - Register interface design - Control logic and data flow - CRC calculation algorithm in hardware

3. Integration Steps - Step-by-step modifications to 5 core files - Address space configuration - Response bus routing - Building and synthesizing

4. Software Development - Device driver implementation in C - Memory-mapped register access - Example applications - Performance comparison (HW vs SW)

5. Testing & Validation - Simulation and verification - FPGA testing on Nexys A7 - Debugging techniques with LED output - Performance benchmarking

---

Key Learning Outcomes

For RISC-V Developers

Custom Peripheral Development: - How to design IP modules for RISC-V systems - Integration with processor bus architectures - Memory-mapped I/O concepts - Hardware/software co-design

For FPGA Engineers

System Integration: - Integrating custom IP with existing systems - Wishbone bus protocol implementation - Timing and resource optimization - FPGA synthesis and place & route

For Students & Researchers

Complete Workflow: - End-to-end custom IP development - Open-source tool usage - Documentation best practices - Community contribution process

---

Open Source Contribution

Publication

GitHub Repository

Official NEORV32: Merged into main repository

Motius Fork: github.com/motius/neorv32

Branch: `add-custom-crc32-module`

Status: Successfully merged into official NEORV32 repository

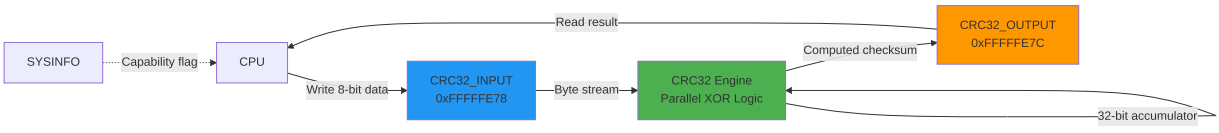
Community Impact

Benefits to RISC-V Community: - Reference for custom IP development - Educational resource for new developers - Demonstrates extensibility of NEORV32 - Lowers barrier to entry for custom peripherals

---

CRC32 Data Flow

The following diagram shows how data flows through the CRC32 module:



Processing Model: 1. CPU writes data bytes sequentially to `CRC32_INPUT` 2. Each byte triggers combinatorial XOR computation 3. Internal state accumulates across all bytes 4. CPU reads final 32-bit result from `CRC32_OUTPUT` 5. Result inverted to match standard CRC32 format

---

Technical Achievements

🔲 Custom IP Development

- ✅ Designed CRC32 module from scratch in VHDL
- ✅ Parallel XOR logic (combinatorial, not iterative)
- ✅ Memory-mapped register interface
- ✅ Developed device driver in C
- ✅ Hardware-software co-design

💉 Integration Success

- ✅ Successfully integrated with NEORV32 core
- ✅ Modified 5 system files systematically
- ✅ Validated on Nexys A7 FPGA
- ✅ Runtime capability detection via SYSINFO
- ✅ Documented complete process

🎓 Educational Value

- ✅ Created comprehensive tutorial
- ✅ Demonstrated extensibility methodology
- ✅ Provided working example code
- ✅ Merged into official repository
- ✅ Contributed to RISC-V community

Performance Benefits

- ✅ Hardware acceleration vs software
- ✅ Zero-cycle parallel computation
- ✅ Immediate results after final byte
- ✅ No CPU overhead during calculation
- ✅ Validated with LED output

Skills Demonstrated

HDL Design

- Verilog/VHDL for custom IP
- Finite state machines
- Bus protocol implementation
- Timing constraints

System Integration

- RISC-V processor architecture
- Wishbone bus protocol
- Memory mapping
- Peripheral interconnect

Software Development

- Device driver development
- Embedded C programming
- Hardware abstraction layers
- Performance optimization

Documentation

- Technical writing
- Tutorial creation
- Code documentation
- Community contribution

Impact on Motius

RISC-V Foundation

This project established our: - Understanding of RISC-V architecture - Custom IP development methodology - Open-source contribution practices - Educational material creation

Building Block

The skills and knowledge gained here directly enabled: - eMil project (Western Digital RISC-V) - Game Engine chip (custom IP to silicon) - Commercial Platform (commercial RISC-V integration)

Open Source Credibility

- Demonstrated technical capability to open-source community
- Established reputation in RISC-V ecosystem
- Created reusable methodologies
- Built foundation for future projects

Lessons Learned

Technical

What worked well: - NEORV32 architecture is well-designed for customization - Wishbone bus is straightforward to implement - Open-source tools are production-ready - Community support is excellent

Challenges: - Timing closure requires careful design - Resource utilization must be optimized - Documentation takes significant effort - Testing and validation is time-intensive

Process

Best Practices: - Start with simple examples - Incremental integration and testing - Comprehensive documentation from start - Community engagement throughout

---

Next Steps from This Project

This tutorial project led to:

1. **eMil (2022)** - More complex RISC-V integration with WD EH1 core
2. **Game Engine Chip (2022-2024)** - Custom IP → manufactured silicon
3. **Commercial Platform (2025)** - Commercial RISC-V platform development

Progression:

```
NEORV32 Tutorial → eMil Research → Game Engine Manufacturing → Commercial Platform
```

← eMil Project

Back to Case Studies