Fixed-Point MAC Peripheral Design Proposal

Design Specification

Our team proposes to design a fixed-point Multiply-Accumulate (MAC) peripheral specifically optimized for the 8-bit RISC CPU architecture. This peripheral will enhance the computational capabilities of the base CPU by providing hardware acceleration for multiplication and accumulation operations.

Key Design Features:

- Implementation of an 8×8→16-bit fixed-point MAC unit with single-cycle operation
- Custom memory-mapped interface with optimized register layout
- Configurable accumulator behavior with saturation protection
- Status flags for overflow detection and operation completion
- Debug output ports for real-time monitoring

System Implementation

MAC Architecture

Our MAC implementation will feature:

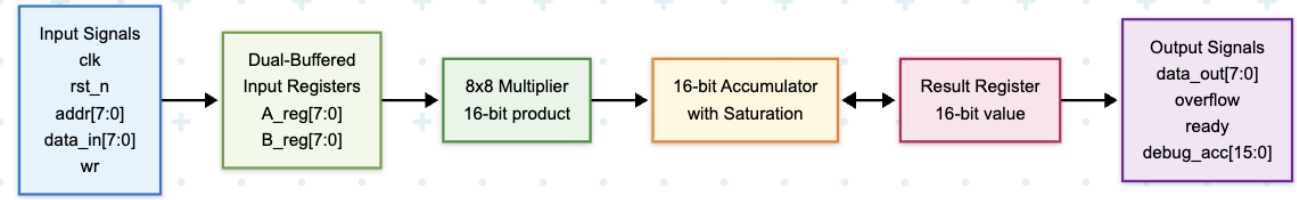
1. **Enhanced Input Buffering**: Dual-buffered input registers to allow loading of next operands while current multiplication is in progress
2. **Configurable Accumulation Modes**:
3. Standard mode: acc += A×B
4. Clear-and-multiply mode: acc = A×B
5. Saturating accumulation to prevent overflow
6. **Status Register**: Provides operation status including overflow flag and ready signal

Memory-Mapped Interface

Our memory map design uses five addresses:

- 0xF0: Input Register A (write-only)
- 0xF1: Input Register B (write-only)
- 0xF2: Control Register (bit 0: start, bit 1: reset, bit 2: mode select)
- 0xF3: Result Low Byte (read-only)
- 0xF4: Result High Byte and Status Flags (read-only, bits 0-7: high byte, bit 8: overflow)

Block Diagram



I/O Definition Table

Signal Name	Direction	Width (bits)	Description
clk	Input	1	System clock
rst_n	Input	1	Active-low reset
addr	Input	8	Address bus
data_in	Input	8	Data input bus
wr	Input	1	Write enable
data_out	Output	8	Data output bus
overflow	Output	1	Overflow indicator

Signal Name	Direction	Width (bits)	Description
ready	Output	1	Result ready flag
debug_acc	Output	16	Debug: Current accumulator value

Test Plan

Software Simulation Environment

- RTL simulation using Verilog/SystemVerilog testbench
- Waveform analysis with GTKWave or similar tool
- Python scripts for test vector generation and result verification

Simulation Test Approach

1. **Functional Verification**
2. Register interface testing with read/write operations
3. Basic multiplication with various input combinations (0×0, 1×1, 255×255)
4. Accumulation mode verification with sequential operations
5. Overflow and saturation logic validation
6. **Timing Analysis**
7. Clock cycle verification at 50MHz (simulated)
8. Operation latency measurement
9. Setup and hold time verification
10. **Corner Case Testing**
11. Boundary value testing
12. Reset during operation
13. Back-to-back operations

Test Implementation

- Automated testbench with self-checking assertions
- Comprehensive test vectors covering normal and edge cases

- Simulation logs and waveform captures for documentation
- Coverage analysis to ensure complete design verification