Bit Manipulation Unit (BMU)

Verification Plan

Lujain Aburajab

9-Dec-2025

Contents

List of Tables and Figures

Abbreviations *Table 1-1: Abbreviations table*

Abbreviation	Description
BMU	Bit Manipulation Unit
RTL	Register Transfer Level
RISC-V	Reduced Instruction Set Computing - V (version)
Zbb	RISC-V Bit Manipulation Extension for instructions like CLZ, CTZ, CPOP, MIN, MAX, etc.
Zbs	RISC-V Bit Manipulation Extension for instructions like BSET, BCLR, BINV, BEXT
Zbp	RISC-V Bit Manipulation Extension for instructions like ROL, ROR, PACK, PACKU, PACKH, GREV, GORC
Zba	RISC-V Bit Manipulation Extension for instructions like SH1ADD, SH2ADD, SH3ADD
clk	Clock (input)
rst_l	Active-low reset (input)
scan_mode	Scan test mode control (input)
valid_in	Instruction valid flag (input)
ap	Operand Control Structure (decoded instruction control signals
csr_ren_in	CSR (Control and Status Register) read-enable signal (input)
csr_rddata_in	CSR read data (input)
a_in	Operand A (input)
b_in	Operand B (input)
result_ff	Final computed result (output)
error	Error indicator (output)
LZD	Leading Zero Detection (used for CLZ operation)
CPOP	Count Set Bits / Population Count
siext_b	Sign Extend Byte
ORC.B	OR-compression for bytes
ROR	Rotate Right
PACK	Combine operands A and B in different formats
PACKU	Unsigned packing instruction
PACKH	Halfword packing instruction
GREV	Byte-Reverse Register operation
AP.lor	Logical OR operation control signal
AP.lxor	XOR operation control signal
AP.srl	Right Logical Shift control signal
AP.sra	Right Arithmetic Shift control signal
AP.ror	Rotate Right control signal

AP.binv	Bit Inverse operation control signal
AP.sh2add	Shift Left by 2 and Add operation control signal
AP.sub	Subtract operation control signal
AP.slt	Set on Less Than operation control signal
AP.cpop	Count Set Bits / Population Count operation control signal
AP.ctz	Count Trailing Zero Bits operation control signal
AP.siext_b	Sign Extend Byte operation control signal
AP.max	Maximum operation control signal
AP.pack	Pack operation control signal
AP.grev	Byte-Reverse Register operation control signal
AP.csr_write	Control and Status Register Write operation control signal
AP.csr_imm	Immediate CSR operation control signal
SHxADD	Shift Left by x and Add operation
SRL	Shift Right Logical (Right Logical Shift)
SRA	Shift Right Arithmetic (Right Arithmetic Shift)
CTZ	Count Trailing Zeros
MAX	Maximum comparison operation
BINV	Bit Inversion (bit manipulation)
SH2ADD	Shift Left by 2 and Add
AP.land	Logical AND operation control
AP.lxor	XOR operation control signal

Introduction

Goals/Purpose of the document

This document outlines a verification plan for the Bit Manipulation Unit (BMU), which is designed to operate as part of a larger processor architecture. The purpose of this plan is to verify that the BMU functions correctly under all supported operating conditions, including normal operation, boundary cases, and invalid or conflicting instruction scenarios, and that it properly reports error conditions when required.

Scope of verification

The verification scope includes the BMU top module and its associated internal logic blocks responsible for arithmetic, shifting, bit manipulation, comparison, and CSR-related operations. The verification covers functional testing of all relevant input and output signals, internal data paths, and interactions between control signals and datapath logic. It is restricted to the following BMU operations: OR, ORN, XOR, XNOR, SRL, SRA, ROR, BINV, SH2ADD, SUB, SLTU, SLT, CTZ, CPOP, siext_b, MAX, PACK, and GREV, as well as supported CSR operations.

The verification will check the correct functionality of control signals for each supported operation and ensure correct result generation under normal and boundary conditions. It will also verify proper error detection for unsupported opcodes, invalid or conflicting control signal combinations, and CSR read conflicts. In addition, random test cases will be used to validate robustness and ensure overall correctness and compliance with the BMU specification for the selected operations.

Assumptions and constraints

Constraints:

- The BMU does not support multi-cycle handshake or back-pressure.
- The verification must assume single-cycle acceptance of each valid instruction.
- All input operands (a_in, b_in) are strictly 32-bit values.

Assumptions:

- The design and verification environment uses UVM .
- Functional coverage goals are achievable and measurable.
- Single active instruction per cycle
- Synchronous design behavior

Design Overview

The Bit Manipulation Unit (BMU) is responsible for performing bit-level arithmetic, logical, shifting, rotation, counting, sign-extension, and data packing operations, compliant with a selected subset of the RISC-V BitManip extension. The BMU supports only the operations covered in this verification plan, including OR, ORN, XOR, XNOR, SRL, SRA, ROR, BINV, SH2ADD, SUB, SLT, SLTU, CTZ, CPOP, siext_b, MAX, PACK, and GREV. It operates as part of the processor's integer execution path and processes operands based on pre-decoded control signals provided by the decode stage.

The BMU receives operand inputs and control signals that select the required operation and mode of execution. Based on these inputs, it performs the requested bit manipulation operation and produces a result along with an error indication if invalid or conflicting conditions are detected. The BMU also supports CSR-related data handling and enforces guard conditions to ensure correct and exclusive operation execution.

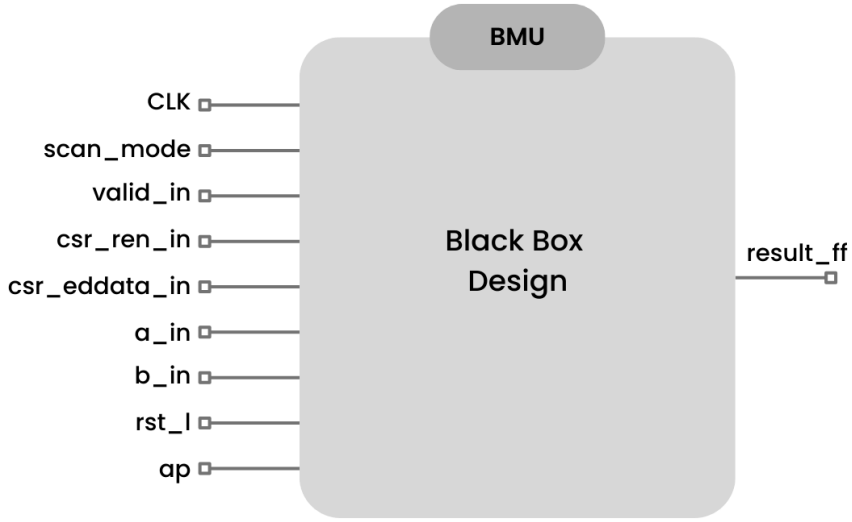
The top-level BMU module is responsible for the following functions:

- **Operation Execution:** Perform the selected bit manipulation operation based on the asserted control signals and input operands.
- **Control Signal Interpretation:** Interpret pre-decoded control signals (e.g., operation enables, mode bits, and CSR controls) to select the appropriate internal datapath.
- **Guard and Exclusivity Enforcement:** Ensure that only one operation is active at a time, required mode bits are correctly set, and CSR read operations do not conflict with bit manipulation execution.
- **Error Detection:** Assert the error signal in the presence of unsupported operations, invalid control signal combinations, or CSR conflicts.

Key Features

- **Operation Execution:** The BMU executes bit manipulation operations based on pre-decoded control signals, supporting OR, ORN, XOR, XNOR, SRL, SRA, ROR, BINV, SH2ADD, SUB, SLT, SLTU, CTZ, CPOP, siext_b, MAX, PACK, and GREV.
- **Control Signal Interpretation:** The BMU interprets incoming control signals to select the appropriate internal functional logic, while enforcing exclusivity of operation enables and validating required mode bits.
- **Operand Processing:** The BMU operates on input operands provided by the execution pipeline and produces a computed result without performing register selection or instruction decoding.
- **CSR Interaction Handling:** The BMU supports CSR-related data paths and ensures that bit manipulation operations do not conflict with CSR read operations.
- **Error Detection:** The BMU detects invalid or unsupported control signal combinations, missing mode requirements, and CSR conflicts, and asserts an error indication to prevent incorrect result propagation.

Block diagram and interface



Interfaces include operand inputs, result outputs, control signals for selecting operations, flush signals for pipeline control, and status outputs for performance monitoring.

The interface and signals details are listed below in the table:

Signal	Width	Direction	Description
clk	1 bit	input	Clock signal to the module
rst_l	1 bit	input	Active-Low synchronous reset , reset all internal registers of the design to its initial state
a_in	32 bit	input	Represents the first operand of BMU module
b_in	32 bit	input	Represents the second operand of BMU module
scan_mode	1 bit	input	Scan test mode control
valid_in	1 bit	input	Instruction valid flag , indicate if the input instruction is valid or not
ap	Struct	input	Decoded instruction control signals (supported instructions by the BMU)
csr_ren_in	1 bit	input	CSR read-enable
csr_rddata_in	32 bit	input	CSR read data
result_ff	32 bit	output	Final computed result
error	1 bit	output	Error Indicator

Verification Strategy

Methodology

UVM will be used to develop a structured, scalable, and reusable verification environment for the Bit Manipulation Unit (BMU), comprising standard UVM components such as drivers, monitors, a reference model with scoreboard, and a functional coverage collector to support both directed and constrained-random stimulus generation. This environment will validate the functionality of the BMU for the selected operations OR, ORN, XOR, XNOR, SRL, SRA, ROR, BINV, SH2ADD, SUB, SLT, SLTU, CTZ, CPOP, siext_b, MAX, PACK, and GREV, while verifying correct result generation under normal operation, enforcement of guard and exclusivity conditions, an

Selection of Verification Tests

Verification tests will cover all relevant aspects of the BMU, including the following:

1. **logical Operations**
   This section defines logical operations performed by the BMU, primarily controlled by the ap.op field. The operation mode depends on the ap.Zbb extension bit to be asserted and requires all

other control fields to be disserted.

- **Standard Bitwise OR Operation**
  - A | B for random values
  - A = 0, B = any
  - A = all ones, B = random
  - A = B
  - Edge cases: MSB=1, LSB differences
  - Guard test: other ap.* bits set → error=1
  - CSR conflict: csr_ren_in=1 → error=1
- **Standard Bitwise Invertor OR Operation**
  - A OR (~B)
  - B = 0, 1, alternating bits
  - Mode bit (ap.zbb) ON vs OFF
  - Guard conditions + error scenarios
- **Standard Bitwise XOR Operation**
  - A ^ B
  - Identical operands → result = 0
  - Alternating patterns
  - MSB = 1
  - Error when overlapping operation
- **Standard Bitwise Invertor XOR Operation**
  - A ^ ~ B
  - Patterns: all zeros, all ones, 1010…, 11110000…
  - Ensure correct inversion

2. Shifting and Masking Operation

The Shifting and Masking operations include logical shifts, arithmetic shifts, rotates, and bit-level operations. Each operation is enabled by a dedicated control signal in the ap register. Only one shifting/masking operation should be active at a time, and all other ap.* fields must be 0, unless explicitly required by the operation.

- **Right Logical Shift Operation (SRL)**
  - Shift amounts: 0, 1, 5, 31
  - Large shift: b_in > 31 (should use b_in[4:0])
  - A = 0, A = max, negative value patterns
  - Guard violations
  - CSR conflict
- **Right Arithmetic Shift Operation (SRA)**
  - Positive vs negative numbers
  - A = 0x80000000 (sign bit=1)
  - Shift amount = 0, 31
  - Shift amount > 31
  - Ensure sign bit replicates properly
- **Rotate Right Operation (ROR)**
  - Rotation by 0, small values, 31
  - Rotation by >31 (masked)
  - All patterns (zeros, ones, alternating)
  - Illegal activation tests

- **Bit Inverse Operation (BINV)**
  - Flip single bit at index b_in[4:0]
  - Index = 0, 15, 31
  - Out-of-range index (>31)
  - Operand = all 0, all 1
  - Multiple operations → error
- **Shift Left by 2 and Add (SH2ADD)**
  - ap.sh2add = 1, ap.zba = 1 (legal)
  - ap.zba = 0 → must trigger error
  - A = random, B = random
  - Overflow scenarios
  - Corner cases: A or B = 0

3. Arithmetic Operations

This section verifies arithmetic functionality supported by the BMU.

- **Subtraction (a – b)**
  - Positive cases
  - Negative results
  - Overflow underflow scenarios
  - Large operand differences
  - A = B → result = 0
  - ap.zba must be 0 (else error per spec)

4. Bit Manipulation Operations

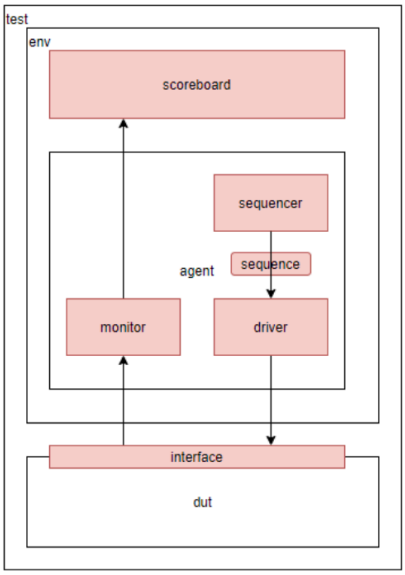
This section verifies comparison, counting, extension, and data manipulation operations supported by the BMU.

- **Set on Less Than (Unsigned SLT)**
  - Negative < Positive
  - Positive < Negative
  - Equal values
  - INT_MIN vs INT_MAX
  - Random signed comparisons
- **Set on Less Than (SLT signed == default)**
  - 0 < 1
  - Large unsigned values
  - Unsigned wrap-around
  - A = B
  - Edge cases (0xFFFFFFFF vs small numbers)
- **Count Trailing Zero Bits (CTZ)**
  - Input = 0
  - Inputs with single 1-bit at positions: 0, 5, 31
  - Alternating patterns
  - Random values
- **Count Set Bits / Population Count (CPOP)**
  - All 0 → count = 0
  - All 1 → count = 32
  - Random patterns
  - Large mixed values

- **Sign Extend Byte (siext_b)**
  - Sign bit of byte = 0 ⟶ zero-extend
  - Sign bit of byte = 1 ⟶ extend ones
  - Extreme values (0x7F, 0x80)
- **Maximum (max)**
  - Signed max
  - A=B
  - Positive vs Negative
  - Large magnitudes
- **Pack (pack)**
  - Combine A[15:0] with B[15:0]
  - A or B = 0
  - Random values
  - Verify byte ordering
- **Byte-Reverse Register(grev)**
  - Valid mode: b_in[4:0] = 24
  - Invalid mode ⟶ error
  - Patterns:
    1. 0x00000000
    2. ○ 0xFFFFFFFF
    3. ○ Alternating bytes
    4. ○ Random

5. **CSR operations**
   - **CSR Write**
     - Immediate mode (csr_imm = 1)
     - Register mode (csr_imm = 0)
     - Correct output of result
   - **CSR Read Enable Conflict**
     - csr_ren_in=1 + any BitManip instruction ⟶ error
     - csr_write + ap.* instruction active ⟶ error
6. **Unsupported Opcode Handling**
   This section verifies proper error detection and signaling for unsupported operations, invalid control signal combinations, missing mode bits, and violations of guard and exclusivity conditions.
7. **Random cases**
   This section verifies BMU robustness using constrained-random stimulus across operands, operation enables, mode bits, and error scenarios to uncover corner cases not explicitly covered by directed tests.

Testbench structure



Tests

For each operation I will implement a test which contains all the pre-mentioned sequences.

Coverage Targets

- Code (line) coverage
- Functional coverage
- Toggle coverage
- Path coverage
- Cross coverage (advanced , check multiple options once and for testing direct sequence ..)

Verification Extra Details

- **Components:** driver, monitor, sequencer, sequence_item, agent, environment,scoreboard, subscriber.
- **Interfaces:** middle-ware interface between the DUT and the top testbench module
- **Tools and technologies:**
  - MobaxTerm Server
  - Integrated Metrics Center (IMC)
  - Cadence Verisium for debugging
  - Visual Studio Code
- **Regression Tests:**
  - Create Regression test to test all cases I have set up and run them together instead of running them individually in the top testbench module.
- **Coverage Reports:** will be attached later
- **Verification Results:** will be attached later